UNIVERSIDADE DE SÃO PAULO
FACULDADE DE FILOSOFIA, LETRAS E CIÊNCIAS HUMANAS

JOSÉ LOPES MOREIRA FILHO

Linguística e Computação em diálogo
para análise de textos e criação de atividades de leitura em língua inglesa
versão corrigida

São Paulo

2015

UNIVERSIDADE DE SÃO PAULO
FACULDADE DE FILOSOFIA, LETRAS E CIÊNCIAS HUMANAS

JOSÉ LOPES MOREIRA FILHO

Linguística e Computação em diálogo
para análise de textos e criação de atividades de leitura em língua inglesa
versão corrigida

Tese apresentada ao Programa de Pós-Graduação em
Linguística do Departamento de Linguística da
Faculdade de Filosofia, Letras e Ciências Humanas
da Universidade de São Paulo para obtenção do
título de Doutor em Letras

Área de Concentração: Semiótica e Linguística
Geral

Orientadora: Profª Dra. Zilda Maria Zapparoli

Coorientador: Prof. Dr. João José Neto

São Paulo

2015

MOREIRA FILHO, José Lopes. Linguística e Computação em diálogo para análise de textos e criação de atividades de leitura em língua inglesa

Tese apresentada ao Programa de Pós-Graduação em Linguística do Departamento de Linguística da Faculdade de Filosofia, Letras e Ciências Humanas da Universidade de São Paulo para obtenção do título de Doutor em Letras

Orientadora: Profª Dra. Zilda Maria Zapparoli

Coorientador: Prof. Dr. João José Neto

Aprovado em: 06/05/2015

Banca Examinadora

Profª Dra. Zilda Maria Zapparoli

Instituição: FFLCH/USP

Prof. Dr. João José Neto

Instituição: EP/USP

Profª Dra. Rosinda de Castro Guerra Ramos

Instituição: PUC-SP - Externo

Prof. Dr. Thiago Alexandre Salgueiro Pardo

Instituição: ICMC/USP

Profª Dra. Neide Ferreira Gaspar

Instituição: Externo

DEDICATÓRIA

A Deus.

A minha esposa Luciana e filhos pela compreensão e carinho.

Aos meus irmãos Witalo e Aline pela motivação.

Aos meus pais, José e Rosa, que sempre me apoiaram.

AGRADECIMENTOS

À Profª Dra. Zilda Maria Zapparoli, minha orientadora, que sempre esteve do meu lado, em todos os momentos com seu apoio incondicional, com a sabedoria que transcende os conhecimentos acadêmicos, que me proporcionou experiências intelectuais e de vida únicas. Agradeço, com muito orgulho e felicidade, a pessoa em que vejo motivos para continuar neste caminho acadêmico.

Ao Profº Dr. João José Neto, meu coorientador, uma excelente pessoa, com visão e conhecimentos ímpares, agradeço pela motivação, conselhos metodológicos e contribuições a este trabalho, por oportunidades de aprendizado no WTA, além da paciência em nossas reuniões.

Ao Profº Dr. Thiago Alexandre Salgueiro Pardo pela disciplina de Introdução ao PLN e pelas valiosas contribuições ao trabalho escrito na qualificação.

À amiga Profª Dra. Neide Ferreira Gaspar pela confiança, uma pessoa que faz a diferença onde quer que esteja, e por acreditar no potencial das pessoas.

À amiga Me. Lilian por ser uma pessoa maravilhosa e por contribuir para meu caminho.

A todos os professores que motivaram e contribuíram para o meu avanço nos estudos.

À Secretaria da Educação do Estado de São Paulo (SEE-SP), pela concessão da bolsa de apoio financeiro do programa Bolsa Mestrado/Doutorado para realização da pesquisa.

À comissão regional do programa Bolsa Mestrado/Doutorado da Diretoria de Ensino Leste 3, o Supervisor Sérgio Vieira de Sousa, o Supervisor José Antonio Moreira da Silva e à Dirigente Regional de Ensino Maria Helena Tambellini, Faustino pelo apoio.

À Supervisora Edna Martins Guellere e à Diretora Patrícia Mendes do Núcleo Pedagógico da Diretoria de Ensino Leste 3 pelo apoio.

A todos os amigos do Núcleo Pedagógico da Diretoria de Ensino Leste 3 pelo auxílio, carinho e motivação.

Aos professores de inglês e colegas PCNP de outras Diretorias pela contribuição na pesquisa.

A toda minha família e a Deus.

“Resolveu Daniel, firmemente, não contaminar-se com as finas iguarias do rei, nem com o vinho que ele bebia; então, pediu ao chefe dos eunucos que lhe permitisse não contaminar-se” (Daniel, 1:8).

RESUMO

MOREIRA FILHO, J. L. **Linguística e Computação em diálogo para análise de textos e criação de atividades de leitura em língua inglesa**. 2015. 324 f. Tese (Doutorado) - Faculdade de Filosofia, Letras e Ciências Humanas, Universidade de São Paulo, São Paulo, 2015.

A coleta e a exploração de *corpora* para a criação de atividades é um tema cada vez mais recorrente, uma vez que a prática visa a garantir materiais de ensino que privilegiam a língua em uso. A disponibilidade de instrumentação computacional para análise de *corpora* é enorme, assim como o seu potencial para o ensino de línguas. Apesar dos benefícios, o uso desses recursos ainda não é uma realidade para a maioria dos professores, principalmente fora do contexto acadêmico. Nesta tese, desenvolve-se um sistema de análise de textos e *corpora* e de criação automática de atividades de leitura e ensino de léxico-gramática em língua inglesa, com base na investigação das possibilidades de construção e exploração de *corpora* da Linguística de Corpus em diálogo com métodos e ferramentas de trabalho das áreas do Processamento de Línguas Naturais e Aprendizado de Máquina. Os objetivos da pesquisa estão relacionados a um estudo inicial que teve como produto final um *software desktop* para a preparação semiautomática de atividades de leitura em inglês, segundo o conceito de atividade-padrão, para facilitar a produção de materiais baseados em *corpora*. A proposta da tese concentra-se na automatização das análises linguísticas para a criação automática de atividades de ensino. Para tanto, a linguagem de programação *Python*, com a biblioteca *Natural Language Toolkit (NLTK)*, foi utilizada na construção de módulos de análise de texto e *corpora*. Os principais *corpora* foram: a. o *corpus* de referência *British National Corpus (BNC)*; b. o *corpus* de referência Floresta Sinta(c)tica do *NLTK*; c. o *corpus* de referência MacMorpho do *NLTK*; d. um *corpus* de estudo/treinamento com 135 textos de anúncios de emprego em inglês da Internet; e. um *corpus* de estudo/treinamento com 771 textos de divulgação científica das revistas eletrônicas *Scientific American* e *NewScientist*. A partir das análises automáticas programadas, foram criados modelos em XML, que extraem informações de texto e *corpus* para a criação de atividades. Uma interface gráfica foi desenhada para implementação do sistema, por meio das linguagens *PHP*, *JavaScript*, *HTML* e *CSS*, e disponibilizada *online* para a avaliação de possíveis usuários finais. A análise das avaliações mostrou-se positiva tanto em relação a aspectos da interface quanto a informações geradas pelo sistema. Os resultados obtidos no trabalho são significativos, pois sugerem que o sistema proposto, que permite a automatização de análises de texto e *corpora* para a criação automática de atividades didáticas de leitura e ensino de léxico-gramática em língua inglesa, apresenta um diferencial em relação a ferramentas disponíveis para análise de textos: fornece análises mais apuradas para a tarefa de elaboração de atividades didáticas, quando comparado, por exemplo, a programas como concordanciadores. Dentre as contribuições do trabalho, destacam-se o percurso do desenvolvimento do sistema como parte integrante da pesquisa, o diálogo entre as Humanidades – a Linguística e a Língua Inglesa – e as Ciências Exatas – a Computação, com o Processamento de Línguas Naturais e o Aprendizado de Máquina –, e a automatização de tarefas de análise de textos para fins de criação de materiais pedagógicos para o ensino de línguas.

Palavras-chave: Linguística de *Corpus*; Processamento de Línguas Naturais; Aprendizado de Máquina; Leitura; Ensino de Línguas.

ABSTRACT

MOREIRA FILHO, J. L. **Dialogue between Linguistics and Computing to analyze texts and create reading activities in English**. 2015. 324 f. Tese (Doutorado) - Faculdade de Filosofia, Letras e Ciências Humanas, Universidade de São Paulo, São Paulo, 2015.

Collecting and analyzing *corpora* in order to create activities is a topic that has risen by leaps and bounds, since practice is aimed at ensuring that the teaching material is focused on the language currently in use. There is a great amount of computer devices available for *corpora* analysis, and its potential for teaching languages is evident. Even though they present huge benefits, most of the teachers do not take advantage of these resources yet, chiefly when they do not take part in the academic environment. Through this thesis, a system for analysis of text and *corpora* and automatic creation of reading and English lexical-grammar teaching activities is developed, based on the investigation of possible ways of collecting and analyzing *corpora* from Corpus Linguistics, in partnership with methods and working tools related to Natural Language Processing and Machine Learning. The purpose for this research is related to an initial study that resulted in a desktop software that semi-automatically prepares reading activities in English, based on the standard-activity concept, in order to facilitate the creation of *corpora*-based material. This thesis puts forward the automatization of linguistic analyses so that teaching activities may be created automatically. For this purpose, the programming language *Python*, together with the *Natural Language Toolkit* (*NLTK*) library, was used for the creation of text-analysis and *corpora* modules. The main *corpora* used were: a. *British National Corpus* (*BNC*) reference *corpus*; b. Floresta Sinta(c)tica reference *corpus* from *NLTK*; c. MacMorpho reference *corpus* from *NLTK*; d. a study/training *corpus* with 135 texts of job announcements in English taken from the Internet; e. a study/training *corpus* with 771 texts of scientific disclosure of the following e-magazines: *Scientific American* and *NewScientist*. As from the scheduled automatic analyses, some XML models were created; they gather information about texts and *corpus* in order to create activities. A graphic interface was designed with the purpose of implementing the system by means of *PHP*, *JavaScript*, *HTML* and *CSS languages* and made available online, so that it could be evaluated by potential final users. The evaluations' analysis had a positive outcome, both in relation to interface aspects and information provided by the system. The outcome of this work is meaningful, since this new system, which allows for the automatization of text and *corpora* analyses in order to create reading and English lexical-grammar activities, has a differential with regard to the tools available to analyze texts: it allows for more accurate analyses for the task of creating teaching activities, when it is compared, for instance, to softwares such as the concordance ones. Among the contributions to this work, it is important to highlight the system development process as an integral part of the research, the dialog between the Humanities – Linguistics and English language – and the Exact Sciences – Computing, with the Natural Language Processing and the Machine Learning –, as well as the automatization of text-analysis tasks in order to create teaching material for language teaching.

Keywords: *Corpus* Linguistics; Natural Language Processing; Machine Learning; Reading; Language Teaching.

LISTA DE FIGURAS

Figura 1 - Programas mais utilizados por linguistas de <i>corpus</i>	32
Figura 2 - Resultados da guia lista de palavras e opções de classificação	34
Figura 3 - Exemplo de palavras marcadas para lematização.....	35
Figura 4 - Palavras-chave de um <i>corpus</i> de anúncios de emprego em inglês	36
Figura 5 - Exemplo de linhas de concordância no <i>Kitconc</i>	37
Figura 6 - Concordâncias da palavra ‘ <i>resume</i> ’	38
Figura 7 - Lista de colocados da palavra <i>resume</i> na posição N-1 no <i>Kitconc</i>	39
Figura 8 - Gráfico de dispersão da palavra ‘ <i>RESUME</i> ’ no <i>Kitconc</i>	39
Figura 9 - Utilização do recurso de filtro em dados anotados.....	40
Figura 10 - Exemplo de texto etiquetado	42
Figura 11 - Ciclo do desenvolvimento de anotação	54
Figura 12 - Exemplo de anotação morfossintática	54
Figura 13 - Exemplo de anotação sintática do <i>corpus Penn Treebank</i>	55
Figura 14 - Exemplo de anotação semântica.....	55
Figura 15 - Divisão do <i>corpus Brown</i> para o desenvolvimento do etiquetador	57
Figura 16 - Esquema de funcionamento do primeiro protótipo	64
Figura 17 - Esquema das etapas básicas para criação de atividade de leitura	66
Figura 18 - Exemplo de itens do léxico para itemização	73
Figura 19 - Exemplo de expressões regulares para itemização.....	74
Figura 20 - Exemplo parcial de um texto itemizado	75
Figura 21 - Exemplos de atributos para classificação de palavras desconhecidas	81
Figura 22 - Declaração de uma classe para criação de um etiquetador.....	81
Figura 23 - Função ‘ <i>tag</i> ’ da classe ‘ <i>POSTagger</i> ’	82
Figura 24 - Fórmula para determinar o valor de chavidade de uma palavra	85
Figura 25 - Gráfico para comparação dos valores de frequência e chavidade	86
Figura 26 - Exemplo parcial de análise de texto com o programa <i>Vocabprofile</i>	87
Figura 28 - Interface de busca do Dicionário Michaelis <i>online</i> - inglês/português.....	91
Figura 29 - Parte da função que baixa os endereços de cada palavra.....	92
Figura 30 - Exemplo de termo baixado	93
Figura 31 - Implementação da fórmula <i>Dice</i> em <i>Python</i>	94
Figura 32 - Exemplo de palavras cognatas anotadas para treinamento e teste.....	95
Figura 34 - Exemplo de gramática livre de contexto e visualização em árvore.....	97
Figura 35 - Exemplo de anotação utilizada para os dados de treinamento	98
Figura 36- Exemplo de autômato para leitura de grupos nominais.....	99
Figura 37 - Exemplo de estrutura XML para armazenar dados	103
Figura 38 - Exemplo de estrutura XML preenchida com dados de análises.....	105
Figura 39 - Gráfico de dispersão da lista de frequência do <i>corpus</i> de anúncios de emprego.....	109
Figura 40 - Agrupamentos na dispersão da frequência de itens por texto.....	114
Figura 41 - Proporção de etiquetas morfossintáticas em categorias no <i>corpus</i> de estudo.....	116
Figura 42 - Comparação de resultados de extração de palavras-chave	119
Figura 43 - Resultado parcial de palavras-chave em gráfico de nuvem.....	120
Figura 44 - Etiquetas morfossintáticas dos 25 grupos verbais mais frequentes	124
Figura 45 - Linhas de concordância da palavra ‘ <i>seeking</i> ’	125
Figura 46 - Dispersão da palavra ‘ <i>seeking</i> ’	126

Figura 47 - Linhas de concordância da palavra 'salary'	127
Figura 48 - Dispersão da palavra 'salary'	127
Figura 49 - Exemplo de anotação para registro da posição de itens	128
Figura 50 - Dispersão de palavras em faixas para comparação.....	129
Figura 51 - Gráficos de nuvem para cada agrupamento de palavras.....	131
Figura 52 - Exemplo de modelo de atividade em XML.....	142
Figura 53 - Exemplo de tradução de modelo de atividade	143
Figura 54 - Níveis do sistema para criação automática de atividades	150
Figura 55 - Arquitetura do sistema.....	151
Figura 56 - Página inicial do Readler.....	153
Figura 57 - Formulário para entrada de texto e opções do usuário	153
Figura 58 - Editor de atividades do Readler.....	154
Figura 59 - Exemplo de visualização de informações do texto na aba Text	155
Figura 60 - Exemplo de seleção de modelos de atividades na aba Templates	156
Figura 61 - Opções da aba Save	156
Figura 62 - Mensagem da página após a publicação da atividade	157
Figura 63 - Resultado parcial da extração de palavras-chave no program <i>AntConc</i>	162
Figura 64 - Candidatos a colocados da palavra 'experience' no programa <i>AntConc</i>	163
Figura 65 - Modelo interpretado de atividade 1	165
Figura 66 - Modelo interpretado de atividade 2	165
Figura 67 - Modelo interpretado de atividade 3	165
Figura 68 - Modelo interpretado de atividade 4	165
Figura 69 - Modelo interpretado de atividade 5	166
Figura 70 - Modelo interpretado de atividade 6	166
Figura 71 - Modelo interpretado de atividade 7	166
Figura 72 - Modelo interpretado de atividade 8	166
Figura 73 - Modelo interpretado de atividade 9	167
Figura 74 - Modelo interpretado de atividade 10	167
Figura 75 - Modelo interpretado de atividade 11	167
Figura 76 - Modelo interpretado de atividade 12.....	167
Figura 77 - Modelo interpretado de atividade 13	168
Figura 78 - Modelo interpretado de atividade 14	168
Figura 79 - Modelo interpretado de atividade 15	168
Figura 80 - Modelo interpretado de atividade 16	169
Figura 81 - Modelo interpretado de atividade 17	169
Figura 82 - Modelo interpretado de atividade 18	169
Figura 83 - Modelo interpretado de atividade 19	169
Figura 84 - Modelo interpretado de atividade 20	170
Figura 85 - Modelo interpretado de atividade 21	170
Figura 86 - Modelo interpretado de atividade 22	170
Figura 87 - Modelo interpretado de atividade 23	170
Figura 88 - Modelo interpretado de atividade 24	170
Figura 90 - Modelo interpretado de atividade 26	171
Figura 91 - Modelo interpretado de atividade 27	171
Figura 92 - Gráfico de frequência dos tipos de atividades elaboradas por professores.....	176
Figura 93 - Gráfico dos tipos de atividades que ocorrem mais de uma vez.....	176
Figura 94 - Formulário para avaliação do sistema de criação de atividades	178
Figura 95 - Gráfico das respostas fechadas	180

Figura 96 - Exemplo de organização de diretórios e código	183
Figura 97 - Arquitetura para melhoria do sistema.....	184
Figura 98 - Programa simples para contar palavras	219
Figura 99 - Execução de um programa em <i>Python</i>	220
Figura 100 - Exemplo de uso de funções de manipulação de <i>strings</i>	226
Figura 101 - Exemplo de uso de expressões regulares.....	227
Figura 102- Exemplo de criação de lista e acesso a seus elementos	229
Figura 103 - Exemplo de criação e uso de um dicionário	229
Figura 104 - Exemplo de uso de um dicionário para contar as ocorrências de palavras.....	230
Figura 105 - Exemplo de código para criar um gráfico de dispersão.....	232
Figura 106 - Exemplo de gráfico de dispersão criado com o módulo <i>matplotlib</i>	232
Figura 107 - Gráfico para visualização de probabilidades criado com o módulo <i>matplotlib</i>	232
Figura 108 - Função para eliminar códigos de resultados do etiquetador <i>VISL</i>	234
Figura 109 - Função para acessar o etiquetador <i>VISL</i>	234
Figura 110 - Função para acessar o etiquetador <i>VISL</i>	235
Figura 111 - Exemplo de código para análise de uma sentença com o <i>NLTK</i>	236
Figura 112 - Representação em árvore de sentença analisada	236
Figura 113 - Exemplo de criação de etiquetadores em sequência.....	238
Figura 114 - Exemplo de sentenças anotadas com etiquetadores.....	238
Figura 115 - Exemplo de criação de classificadores	241
Figura 116 - Avaliação de classificações e dados de árvore de decisão em texto.....	242
Figura 117 - Exemplo de utilização do algoritmo <i>KMeansClusterer</i> do <i>NLTK</i>	243
Figura 118 - Exemplo de agrupamentos com ' <i>KMeansClusterer</i> ' do <i>NLTK</i>	243

LISTA DE TABELAS

Tabela 1 - Acurácia de métodos de etiquetagem de alto desempenho	78
Tabela 2 - Desempenho dos etiquetadores treinados	83
Tabela 3 - Exemplo de tabela para cálculo de chavicidade.....	85
Tabela 4 - As 100 bases de palavras mais frequentes identificadas	91
Tabela 5 - Resultados dos experimentos para identificação de palavras cognatas.....	96
Tabela 6 - Resultados da avaliação do etiquetador de grupos nominais simples	100
Tabela 7 - Frequência das etiquetas morfossintáticas no <i>corpus</i>	115
Tabela 8 - Sufixos identificados no <i>corpus</i> de anúncios de empregos.....	122
Tabela 9 - Dispersão de 15 palavras do <i>corpus</i> de estudo em faixas	129
Tabela 10 - Exemplo de dados para a seleção de itens lexicais para criação de atividades	160
Tabela 11 - Candidatos a colocados da palavra ' <i>experience</i> ' na posição R1	163
Tabela 13 - Exemplo de dados classificados e sua conversão em números.....	240
Tabela 14 - Dados reorganizados a partir do agrupamento realizado	244

LISTA DE QUADROS

Quadro 1 - Principais tipos de padronização.....	27
Quadro 2 - Evolução dos sistemas de PLN.....	48
Quadro 3 - Estágios para elaboração de atividades de leitura.....	61
Quadro 5 - Conjunto de etiquetas do <i>corpus Penn Treebank</i> no <i>NLTK</i>	77
Quadro 6 - Descrição dos algoritmos para identificação de palavras cognatas.....	95
Quadro 7 - Regras simbólicas para identificação de grupos verbais.....	100
Quadro 8 - Exemplos de regras para a identificação de palavras com sufixos	101
Quadro 9 - Estrutura em <i>XML</i> para representar textos analisados	103
Quadro 10 - Composição do <i>corpus</i> de anúncios de emprego	108
Quadro 11 - Itens mais frequentes do <i>corpus</i> de anúncios de emprego	109
Quadro 12 - Os 100 itens mais frequentes do <i>corpus</i> de referência.....	111
Quadro 13 - Itens que ocorrem em mais de 9% dos textos do <i>corpus</i> de anúncios de emprego.....	113
Quadro 14 - Resultado parcial da extração de palavras-chave do <i>corpus</i> de estudo.....	117
Quadro 15 - As 100 palavras cognatas mais frequentes do <i>corpus</i> de estudo.....	121
Quadro 16 - Os 100 grupos nominais mais frequentes do <i>corpus</i> de anúncios de emprego	122
Quadro 17 - Os 100 grupos verbais mais frequentes do <i>corpus</i> de anúncios de emprego	123
Quadro 18 - Resultado de agrupamento a partir da dispersão das palavras	130
Quadro 19 - Composição do <i>corpus</i> de textos de divulgação científica	132
Quadro 20 - Os 100 itens com maior valor de chavicidade do <i>corpus</i> de divulgação científica	132
Quadro 21 - Exemplos de resultados da função <i>concordance_get</i>	139
Quadro 22 - <i>Tags</i> disponíveis para criação de modelos de atividades	140
Quadro 23 - Código XML do modelo de atividade <i>pt_affixes_a</i>	144
Quadro 24 - Código XML do modelo de atividade <i>pt_cognates_identification_a</i>	144
Quadro 25 - Código XML do modelo de atividade <i>pt_grammar_pos_a</i>	145
Quadro 26 - Código XML do modelo de atividade <i>pt_grammar_pos_a</i>	146
Quadro 27 - Código XML do modelo de atividade <i>pt_paragraphs_keywords_b</i>	147
Quadro 28 - Código XML do modelo de atividade <i>pt_references_a</i>	147
Quadro 29 - Código XML do modelo de atividade <i>pt_previous_knowledge_b</i>	148
Quadro 30 - Código XML do modelo de atividade <i>pt_typographic_marks_a</i>	148
Quadro 31 - Relação de módulos de análise programados	158
Quadro 32 - Exemplo de unidade didática gerada automaticamente	172
Quadro 33 - Tipos de atividades elaboradas por professores	175
Quadro 34 - Questões do formulário de avaliação	178
Quadro 35 - Respostas às perguntas abertas	181
Quadro 36 - Mensagem 1	181
Quadro 37 - Mensagem 2	182
Quadro 38 - Mensagem 3	182
Quadro 39 - Operadores da linguagem <i>Python</i>	221
Quadro 40 - Principais funções de manipulação de <i>strings</i>	225
Quadro 41 - Exemplo de cadeias de caracteres e expressões regulares	227

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de Máquina
AWL	<i>Academic Word List</i>
BNC	<i>British National Corpus</i>
COCA	<i>Corpus of Contemporary American English</i>
CSS	<i>Cascading Style Sheets</i>
ESP	<i>English For Specific Purposes</i>
IDE	<i>Integrated Development Environment</i>
HD	<i>Hard Disk</i>
HTML	<i>HyperText Markup Language</i>
KWIC	<i>Key Word In Context</i>
LC	Linguística de <i>Corpus</i>
LEM	Língua Estrangeira Moderna
MB	<i>Megabyte</i>
NLTK	<i>Natural Language Toolkit</i>
PCNP	Professor Coordenador do Núcleo Pedagógico
PDF	<i>Portable Document Format</i>
PHP	<i>Hypertext Preprocessor</i>
PLN	Processamento de Línguas Naturais
POS	<i>Part-Of-Speech</i>
RAM	<i>Random Access Memory</i>
RCB	<i>Reading Class Builder</i>
SPLN	Sistema de Processamento de Língua Natural
VISL	<i>Visual Interactive Syntax Learning</i>
WTA	Workshop de Tecnologia Adaptativa
XML	<i>eXtensible Markup Language</i>

SUMÁRIO

1 INTRODUÇÃO	18
1.1 Contextualização	18
1.2 Motivação e relevância.....	19
1.3 Objetivos	22
1.4 Premissas.....	23
1.5 Metodologia	24
1.6 Contribuições.....	25
1.7 Organização da tese	25
2 FUNDAMENTOS TEÓRICO-METODOLÓGICOS.....	27
2.1 Linguística de <i>Corpus</i>	27
2.1.1 Breve histórico sobre a Linguística de <i>Corpus</i>	28
2.1.2 Definição e uso de <i>corpus</i>	29
2.1.3 O computador e a Linguística de <i>Corpus</i>	30
2.1.4 Principais ferramentas em Linguística de <i>Corpus</i>	32
2.1.5 Outros programas	40
2.1.6 Ferramentas computacionais em metodologia básica de pesquisa	42
2.1.7 Linguística de <i>Corpus</i> e ensino	43
2.2 Processamento de Línguas Naturais.....	46
2.3 Aprendizado de Máquina	51
2.4 Avaliação de resultados	57
2.5 Ensino de leitura em língua estrangeira e criação de atividades	59
2.6 Estudo de criação automática de atividades	63
3 DESENVOLVIMENTO	67
3.1 Linguagem <i>Python</i> e Pesquisa Linguística	67
3.2 <i>Corpora</i> da pesquisa	69
3.3 Criação dos módulos de análise de texto e <i>corpora</i>	70
3.3.1 Segmentação	71
3.3.2 Itemização	72
3.3.3 Método de <i>stemming</i>	76
3.3.4 Etiquetagem morfossintática.....	76
3.3.5 Contagem de frequência das palavras.....	84
3.3.6 Extração de palavras-chave.....	84

3.3.7 Identificação de faixas de frequência lexical.....	86
3.3.8 Identificação de palavras cognatas	89
3.3.9 Identificação de grupos nominais	97
3.3.10 Identificação de grupos verbais	100
3.3.11 Identificação de entidades nomeadas	101
3.3.12 Identificação de sufixos.....	101
3.3.13 Estrutura de dados para armazenar as análises.....	102
3.4 Aplicação dos módulos de análise aos <i>corpora</i> de estudo.....	105
3.4.1 Criação de um analisador de <i>corpus</i>	106
3.4.2 Análise do <i>corpus</i> de anúncios de emprego em inglês	107
3.4.3 Composição do <i>corpus</i> de textos de divulgação científica.....	132
3.5 Sistema de criação automática de atividades.....	133
3.5.1 Componentes para o desenvolvimento do sistema	133
3.5.2 Módulo readler	134
3.5.3 Módulo xmltextreader	136
3.5.4 Módulo xcorpusanalyser.....	138
3.5.5 Tags em <i>XML</i> para a criação de modelos de atividades	140
3.5.6 Módulo xinterpreter.....	142
3.5.7 Modelos de atividades	143
3.5.8 Módulo xbuilder	148
3.5.9 Arquitetura do sistema	151
3.5.10 Interface do sistema de criação de atividades.....	152
4 RESULTADOS E DISCUSSÃO	158
4.1 Módulos de análise de textos	158
4.2 Módulo de análise de <i>corpora</i>	160
4.3 Modelos de atividades interpretados.....	164
4.4 Exemplo de unidade didática gerada automaticamente	171
4.5 Atividades elaboradas sem o uso do sistema.....	174
4.6 Avaliação do sistema por usuários.....	177
4.7 Adaptações do sistema para uso de textos em língua portuguesa.....	182
4.8 Possíveis melhorias ao sistema	183
4.9 Avanços do sistema desenvolvido.....	185
5 CONCLUSÃO	187
REFERÊNCIAS	192

ANEXO A - Questões da atividade-padrão	196
ANEXO B - Interface do primeiro protótipo de RCB	197
ANEXO C - Exemplo de atividade preparada pelo RCB	198
ANEXO D - Lista de etiquetas morfossintáticas do corpus Penn Treebank	200
ANEXO E - Texto de anúncio de emprego em inglês utilizado em orientação técnica.....	201
ANEXO F - Texto de divulgação científica utilizado em orientação técnica	203
ANEXO G - Atividade preparada por professores 1	204
ANEXO H - Atividade preparada por professores 2	205
ANEXO I - Atividade preparada por professores 3.....	206
ANEXO J - Atividade preparada por professores 4	207
ANEXO K - Atividade preparada por professores 5	208
ANEXO L - Atividade preparada por professores 6.....	209
ANEXO M - Atividade preparada por professores 7	211
ANEXO N - Atividade preparada por professores 8	212
ANEXO O - Atividade preparada por professores 9	213
ANEXO P - Atividade preparada por professores 10.....	214
ANEXO Q - Atividade preparada por professores 11	215
ANEXO R - Atividade preparada por professores 12	216
ANEXO S - Atividade preparada por professores 13.....	217
ANEXO T - Atividade preparada por professores 14.....	218
APÊNDICE A - Recursos da linguagem Python para a pesquisa linguística	219
APÊNDICE B - Código do módulo affixes.....	246
APÊNDICE C - Código do módulo cogtagger	247
APÊNDICE D - Código do módulo counter	248
APÊNDICE E - Código do módulo keywordsextractor.....	249
APÊNDICE F - Código do módulo netagger	250
APÊNDICE G - Código do módulo nptagger	251
APÊNDICE H - Código do módulo postagger.....	253
APÊNDICE I - Código do módulo segmenter	257
APÊNDICE J - Código do módulo stemmer.....	258
APÊNDICE K - Código do módulo tokenizer	259
APÊNDICE L - Código do módulo vgtagger	261
APÊNDICE M - Código do módulo vocabpro.....	263
APÊNDICE N - Código do módulo corpusanalyser.....	266
APÊNDICE O - Código do módulo interpreter	291
APÊNDICE P - Código do módulo readler.....	296

APÊNDICE Q - Código do módulo builder	303
APÊNDICE R - Código do módulo xmltextreader	305

1 INTRODUÇÃO

"Na Era da Informação, a absorção de novas tecnologias também se faz presente nos estudos da linguagem humana, exigindo do pesquisador um novo tipo de posicionamento frente ao seu objeto de estudo." (ZAPPAROLI, 2010, p. 10)

Os estudos da linguagem por meio de *corpora*, impulsionados pelo advento do computador, especificamente em Linguística de *Corpus*, têm influenciado diversas áreas, em especial, o ensino de línguas, que tem, nesse domínio, quatro principais áreas de interesse: descrição da linguagem nativa, descrição da linguagem do aprendiz, transposição de metodologias de pesquisa acadêmica para a sala de aula e desenvolvimento de materiais de ensino, currículo e abordagens (BERBER SARDINHA, 2004, p. 254). A ampliação de tais áreas de interesse é pretendida nesta tese, a partir do estudo e criação de ferramentas computacionais para a análise de textos e *corpora*, com o objetivo de otimizar o desenvolvimento de materiais de ensino em um posicionamento pouco provável, em que o pesquisador linguista se envereda pela análise computacional, aprendendo a programar e a explorar os recursos de áreas correlatas, como o Processamento de Línguas Naturais (PLN) e Aprendizagem de Máquina (AM), a fim de tornar a pesquisa linguística mais produtiva. A partir de um sistema que realiza automaticamente análises linguísticas em texto e *corpora* e extrai dados para a criação de atividades, a tese é a de que a preparação de materiais didáticos que privilegiem a língua em uso, em especial para o ensino de leitura, pode ser auxiliada e potencializada por ferramentas computacionais projetadas especificamente para a tarefa, por meio do aproveitamento de tecnologias das áreas da Linguística e da Computação.

1.1 Contextualização

A coleta e exploração de *corpus* para a criação de atividades de ensino é tema recorrente da Linguística de *Corpus*. No ensino de língua estrangeira, nos anos recentes, há uma gama de pesquisas sobre a análise, o desenvolvimento e o emprego de materiais de ensino baseados em *corpora*, uma vez que privilegiam a língua em uso.

Uma grande quantidade de trabalhos defende a utilização da instrumentação da Linguística de *Corpus* no desenvolvimento de material didático para o ensino de línguas: o uso de ferramentas de exploração de *corpora* como auxílio para a elaboração de cursos (FLOWERDEW, 1993); o desenvolvimento de material didático (THURSTUN &

CANDLIN, 1998); e as possibilidades de uso dos dados de pesquisa de *corpora* para ensino de vocabulário contextualizado, gramática e pragmática (FOX, 1998).

No contexto de pesquisa brasileira, pode ser citada uma série de trabalhos que produziram atividades de ensino com *corpus* em vários âmbitos do ensino de língua estrangeira: no ensino de inglês geral, os trabalhos de Condi de Souza (2005), Vicentini (2006), Bértoli Dutra (2002) e Amarante (2005); no ensino de Inglês Instrumental, os trabalhos de Ferrari (2004) e Barbosa (2004); no ensino de língua espanhola, Jacobi (2001) e Alonso (2006). Esses estudos, no entanto, não abordam a automatização do processo de exploração de *corpora* para a criação de atividades de ensino.

Nesta pesquisa, investigam-se as ferramentas de exploração de *corpus* da Linguística de *Corpus* em diálogo com técnicas, métodos e recursos do Processamento de Línguas Naturais e Aprendizado de Máquina, para o desenvolvimento de um sistema de análise de textos e *corpora* em língua inglesa, com vistas à criação automática de atividades de leitura e ensino de léxico-gramática.

1.2 Motivação e relevância

Embora o uso de materiais baseados em *corpus* seja positivo para o processo de ensino e aprendizagem, a sua preparação e elaboração ainda não é uma realidade comum fora do contexto acadêmico. Mesmo com as mais variadas ferramentas computacionais para análise de *corpora*, o processo de preparação de unidades didáticas inteiras, e até mesmo de atividades menores, pode ser considerado problemático para a maioria dos professores.

A tarefa, que geralmente leva tempo, é realizada apenas por pesquisadores; muitas vezes, requer a análise prévia de grandes quantidades de dados por programas de computador especializados, como concordâncias, listas de frequência, listas de palavras-chave, anotação¹ de *corpus*, entre outros tipos. Podemos citar, como exemplo, a pesquisa de Condi de Souza (2005), que descreveu todo o percurso do uso de dois *corpora* na elaboração de uma tarefa para ensino de inglês por meio de análises propiciadas por essas ferramentas.

¹ Adição manual ou automática de informações linguísticas (gramatical, lexical sintática, etc.) a serviço de um objetivo teórico ou prático.

Não é possível esperar que todo professor seja um especialista em Linguística de *Corpus* para que possa aproveitar os benefícios do uso de *corpora* e de suas ferramentas computacionais de análise em atividades de sala de aula.

Devido a esses motivos, professores podem ter dificuldades na preparação de tais materiais e, em decorrência, não utilizá-los com certa frequência e/ou fazer uso de materiais tradicionais não significativos² para a aprendizagem dos alunos.

Assim, a partir da investigação linguística, desenvolvimento, aplicação e análise de um sistema de análise de textos e de criação automática de atividades de leitura em língua inglesa com *corpora*, por meio do uso de técnicas de análise de Processamento de Línguas Naturais (NUGUES, 2006; DIAS DA SILVA et al., 2007), de práticas de análise de *corpus* para o ensino de línguas (BERBER SARDINHA, 2004; HUNSTON, 2002; KENNEDY, 1998; JOHNS, 1991; FLOWERDEW, 1993; FOX, 1998; TRIBLE & JONES, 1990) e Aprendizado de Máquina (MARSLAND, 2009; HARRINGTON, 2012; PUSTEJOVSKY & STUBBS, 2012; LUGER, 2013), esta pesquisa tem como um de seus objetivos oferecer a possibilidade de criação de materiais baseados em *corpora* a professores de língua estrangeira que não estejam familiarizados com o uso de ferramentas de processamento e exploração de *corpora* e/ou que não possuam muito tempo para preparar atividades.

A investigação está baseada em um estudo inicial realizado em uma pesquisa de mestrado (MOREIRA FILHO, 2007), que teve como produto final um *software desktop* para preparação semiautomática de atividades de leitura em inglês. O produto final do estudo considera como entrada um texto selecionado pelo usuário e, através de etapas, como um assistente eletrônico, leva-o até a publicação de uma unidade-padrão.

Nos primeiros protótipos, com base no conceito de “atividade-padrão”³ (SCOTT et al., 1984, p. 1) para o ensino de leitura em *English for Especific Purposes (ESP)*⁴, inglês para fins específicos, um conjunto fixo de atividades é preparado automaticamente, incluindo atividades baseadas em concordâncias, *data-driven learning*⁵, predição, léxico-gramática e questões para leitura crítica. Para tanto, por meio de fórmulas estatísticas, o programa faz

²Materiais não autênticos, baseados em achismos, propensos a desvincular a língua de seus usos e de seus usuários.

³Tradução nossa. Em inglês: *standard exercise*.

⁴Em português, é comum o uso do termo inglês instrumental.

⁵Abordagem de ensino conhecida como ensino movido a dados.

várias análises automáticas do texto selecionado: lista de frequência, palavras-chave, possíveis palavras cognatas⁶, etiquetagem morfológica, possíveis padrões e densidade lexical⁷ do texto.

Embora os resultados obtidos tenham demonstrado a viabilidade e o potencial de, por meio do computador, analisar textos e gerar automaticamente determinados tipos de exercícios para ensino de leitura, considerou-se que há ainda a necessidade de muita pesquisa e desenvolvimento de melhorias para a consolidação de uma ferramenta que possa ser usada pelo usuário final. Há situações em que o programa gera exercícios com erros, tal como incluir na lista de palavras gramaticais uma palavra de conteúdo. Nesses casos, há a possibilidade de intervenção do usuário, mas, se a quantidade de erros for numerosa e exigir constantemente esse tipo de intervenção do usuário/professor, o sistema pode perder sua utilidade, impossibilitando seu uso pedagógico.

Outro ponto essencial é o aumento da variedade de exercícios disponíveis e a quebra da limitação do usuário a um modelo fixo. O conceito de atividade-padrão está relacionado à necessidade de treinar a compreensão leitora do aprendiz a partir de um conjunto fixo de questões que podem ser utilizadas a quase qualquer texto. A invariabilidade do modelo padrão faz com que o programa tenha sua funcionalidade limitada e estática. A adição de recursos adicionais significaria toda a sua reprogramação.

Em uma nova proposta de sistema, a geração de exercícios em relação a outros itens comuns no ensino de leitura e léxico-gramática, adequados a uma gama maior de textos, é pretendida. Em vez de um modelo fixo e estático, procura-se desenvolver um ambiente propício com um conjunto de instruções, parâmetros e regras que possibilitem uma funcionalidade flexível e dinâmica, em que a criação das atividades seja feita de acordo com o texto de entrada, sem a necessidade de reprogramação, em contraponto a um modelo fixo de unidade didática.

Desse modo, buscando estender os objetivos da pesquisa de mestrado (MOREIRA FILHO, 2007), empregam-se, em diálogo, pressupostos teóricos e metodológicos, além de técnicas das áreas de Linguística de *Corpus*, Processamento de Línguas Naturais e Aprendizagem de Máquina em todos os níveis aplicáveis de um Sistema de Processamento de Língua Natural

⁶ Palavras em inglês com grafias semelhantes a palavras em português.

⁷ Proporção de vocábulos lexicais em um texto.

(SPLN), que, a partir do fornecimento de um texto e de algum tipo de entrada preestabelecida (opções do usuário), deverá gerar atividades didáticas de leitura em língua inglesa.

Considerou-se relevante a exploração do trabalho em quatro níveis/camadas do sistema: opções do usuário, análise linguística do texto de entrada, análise pedagógica do texto e seleção de modelos de atividade. As atividades de ensino a serem geradas levam em consideração as informações de todos os níveis. O produto final está condicionado às opções do usuário (seleção de itens de ensino, tipos de exercício e itens léxico-gramaticais incluídos), análise linguística de textos (informações de frequência, palavras-chave, anotação morfossintática, entre outras), análise pedagógica da entrada (a partir da análise linguística, quais tipos de exercícios são possíveis e adequados), seleção e montagem de atividades (como a extração e a organização de itens linguísticos).

Embora ainda haja muito trabalho a ser realizado, a investigação é importante para o estabelecimento de um diálogo concreto entre Linguística de *Corpus*, Processamento de Línguas Naturais e Aprendizagem de Máquina, especificamente no que diz respeito ao desenvolvimento de aplicações para análise de *corpora* e ensino de línguas.

1.3 Objetivos

O objetivo geral desta pesquisa é conceber um Sistema de Processamento de Língua Natural capaz de analisar *corpora* e textos para gerar, automaticamente, atividades de leitura e ensino de padrões (léxico-gramática) em língua inglesa a partir de um texto e *corpus*.

Os objetivos específicos envolvem:

- a. estudar as abordagens e ferramentas de análise de *corpora* para a criação de materiais de ensino na área de Linguística de *Corpus*;
- b. estudar as abordagens, técnicas, algoritmos e recursos das áreas da Computação, como Processamento de Línguas Naturais e Aprendizado de Máquina;
- c. estudar a implementação de algoritmos para análise linguística por meio de programação em linguagem *Python*;

- d. coletar e analisar a padronização de *corpora* em gêneros específicos a fim de serem utilizados no desenvolvimento e avaliação de um sistema de análise de *corpora* e texto;
- e. construir módulos de análise de texto e *corpora* para obtenção de recursos e dados linguísticos para avaliação pedagógica de um texto e criação de exercícios;
- f. avaliar, por meio de experimentos práticos, os módulos de análise construídos, além de sua eficácia na criação de atividades;
- g. contrastar os benefícios de uso do sistema em relação às ferramentas de programas de análise de *corpora*, especificamente os concordanciadores;
- h. formalizar a arquitetura do sistema;
- i. realizar a implementação do sistema por meio de uma interface gráfica para utilização;
- j. avaliar o sistema implementado com possíveis usuários finais.

1.4 Premissas

As premissas adotadas nesta tese são as de que:

- a. os concordanciadores atuais, por serem desenhados para uso geral, com diferentes propósitos, podem demandar grande esforço de adaptação de suas funcionalidades para a pesquisa e construção de materiais baseados em dados de análise de *corpora* por professores;
- b. a preparação de materiais no ensino de línguas pode ser enriquecida com o auxílio de recursos computacionais que contribuam para a otimização dos estudos linguísticos em larga escala, tanto em relação ao tempo e esforço, quanto em qualidade de seus resultados;
- c. a automatização de análises e exploração de *corpora* para fins específicos, como a extração de dados para elaboração de materiais didáticos e atividades para o ensino de línguas, pode ser uma realidade a partir do estudo das ferramentas computacionais disponíveis e o diálogo com outras áreas de estudo;
- d. os dados de análises básicas, comuns no Processamento de Línguas Naturais, podem ser extrapolados na criação de atividades de leitura e ensino de léxico-gramática;
- e. o uso de uma abordagem específica de análise automática de *corpus* pode trazer benefícios significativos para a tarefa de elaboração de materiais de ensino baseados em *corpora*;

- f. um maior diálogo da Linguística de *Corpus* com outras áreas, consideradas próximas, as quais também fazem uso de *corpora*, especificamente o Processamento de Línguas Naturais e Aprendizagem de Máquina, pode otimizar os procedimentos de pesquisa e análise de *corpora*;
- g. a formalização da necessidade de análise automática de *corpora* pode ser objeto interessante de pesquisa para o linguista de *corpus* em novos caminhos de investigação;
- h. a extensão dos achados às demais línguas a partir da investigação de *corpora* de determinada língua, conforme os objetivos da pesquisa, de forma a não se restringir apenas à língua dos *corpora* de estudo da pesquisa.

1.5 Metodologia

A abordagem de desenvolvimento do sistema de análise de textos e *corpora* em língua inglesa para a criação automática de atividades de leitura e ensino de padrões baseia-se nas funcionalidades das ferramentas existentes de análise e exploração de *corpora*, como ponto de partida.

A metodologia para a realização desta tese consistiu em:

- a. realizar revisão bibliográfica associada aos assuntos abordados na tese e sobre trabalhos correlatos;
- b. analisar os elementos constituintes das principais ferramentas de análise e exploração de *corpora* e as metodologias de uso de tais ferramentas no âmbito do ensino de línguas relacionado à Linguística de *Corpus*;
- c. coletar e compilar os *corpora* de estudo, desenvolvimento e teste para todo o processo de desenvolvimento do sistema proposto;
- d. estudar, construir e adaptar métodos, técnicas e algoritmos existentes para análise de *corpora* e textos, necessários para as análises pretendidas;
- e. anotar dados de *corpora* para o treinamento e teste de algoritmos;
- f. elaborar módulos de análise de apoio, por exemplo, módulos de itemização (*tokenization*), etiquetagem, contagem de palavras, extração de palavras-chave, extração de n-gramas entre outros, para a análise de texto e *corpora*;

- g. aplicar os módulos de análise nos *corpora* de estudo, desenvolvimento e teste, a fim de obter a descrição dos *corpora* e possibilidades de dados e recursos a serem utilizados para a criação de atividades;
- h. avaliar e otimizar os módulos de análise a partir de sua aplicação nos *corpora* de desenvolvimento e teste;
- i. explorar, criar e adaptar os módulos de análise na construção de algoritmos de criação de atividades de ensino;
- j. desenvolver e avaliar um sistema de criação automática de atividades.

O detalhamento das metodologias específicas empregadas em cada etapa do desenvolvimento é descrito no capítulo 2, **FUNDAMENTOS TEÓRICO-METODOLÓGICOS**.

1.6 Contribuições

Os resultados da investigação podem oferecer contribuições para fomentar o interesse em relação ao desenvolvimento de ferramentas computacionais para exploração de *corpora* na automatização da extração de dados linguísticos para a elaboração de materiais de ensino e currículos.

Considera-se o percurso do desenvolvimento do sistema como parte integrante da pesquisa, uma vez que seu cenário pode ser considerado pouco usual, dada a natureza interdisciplinar do estudo e suas exigências, tendo o pesquisador linguista de enveredar pela análise computacional por meio de programação.

O esforço pode propiciar um contato maior com as diferentes áreas e trazer benefícios para a construção do diálogo pretendido, não só neste estudo, mas também nas pesquisas de linguagem via computador em geral, em que há separação entre equipes: de um lado, aquelas que fazem a programação para a realização de tarefas automaticamente; de outro, as que apenas fazem uso dos instrumentos.

1.7 Organização da tese

O presente trabalho apresenta-se distribuído da seguinte maneira:

Capítulo 1. INTRODUÇÃO: apresenta-se a contextualização do tema da tese, motivação e relevância, objetivos, premissas, metodologia, contribuições e a organização do texto.

Capítulo 2. FUNDAMENTOS TEÓRICO-METODOLÓGICOS: relatam-se os fundamentos teórico-metodológicos das diversas áreas de Linguística de *Corpus*, Processamento de Línguas Naturais e Aprendizado de Máquina, a fim de situar o trabalho no âmbito interdisciplinar, descrevem-se as metodologias de avaliação e os estudos-piloto que contribuíram como suporte e motivação para o desenvolvimento da pesquisa.

Capítulo 3. DESENVOLVIMENTO: expõe-se a linguagem de programação, o ambiente de desenvolvimento e os recursos disponíveis para pesquisa linguística, faz-se a apresentação dos *corpora* utilizados na pesquisa, descreve-se o desenvolvimento dos principais módulos de análise de texto e *corpora* necessários para o sistema de criação automática de atividades, a aplicação dos módulos de análise aos *corpora* de estudo, juntamente com dados de sua descrição linguística, a arquitetura do sistema de criação automática de atividades, as principais funções de seus módulos e a interface com o usuário.

Capítulo 4. RESULTADOS E DISCUSSÃO: faz-se a exposição e a avaliação dos resultados obtidos e principais contribuições. Tecem-se, ainda, sugestões para possíveis adaptações do sistema para uso de textos em língua portuguesa.

Capítulo 5. CONCLUSÕES E TRABALHOS FUTUROS: apresenta-se a conclusão sobre a pesquisa e as considerações finais, além de possíveis trabalhos futuros.

2 FUNDAMENTOS TEÓRICO-METODOLÓGICOS

A pesquisa encontra suporte teórico e metodológico em diversas áreas do saber: Linguística de *Corpus* (LC), Processamento de Línguas Naturais (PLN) e Aprendizado de Máquina (AM).

2.1 Linguística de *Corpus*

A Linguística de *Corpus* é uma área que estuda a língua por meio da observação de grandes quantidades de dados linguísticos reais, isto é, de textos falados ou escritos provenientes da comunicação no mundo real - língua em uso -, com o auxílio de ferramentas computacionais (BERBER SARDINHA, 2004, p. 3).

O conjunto de dados linguísticos reais criteriosamente coletados e utilizados em estudos de Linguística de *Corpus* é chamado de *corpus* (plural: *corpora*). O *corpus* deve ser constituído de dados autênticos (não inventados), legíveis por computador e representativos de uma língua ou variedade da língua que se deseja estudar.

A Linguística de *Corpus* faz uso de uma abordagem empirista e tem como central a noção de língua enquanto sistema probabilístico. De acordo com essa noção, os traços linguísticos não ocorrem de forma aleatória, sendo possível evidenciar e quantificar regularidades (padrões). É comum na área afirmar que a linguagem é padronizada (*patterned*), isto é, que existe uma correlação entre os traços linguísticos e os contextos situacionais de uso da língua.

Na Linguística de *Corpus*, a padronização evidencia-se por colocações, coligações ou estruturas que se repetem significativamente (BERBER SARDINHA, 2004, p. 40). O quadro a seguir traz um exemplo ilustrativo de um padrão identificado em um *corpus* de anúncios de emprego:

Quadro 1 - Principais tipos de padronização

Colocação	Coligação	Prosódia Semântica
<i>a + minimum + of + # + years</i>	$P(a) + NN(minimum) + P(of) + CD(\#) + NN(years)$	prosódia semântica neutra

Fonte: Produção do próprio autor

Conforme os exemplos do quadro, a colocação é a junção significativa, co-ocorrência, entre palavras. A coligação é a relação gramatical mantida pelas palavras (P=preposição; NN=substantivo; CD=numeral). A prosódia semântica é a associação entre itens lexicais e as conotações: negativa, positiva ou neutra.

Algumas das áreas de interesse da Linguística de *Corpus* são: compilação de *corpora* e desenvolvimento de ferramentas para sua análise, descrição de línguas, exploração do uso de descrições baseadas em *corpora* para várias aplicações, tal como o ensino de línguas, processamento de línguas naturais, reconhecimento de voz e tradução.

2.1.1 Breve histórico sobre a Linguística de *Corpus*

Conforme Pustejovsky e Stubbs (2012, p. 5), na metade do século XX, no campo da Linguística, predominavam os estudos descritivos, propriedades estruturais e variações tipológicas entre línguas. Em outras ciências sociais, a coleta e análise de dados também estava sujeita a técnicas quantitativas e estatísticas.

Nos anos de 1940, linguistas como Bloomfield estavam começando a pensar que a língua poderia ser explicada em termos comportamentais e probabilísticos.

Nos anos de 1950, métodos empíricos e estatísticos tornaram-se populares, e a teoria de Shannon para análise da linguagem fornecia uma abordagem quantitativa sólida para modelar descrições qualitativas de estruturas linguísticas.

Tais estudos, no entanto, encontraram uma barreira nos anos de 1950. Os fatores principais de dificuldade eram: i. o problema de disponibilidade de dados; ii. mudança de paradigma nos estudos das ciências sociais de descrições orientadas por dados do comportamento humano para a modelagem introspectiva de funções cognitivas.

Nesse cenário, o linguista Noam Chomsky focava em uma metodologia formal e uma teoria linguística que, além de ignorar dados linguísticos quantitativos, pregava a obtenção de conceitos provenientes da intuição e não mediante a consulta empírica.

Essa visão foi muito influente durante os anos de 1960 e 1970, com o desenvolvimento de sofisticados modelos de linguagem baseados em regras a partir de dados introspectivos. Como a construção de teoria envolve o teste e a avaliação de hipóteses em relação a fenômenos observados, conforme o crescimento da disponibilidade de dados linguísticos para pesquisa e de ferramentas poderosas para sua análise, o modo como os linguistas vem os dados tem mudado significativamente.

Atualmente, os fenômenos da linguagem podem ser visualizados em bilhões ou trilhões de textos, o que permite não deixar sombra de dúvidas sobre os benefícios de técnicas quantitativas no teste e criação de modelos de língua. Isso nos traz à era moderna da Linguística de *Corpus*.

Como resultado, recentemente, o *corpus* tem sido a porta de entrada para a maioria das análises linguísticas.

2.1.2 Definição e uso de *corpus*

A coleta e exploração de *corpus* são tarefas indissociáveis da Linguística de *Corpus*. Ao longo do desenvolvimento da disciplina, uma gama de *corpora* foi compilada, com propósitos e características variadas, contribuindo para o seu estabelecimento. Berber Sardinha (2004, p. 18) propõe uma definição mais completa de *corpus*:

“Um conjunto de dados lingüísticos (pertencentes ao uso oral ou escrito da língua, ou ambos), sistematizados segundo determinados critérios, suficientemente extensos em amplitude e profundidade, de maneira que sejam representativos da totalidade do uso lingüístico ou de algum de seus âmbitos, dispostos de tal modo que possam ser processados por computador, com a finalidade de propiciar resultados vários e úteis para a descrição e análise.”

Conforme a definição apresentada, alguns critérios são essenciais para a compilação de um *corpus*. Primeiro, os textos devem ser autênticos, ou seja, não produzidos artificialmente para os propósitos da pesquisa. Segundo, os textos do *corpus* devem ser produzidos por falantes nativos, se não, a informação deve ser explícita de que se trata de textos produzidos por aprendizes. Terceiro, o *corpus* deve ser representativo de uma língua ou variedade para estudo linguístico. Quarto, os textos devem estar em formato eletrônico para permitir o uso de

ferramentas computacionais para análise. Por último, o *corpus* compilado deve seguir o propósito específico da investigação linguística, o qual influencia suas características.

O uso mais comum de *corpora* tem sido a geração de concordâncias, *Key Word in Context* (KWIC), que permite o estudo de uma palavra ou item linguístico do ponto de vista sintático e semântico. Nos anos de 1950 e 1960, programas foram escritos para gerar automaticamente concordâncias. Atualmente, há diversos programas que realizam tal tarefa, comumente chamados de concordanciadores.

Em outras áreas, o uso efetivo de *corpora* para modelar fenômenos linguísticos e treinar algoritmos iniciou-se nos anos de 1980 com pesquisas de reconhecimento de fala, a partir de transcrições, utilizando n-gramas⁸ e o modelo de Cadeias Ocultas de Markov⁹ (*Hidden Markov Models*) no reconhecimento de vocabulário de palavras (em um pequeno domínio específico).

Nos anos de 1990, o trabalho com tradução automática começou a receber a influência, cada vez mais, de grandes bancos de dados e, com isso, o surgimento de um modelo de língua estatístico para a tradução.

O desenvolvimento de memória e *hardware* do computador propicia o trabalho de coleta e análise de grandes quantidades de dados linguísticos, o qual permite a criação de modelos de língua estatísticos que executam diferentes tarefas no Processamento de Línguas Naturais com precisão satisfatória.

2.1.3 O computador e a Linguística de *Corpus*

Embora a Linguística de *Corpus* não se tenha iniciado com o desenvolvimento dos computadores, mas com a importante contribuição das análises manuais (KENNEDY, 1998, p. 5), é reconhecido que o advento do computador impulsionou a área, trazendo mudanças para os estudos baseados em textos.

⁸ Uma subsequência de n itens de uma determinada sequência.

⁹ Um modelo probabilístico que permite fazer previsões de um próximo elemento de certa sequência de elementos

Os desenvolvimentos tecnológicos mudaram a maneira como trabalhamos com *corpus*. O computador introduziu velocidade, precisão, confiabilidade estatística e habilidades para lidar com grandes quantidades de dados (KENNEDY, 1998). Segundo Fox (1998, p. 25), atualmente não há restrições para o número de textos em um *corpus*.

Berber Sardinha (2004, p. 85) cita uma metáfora para ressaltar o papel do computador na Linguística de *Corpus*:

“O desenvolvimento do computador com memória poderosa seria para a lingüística o que o desenvolvimento do microscópio com lentes poderosas foi para a biologia – uma oportunidade não somente de ampliar nosso conhecimento, mas de transformá-lo.”

De acordo com Kennedy (1998), os computadores facilitaram o desenvolvimento de bases matemáticas para o processamento automático de linguagem natural, além de trazer para os estudos linguísticos um grande grau de precisão, importante para todas as ciências.

Berber Sardinha (2004, p. 85) aponta um maior emprego de computadores na investigação da linguagem por três razões. Primeiro, porque os computadores garantem a consistência da análise em tarefas que não sejam adequadas para o pesquisador, como contagens exaustivas de itens linguísticos. Segundo, por permitirem maior abrangência na quantidade de dados analisáveis. Terceiro, por possibilitarem a descoberta de fatos novos ou contestarem crenças estabelecidas sobre a linguagem.

Hunston (2002, p. 3) afirma que o *corpus* em si não pode fazer muito. O *corpus* não contém informações novas sobre a linguagem, mas as ferramentas eletrônicas utilizadas para reorganizar as informações contidas no *corpus* podem oferecer novas perspectivas para o pesquisador. As descobertas a partir do estudo da linguagem são favorecidas pela análise das informações e representações privilegiadas pela instrumentação eletrônica.

Dessa forma, o computador é essencial para o desenvolvimento de pesquisas em Linguística de *Corpus*. Há uma grande variedade de *software* para os estudos em Linguística de *Corpus*. A maioria processa os dados de um *corpus* exibindo frequências, fraseologias e colocações. Os mais conhecidos são listadores de palavras e concordanciadores.

2.1.4 Principais ferramentas em Linguística de *Corpus*

O uso do computador e *corpora* permite a ampliação das análises linguísticas, tendo em vista que determinadas tarefas são mais apropriadas para a máquina, uma vez que pode evitar os esforços repetitivos e cansativos, aumentar a velocidade de processamento de informações e garantir maior consistência aos fenômenos linguísticos pesquisados.

Diversos programas de computador estão disponíveis e são utilizados em pesquisas na área de Linguística de *Corpus*. Muitos programas apresentam características e funcionalidades semelhantes. De maneira geral, esses programas disponibilizam ferramentas para a contagem de itens, cálculos estatísticos, comparações e organizações de resultados em visualizações privilegiadas, a fim de que padrões linguísticos sejam identificados com maior facilidade pelo linguista.

Anthony (2013) cita os resultados de uma pesquisa com 891 linguistas de *corpus*, perguntando "qual programa de computador você utiliza para análise de *corpora*?", a fim de evidenciar ferramentas mais populares utilizadas na área. O resultado é mostrado no seguinte gráfico:

Figura 1 - Programas mais utilizados por linguistas de *corpus*

Fonte: (ANTHONY, 2013, p. 154)

Destacam-se os três programas mais populares:

- a. *corpus.byu.edu* (DAVIES, 2012) - *site* composto por um conjunto de funcionalidades de análise e *corpora* online, tal como o *Corpus of Contemporary American English (COCA)*, com mais de 425 milhões de palavras. Não possibilita o uso de suas ferramentas com *corpora* de usuários;
- b. *AntConc* (ANTHONY, 2012) - um concordanciador *desktop* gratuito para diversos sistemas operacionais, para usos gerais de análise de *corpora*;
- c. *Wordsmith tools* (SCOTT, 1996) - um concordanciador *desktop* comercial para o sistema operacional *Windows*, para usos gerais de análise de *corpora*.

Tais programas possuem em comum algumas das ferramentas mais utilizadas em análise de *corpora*: listadores de palavras e concordanciadores. Outras funcionalidades relacionadas são a extração de palavras-chave e lista de colocados, geralmente a partir dos resultados de ambas as ferramentas.

Ferramentas para análise da padronização, como listadores de palavras e concordanciadores, permitem a busca por palavras específicas em um *corpus*, fornecem exaustivas listas para as ocorrências da palavra em contexto. Geralmente, esses programas são encontrados em uma única solução/*software*. É o caso de *software* como o *Wordsmith Tools*, com versões disponíveis para download em <http://www.lexically.net/wordsmith/>, um dos mais utilizados em pesquisas de Linguística de *Corpus*, cuja versão mais estável possui três ferramentas principais: *Wordlist* (lista palavras por frequência no *corpus*), *KeyWords* (extrai palavras-chave do *corpus*) e *Concord* (gera linhas de concordância).

Tendo em vista a grande influência de tais ferramentas na análise de *corpora*, há a necessidade de se detalharem suas funcionalidades.

No que se segue, apresenta-se cada uma dessas ferramentas em uma perspectiva de utilidade para o ensino de línguas, usando como exemplo o programa *Kitconc 4.0* (MOREIRA FILHO, 2008), que é um concordanciador *desktop* simples e gratuito para *Windows* (requer a tecnologia *.NET Framework* da *Microsoft*), com interface em língua portuguesa e com listas de referência para extração de palavras-chave em *corpora* de língua inglesa e portuguesa.

A seleção do programa *Kitconc*, como exemplo, justifica-se por possuir interface e utilitários semelhantes aos do programa *AntConc*, um dos mais utilizados como mencionado

anteriormente, com diferenciação em relação à língua e à inclusão de listas de referência. É importante mencionar que o programa *Kitconc* foi desenvolvido como opção para facilitar o uso de um concordanciador por professores do contexto brasileiro.

A partir de listadores de palavras, é possível descobrir quais são as palavras mais utilizadas em um texto ou conjunto de textos. Uma das opções permite listar também as expressões mais recorrentes formadas por duas ou mais palavras, chamadas de n-gramas, de um texto ou conjunto de textos.

A lista de palavras é uma listagem ordenada por frequência de todas as formas que ocorrem em um *corpus*. A partir da lista de frequência, pode-se definir quais são as palavras mais interessantes para análise do *corpus*. No exemplo a seguir, é apresentada parte da lista de frequência de um *corpus* de anúncios de emprego em língua inglesa.

N	Palavra	Frequência	%	Marca
1	AND	1412	05,80	
2	THE	831	03,41	
3	TO	725	02,98	
4	OF	620	02,54	
5	A	492	02,02	
6	IN	467	01,92	
7	WITH	365	01,50	
8	FOR	325	01,33	
9	IS	220	00,90	
10	WILL	182	00,75	
11	EXPERIENCE	179	00,73	
12	OR	170	00,70	
13	AS	159	00,65	
14	AN	138	00,57	
15	SKILLS	124	00,51	
16	BE	120	00,49	
17	WORK	119	00,49	
18	ON	114	00,47	

Figura 2 - Resultados da guia lista de palavras e opções de classificação

Fonte: (MOREIRA FILHO, 2011)

A ideia é a de que palavras que possuem uma ocorrência maior são mais relevantes, visto que há uma probabilidade maior de serem encontradas em outros *corpora* ou textos. Por exemplo, para um aprendiz inicial de língua estrangeira, aprender palavras mais frequentes é extremamente essencial, visto que elas podem aparecer em diversos contextos.

Ao analisar a ocorrência das palavras gramaticais, pode-se tentar identificar quais palavras se destacam em relação ao tipo de *corpus*, texto, gênero ou registro. Geralmente, a palavra mais frequente (número um da lista) em textos em língua inglesa é o artigo ‘*the*’. Se alguma outra palavra gramatical ocupar essa posição, será uma ocorrência marcada e merecedora de verificação.

A manipulação das listas por meio de funções de classificação permite, quando o pesquisador julgar necessário, juntar frequências de palavras, como é o caso da lematização, redução da forma de uma palavra a sua raiz. No exemplo abaixo, o pesquisador poderia ter a necessidade de juntar todas as frequências das diferentes formas com a raiz ‘*WORK*’ sob uma única forma:

N	Palavra	Frequência	%	Marca
22	WORKPLACE	2	00,01	
23	WORKING	55	00,23	
24	WORKERS	4	00,02	
25	WORKER	3	00,01	
26	WORKED	3	00,01	
27	WORK	119	00,49	

Figura 3 - Exemplo de palavras marcadas para lematização

Fonte: (MOREIRA FILHO, 2011)

É possível também levantar informações estatísticas gerais do *corpus*, como número total de palavras/itens (*tokens*), número de vocábulos/formas (*types*), razão forma/item (*type/token ratio*), que pode ser calculada da seguinte maneira: (número total de formas / número de itens)*100). A porcentagem resultante pode indicar a proporção de repetição das palavras no *corpus*. Um valor baixo da razão forma/item (*type/token ratio*) indica que há muita repetição no *corpus*, ou seja, um número menor de formas. Um valor alto da razão forma/item indica pouca repetição das palavras no *corpus*, um número maior de formas.

Outra maneira de se fazer um recorte em relação às palavras que devem ser analisadas é a extração de palavras-chave. Muitas vezes, a lista de palavras-chave fornece uma filtragem mais apurada das palavras que se destacam em um determinado *corpus* ou texto.

Em um *corpus* formado por textos de anúncios de emprego em inglês, palavras como ‘*EXPERIENCE*’, ‘*SKILLS*’, ‘*ABILITY*’, ‘*REQUIREMENTS*’ e ‘*SALARY*’ seriam palavras-chave, uma vez que podem ser consideradas específicas do gênero, revelando parte de sua estrutura padronizada:

The screenshot shows the 'kitconc 4.0 - [jobs]' window with the 'Palavras-chave' tab selected. It displays a table with the following data:

N	Palavra	Frequência	%	Chavacidade	Mar
1	COM	107	00,44	1316,34	
2	EXPERIENCE	179	00,73	928,78	
3	SKILLS	124	00,51	758,95	
4	AND	1412	05,80	726,66	
5	SALES	96	00,39	511,26	
6	ABILITY	91	00,37	501,88	
7	WEB	48	00,20	459,45	
8	ONLINE	46	00,19	437	
9	MANAGEMENT	95	00,39	370,28	
10	IT	20	00,08	361,79	
11	WAS	10	00,04	339,26	
12	DEVELOPMENT	99	00,41	324	
13	TEAM	83	00,34	321,62	
14	THE	831	03,41	307,77	
15	REQUIREMENTS	54	00,22	286,59	
16	CANDIDATE	48	00,20	283,55	
17	EMAIL	22	00,09	281,57	
18	INCLUDING	79	00,32	268,33	

Figura 4 - Palavras-chave de um *corpus* de anúncios de emprego em inglês

Fonte: (MOREIRA FILHO, 2011)

Em programas que executam extração de palavras-chave, é comum haver um recurso que possibilita a comparação automática de uma lista de frequência de palavras do *corpus* de estudo com uma lista de frequência de palavras de um *corpus* de referência, geralmente muito maior, a partir de um *corpus* de língua geral.

O resultado da comparação retorna uma lista classificada em que as principais palavras do *corpus* de estudo estarão no topo, geralmente palavras de conteúdo, em contraponto a uma lista de frequência em que as primeiras palavras são gramaticais.

Concordanciadores são programas utilizados para extração e exibição de linhas de concordância. O termo concordância refere-se à listagem das ocorrências de uma palavra de busca de um *corpus*, a qual fica centralizada, com uma quantidade definida de contextos em ambos os lados (esquerda e direita).

Em um primeiro momento, é comum associar/confundir o termo concordância apresentado aqui com o termo utilizado em gramática. Quando se utiliza o termo concordância, queremos nos referir a um tipo de visualização privilegiada do uso de palavras, conforme já descrito. No exemplo abaixo, exibem-se linhas de concordância da palavra ‘*minimum*’ em seu formato mais comum, *KWIC* (*Key Word In Context*).

Figura 5 - Exemplo de linhas de concordância no *Kitconc*

Fonte: (MOREIRA FILHO, 2011)

Diferentemente de um texto normal, as linhas de concordância são geralmente lidas a partir de seu nóculo, palavra ou expressão de busca, que fica centralizada para análise. Na leitura, busca-se a existência de padrões de uso por meio da análise de colocados, palavras à esquerda e à direita que ocorrem significativamente com o nóculo. Por isso, num primeiro momento na análise, não é aconselhável buscar a totalidade dos fragmentos para completar as frases/sentenças.

A fim de determinar padrões, pode-se primeiramente buscar quais palavras da esquerda e da direita geralmente se colocam com a palavra de busca. Em seguida, pode-se analisar quais tipos de palavras formam tais padrões, classe gramatical, campo semântico etc. Por fim, identificam-se os sentidos de cada padrão.

Vejamos a padronização da palavra ‘*resume*’ nas linhas de concordância a seguir:

red. CONTACT: Please <u>forward</u> cover letter and	resume	to: hr@biadvisors.com
ested persons should <u>submit</u> a cover letter and	resume	, or employment application, to the Human Resov
es are asked to <u>submit</u> a letter of interest and	resume	to: RN/LPN Search, 770 Central Avenue, Dover, NE
c 1, 2005 Please <u>Submit</u> letter of application,	resume	, 3 letters of reference, official transcripts
re salary and full benefit package. Please <u>fax</u>	resume	w/salary req. to: Bilt-Rite Construction, Inc.
fts & lift 70 pounds (unaided) are req'd. <u>Fax</u>	resume	immediately to 617-567-2247. EOE
ent benefits. Interested candidate, please <u>send</u>	resume	to: HR, Avecia Biotechnology, Inc. 125 Fortune B
-12 noon. Great benefits. Compet salary. <u>Send</u>	resume	: 134 South Ave., Weston MA 02493 or Fax: 781-7
illing to work w/ radioactive materials. <u>Send</u>	Resume	to: Nancy Clinton, Administrator, Boston Unive
to recognize serious chromosome defects. <u>Send</u>	resume	to Judy Heck, Administrator, Boston University
.red. Interested candidates should <u>email</u> their	resume	to mfoley@sagents.com or fax to (781) 419-070
! Opportunities, please <u>contact</u> or <u>forward</u> your	resume	to; Recruiter@tasctech.com TASC Technical Serv
y, we invite you to apply! Please <u>e-mail</u> your	resume	to hr@bestdoctors.com or fax to the number giv
of our Marketing positions, please <u>send</u> your	resume	and cover letter, indicating position of inter
choices are: Apply for this job online <u>Send</u> your	resume	as an attachment via e-mail More information a
ates. If you are interested please <u>submit</u> your	resume	to: mxwllmktg@aol.com - No Attachments - or ca
um Pro Interested candidates please <u>email</u> your	resume	to jobs@homesite.com Contact: 99 Bedford Street
If you like what you see, please <u>forward</u> your	resume	to hr@barneys.com or you may fax it to (212) 4
Salary ranges from 50K-60K. Please <u>email</u> your	resume	(as a Word attachment) to jleonard@johnleonar
ix talents to work for us. Please <u>forward</u> your	resume	along with cover letter and salary requirement

Figura 6 - Concordâncias da palavra ‘*resume*’

Fonte: (MOREIRA FILHO, 2011)

Pode-se analisar linhas de concordância e identificar três tipos de padrões: colocação, coligação e prosódia semântica. Colocação refere-se à junção significativa, co-ocorrência, entre palavras (*send/forward/submit/fax/email* + *resume* + *to*). Coligação é a relação gramatical mantida pelas palavras (verbo + substantivo [*resume*] + preposição). Prosódia é a recorrência entre itens lexicais que pode indicar uma conotação semântica negativa, positiva

ou neutra (o padrão '*send/fax resume to*' ocorre tipicamente com palavras de semântica e contextos neutros).

A identificação de padrões pode ser auxiliada também por outros tipos de visualização, como lista de colocados:

N	Palavra	Frequência	Marca
1	YOUR	9	
2	SEND	4	
3	AND	3	
4	FAX	3	
5	BOSTONWORKS	1	
6	APPLICATION	1	
7	PREF	1	
8	PROVIDEEMAIL	1	
9	ACCOUNTING	1	
10	THEIR	1	

Figura 7 - Lista de colocados da palavra *resume* na posição N-1 no *KitconC*

Fonte: (MOREIRA FILHO, 2011)

Na lista de colocados, é possível identificar a ocorrência das palavras '*send*' e '*fax*', colocados que fazem parte do padrão identificado.

Outra ferramenta é a exibição de gráficos de dispersão de palavras nos textos do *corpus*:

Figura 8 - Gráfico de dispersão da palavra '*RESUME*' no *KitconC*

Fonte: (MOREIRA FILHO, 2011)

Na figura, os gráficos indicam que a palavra '*resume*' ocorre mais no final dos textos. Tal tipo de visualização privilegiada pode ser útil para a análise da organização textual.

2.1.5 Outros programas

Quando há a necessidade de uso de algum recurso não encontrado nos programas mencionados na seção anterior, é comum a utilização de programas complementares. Um desses programas pode ser a planilha eletrônica do *Excel*, que disponibiliza opções de filtro, classificação, cálculos de fórmulas, entre outras. Vejamos a filtragem de substantivos, identificados pela etiqueta 'NN', a partir de um texto etiquetado:

Figura 9 - Utilização do recurso de filtro em dados anotados

Fonte: Produção do próprio autor

Embora o exemplo de utilização pareça simples, em conjunto com outras funções, o aplicativo pode suprir necessidades de determinadas pesquisas.

Um exemplo de utilização para pesquisa linguística é o trabalho de Silva e Teixeira (2010), que comparou os resultados de diferentes ferramentas de análise de *corpus* na tarefa de identificação de palavras candidatas a termo em textos sobre câncer de mama e desenvolveu uma metodologia com o auxílio do programa *Excel* para a filtragem dos candidatos mais comuns em todas as ferramentas e candidatos exclusivos de cada uma.

Programas para coleta de *corpus*, como o *WinHTTrack*¹⁰, um off-line browser, disponível em <http://www.httrack.com/>, permite o download de um *site* inteiro da Internet para o diretório de um computador local, incluindo *HTML*, imagens e outros tipos de arquivos. Em suas opções avançadas, é possível especificar os tipos de arquivos e condições de busca para a coleta.

Embora o programa não seja destinado especificamente para a pesquisa linguística, suas funções vão ao encontro da necessidade da coleta de grandes quantidades de textos na área, a qual também pode ser conseguida por meio de outros programas e *scripts* de linguagem de programação.

Outro tipo de programa, geralmente necessário após uma coleta automática de textos na Internet, é um conversor de *HTML* para textos sem formatação e códigos de *script* característicos. Tais programas podem ser encontrados facilmente na *Web*.

Também é possível escrever *scripts* em linguagem de programação *SHELL* ou *Python*, por exemplo, que utilizem expressões regulares para a remoção de determinados códigos e textos indesejados, fazendo uma ‘limpeza’ nos textos do *corpus* para o prosseguimento das análises.

Para os usuários do sistema operacional *Windows*, uma opção é a instalação do programa *Cygwin*, um emulador do sistema operacional *Unix* para *Windows*. No *Cygwin*, os *scripts* podem ser escritos e executados em um terminal de comandos.

Etiquetadores *online* e *desktop*, tais como o etiquetador do *site VISL (Visual Interactive Syntax Learning)*, *Brill* e *TreeTagger*, são utilizados para inserir automaticamente etiquetas que podem indicar, dependendo do tipo de etiquetagem, marcações morfossintáticas, sintáticas, semânticas ou discursivas. Um exemplo de tais marcações é ilustrado a seguir:

¹⁰ Outros exemplos são os *crawlers*, *bots* e *web spiders*, que varrem páginas da Internet para indexação e cópia.

```

([[] PU @PU #1->0
Reuters [Reuters] N P NOM @APP #2->0
)][] PU @PU #3->0
-[] PU @PU #4->0
The [the] ART S/P @>N #5->7
Arctic [Arctic] ADJ POS @>N #6->7
zone [zone] N S NOM @SUBJ> #7->8
has [have] <aux> V PR 3S @FS-ACC> #8->38
moved [move] <mv> V PCP2 AKT @ICL-AUX< #9->8
into [into] PRP @<SA #10->9
a [a] ART S @>N #11->12
warmer [warm] ADJ COM @P< #12->10
, [] PU @PU #13->0
greener [green] ADJ COM @>N #14->19
« [«] PU @PU #15->0
new [new] ADJ POS @>N #16->19
normal [normal] ADJ POS @>N #17->19
» [»] PU @PU #18->17
phase [phase] N S NOM @SUBJ> #19->0
, [] PU @PU #20->0
which [which] <rel> INDP S @SUBJ> #21->22
means [mean] <mv> V PR 3S @FS-N< #22->19
less [little] DET COM S @>N #23->24

```

Figura 10 - Exemplo de texto etiquetado

Fonte: (VISL, 2011)

A maioria das ferramentas disponíveis permite uma utilização avançada, em que o conjunto de etiquetas pode ser aumentado e/ou o usuário pode fazer um treinamento dos dados para melhor satisfazer as necessidades específicas de sua pesquisa.

2.1.6 Ferramentas computacionais em metodologia básica de pesquisa

As ferramentas descritas anteriormente podem ser aplicadas nos seguintes passos em pesquisas, compondo uma metodologia básica, em nível menos profundo, para a elaboração de cursos e extração de material para confecção de atividades pedagógicas. É possível que os passos também sejam utilizados em pesquisas com diferentes objetivos além dos especificados, tais como lexicografia e análise de *corpora* de aprendizes.

Vejam os passos:

- a. coleta do *corpus* – manual, ou automática a partir de programas específicos;
- b. preparação do *corpus* para as análises;
- c. limpeza e organização dos textos;
- d. análise das frequências das palavras do *corpus*;
- e. busca de indícios de palavras que se destacam no *corpus*.
- f. análise de palavras-chave do *corpus*;
- g. busca de palavras que se destacam no *corpus*;
- h. delimitação de um número de palavras para possível análise;

- i. análise da padronização das palavras selecionadas por meio de concordâncias, listas de colocados e n-gramas;
- j. descrição da padronização das palavras selecionadas como resultado;
- k. utilização dos padrões em objetivos seguintes (confeção de atividades, por exemplo).

A pequena metodologia descrita é ponto de partida inicial para uma gama de pesquisas em Linguística de *Corpus*. Os passos podem ser cíclicos e envolver algum tipo de recurso adicional durante o processo.

A metodologia, ainda que seja básica, exige grande esforço e trabalho do pesquisador, tanto na preparação do material, *corpus*, como na análise, identificação de padrões, por exemplo.

2.1.7 Linguística de *Corpus* e ensino

Dado o crescimento das pesquisas em Linguística de *Corpus*, com a compilação de *corpora* e disponibilidade de ferramentas para análise, há um grande interesse de sua utilização no ensino de línguas. Conforme Berber Sardinha (2004, p. 254), podem-se destacar quatro principais áreas:

- a. descrição da linguagem nativa;
- b. descrição da linguagem do aprendiz;
- c. transposição de metodologias de pesquisa acadêmica para sala de aula;
- d. desenvolvimento de materiais de ensino, currículos e abordagens.

A primeira, com maior concentração de pesquisas, tem como grande tarefa informar a produção de materiais, tais como livros didáticos, dicionários e gramáticas. A segunda tem como objetivo a análise da produção do aprendiz, textos escritos ou transcrições de fala de não-nativos (*corpora* de aprendizes), geralmente por meio dos conceitos de sobreuso e subuso em relação à linguagem do falante nativo. A terceira está relacionada ao uso direto do material de análise, como linhas de concordância e listas de palavras em sala de aula. A quarta área está relacionada a metodologias ou abordagens de ensino, tal como:

- a. o Currículo Lexical (*lexical syllabus*);
- b. a Abordagem Lexical (*lexical approach*)
- c. o Ensino Movido a Dados (*data-driven learning*).

O Currículo Lexical tem como proposta o ensino dos itens léxico-gramais (colocações) mais comuns da língua a partir de corpora formados pela linguagem de falantes nativos, tendo como princípio o ensino por meio da descoberta. A Abordagem Lexical tem como base o ensino de itens léxico-gramaticais, em que léxico e gramática não são vistos de maneira dicotômica. As atividades são centradas no conteúdo léxico-gramatical e o aluno tem o papel de descobridor. O Ensino Movido a Dados tem como princípio o desenvolvimento da habilidade de descoberta do aluno a partir da análise de dados, como linhas de concordância e listas de palavras.

O uso de *corpora* no ensino pode se dar a partir das seguintes maneiras: i. ensino sobre *corpora*; ii. exploração de *corpora* para ensinar língua, linguística e/ou outras disciplinas; iii. ensino para exploração de *corpora*. A primeira está mais relacionada ao estudo teórico sobre a área. A segunda refere-se ao aproveitamento da instrumentação e dados de análise, geralmente a identificação de itens de ensino e produção de materiais. A terceira é a favor de se ensinar os aprendizes para que possam explorar *corpora* autonomamente.

Em relação à exploração de *corpora* para ensinar língua, é possível evidenciar as contribuições e possibilidades do uso de *corpora* de língua geral, grandes *corpora*, e *corpora* especializados, de determinado gênero ou registro definido, para o que se deve ensinar aos aprendizes e para extração de recursos a serem incluídos no material de ensino (ASTON, 2000).

Análises de grandes *corpora* mostram que os currículos e materiais tradicionais ignoram muitos itens linguísticos de grande frequência na linguagem nativa e dão ênfase a itens relativamente raros. As análises também revelam discrepâncias em padrões gramaticais em relação ao currículo e materiais de ensino, quando baseados unicamente em categorias gramaticais, sem a consideração da influência de escolhas de itens lexicais. A proposta de uso de *corpora* para a criação de atividades nesta pesquisa vai ao encontro de soluções que evitam os problemas mencionados e potencializam o desenvolvimento de currículos e a elaboração de materiais didáticos adequados às necessidades de aprendizes.

Os *corpora* especializados são adequados para o levantamento sobre o que ensinar no ensino de línguas para fins específicos, por meio da definição de padrões lexicais e gramaticais. Um

benefício da utilização de *corpora* especializados é a facilidade de se relacionar os itens léxico-gramaticais a funções pragmáticas específicas, dada a recorrência de tais funções, geralmente em posições similares. Por exemplo, em um *corpus* de anúncios de emprego, a informação sobre a forma de envio do currículo de um candidato pode ser veiculada pelo conjunto de verbos: '*forward*', '*submit*', '*fax*', '*send*', '*email*' e '*contact*'.

Embora possam apresentar coligações diferentes, as palavras contribuem para uma função pragmática semelhante, específica do gênero: indicar como o interessado deve se candidatar a uma vaga oferecida. Tal fato abre oportunidades para a descoberta de significados e formas associadas ao gênero, marcando uma tendência no ensino de línguas em relação à definição do que ensinar em termos de semântica e pragmática, em contraste a uma abordagem puramente formal.

O exemplo apresentado relaciona-se intimamente com possíveis abordagens gerais de análise de *corpora* especializados tanto para sua interpretação como para sua construção. Os padrões evidenciados nas linhas de concordância da Figura 6 podem ser de relevância para o ensino do gênero anúncios de emprego, especificamente se o objetivo for ensinar o aprendiz a ler ou a escrever um anúncio de emprego. A identificação e extração de padrões desse tipo podem ser facilmente realizadas a partir de um *corpus* criteriosamente compilado.

Um procedimento geral utilizado na análise de *corpora* especializados é o uso de programas para extração de palavras que ocorrem significativamente mais no *corpus* especializado do que em um *corpus* de língua geral (referência). A comparação promovida por tais programas possibilita a identificação das palavras-chave do *corpus*, as quais estão associadas aos significados e funções que caracterizam o gênero do *corpus* especializado. No *corpus* de anúncios de emprego, por exemplo, palavras como '*experience*', '*skills*', '*ability*', '*requirements*', '*candidate*', '*salary*' e '*resume*' seriam possíveis primeiras palavras a serem identificadas.

Nesse sentido, Flowerdew (1993) descreve o uso de programas aplicados a um *corpus* especializado, com tamanho de 104.483 itens e 4.232 formas, na elaboração de um curso de inglês para fins específicos. Em sua proposta, a partir dos resultados de um programa listador de palavras, uma série de critérios e filtros relacionados à frequência das formas no *corpus* é utilizada em uma análise manual realizada pelo linguista para definir os itens a serem

ensinados, além de o material servir para a preparação das atividades por meio de um concordanciador.

Nesta pesquisa, considera-se que parte dos procedimentos do processo de análise de *corpora* para fins de ensino de línguas pode ser automatizada a partir de soluções específicas. Regras e filtros automáticos, como a restrição de itens por frequência, são exemplos de dispositivos para sua implementação. O uso de algoritmos, técnicas e métodos do Processamento de Línguas Naturais e do Aprendizado de Máquina colabora para o aperfeiçoamento da automatização das análises linguísticas.

Desse modo, o uso de *corpora* como recurso para materiais de ensino vai ao encontro das necessidades dos aprendizes em relação ao aprendizado do uso real e significativo da língua.

2.2 Processamento de Línguas Naturais

O Processamento de Línguas Naturais (PLN) é uma área da Ciência da Computação e Engenharia que se desenvolveu a partir do estudo da linguagem e da Linguística Computacional dentro do campo da Inteligência Artificial. De acordo com Nunes (2008, p. 2), uma definição abrangente é: “Processamento de Línguas Naturais (PLN) é o nome que se dá à área de pesquisa que se dedica a investigar, propor e desenvolver formalismos, modelos, técnicas, métodos e sistemas computacionais que têm a língua natural como objeto primário”.

A partir de tal definição, a área de PLN tem como áreas correlatas a Linguística Computacional, a Linguística Aplicada, a Linguística de *Corpus*, a Mineração de Textos (*Text Mining*). Ainda conforme Nunes (2008, p.2), é comum a diferenciação entre a Linguística de *Corpus* e a área de PLN a partir das perspectivas linguística ou computacional que cada uma delas pode ter em relação ao mesmo objeto.

Em PLN, é comum a ênfase no tratamento de língua escrita e suas questões linguísticas, uma vez que o tratamento da fala, por questões tecnológicas relacionadas à síntese e ao reconhecimento do som, pode estar geralmente ligado a áreas da Engenharia Elétrica, o que não exclui os trabalhos de processamento de fala na área.

Com a introdução do computador em nossa cultura, as possibilidades de seu uso para diversas atividades e avanços nos mais variados campos da ciência, inclusive os novos ramos de pesquisa criados, a necessidade de realizar a comunicação com a máquina para que instruções sejam executadas na consecução de tarefas levou à criação de linguagens de programação, tal como as linguagens *Lisp* e *Prolog*, que se diferenciam das línguas naturais por serem criadas artificialmente e possuírem uma característica rígida e invariável (DIAS DA SILVA et al., 2007, p. 4). O passo seguinte foi a criação de interfaces gráficas para que o usuário não tivesse de digitar dezenas de comandos em linguagem de programação para realizar as tarefas desejadas.

Embora a comunicação com a máquina tenha evoluído a partir da criação de interfaces gráficas e recursos simbólicos para facilitar o uso do computador, fazer com que a máquina compreenda mensagens codificadas em língua natural é ainda um desafio e motivo para diversas pesquisas em PLN.

Dado o desafio, muitos autores consideram que o marco inicial do uso do computador para a investigação das línguas naturais foi a tarefa de tradução automática (DIAS DA SILVA et al., 2007, p. 5). Inicialmente, os primeiros trabalhos levantaram grandes expectativas, mas resultados não favoráveis levaram a um período de hibernação da área até sua retomada.

O estímulo a pesquisas em PLN teve especial contribuição de investigações como a de Winograd, em uma abordagem a *micromundos*, que desenvolveu um sistema chamado *SHRDLU*, que conseguia conversar sobre um mundo de blocos de diferentes formas e cores. O sistema podia responder a perguntas em inglês, como "*O que está sobre o bloco vermelho*", além de lidar com referências pronominais e compreender elipses. Com a aplicação limitada a conhecimento mínimo do domínio (LUGER, 2013, p. 514), o sistema *SHRDLU* demonstrou a possibilidade de se comunicar em língua natural com a integração de sintaxe e semântica.

Como breve histórico, Dias da Silva et al. (2007, p. 8) apresenta um quadro com a síntese da evolução dos estudos do PLN a partir da tradução automática:

Quadro 2 - Evolução dos sistemas de PLN

Década de 50: A Tradução automática

- _ sistematização computacional das classes de palavras da gramática tradicional;
- _ identificação computacional de poucos tipos de constituintes oracionais.

Década de 60: Novas aplicações e criação de formalismos

- _ primeiros tratamentos computacionais das gramáticas livres de contexto;
- _ criação dos primeiros analisadores sintáticos.
- _ primeiras formalizações do significado em termos de redes semânticas

Década de 70: Consolidação dos estudos do PLN

- _ implementação de parcelas das primeiras gramáticas e analisadores sintáticos;
- _ busca de formalização de fatores pragmáticos e discursivos.

Década de 80: Sofisticação dos sistemas

- _ desenvolvimento de teorias lingüísticas motivadas pelos estudos do PLN.

Década de 90: Sistemas baseados em “representações do conhecimento”

- _ desenvolvimento de projetos de sistemas de PLN complexos que buscam a integração dos vários tipos de conhecimentos lingüísticos e extralingüísticos e das estratégias de inferência envolvidos nos processos de produção, manipulação e interpretação de objetos lingüísticos.

Fonte: (DIAS DA SILVA et al., 2007, p. 8)

Assim, os objetivos do PLN têm, inicialmente, relação com o desenho e construção de aplicações que facilitem a interação com máquinas e outros dispositivos por meio do uso de línguas naturais. De acordo com Nunes (2008, p. 4), recentemente, o PLN estuda o tratamento de problemas mais pontuais, dedicando-se a tarefas básicas, como: pré-processamento de textos, classificação automática (etiquetagem) e mapeamento de representações (sintática, semântica ou discursiva). Os propósitos do PLN também são variados atualmente (extração de informações, geração automática de textos, correção ortográfica etc.).

O desenvolvimento de aplicações de PLN geralmente pode incorporar uma ou mais das tarefas básicas, em uma abordagem de fragmentação do problema a ser solucionado por meio da combinação das soluções das partes. Conforme Nunes (2008, p. 5), uma aplicação é entendida como "qualquer sistema computacional que, a partir de uma entrada lingüística, realiza um processamento que a transforma de tal maneira a produzir um determinado resultado".

As áreas de desenvolvimento de aplicações mais comuns do PLN (DIAS DA SILVA et al., 2008) são: Sistemas de Perguntas e Respostas, Sumarização, Tradução Automática, Reconhecimento de Fala, Classificação de Documentos, Sistemas de Recuperação de Informação, Sistemas de Extração de Informação, Sistemas de Diálogos, Sistemas de Auxílio à Escrita (editores e revisores de textos), Sistemas de Automação de Tarefas Administrativas,

Programação Automática, Sistemas de Processamento de Textos Científicos e Sistemas Especializados.

Para a construção de tais sistemas, recursos linguísticos e computacionais são necessários, tais como *corpora*, léxicos, dicionários, ontologias e gramáticas computacionais. A construção de módulos de análise para muitas das camadas da descrição linguística requer tais recursos, que podem ser específicos para os modelos de língua adotados. Por exemplo, para análises sintáticas, gramáticas livres de contexto (probabilísticas ou não) que se constituem em um conjunto de regras de reconhecimento e produção (Ex.: $S \Rightarrow np\ vp^{11}$) podem ser utilizadas.

Em relação a questões conceituais e metodológicas, a partir das contribuições de Winograd, o PLN assume a complexidade do entendimento e simulação de uma língua natural pelo computador e reconhece a necessidade de fornecimento de um modelo detalhado do domínio específico do discurso, com informações e análises dos diversos níveis de processamento da língua natural: fonético-fonológico, morfológico, sintático, semântico e pragmático-discursivo, além do conhecimento de mundo. Assume-se também que os modelos de PLN são geralmente incompletos no sentido de compreender toda a complexidade da língua natural, porém considera-se que seus estudos podem auxiliar na compreensão de fatos da língua.

Conforme Dias da Silva et al. (2007, p. 12), a simulação de uma língua natural requer vários sistemas de conhecimento, além da realização de uma série de atividades cognitivas. Sistemas de PLN em tal cenário incluem tarefas automatizadas para revisões ortográficas de textos, análises sintáticas, análises de texto, tradução, entre outras.

O processo de construção de sistemas de PLN, como os já citados, envolve etapas sistemáticas, que são desenvolvidas sucessiva, progressiva e ciclicamente. Dias da Silva (1996) argumenta que os projetos de desenvolvimento de sistemas de PLN precisam passar por níveis de processamento gramatical e pragmático-discursivo em três domínios: o Linguístico, o Representacional (Linguístico-computacional) e o Implementacional (Computacional).

Desse modo, as fases de análise propostas são:

¹¹ Os elementos da regra exemplificada são: *S* (*Sentence*); *np* (*noun phrase*); *vp* (*verb phrase*).

- a. fase de representação linguística - conceituação do objeto linguístico a ser modelado;
- b. fase de representações e algoritmos - modelagem da conceituação;
- c. fase da construção do sistema - implementação do modelo.

Em PLN, projetar sistemas para manipulação da língua natural é uma tarefa árdua, uma vez que a ambiguidade natural da língua é uma realidade (NUNES, 2008, p.4). Tarefas básicas como etiquetagem morfofssintática ou análises sintáticas de uma sentença são exemplos da problemática enfrentada na área. Muitas vezes, pesquisadores e projetistas deparam-se com situações em que uma palavra pode possuir mais de um significado, mais de uma etiqueta morfofssintática e/ou uma sentença pode ter mais de uma análise sintática válida.

Desse modo, os sistemas em PLN devem ser capazes de realizar desambiguações e decidir quais análises são adequadas dado o problema. Uma série de abordagens e estratégias são estudadas para a resolução de ambiguidades na área, seja por processos estocásticos ou simbólicos. O uso de dados de *corpora* tem mostrado resultados relevantes para tais tarefas.

A avaliação dos sistemas em PLN também é um dos problemas de estudo na área, uma vez que as soluções não são completas e que pode ser difícil identificar quais soluções ou partes contribuíram ou não para o desempenho. Somam-se ainda críticas na utilização de humanos para a tarefa de avaliação, relacionadas ao processo científico, principalmente em relação à replicação dos resultados. Como exemplo, para a tarefa de etiquetagem morfofssintática, pode haver dificuldades de especialistas humanos em chegar a um consenso de 100% em relação a determinadas análises.

Neste trabalho, para o desenvolvimento dos módulos de análise de textos, realiza-se um estudo, aplicação e experimentos dos principais métodos de PLN relacionadas às necessidades da pesquisa, tal como a divisão das análises em diferentes módulos específicos e sua aplicação em sequência no processamento de textos. O processo de construção e uso da metodologia de sistemas de PLN, como as etapas e fases sistemáticas, estão presentes no desenvolvimento, muito embora não estejam explicitamente marcadas, dada a diversidade de análises requeridas.

2.3 Aprendizado de Máquina

Atualmente, há uma grande diversidade de tarefas e problemas que os computadores conseguem resolver a partir de um dado aprendizado, como o funcionamento de simuladores, jogos, sistemas de buscas que aprendem as preferências dos usuários, detecção automática de *spams*, detecção automática de rostos em fotos, reconhecimento de escrita, recomendação de produtos, além de sistemas de localização e direção, mas ainda há muitas tarefas que os computadores não conseguem realizar, principalmente no campo das línguas naturais.

O Aprendizado de Máquina é a área da Inteligência Artificial que se ocupa do desenvolvimento de algoritmos que aprendem ou melhoram seu desempenho a partir de dados de experiência armazenados (PUSTEJOVSKY e STUBBS, 2012, p. 18) e está entre a Ciência da Computação, a Engenharia e a Estatística como um meio de explorar a grande quantidade de dados gerada atualmente para transformar dados em soluções úteis, com aplicações a diversos campos e áreas.

Dentro da área, é possível encontrar diversos métodos para o Aprendizado de Máquina: simbólico, conexionista, genético e estocástico (LUGER, 2013). Métodos estatísticos também são comumente utilizados em algoritmos de aprendizado de máquina como fonte de aprendizado para extrapolação do que já é sabido, estendendo o conhecimento adquirido a dados que ainda não foram vistos ou anotados, isto é, a dados de teste ou de aplicação real do que foi aprendido.

Técnicas de Aprendizado de Máquina dependem de dados de aprendizado relevantes para sua efetividade, mais do que de grandes quantidades de dados brutos. Tais dados relevantes podem vir na forma de anotações, metadados do texto, que são necessários para ensinar o computador a desempenhar com sucesso determinada tarefa. É preciso cuidado no processo de construção de dados de qualidade para algoritmos de Aprendizado de Máquina.

Para os objetivos da pesquisa, focada no aprendizado, os dados utilizados são os de língua natural, que podem estar na forma de texto, geralmente anotados com etiquetas relacionadas a atributos específicos e relevantes para uma determinada tarefa.

Ao lidar com Aprendizado de Máquina em PLN, três diferentes tipos de algoritmo podem ser encontrados (MARSLAND, 2009):

- a. aprendizado supervisionado - informa ao algoritmo o que prever (classificação e regressão);
- b. aprendizado não supervisionado - não informa o rótulo ou o valor alvo para os dados (*clustering*);
- c. aprendizado semi-supervisionado - trabalha a partir de dados rotulados e não rotulados.

Uma pequena descrição das principais tarefas e algoritmos é apresentada (PUSTEJOVSKY e STUBBS, 2012, p. 19):

- a. classificação - tarefa de identificar o rótulo de uma única entidade a partir de um conjunto de dados. Os algoritmos mais comuns são: *Maximum Entropy (Maxent)*, *Naive Bayes*, *Decision trees*, e *Support Vector Machines (SVMs)*;
- b. *clustering* - algoritmos que encontram agrupamentos e padrões a partir de dados de entrada, sem qualquer tipo de rotulagem ou dados de treinamento. Os algoritmos mais comuns são: *K-means*, *Hierarchical clustering*, *Kernel Principle Component Analysis*, e *Fuzzy C-means*;
- c. *structured pattern induction* - mais que o aprendizado de rótulos ou categoria de uma única entidade, mas o aprendizado de uma sequência de rótulos, ou outras dependências estruturais entre os itens rotulados. Os algoritmos mais comuns são: *Hidden Markov Models (HMMs)*, *Conditional Random Fields (CRFs)*, e *Maximum Entropy Markov Models*.

A escolha do algoritmo de aprendizado adequado está relacionada ao objetivo e dados disponíveis, isto é, ao que se quer conseguir e a quais dados se tem ou deseja coletar. Por exemplo, em relação ao objetivo, conseguir a probabilidade de chover no dia ou identificar os grupos de eleitores com preferências semelhantes podem levar a diferentes escolhas de algoritmo. Encontrar o melhor algoritmo é questão de um processo de tentativa e erro.

Técnicas podem ser utilizadas para melhorar o desempenho de algoritmos, dentre elas a anotação de dados, técnica comum para treinamento de algoritmos de aprendizado em PLN.

Embora os computadores possuam um grande poder de processamento, servindo como fonte de todo tipo de informações e meio de comunicação, ainda não possuem poder de entendimento da linguagem na mesma proporção com que lidam com ela, isto é, consegue-se processar frações específicas da linguagem por meio dos avanços em PLN.

Linguistas computacionais e teóricos têm a preocupação de desvendar a natureza profunda da língua e capturar as propriedades computacionais de estruturas linguísticas. O intenso uso de computadores, principalmente com a Internet, tem fornecido grandes quantidades de dados linguísticos que possibilitam a modelagem de problemas linguísticos em tarefas de aprendizado de máquina de uma maneira nunca antes vista. Não é suficiente, no entanto, apenas fornecer grandes quantidades de dados ao computador a fim de que ele consiga entender plenamente qualquer aspecto da linguagem humana no desempenho de determinada tarefa. Há a necessidade de seleção e preparação de dados para que o computador possa encontrar padrões e fazer inferências.

Para que o computador possa obter dados significativos para o processo de aprendizado, é necessário fornecer metainformações para o conjunto de dados. O emprego de marcas/etiquetas aos elementos do conjunto de dados é chamado de anotação. A anotação é, pois, o processo de adição de metainformações ao texto com o objetivo de melhorar o desempenho de algoritmos de aprendizado (PUSTEJOVSKY e STUBBS, 2012, p. 28). Para que algoritmos de aprendizado de máquina sejam efetivos, os dados de anotação devem ser precisos e relevantes para a tarefa que a máquina deve executar. Assim, é vital que haja um cuidado especial na geração de dados para o aprendizado.

O conjunto de dados linguísticos, em língua natural, pode também ser chamado de *corpus*. Um *corpus* anotado pode servir para o treinamento de algoritmos de aprendizado de máquina.

O processo de anotação para uso em aprendizado de máquina não é linear, visto que depende de objetivos particulares e de adequações necessárias ao longo do trabalho, tal como a revisão da teoria utilizada e o nível de detalhamento das informações. Contudo, tal processo pode ser resumido em cinco etapas básicas: modelagem, anotação de treinamento, teste, avaliação e revisão (PUSTEJOVSKY e STUBBS, 2012, p. 21). Na figura a seguir, apresenta-se o esquema do processo de anotação:

Figura 11 - Ciclo do desenvolvimento de anotação

Fonte: (PUSTEJOVSKY e STUBBS, 2012, p. 21)

Considerando as diferentes camadas de descrição linguística, há diferentes tipos de anotação de acordo com os objetivos e características do tipo de conhecimento que se pretende abordar.

Um dos processos mais básicos de anotação é o de etiquetagem morfossintática (*POS*¹² *tagging*), em que códigos/etiquetas que indicam a classe gramatical de cada palavra são assinalados (substantivo, verbo, adjetivo etc.). O número de etiquetas pode variar de acordo com o nível de detalhamento desejado. No *corpus Brown*, por exemplo, são encontradas 77 diferentes etiquetas.

A etiquetagem morfossintática é uma etapa essencial para muitas aplicações de Processamento de Línguas Naturais, uma vez que subsequentes análises poderão ser realizadas. Vejamos um exemplo de anotação morfossintática a partir de uma sentença do *corpus Brown* nos recursos do *NLTK*.

The/AT girl/NN was/BEDZ in/IN critical/JJ condition/NN with/IN
burns/NNS over/IN 90/CD per/IN cent/NN of/IN her/PP\$ body/NN ./.

Figura 12 - Exemplo de anotação morfossintática

Fonte: (BIRD; KLEIN; LOPER, 2009)

No exemplo, cada item/palavra recebe um código. A descrição dos códigos no exemplo é a seguinte: 'AT' (artigo), 'NN' (substantivo, singular, comum), 'BEDZ' (verbo *to be*, passado, 1ª e 3ª pessoas do singular), 'JJ' (adjetivo), 'IN' (preposição), 'NNS' (substantivo, plural, comum), 'CD' (numeral, cardinal), 'PP\$' (determinante, possessivo).

¹² *Part-Of-Speech*.

Outro tipo de anotação pode considerar as funções sintáticas de itens em uma sentença. As sequências específicas de palavras podem ser anotadas por meio de parênteses. Esse tipo de anotação leva em consideração dois tipos de relações: ordem (precedência) e hierarquia (dominância), em representação de árvore (STUBBS e PUSTEJOVSKY, 2012, p. 16):

```
( (S (NP-SBJ-1 (NP Rudolph Agnew)
    ,
    (UCP (ADJP (NP 55 years)
    old)
    and
    (NP (NP former chairman)
    (PP of
    (NP Consolidated Gold Fields PLC))))
    ,)
    (VP was
    (VP named
    (S (NP-SBJ *-1)
    (NP-PRD (NP a nonexecutive director)
    (PP of
    (NP this British industrial conglomerate))))))
    .))
```

Figura 13 - Exemplo de anotação sintática do *corpus Penn Treebank*

Fonte: (BIRD; KLEIN; LOPER, 2009)

No exemplo, parênteses e etiquetas são utilizadas para a representação da análise sintática de uma sentença. Aplicações de Processamento de Línguas Naturais mais sofisticadas podem demandar o processamento sintático, seja completo (*full parsing*) ou parcial (*shallow parsing*).

Na anotação semântica, uma palavra ou frase na sentença é etiquetada com um identificador a partir de um vocabulário ou ontologia, por exemplo, organização, pessoa, lugar, tempo etc., de acordo com seu papel semântico. Abaixo, é apresentada a anotação de uma sentença:

```
<sentence>
During the <entity type='TIME'>2008</entity> presidential election,
importers tried to stuff drugs into fake <entity type='ORGANIZATION'>Nike</entity>
sneakers emblazoned with <entity type='PERSON'>Barack Obama</entity> 's "Yes we can"
campaign slogan.
</sentence>
```

Figura 14 - Exemplo de anotação semântica

Fonte: Produção do próprio autor

No exemplo, a anotação é feita por meio da linguagem *XML*. Em *XML*, não há um conjunto de *tags* predefinido, mas o programador o define de acordo com sua necessidade para estruturar o documento, como, no caso citado, *tags* para indicar a sentença e entidades nomeadas.

Na pesquisa, em diversas etapas do desenvolvimento, fez-se necessária a anotação de dados para o treinamento de algoritmos de aprendizado. A linguagem *XML* foi extensamente utilizada na atividade, tendo em vista a estruturação dos documentos de texto (título, corpo do texto, parágrafos, sentenças, itens etc.) e a combinação de diferentes tipos de informação.

Os passos básicos para o uso de algoritmos de aprendizado de máquina são:

- a. coleta de dados;
- b. preparação dos dados de entrada;
- c. análise dos dados de entrada;
- d. treinamento do algoritmo;
- e. teste do algoritmo;
- f. uso do algoritmo.

A maneira mais comum de realizar o treinamento e teste é dividir os dados em duas partes: *corpus* de desenvolvimento (*development corpus*) e *corpus* de teste (*test corpus*). O *corpus* de desenvolvimento pode ser subdividido em duas partes: treinamento (*training*) e desenvolvimento/teste (*development-test*).

Como exemplo, nos experimentos de desenvolvimento de um etiquetador, utilizou-se o *corpus Brown*, composto por 1.161.192 itens, divididos da seguinte forma: para treinamento, 928.954; desenvolvimento, 116.118; e teste, 116.120. A seguir, uma representação gráfica da divisão do *corpus* para tal fim é apresentada:

Figura 15 - Divisão do *corpus Brown* para o desenvolvimento do etiquetador

Fonte: Produção do próprio autor

Os dados de treinamento são utilizados para treinar o algoritmo para a tarefa. Uma vez treinado, o algoritmo é aplicado ao *corpus* de desenvolvimento. A execução do algoritmo geralmente gera erros, que podem ser identificados e localizados para uma análise. Após a identificação e análise dos erros, o algoritmo é ajustado para uma nova execução no *corpus* de desenvolvimento. Tal processo pode ser repetido até que se obtenham os resultados esperados. Tendo completado o treinamento e os testes no *corpus* de desenvolvimento, o algoritmo é executado no *corpus* de teste. O teste possibilita verificar o desempenho do algoritmo em novos dados que ainda são desconhecidos pelo aprendizado realizado. Esse procedimento permite simular e avaliar o uso do algoritmo em situações mais comuns em que sua execução será realizada.

2.4 Avaliação de resultados

Nesta seção, apresentamos as principais abordagens utilizadas para a avaliação de resultados da pesquisa em relação aos módulos e algoritmos de análise linguística.

A maneira mais comum de avaliar o desempenho de um algoritmo é calcular o quanto ele consegue classificar adequadamente os dados (PUSTEJOVSKY e STUBBS, 2012, p. 26). Para tanto, podem ser utilizadas as técnicas de cálculo da precisão (*precision*) e abrangência/cobertura (*recall*).

O cálculo é feito dividindo-se os dados em dois conjuntos, um considerado relevante/correto e outro, irrelevante/incorreto. A métrica de precisão é calculada pela diferença entre as instâncias corretas em relação às incorretas. A métrica de abrangência é calculada pela diferença entre as instâncias corretas que realmente pertencem ao conjunto de dados relevantes.

Há ainda uma medida que combina os resultados de precisão e abrangência, chamada de *F-measure*. *F-measure* é a média harmônica das medidas de precisão e abrangência.

Assim, há as seguintes equações a partir dos valores verdadeiro positivo (vp), verdadeiro negativo (vn), falso positivo (fp) e falso negativo (fn):

Precisão: $P = vp / (vp + fp)$

Abrangência/Cobertura: $C = vp / (vp + fn)$

Acurácia: $A = vp + vn / (vp + vn + fp + fn)$

F-measure: $F = 2 * (P * C) / (P + C)$

Para ilustração, seguem os resultados de um sistema que identifica palavras cognatas em um texto. O sistema, ao analisar um texto de 350 palavras que possui 10 palavras cognatas, identifica 15, sendo que apenas 7 das palavras identificadas são realmente cognatas. O número de itens identificados é 15, e o número de itens relevantes não identificados, 3, de onde o seguinte cálculo:

Precisão: $P = 7 / (7 + 3) \Rightarrow 0.7$

Abrangência/cobertura: $C = 15 / (15+3) \Rightarrow 0.83$

Acurácia: $A = 7 + 343 / (7+343+8+3) \Rightarrow 0.97$

F-measure: $F = 2 * ((0.7 * 0.83) / (0.7 + 0.83)) \Rightarrow 0.76$

A obtenção de valores mais confiáveis, a partir da divisão dos conjuntos de treinamento e teste, pode ser conseguida por meio de aplicações de métodos como *holdout* e validação cruzada.

O método *holdout* está relacionado à divisão dos dados em dois subconjuntos mutuamente exclusivos, sendo um para treinamento e outro para teste e validação. Geralmente, são utilizados "dois terços dos dados para treinamento e um terço para teste" (DOMINGUES, 2011, p. 49). Tal método é mais indicado quando há uma grande quantidade de dados.

O método de validação cruzada, também conhecido como *k-fold*, indicado quando a quantidade de dados é limitada, basicamente é a repetição do método *holdout*. No método, todos os dados são utilizados para treinamento e teste. Os procedimentos envolvem a divisão randômica dos dados em *k* partes de mesmo tamanho. Em cada iteração de *k*, uma das partes é utilizada para teste o restante para treinamento, sendo calculada e armazenada a taxa de erro. O resultado final é a média das taxas calculadas.

Na pesquisa, os métodos de avaliação foram amplamente empregados conforme as especificidades das análises nos módulos de análise linguística e nos experimentos.

2.5 Ensino de leitura em língua estrangeira e criação de atividades

O ensino de leitura em língua estrangeira é um tópico de grande relevância, uma vez que é comum os aprendizes da língua terem uma forte necessidade, como geralmente seu contexto demanda, de ler para obter informações, para fins de estudo ou atividade profissional.

Solé (1998) aponta a inesgotável diversidade de objetivos de leitura frente a um texto, tendo em vista os diferentes leitores, situações e momentos, como ler para obter uma informação precisa, ler para seguir instruções, ler para obter uma informação de caráter geral, ler para aprender, ler para revisar um escrito próprio, ler por prazer, ler para comunicar um texto a um auditório, ler para praticar a leitura em voz alta e ler para verificar o que se compreendeu.

Para aprendizes de língua estrangeira, a necessidade de ler textos acadêmicos da área de estudo ou ler textos de uma prova de vestibular para responder a perguntas, por exemplo, ilustra o anseio de se adquirir muito mais a habilidade de leitura, privilegiando nesses casos apenas uma habilidade, tendo em vista os propósitos específicos.

A valorização da leitura está também presente no fato de que o trabalho com textos leva a bons modelos de escrita, além de fornecer oportunidades para introdução de novos tópicos

para discussão e estudo de aspectos formais da língua (RICHARDS e RENANDYA, 2002, p. 273).

Conforme Holden (2009, p. 56), a compreensão de um texto requer diferentes tipos de conhecimento: conhecimento de mundo, conhecimento da linguagem, percepção textual (conhecimento de gêneros textuais) e percepção não textual (elementos tipográficos). Dentre tais conhecimentos, é destacada a importância de se expor o aprendiz a uma variedade ampla de gêneros para tornar suas habilidades de leitura mais flexíveis.

Sobre o ensino de gêneros textuais em cursos de inglês para fins específicos, Ramos (2004) considera que o trabalho pode auxiliar o professor tanto em seu melhor entendimento do texto (enquanto gênero), quanto em relação ao que os alunos têm de fazer linguisticamente, e propõe uma prática de ensino a partir de três fases: apresentação, detalhamento e aplicação.

Na fase de apresentação, deve-se criar condições para que o gênero a ser trabalhado seja observado de uma perspectiva ampla (contextualização, conscientização e familiarização). Na fase de detalhamento, deve-se fornecer condições satisfatórias para a compreensão/produção geral e detalhada dos textos, como explorar a função discursiva e os componentes léxico-gramaticais particulares ao gênero em pauta, além de levar à compreensão dos significados e à relação entre um texto e seu contexto de situação. Na fase de aplicação, deve-se fazer o aluno trabalhar com o gênero como um todo, reintegrando os vários conhecimentos que veio adquirindo (em fases de consolidação¹³ e apropriação¹⁴). A implementação das fases pode envolver atividades com perguntas e preenchimento de quadros.

Para a criação de atividades de leitura, Grellet (1981) apresenta um conjunto de itens para a confecção de uma aula de leitura baseada em texto: utilização de estratégias como *skimming*¹⁵ e *scanning*¹⁶; interpretação do texto; algum tipo de produção que considere o texto como um todo. Dentre os itens, para auxiliar a interpretação do texto, atividades que explorem informações não textuais, habilidades para lidar com vocabulário desconhecido e características do texto podem ser utilizadas. Os itens formam um quadro conceitual flexível e adaptável para elaboração de planos de aula para o ensino de leitura.

¹³ Fase para retomar o conceito do todo.

¹⁴ Fase de transferência do gênero para situações da vida real.

¹⁵ Estratégia de leitura em que se observa o texto rapidamente para detectar o assunto geral.

¹⁶ Estratégia de leitura em que se corre rapidamente os olhos pelo texto para identificação de informações específicas.

Holmes (1982) fornece um guia para a preparação de materiais para leitura em um curso de inglês para fins específicos, o qual também aborda questões de estratégias de leitura e atividades. Nele, três pontos principais são desenvolvidos em relação à elaboração de atividades de leitura: níveis de compreensão, estratégias para facilitar a leitura e tipos de exercícios. Para a preparação de materiais de ensino, são necessárias a análise da matéria-prima (texto) e o conhecimento de tipos de exercícios.

As atividades devem ser estruturadas a partir de níveis de compreensão, com a análise do texto para exercícios de predição, compreensão geral, compreensão de pontos principais e compreensão detalhada. O quadro a seguir mostra os estágios estruturantes para a preparação da atividade com o texto:

Quadro 3 - Estágios para elaboração de atividades de leitura

Estágio	Objetivos	Possíveis formas de atividade
Estágio 1: Predição	<ul style="list-style-type: none"> - preparar os alunos mentalmente para o texto; - encorajá-los a pensar sobre o tópico; - deixá-los interessados e motivados com o texto. 	Discussões gerais e discussões sobre vocabulário.
Estágio 2: compreensão geral	- ensinar as estratégias de <i>skimming</i> e <i>scanning</i> .	Perguntas para <i>skimming</i> e <i>scanning</i> .
Estágio 3: compreensão de pontos principais	- tomar nota das principais ideias do texto.	Realização de anotações, preenchimento de diagramas, conectores de parágrafos, classificação (comparação contraste), identificação de tópicos, sinonímia e hiponímia ¹⁷ , escrita de resumo, reorganização de parágrafos e identificação de sentenças irrelevantes ou supérfluas.
Estágio 4: compreensão detalhada	<ul style="list-style-type: none"> - decidir conscientemente se o texto merece uma compreensão detalhada; - definir um propósito para o texto; - ler praticamente palavra por palavra, inferindo o significado de palavras desconhecidas a partir de pistas contextuais/gramaticais. 	Desenrolando frases longas, identificação do núcleo e adjetivos em sintagmas nominais, identificação de elementos coesivos, gramática, questões de verdadeiro ou falso e resumo.

Fonte: (Holmes, 1982)

¹⁷ Relação semântica entre uma palavra de sentido mais específico e outra de sentido mais genérico.

Scott et al. (1984), com experiência em projetos de ensino de inglês para fins específicos no Brasil, relatam o uso de uma atividade padrão para treinar aprendizes em estratégias de leitura eficientes que poderiam ser utilizadas com quase qualquer texto. Embora o procedimento estivesse relacionado ao ensino de inglês para fins específicos, seria possível utilizá-lo em cursos de inglês geral. Tendo em vista a problemática da produção de novas e significativas atividades de leitura e exercícios, uma vez que elas podem perder seu valor pedagógico após certo tempo e características de diferentes aprendizes, a solução a partir do uso de uma atividade padrão maximizaria o número de atividades de leitura disponíveis para a escolha do aprendiz, para uma abordagem *self-access*¹⁸, em uma concepção de leitura baseada em um processo descendente (*top-down*), na qual o aprendiz deve evitar uma "visão de túnel" (SCOTT et al., 1984, p. 115, tradução nossa) e se apoiar em hipóteses e previsões a partir de uma leitura rápida e ampla do texto. A proposta de atividade padrão exemplifica um rol de 13 questões sobre o texto, traduzidas do português, conforme é possível verificar no ANEXO A (SCOTT et al., 1984, p. 116).

Ainda no contexto brasileiro, a análise de alguns livros didáticos para ensino de leitura de inglês em abordagem instrumental revela alguns dos itens de ensino recorrentes que podem ser utilizados como referência para ampliação das possibilidades de criação de exercícios conforme permite o conjunto de análises e instrumentação computacional disponível:

- a. reconhecimento de gêneros textuais;
- b. níveis de compreensão de leitura:
 - compreensão geral;
 - compreensão de pontos principais;
 - compreensão detalhada.
- c. estratégias de leitura:
 - identificação de cognatos;
 - identificação de marcas tipográficas e informação não verbal;
 - identificação de palavras repetidas;
 - predição;
 - *skimming*;
 - *scanning*;
 - seletividade;

¹⁸ Abordagem de ensino em que os aprendizes estudam de forma independente a partir de diferentes recursos disponíveis.

- identificação de palavras-chave;
- identificação de pontos principais;
- função retórica do texto.

d. aspectos léxico-gramaticais:

- afixos;
- grupos nominais;
- classes de palavras;
- grau de adjetivos;
- tempos verbais;
- verbos modais;
- referência contextual;
- referência pronominal;
- elementos de ligação.

Além dos pontos elencados anteriormente, há também a possibilidade de adição de novos aspectos a serem explorados, como a criação de exercícios baseados em *data-driven learning* (ensino movido a dados), uma vez que as análises automáticas retornam dados de frequência e padronização da linguagem.

As práticas e propostas apresentadas são possibilidades válidas para a preparação de atividades de leitura em língua estrangeira. Não é objetivo da pesquisa buscar um modelo específico ou determinar quais atividades são mais eficientes. Tendo em vista tal variedade de propostas e possibilidades, e embora tenha sido dada uma atenção maior ao ensino de inglês para fins específicos, a intenção não é direcionar a criação de atividades de leitura a uma proposta específica, mas explorar as possibilidades de criação de atividades a partir de análises automáticas de texto e *corpora*.

2.6 Estudo de criação automática de atividades

O *software desktop* desenvolvido na pesquisa de mestrado (MOREIRA FILHO, 2007), *Reading Class Builder (RCB)*, para o sistema operacional *Windows*, na linguagem de programação *Visual Basic 6*, permite a criação de atividades a partir de modelos estáticos e de

um assistente (*wizard*) que auxilia o usuário até a preparação das atividades. Tal *software* foi o primeiro protótipo construído a partir de modelos fixos de unidades didáticas (ANEXO B).

O assistente eletrônico guia o usuário por meio de três etapas. Na primeira etapa, o usuário escolhe que tipo de atividade de leitura será preparada. O programa fornece alguns modelos padronizados. Na segunda etapa, o usuário seleciona um texto de sua escolha para preparação da atividade. Na terceira etapa, o programa exibe informações sobre o texto escolhido e a atividade quase pronta. Após verificar as informações do texto e editar a atividade (opcional), o usuário pode salvá-la.

Os exercícios das atividades estão relacionados a concordâncias (*data-driven*), predição, léxico-gramática e questões para leitura crítica. Um modelo fixo com questões e lacunas para inserção de itens lexicais do próprio texto é utilizado para gerar os exercícios na atividade.

Para tanto, a partir do texto selecionado, o programa realiza análises automáticas por meio de fórmulas estatísticas: lista de frequência, palavras-chave, possíveis palavras cognatas, *POS*, possíveis padrões (n-gramas) e densidade lexical do texto. O esquema a seguir sintetiza o funcionamento do sistema:

Figura 16 - Esquema de funcionamento do primeiro protótipo

Fonte: Produção do próprio autor

Tendo em vista que a linguagem de programação utilizada, *Visual Basic 6*, não é compatível com os sistemas operacionais e configurações mais atuais, a inclusão de bibliotecas específicas por ela requerida demonstrou ser um entrave para sua utilização. Por isso, foi criada uma segunda versão *desktop*.

A segunda versão foi escrita em *C#*, com o objetivo de manter o registro do trabalho realizado. O programa incorporou algumas mudanças em relação aos modelos utilizados para a criação das atividades e ao número de etapas.

O funcionamento se deu a partir de três etapas: Etapa 1 – escolha do modelo de atividade; Etapa 2 – escolha do texto e do *corpus* de referência, e extração de exemplos; Etapa 3 – visualização de informações do texto e atividade preparada para edição e publicação.

Os modelos podem ser criados pelo próprio usuário por meio da utilização de um conjunto finito de códigos, tal como <XC010>, para retornar dez possíveis palavras cognatas do texto, ou <XK0#>, para retornar um determinado número de palavras-chave do texto (substituir # pelo número desejado). Na ajuda do programa, há uma listagem de todos os códigos disponíveis e sua descrição.

Nesses primeiros protótipos, embora os resultados obtidos tenham demonstrado a viabilidade e o potencial da solução, considerou-se a necessidade de pesquisas adicionais e desenvolvimento para melhorias por meio da avaliação das análises realizadas: diminuição de erros de análise, aumento da variedade de exercícios disponíveis e adequação do conjunto de exercícios ao texto de entrada.

Com base na experiência de desenvolvimento dessas duas versões, pretendeu-se uma investigação maior em relação às análises de texto e *corpora*, por meio da instrumentação metodológica computacional de áreas com o Processamento de Línguas Naturais e Aprendizagem de Máquina, a fim de explorar as possibilidades de análises linguísticas e a criação automatizada de atividades de forma dinâmica.

Dessa forma, a atual pesquisa aborda o desenvolvimento de uma aplicação dinâmica, em que o conjunto de exercícios seja variado, flexível e adequado à entrada (texto/gênero), com a possibilidade de o usuário selecionar modelos de atividades para compor uma unidade didática. Em contraponto a um modelo fixo de atividade, como no primeiro protótipo, propõe-se um novo esquema para criação de atividades.

O esquema representado a seguir ilustra as etapas básicas para o funcionamento do sistema na geração automática de atividades de leitura, tendo como partida um texto e opções pré-definidas de um usuário:

Figura 17 - Esquema das etapas básicas para criação de atividade de leitura

Fonte: Produção do próprio autor

As etapas apresentadas no esquema referem-se à necessidade de inserção de um texto de entrada, análise linguística do texto, instanciação de *scripts* para análise pedagógica e interpretação dos *scripts* para o resultado final: atividades criadas.

No capítulo a seguir, descreve-se todo o processo de desenvolvimento do sistema.

3 DESENVOLVIMENTO

3.1 Linguagem *Python* e Pesquisa Linguística

Nesta seção, apresentam-se as linguagens de programação, o ambiente de desenvolvimento e seus recursos disponíveis para a pesquisa.

Para o desenvolvimento da pesquisa, utilizou-se a linguagem de programação *Python*, com sua *IDE* (*Integrated Development Environment*), ou ambiente integrado de desenvolvimento, *IDLE* para escrita dos códigos, além do ambiente de desenvolvimento *Eclipse*, que suporta várias outras linguagens por meio de *plugins*, configurado para uso da linguagem *Python*. As *IDEs* facilitam o desenvolvimento rápido de aplicativos, provendo maior produtividade.

A versão da linguagem *Python* utilizada foi a 2.7, instalada em um computador com sistema operacional *Windows7*, 64x, processador Intel i7, com 8 Gigabytes de *RAM* e *HD* de 750 *Gigabytes*. Com a linguagem *Python*, foi feito o uso do *Natural Language Toolkit (NLTK)*, criado pelo Departamento de Computação e Ciência da Informação da Universidade da Pennsylvania (BIRD; KLEIN; LOPER, 2009), e de outras bibliotecas e módulos não nativos ao *Python*, tais como as bibliotecas *NumPy*, *Matplotlib* e *MySQL-Python*.

Python é uma linguagem de alto nível, fácil de aprender e poderosa, principalmente para manipulação de *strings*¹⁹. Há uma série de bibliotecas, módulos e *toolkits* escritos para o Processamento de Línguas Naturais e aprendizado de máquina, tal como o *NLTK*. A maioria desses recursos está disponível gratuitamente para uso em pesquisas, como é o caso do *NLTK*, e recebe atualizações periódicas.

Embora houvesse a possibilidade de uso de outras linguagens de programação de conhecimento do pesquisador, como *C#*²⁰, *PHP* ou *SHELL*, a decisão pela linguagem *Python* levou em consideração os seguintes aspectos: simplicidade de escrita e leitura do código, alta produtividade, funcionalidade para o processamento de dados linguísticos, vasta coleção de bibliotecas e componentes para programação, tradição de uso para pesquisas linguísticas.

¹⁹Sequência de caracteres.

²⁰*C Sharp*.

Além da linguagem *Python* e suas *IDEs*, para o desenvolvimento da interface do sistema, foram utilizadas as linguagens *PHP* (com o banco de dados *MySQL*), *HTML*, *JavaScript* e *CSS* (*Cascading Style Sheets*). A linguagem *Python* é especificamente utilizada para as análises do sistema de Processamento de Línguas Naturais, embora pudesse realizar o gerenciamento das páginas na interface, visto que tal tarefa parece ser naturalmente melhor realizada via *PHP*, conforme a experiência e habilidades de programação do pesquisador.

O desenvolvimento da interface foi auxiliado pela instalação de um servidor no próprio computador de desenvolvimento, o que facilitou os testes durante a criação, uma vez que não é necessário fazer o *upload* dos arquivos atualizados para um servidor na Internet. Para isso, o servidor disponibilizado pelo *software XAMPP* foi utilizado. *XAMPP* é um *software* livre que consiste no servidor *web Apache*, a base de dados *MySQL*, e nos interpretadores para as linguagens de *script*: *PHP* e *Perl*. Houve a necessidade de apontar o caminho do servidor instalado, por meio da alteração de seu arquivo de configuração, para o diretório de desenvolvimento do Eclipse, dado que o caminho padrão do Apache na instalação do *XAMPP* era '*c:/xampp/htdocs*', assim como a necessidade de ativar um módulo para a execução de *scripts* na linguagem *Python*.

Tendo realizado os ajustes na máquina, conseguiu-se um ambiente de desenvolvimento adequado para os objetivos da pesquisa. Além do servidor instalado na máquina de desenvolvimento, foi adquirido um serviço de hospedagem de um servidor virtual privado, gerenciado pelo próprio pesquisador, e um domínio para disponibilizar o sistema criado na Internet.

A necessidade técnica²¹ de um servidor virtual privado está relacionada ao uso do *NLTK* e à biblioteca *NumPy*. Os serviços de hospedagens de *sites* simples, por oferecerem um ambiente de hospedagem compartilhado, geralmente possuem uma série de restrições e limitações. Essas limitações impedem a instalação dos módulos requeridos pelo sistema. Com um servidor virtual privado, há a possibilidade de instalar todos os módulos requeridos, além de obter maior controle em relação ao uso dos recursos.

Desse modo, foi possível a disponibilização do sistema criado a fim de que usuários pudessem utilizá-lo e avaliá-lo, conforme uma das etapas da pesquisa.

²¹ Foi pretendida a realização de testes práticos com usuários para a avaliação do sistema.

O Apêndice A expõe os principais recursos que justificam a escolha da linguagem de programação *Python* para a pesquisa linguística, além de conceitos e convenções utilizadas nas descrições de códigos e algoritmos da pesquisa.

3.2 *Corpora* da pesquisa

Para o desenvolvimento da pesquisa, os seguintes *corpora* de referência e estudo foram utilizados:

Quadro 4 - Caracterização dos *corpora* da pesquisa

Tipo	Composição	Itens	Formas
Referência	<i>BNC</i>	109.165.999	619.281
Referência	Floresta Sintá(c)tica	211.852	29.421
Referência	MacMorpho	1.170.095	67.696
Estudo	135 anúncios de emprego coletados da Internet	37.700	5.075
Estudo	771 textos de divulgação científica de revistas eletrônicas (<i>Scientific American</i> e <i>NewScientist</i>)	575.713	30.328

Fonte: Produção do próprio autor

O quadro apresenta a caracterização dos *corpora* da pesquisa. A classificação dos *corpora* está relacionada à sua função principal. Todos os *corpora* foram utilizados no desenvolvimento dos módulos de análise do sistema, podendo também, em tal caso, serem considerados de treinamento e teste. Os *corpora* de língua portuguesa foram especificamente utilizados para o desenvolvimento do módulo de identificação de palavras cognatas. O *corpus BNC* forneceu recursos para a criação do módulo de extração de palavras-chave e do módulo de etiquetagem morfossintática. Os *corpora* de estudo foram utilizados em todo o desenvolvimento e teste do sistema para a criação de atividades.

O *BNC* foi um dos primeiros *corpora* de língua geral a atingir mais de 100 milhões de palavras e possui uma versão comercial. O *BNC* é composto por 4.054 arquivos de textos,

sendo 3.144 textos escritos (90%) e 910 textos falados (10%). A pesquisa faz uso de 4.023 arquivos de textos do *BNC*, provenientes de um primeiro estudo na pesquisa de mestrado.

O Floresta Sinta(c)tica e o MacMorpho utilizados são amostras de língua portuguesa provenientes do *NLTK*. O Floresta Sinta(c)tica é um *corpus* anotado que contém textos do português do Brasil e de Portugal, pertencente a um projeto de parceria entre a Linguateca²³ e o projeto *VISL*. O MacMorpho é um *corpus* anotado composto por artigos do jornal Folha de São Paulo.

Os *corpora* de estudo foram coletados na Internet. Houve a necessidade de fazer a limpeza dos textos com a substituição de códigos *HTML* e de caracteres indesejados. Os textos dos *corpora* foram salvos em arquivos com a extensão ‘.txt’, texto sem formatação, em pastas distintas. Para o *corpus* de anúncios de emprego, cada anúncio do *corpus* foi salvo em um arquivo separado, totalizando os 135 textos. Para o *corpus* de textos de divulgação científica, visando uma melhor organização e possíveis necessidades de análise os textos foram salvos com nomes de acordo com a seção temática ou assunto categorizado pelo próprio *site* (exemplos: *Addiction_&_Recovery_01.txt*, *Biodiversity_01.txt*, *Climate_01.txt*, *Dinosaurs_01.txt* etc.). As seções compuseram 138 categorias, com média de cinco textos para cada seção.

Em relação a questões de padronização e características dos gêneros dos *corpora*, é interessante ressaltar que o *corpus* de textos de divulgação científica possui uma variação temática maior e é composto por artigos e notícias científicas²⁴ de revistas eletrônicas. As características dos dois *corpora* projetaram cenários diferentes para as análises dos itens lexicais, o que foi positivo para os experimentos.

3.3 Criação dos módulos de análise de texto e *corpora*

Nesta seção, descreve-se o desenvolvimento dos principais módulos de análise de texto e *corpora* necessários para o sistema de criação automática de atividades.

²³ O sítio do projeto é o < <http://www.linguateca.pt/floresta/> >.

²⁴ Na coleta semi-automática, não foi possível fazer a distinção entre artigo e notícia, tendo em vista que os sítios de coleta não fazem a distinção nos resultados de busca.

Embora haja uma série de outras ferramentas e módulos disponíveis para muitas das análises requeridas, optou-se por implementar módulos próprios para necessidades de funcionalidades específicas e pela facilidade de integração dos módulos no projeto de desenvolvimento.

3.3.1 Segmentação

A primeira etapa para iniciar qualquer tipo de processamento de textos é a identificação de unidades de análise. Em termos computacionais, tal tarefa é chamada de segmentação.

A segmentação depende dos objetivos de análise, que podem ser a identificação de palavras, sentenças ou elementos lógicos estruturais de um texto. Dependendo do tipo de segmentação, determinados conhecimentos ou análises são requeridos. Por exemplo, para a segmentação de um texto em sentenças, o simples reconhecimento de caracteres que indicam o final de uma sentença (./;!/?:) pode alcançar resultados satisfatórios, enquanto que, para a identificação de unidades discursivas, análises prévias mais complexas, como análises sintáticas, devem ser realizadas.

Os tipos mais comuns de segmentação realizados para uma série de análises computacionais são a itemização e a identificação de sentenças. A itemização, geralmente, procura segmentar o texto em palavras. Ambos os tipos de segmentação fazem parte da maioria das ferramentas utilizadas em Linguística de *Corpus*, como os concordanciadores, e também de qualquer tarefa do PLN.

Há várias abordagens para executar tarefas de itemização e identificação de sentenças, as quais podem não obter 100% de precisão, mas o nível de acurácia conseguido é suficiente para a realização de processamentos posteriores.

Para esta pesquisa, a segmentação é feita pelos módulos '*segmenter*' e '*tokenizer*', que realizam a quebra do texto em parágrafos, sentenças e palavras por meio da itemização do texto.

O módulo '*segmenter*' possui apenas a função '*segment*', que toma como entrada um texto em formato *string*, remove os caracteres de nova linha '\n' duplicados, considera os caracteres de nova linha não duplicados como separadores de parágrafos e, para cada parágrafo identificado, por meio de um itemizador, identifica as sentenças do texto. Como resultado, a

função retorna a segmentação do texto em parágrafos e sentenças em uma variável do tipo lista.

3.3.2 Itemização

Em PLN, a itemização é um processo básico e importante do processamento de textos. As demais análises possíveis dependem geralmente da normalização dos textos. A normalização está relacionada a questões sobre como segmentar o texto em parágrafos, sentenças e palavras.

No processo, uma série de decisões deve ser tomada. Pode-se considerar a itemização como uma etapa de anotação na qual se divide um texto em unidades chamadas itens, como preparação para subseqüentes análises e anotações. Um item pode ser constituído por uma palavra, pontuação, caractere especial ou número, de acordo com os objetivos da pesquisa ou aplicação que se pretende desenvolver.

Como exemplo, as decisões a serem tomadas podem influenciar diretamente na contagem de palavras de um sistema em que tal tarefa seja primordial para a funcionalidade da aplicação. Ao considerar a sentença *'New York is a state in the Northeastern and Mid-Atlantic regions of the United States'* como foco de análise, as seguintes questões para itemização surgem:

- a. *'New York'* e *'United States'* devem ser considerados cada um único item?
- b. palavras com hífen '-' devem ser divididas?
- c. todas as palavras devem ser convertidas para caixa baixa ou alta?
- d. o que fazer com números, símbolos e pontuações?

Desse modo, podem-se obter diferentes contagens de itens da mesma sentença. Se fossem considerados *'New York'* e *'United States'*, haveria 14 itens e 13 formas (*'the'* ocorre duas vezes). Caso contrário, 16 itens e 15 formas. Outra variação poderia ocorrer se fossem divididas palavras com '-'.

Enfim, como se pode ver, a tarefa de segmentar o texto em itens, sentenças e parágrafos não é trivial. Ambiguidades causadas pela pontuação, abreviações, datas e palavras compostas podem dificultar as análises.

Conforme Nugues (2006, p. 89), pode ser definida a seguinte gramática para a itemização de textos:

- a. um item é uma sequência de caracteres alfabéticos ou dígitos;
- b. outros caracteres, como espaços em branco, tabulações e pontuação, marcam o término de um item;
- c. uma sentença é uma sequência de itens terminada por um ponto final, exclamação, dois pontos ou interrogação;
- d. um parágrafo é uma sequência de sentenças terminada por um caractere de nova linha.

Em um experimento, antes da decisão de utilizar o *corpus Penn Treebank* para treinar o etiquetador morfossintático para as análises, a itemização foi realizada como um processo de constante transformação do texto de entrada. Para possibilitar a realização dessas transformações por meio de um algoritmo, foram criados dois arquivos, um arquivo de léxico e um arquivo com expressões regulares. Além dos arquivos, uma lista de caracteres de pontuação também é compilada internamente no código.

O arquivo de léxico foi gerado a partir do *corpus Brown*, já itemizado e etiquetado do *NLTK*. No processo de geração do léxico, percorreu-se automaticamente o *corpus*, item por item, a partir da localização de caracteres específicos (. ; , / %), extraíndo aqueles que poderiam ser considerados abreviações, acrônimos, números etc. Os itens identificados foram salvos em um arquivo para uso no processo de itemização.

```

1 U.S.
2 Ore.
3 61.2%
4 fig.
5 kc.
6 7.19
7 2:30-:34.3
8 Y.M.C.A.
9 $12.50
10 $10.50
11 cm.
12 0.025-in.
13 Sept.
14 7.6%
15 Ky.
```

Figura 18 - Exemplo de itens do léxico para itemização

Fonte: Produção do próprio autor

O arquivo de expressões regulares foi criado manualmente para casos em que os itens do léxico podem não dar conta, uma vez que são mais específicos que as regras das expressões regulares. Grefenstette e Tapanainen (1994) relatam o uso de regras e léxico para a desambiguação de palavras e sentenças. Exemplos de expressões regulares no arquivo são mostrados a seguir:

```

1 ^[A-Z]\.$ -> abbreviation (ex.: W.)
2 ^[A-Z]\.[A-Z]\.$ -> abbreviation (ex.: W.C.)
3 ^[A-Z]\.[A-Z]\.[A-Z]\.$ -> abbreviation (ex.: W.C.)
4 ^[0-9]+\.[0-9]+$ -> numbers (ex.: 10.00)
5 ^[0-9]+\,[0-9]+$ -> numbers (ex.: 10,00)
6 ^\$[0-9]+$ -> currency (ex.: $100)
7 ^\$[0-9]+\.[0-9]+$ -> currency (ex.: $10.00)
8 ^\$[0-9]+\,[0-9]+$ -> currency (ex.: $10,00)
9 ^[0-9]+\.[0-9]+% -> percent (ex.: 61.2%)
10 ^[0-9]+\/[0-9]+\/[0-9]+$ -> date (ex.: 02/01/1982)
11 ^[A-Z]+\/[A-Z]+$ -> (ex.: WORD/WORD)
12 ^[a-z]+\/[a-z]+$ -> (ex.: word/word)
13 ^[A-Z][a-z]+\/[A-Z][a-z]+$ -> (ex.: Word/Word)
14 ^[A-Z][a-z]+\/[a-z]+$ -> (ex.: Word/word)
15 ^[A-Z]+\/[a-z]+$ -> (ex.: S/he)
16 ^[a-z]+\@[a-z]+\.[a-z]+$ -> email (jlopes@uol.com)

```

Figura 19 - Exemplo de expressões regulares para itemização

Fonte: Produção do próprio autor

No arquivo, as expressões regulares são listadas juntamente com um comentário explicativo de sua função. Tanto o arquivo de léxico como o arquivo de expressões regulares podem sofrer alterações manuais conforme a necessidade de inclusão de novos itens e regras com o uso e experiência do sistema.

Após criados os dois arquivos de regras e léxico, um algoritmo foi escrito para a itemização, aproveitando tais dados. O algoritmo de itemização criado realiza as seguintes ações por meio da função '*tokenize(text)*':

- carrega os dados dos arquivos de léxico e expressões regulares;
- remove os espaços e caracteres de nova linha ao redor do texto de entrada;
- remove os caracteres não desejados como tabulações '\t' e marcadores;
- identifica possíveis parágrafos, substituindo caracteres de nova linha '\r\n' pelo caractere '|'. Os caracteres duplicados são removidos;

- e. coloca espaços ao redor de aspas e parênteses;
- f. remove espaços duplicados;
- g. aplica as expressões regulares e adiciona os itens identificados à lista do léxico carregado;
- h. coloca espaços ao redor de cada pontuação e remove novamente os espaços duplicados;
- i. utiliza o arquivo de léxico para restaurar possíveis abreviações, números e itens identificados pelas expressões regulares;
- j. identifica e restaura reticências;
- k. remove possíveis espaços em branco e caracteres duplicados;
- l. quebra o texto em uma lista, utilizando o caractere de espaço em branco como delimitador, e salva os itens em uma variável do tipo lista a ser retornada pela função.

Na figura a seguir, para fins de ilustração, apresenta-se o resultado impresso de um texto itemizado.

```
1 Great
2 Opportunities
3 !
4 TSC
5 is
6 a
7 global
8 leader
9 in
10 the
11 Staffing
12 Services
13 Industry
14 .
15 |
16 We
17 currently
18 have
19 several
20 staff
21 openings
22 at
23 our
24 main
25 office
26 in
27 Neverland
```

Figura 20 - Exemplo parcial de um texto itemizado

Fonte: Produção do próprio autor

No exemplo impresso, cada item ocupa uma linha. O caractere '|' indica um parágrafo. O caractere indicador de parágrafo é removido nas demais etapas de processamento, com a introdução de marcações mais estruturadas.

Como mencionado no início desta seção, o itemizador descrito foi substituído pelo '*TreebankWordTokenizer*' do *NLTK*, tendo em vista o uso do *corpus Penn Treebank*. O módulo '*tokenizer*' carrega o '*TreebankWordTokenizer*' e, por meio da função '*tokenize*', faz a itemização de sentenças. No módulo, também há a função '*__chr*', que remove e normaliza caracteres no texto de entrada para evitar erros com o módulo que constrói a anotação do texto analisado em *XML* (APÊNDICE K).

Após a itemização, a próxima etapa da análise foi a etiquetagem morfológica de cada item.

3.3.3 Método de *stemming*

O método de *stemming* está relacionado a tarefas comuns de normalização de palavras. Em PLN, a normalização é utilizada em diversas aplicações: análise de sentimentos, tradução automática, sumarização etc.

A tarefa de normalização pode significar uma série de procedimentos, como tornar todas palavras em caixa baixa ou alta, decidir remover todos os caracteres de pontos em abreviações ou reduzir as formas das palavras em radicais.

Nesta pesquisa, para a extração dos radicais das palavras, utilizou-se o método de *stemming* fornecido pelo *NLTK*. O algoritmo selecionado para a tarefa é o '*PorterStemmer*', amplamente conhecido e utilizado para a língua inglesa.

O funcionamento do '*PorterStemmer*' se dá por meio de regras de substituição de sufixos das palavras em etapas ou passos.

3.3.4 Etiquetagem morfossintática

Com o texto itemizado, passa-se para a etapa de etiquetagem de cada item, a qual é feita utilizando o módulo '*postagger*'.

A tarefa de agrupar as palavras de uma língua em classes ou categorias (substantivos, adjetivos, verbos etc.), seja por características sintáticas ou semânticas, não é recente, podendo-se citar a ideia de categorização de Aristóteles (*parts of speech*).

A expressão '*part-of-speech tagging*' ou '*POS tagging*' é utilizada para referir-se à etiquetagem morfosintática, tarefa básica do PLN. A tarefa consiste em atribuir automaticamente, por meio de recursos computacionais, classificações ou símbolos (*DET*, *JJ*, *NN* etc.) a palavras de uma sentença:

- a. Candidates/*NNS* with/*IN* 2+/*CD* years/*NNS* of/*IN* experience/*NN* in/*IN* a/*DT* similar/*JJ* position/*NN* are/*VBP* encouraged/*VBN* to/*TO* apply/*VB* ./.
- b. Must/*MD* be/*VB* willing/*JJ* to/*TO* work/*VB* w//*IN* radioactive/*JJ* materials/*NNS* ./.

As sentenças apresentadas são exemplos de sentenças anotadas com o conjunto de etiquetas do *corpus Penn Treebank* em formato de texto do *NLTK*. O conjunto completo de etiquetas e sua descrição é encontrado no *NLTK* a partir da função '*nlk.help.upenn_tagset()*'. O quadro a seguir apresenta as etiquetas da documentação do *NLTK*:

Quadro 5 - Conjunto de etiquetas do *corpus Penn Treebank* no *NLTK*

#	POS	#	POS	#	POS
1	\$: dollar	16	JJR: adjective, comparative	31	RP: particle
2	": closing quotation mark	17	JJS: adjective, superlative	32	SYM: symbol
3	(: opening parenthesis	18	LS: list item marker	33	TO: "to" as preposition or infinitive marker
4	): closing parenthesis	19	MD: modal auxiliary	34	UH: interjection
5	,: comma	20	NN: noun, common, singular or mass	35	VB: verb, base form
6	--: dash	21	NNP: noun, proper, singular	36	VBD: verb, past tense
7	.: sentence terminator	22	NNPS: noun, proper, plural	37	VBG: verb, present participle or gerund
8	:: colon or ellipsis	23	NNS: noun, common, plural	38	VBN: verb, past participle
9	CC: conjunction, coordinating	24	PDT: pre-determiner	39	VBP: verb, present tense, not 3rd person singular
10	CD: numeral, cardinal	25	POS: genitive marker	40	VBZ: verb, present tense, 3rd person singular
11	DT: determiner	26	PRP: pronoun, personal	41	WDT: WH-determiner
12	EX: existential there	27	PRP\$: pronoun, possessive	42	WP: WH-pronoun
13	FW: foreign word	28	RB: adverb	43	WP\$: WH-pronoun, possessive
14	IN: preposition or conjunction, subordinating	29	RBR: adverb, comparative	44	WRB: Wh-adverb
15	JJ: adjective or numeral, ordinal	30	RBS: adverb, superlative	45	``: opening quotation mark

Fonte: (*NLTK*)

O conjunto de etiquetas no quadro é o utilizado no módulo '*postagger*' construído para realizar as análises morfosintáticas para o sistema. Embora o *NLTK* possua um etiquetador-padrão também treinado com o *corpus Penn Treebank*, dada a necessidade de obter resultados com maior precisão e tendo em vista que sua aplicação aos textos dos *corpora* de estudo

apresentaram erros nas análises para os *corpora* específicos da pesquisa, optou-se por treinar um etiquetador com parte dos dados dos *corpora* de estudo.

Em relação ao desempenho da tarefa de etiquetagem, várias abordagens e métodos ainda são estudados. Podem-se considerar três principais abordagens para o desenvolvimento de etiquetadores morfossintáticos: abordagem baseada em regras, abordagem probabilística e abordagem híbrida (DOMINGUES, 2011).

Na abordagem baseada em regras, um conjunto de regras codificadas, manual ou automaticamente, é utilizado para desambiguação. Na abordagem probabilística, algoritmos estocásticos são empregados para atribuir as etiquetas mais prováveis para as palavras. Os métodos estocásticos compreendem: modelos de n-gramas, Modelo de Markov Oculto (*Hidden Markov Model*), árvores de decisão probabilísticas, *Memory-Based Learning (MBL)*, *Maximum Entropy*, inferência bidirecional, entre outros. Na abordagem híbrida, há a combinação do uso de regras e probabilidade. Como exemplo, etiquetadores como o de Brill (1995) aprendem automaticamente regras baseadas em transformação dirigidas por erro. Considerando todas as abordagens, a taxa de desempenho para a língua inglesa fica geralmente em torno de 96%. Em Manning e Jurafsky (2012), são mencionados os seguintes resultados de desempenho:

Tabela 1 - Acurácia de métodos de etiquetagem de alto desempenho

Método de etiquetagem	Acurácia	Acurácia em palavras desconhecidas
<i>Most freq tag</i>	~90%	~50%
<i>Trigram HMM</i>	~95%	~55%
<i>Maxent P(t/w)</i>	93.7%	82.6%
<i>TnT (HMM++)</i>	96.2%	86.0%
<i>MEMM tagger</i>	96.9%	86.9%
<i>Bidirectional dependencies</i>	97.2%	90.0%
<i>Upper bound</i>	~98% (<i>human</i>)	

Fonte: (JURAFSKY e MANNING, 2012)

A tabela mostra a acurácia dos principais métodos de etiquetagem de alto desempenho. Computacionalmente, conseguir taxas de desempenho maiores é uma tarefa árdua, uma vez que mesmo a tarefa realizada por humanos não chega a um acordo de 100%.

O etiquetador treinado nesta pesquisa faz uso de diversos métodos de etiquetagem, isto é, combina classificadores e etiquetadores, conforme a experiência e conhecimento do pesquisador em relação ao *NLTK*. A partir de uma sentença itemizada, a combinação de métodos utilizados pode ser resumida nas seguintes etapas (não necessariamente ordenadas por execução²⁵):

1. etiquetagem de palavras a partir de um classificador que leva em consideração a palavra atual e as palavras ao redor (W, W_{-1}, W_1);
2. etiquetagem de palavras conhecidas a partir de um dicionário (palavra/etiqueta);
3. etiquetagem de palavras desconhecidas por meio de um classificador que utiliza atributos que levam em consideração características específicas da palavra (tamanho, formato, sufixos etc.);
4. etiquetagem de palavras por meio de bigramas com o etiquetador '*BigramTagger*' do *NLTK*;
5. refinamento da etiquetagem por meio de regras contextuais com o etiquetador '*BrillTagger*' do *NLTK*;

A arquitetura do etiquetador está relacionada a uma combinação sequencial, em que o método em cada uma das etapas, ao esgotar seus recursos, passa para o próximo etiquetador a fim de obter o máximo de desempenho. As etapas 2 e 3 consideram apenas a palavra em si e suas características. As etapas 1, 4 e 5 levam em consideração informações contextuais.

O treinamento do etiquetador ocorreu com uma série de experimentos até se chegar ao seu formato final, com vários descartes e aproveitamentos de códigos e léxicos criados. Apresentam-se aqui os principais passos pertinentes ao funcionamento do sistema.

Inicialmente, selecionaram-se os *corpora Penn Treebank* do *NLTK* e *BNC* para o treinamento, com a intenção de utilizar o conjunto de etiquetas do primeiro e de aumentar a cobertura de palavras desconhecidas a partir do segundo, uma vez que elas influenciam consideravelmente na acurácia. Um primeiro etiquetador foi treinado com sentenças do *Penn Treebank*, divididas em 78% para treinamento, 11% para desenvolvimento e 11% para teste. Em seguida, utilizou-se tal etiquetador para anotar o *corpus BNC* e normalizar as etiquetas entre os *corpora*.

²⁵ Com exceção do etiquetador '*BrillTagger*', os outros classificadores e etiquetadores podem ser alternados em execução.

Com o *BNC* etiquetado e as sentenças de treinamento do *Penn Treebank*, um dicionário de palavras e etiquetas foi compilado ('*wordtag_dict.pkl*'). Para o dicionário, como filtro para o grande número de entradas do *BNC*, foram consideradas apenas as palavras mais frequentes (com um valor de corte), e palavras identificadas com caracteres numéricos foram descartadas.

Depois, apenas a partir dos dados de treinamento e desenvolvimento do *Penn Treebank*, por meio do algoritmo do classificador '*NaiveBayesClassifier*', um classificador para palavras desconhecidas e um classificador com base em trigramas de palavras foram treinados. Os atributos utilizados no classificador de trigramas consideram a palavra corrente e as palavras de contexto da esquerda e da direita. Os atributos utilizados no classificador de palavras desconhecidas foram:

- a. o formato da palavra (*ws*);
- b. o tamanho da palavra (*sz*);
- c. se a palavra é capitalizada (*cap*);
- d. se há dígitos na palavra (*num*);
- e. se a palavra possui hífen (*hyp*);
- f. se a palavra possui apóstrofe (*apo*);
- g. se a palavra termina em 's' (*s*);
- h. o prefixo da palavra (*pre*);
- i. o sufixo da palavra (*suf*).

Jurafsky e Manning (2012) salientam resultados com grande acurácia, alcançados por meio do uso de atributos da própria palavra (prefixos, sufixos, caixa alta/baixa e o formato da palavra). Para o treinamento, o conjunto de atributos é armazenado em um dicionário, como requerido pelo *NLTK*, para cada instância dos dados. A figura a seguir ilustra exemplos de atributos para classificação:

```

HGSE
{'pre': 'HGS', 'suf': 'GSE', 'sz': 4, 'cap': 0, 'hyp': 0, 's': 0, 'num': 0, 'ws': 'x', 'apo': 0}

facility-related
{'pre': 'fac', 'suf': 'ted', 'sz': 16, 'cap': 0, 'hyp': 1, 's': 0, 'num': 0, 'ws': 'x-x', 'apo': 0}

K12
{'pre': '~', 'suf': '~', 'sz': 3, 'cap': 0, 'hyp': 0, 's': 0, 'num': 1, 'ws': 'xd', 'apo': 0}

nights/weekends
{'pre': 'nig', 'suf': 'nds', 'sz': 15, 'cap': 0, 'hyp': 0, 's': 1, 'num': 0, 'ws': 'x/x', 'apo': 0}

24394
{'pre': '243', 'suf': '394', 'sz': 5, 'cap': 0, 'hyp': 0, 's': 0, 'num': 1, 'ws': 'd', 'apo': 0}

09/13/2005
{'pre': '09/', 'suf': '005', 'sz': 10, 'cap': 0, 'hyp': 0, 's': 0, 'num': 1, 'ws': 'd/d/d', 'apo': 0}

triaging
{'pre': 'tri', 'suf': 'ing', 'sz': 8, 'cap': 0, 'hyp': 0, 's': 0, 'num': 0, 'ws': 'x', 'apo': 0}

```

Figura 21 - Exemplos de atributos para classificação de palavras desconhecidas

Fonte: Produção do próprio autor

Na figura, apenas para facilitar o entendimento, a palavra analisada é impressa acima do dicionário com os atributos, mas não faz parte dos dados. Além dos atributos, a etiqueta relacionada também é passada, formando uma tupla com o dicionário e a etiqueta em tipo *string*.

Tendo os classificadores criados e salvos em arquivos ('wordtag_dict.pkl', 'unknown_classifier.pkl' e '3words_dict.pkl'), uma classe nomeada como 'POSTagger' com a interface 'TaggerI' do *NLTK* foi criada para utilizar os classificadores na etiquetagem, como mostra parte do código na figura a seguir:

```

1  class POSTagger(nltk.TaggerI):
2
3      def __init__(self, backoff=None):
4          if backoff is None:
5              self._taggers = [self]
6          else:
7              self._taggers = [self] + backoff._taggers
8          self.__wordtag_dict = self.__load_classifier('wordtag_dict.pkl')
9          self.__unk_classifier = self.__load_classifier('unknown_classifier.pkl')
10         self.__3words_dict = self.__load_classifier('3words_dict.pkl')

```

Figura 22 - Declaração de uma classe para criação de um etiquetador

Fonte: Produção do próprio autor

Basicamente, a classe 'POSTagger' pode ser considerada um etiquetador personalizado. O uso da interface 'TaggerI', juntamente com suas funções requeridas, permite a combinação com outros etiquetadores do *NLTK*.

Dentro da classe, uma das funções essenciais é a função '*tag*', que retorna a sentença etiquetada a partir de seus itens, como mostra a figura:

```

1  def tag(self,tokens):
2      tagged=[]
3      for i in range(0, len(tokens)):
4          tri = cfe.get_3words_from_tokens(tokens, i)
5          if tri in self.__3words_dict:
6              tag = self.__3words_dict[tri]
7          else:
8              if tokens[i] in self.__wordtag_dict :
9                  tag = self.__wordtag_dict[tokens[i]]
10             elif tokens[i].lower() in self.__wordtag_dict:
11                 tag = self.__wordtag_dict[tokens[i].lower()]
12             else:
13                 features = (wfe.get_word_features(tokens[i]))
14                 tag = self.__unk_classifier.classify(features)
15             tagged.append((tokens[i],tag))
16     return tagged

```

Figura 23 - Função '*tag*' da classe '*POSTagger*'

Fonte: Produção do próprio autor

O algoritmo da função é: para cada item do argumento '*tokens*', verifica se o trigramma pertencente ao item é encontrado no dicionário e atribui a etiqueta à variável '*tag*'. Se não é encontrado, procura o item no dicionário de palavras e etiquetas, tentando a forma normal e em caixa baixa. Se o item é encontrado, a etiqueta é atribuída. Se novamente não é encontrado, o classificador de palavras desconhecidas atribui a etiqueta à variável, passando para o processamento do próximo item. Ao final, uma lista de tuplas é retornada com a sentença etiquetada.

O etiquetador criado foi então combinado com os etiquetadores '*BigramTagger*' e '*BrillTagger*' do *NLTK*, treinados em sequência, utilizando-se o argumento '*backoff*'. O treino final utilizou os dados de treinamento e desenvolvimento, compreendendo 89% do *corpus Penn Treebank*.

Posteriormente, outro etiquetador foi criado, com aproveitamento do dicionário de palavras/etiquetas e do classificador para palavras desconhecidas, além da substituição do refinamento contextual pelo algoritmo do etiquetador '*BrillTagger*' por um classificador contextual que leva em consideração os atributos: W, T, W_1, W1, T_1, T1, T_2, T2, T_3 e T3. O classificador foi treinado com o algoritmo do '*NaiveBayesClassifier*', e seus resultados influenciaram a sua escolha para utilização no sistema, tendo em vista que sua implementação

no código é mais simples (não há necessidade de combinar classificadores com etiquetadores).

Os resultados de desempenho dos etiquetadores treinados com os dados (treinamento (T), desenvolvimento (D) e teste (TT)) são mostrados na tabela a seguir:

Tabela 2 - Desempenho dos etiquetadores treinados

Dados para avaliação	<i>POSTagger</i> (inicial)	<i>POSTagger, BigramTagger e BrillTagger</i>	<i>Dicionário e Classificadores</i>
T	0.949076466908	0.974905470471	0.97475585539
D	0.953244274809	0.978053435115	0.977576335878
TT	0.923130331283	0.945546518548	0.950208292006
T+D	0.94959643818	0.975298207186	0.97510773553
T+D+TT	0.946760341822	0.972110029335	0.972439522129

Fonte: Produção do próprio autor

Os resultados mostraram um desempenho próximo das taxas de acurácia de etiquetadores de alto desempenho. Realizaram-se também testes com o etiquetador padrão do *NLTK*, '*pos_tag*', a fim de comparar os resultados. O etiquetador padrão do *NLTK*, por ser treinado com o mesmo *corpus*, obtém taxas em torno de 98% e 99% de acurácia. Embora a taxa seja alta, o desempenho nos *corpora* da pesquisa é diferente. Em um teste com 13 textos, etiquetados automaticamente e revisados manualmente, do *corpus* de anúncios de emprego (10% dos textos do *corpus*), o etiquetador padrão do *NLTK* obteve apenas 84% de acurácia, enquanto que o etiquetador treinado nesta pesquisa obteve 92% de acurácia. O resultado deve-se, em grande parte, à cobertura de palavras conhecidas pelo etiquetador treinado. Uma desvantagem é o tamanho final dos arquivos, que somam quase 10MB, o que pode influenciar a velocidade de processamento dos textos.

Para aumentar a acurácia do etiquetador nos *corpora* da pesquisa, parte dos dados foi manualmente anotada para inclusão no treinamento do etiquetador, uma vez que aplicações específicas tendem a obter resultados melhores. Tal prática mostrou-se satisfatória para correção de erros específicos de etiquetagem dos *corpora* de estudo.

Os sistemas de etiquetagem descritos fazem parte dos módulos análises '*postagger*' e '*clstagger*'. Outro módulo criado, '*processor*', é responsável por reunir os módulos de

segmentação, itemização e etiquetagem para retornar o texto etiquetado para as demais etapas do processo de análise a serem explicadas.

3.3.5 Contagem de frequência das palavras

Após a etiquetagem das palavras pelo módulo '*processor*', o resultado é passado para o módulo '*counter*', que basicamente possui funções de contagem requeridas pelo sistema, como contagem de itens a partir de uma lista ou *string*, contagem de caracteres e contagem de sílabas. A contagem das palavras é realizada e retornada como uma lista.

Na contagem de palavras, utiliza-se como padrão a conversão dos itens para caixa baixa. Cada item do texto recebe um valor de frequência, inclusive pontuações e símbolos. A quantidade de itens, formas, proporção de formas em relação aos itens, número de parágrafos e sentenças, entre outras contagens, também são realizadas.

A tarefa de contagem de itens é frequentemente utilizada na pesquisa. Para a realização das contagens, foram utilizadas variáveis do tipo dicionário, em que os itens são as chaves e a frequência são os valores do dicionário.

3.3.6 Extração de palavras-chave

Nesta pesquisa, palavra-chave é entendida como algo que é importante para entender o texto (SCOTT, 1998), podendo responder a uma pergunta muito comum: o texto é sobre o quê? Assume-se que, a partir de palavras-chave automaticamente extraídas por meio de comparações estatísticas, seja possível, por exemplo, identificar os tópicos ou assuntos de textos, além de itens lexicais específicos de seu gênero.

Na pesquisa, a extração de palavras-chave é realizada a partir da lista de frequência do texto e da lista do *corpus* de referência *BNC* pelo módulo '*keywordsextractor*'. A função '*extract*' do módulo compara as listas por meio de uma fórmula estatística '*log-likelihood*'.

A fórmula, adaptada a partir da implementação de Rayson e Garside (2000), calcula o quanto a frequência de determinada palavra é esperada em relação à lista do *corpus* de referência. Os argumentos da função para cada palavra da lista são:

- frequência da palavra no *corpus* de estudo (*ifreqiinsc*);
- frequência da palavra no *corpus* de referência (*ifreqincr*);
- número total de itens no *corpus* de estudo (*ntokensinsc*);
- número total de itens no *corpus* de referência (*ntokensinsc*).

O código abaixo ilustra a função que calcula o valor de 'log-likelihood':

```
def __loglikelihood(self,ntokensinsc,ifreqiinsc, ntokensincr,ifreqincr):
    nt1 = float(ntokensinsc)+1
    nt2 = float(ntokensincr)+1
    f1 = float(ifreqiinsc)+1
    f2 = float(ifreqincr) +1
    e1 = nt1 * ((f1+f2) / (nt1+nt2))
    e2 = nt2 * ((f1+f2) / (nt1+nt2))
    ll = 2 * ((f1 * math.log( f1/e1,2 ) ) + (f2 * math.log(f2/e2,2) ))
    return round(ll)
```

Figura 24 - Fórmula para determinar o valor de chavicidade de uma palavra

Fonte: Produção do próprio autor

O cálculo é feito para cada palavra na lista de frequência do texto. Uma nova lista é gerada e armazenada também em uma variável de tipo dicionário. As palavras que obtêm maior valor são consideradas chave.

Na tabela a seguir, ilustram-se os resultados obtidos pelo uso da fórmula em 15 palavras de um texto com 331 itens, comparadas com a lista do *corpus* de referência de *BNC*, 100.000.000 de itens:

Tabela 3 - Exemplo de tabela para cálculo de chavicidade

N	PALAVRA	FREQ-ESTUDO	FREQ-REFERÊNCIA	CHAVICIDADE
1	and	29	2596180	1181
2	business	13	34944	110025
3	the	13	5998572	53
4	of	11	3019062	16
5	to	9	2576514	12
6	in	7	1928189	26
7	new	7	123212	14928
8	development	5	31729	49726
9	for	5	872378	329
10	industry	5	19627	82899
11	opportunities	5	5734	295246
12	a	4	2151187	54
13	about	4	190126	4648
14	analysis	4	12946	105895
15	management	4	21592	61781

Fonte: Produção do próprio autor

Na tabela, as palavras gramaticais têm valor de chavicidade muito menor em relação às palavras de conteúdo, embora possam ter alta frequência tanto no *corpus* de estudo como no *corpus* de referência. Os gráficos de nuvem a seguir confirmam a observação:

Figura 25 - Gráfico para comparação dos valores de frequência e chavicidade

Fonte: Produção do próprio autor

Os gráficos de nuvens foram criados a partir dos dados da tabela apresentada. Para o gráfico de nuvem 1, utilizaram-se as frequências do *corpus* de estudo (texto). O gráfico de nuvem 2 foi gerado a partir dos valores de chavicidade. No gráfico de nuvem 1, o destaque é dado às palavras gramaticais, enquanto que, no gráfico de nuvem 2, as palavras de conteúdo são destacadas. Dependendo do texto, o recurso de análise pode ser útil para levantar as palavras que carregam o sentido/assunto do texto.

No exemplo ilustrativo, o procedimento de extração de palavras-chave foi utilizado para análise de um texto. Na pesquisa, esse tipo de análise é central para a escolha de itens para a preparação de atividades e é realizado tanto para a análise de textos individuais como na análise de *corpora*.

3.3.7 Identificação de faixas de frequência lexical

Para a identificação de faixas de frequência lexical nos textos, adaptamos parte do conceito de funcionamento do programa *Vocabprofile* (SILVA, 2011), disponível no site <<http://www.lextutor.ca/vp/eng/>>.

O *Vocabprofile* é um programa capaz de dividir o texto em várias faixas de frequência lexical: a. as primeiras 1.000 palavras mais frequentes da língua inglesa (*K1*), consideradas fundamentais; b. as outras 1.000 palavras seguintes mais frequentes da língua inglesa (*K2*); palavras utilizadas em textos científicos de diversas áreas (*AWL*²⁶); c. palavras não encontradas nas três listas (*OFF-list*). Na figura abaixo, apresenta-se um exemplo parcial da análise de um texto de anúncio de emprego do *corpus* de estudo:

WEB VP OUTPUT FOR FILE: job ad (4.06 kb)

Words recategorized by user as 1k items (proper nouns etc): NONE (total 0 tokens)

	Families	Types	Tokens	Percent
K1 Words (1-1000):	104	128	361	62.89%
Function: ...	...	...	(157)	(27.35%)
Content: ...	...	...	(204)	(35.54%)
> Anglo-Sax	...	...	(75)	(13.07%)
=Not Greco-Lat/Fr Cog:	...	...		
K2 Words (1001-2000):	25	31	54	9.41%
> Anglo-Sax:	...	...	(14)	(2.44%)
1k+2k	...	...	...	(72.30%)
AWL Words (academic):	56	68	99	17.25%
> Anglo-Sax:	...	...	(2)	(0.35%)
Off-List Words:	?	46	60	10.45%
	185+?	273	574	100%

Current profile	
%	Cumul.
62.89	62.89
9.41	72.30
17.25	89.55
10.45	100.00

Figura 26 - Exemplo parcial de análise de texto com o programa *Vocabprofile*

Fonte: (*Vocabprofile*, 2014)

Conforme a análise do programa, o texto utilizado no exemplo possui 574 itens e 273 formas. A divisão das faixas retornou *K1*=62,89%, *K2*=9,41, *AWL*=17,25 e *OFF*=10,45. O programa também identificou as famílias em cada faixa (*K1*=104, *K2*=25 e *AWL*=56), palavras que compartilham a mesma raiz lexical. Os dados sugerem que boa parte das palavras do texto são fundamentais e frequentes da língua inglesa e quase um terço do restante das palavras está relacionado a itens lexicais de textos científicos e fora das demais categorias.

As aplicações do programa para o ensino de língua inglesa estão relacionadas a avaliar se determinado texto é adequado para os aprendizes. Uma das formas é avaliar a possível dificuldade que os aprendizes teriam em relação ao perfil lexical do texto. Textos com uma

²⁶ Academic Word List.

proporção maior de palavras nas faixas *K1* e *K2* poderiam ser mais adequados para iniciantes, enquanto que textos nas faixas *AWL* e *OFF* poderiam apresentar um pouco mais de desafio para os aprendizes em nível avançado, fornecendo oportunidades para aprendizado de 'vocabulário novo'.

Para a implementação das divisões de faixas de frequência lexical, reuniram-se as listas de frequência do *corpus BNC* e uma lista de palavras de textos acadêmicos (*Academic Word List*), disponível no site <<http://www.victoria.ac.nz/lals/resources/academicwordlist/>>. Ambas as listas foram processadas para fornecer os recursos adequados para o desenvolvimento do módulo.

A criação dos recursos para o desenvolvimento do módulo para classificar as palavras de acordo com as faixas de frequência lexical foi realizada a partir das seguintes tarefas:

- a. Seleção em lista ordenada das 2.000 palavras mais frequentes do *corpus BNC*;
- b. Preparação das palavras de textos acadêmicos em lista;
- c. Uso do método de *stemming* para extrair o radical das palavras de ambas as listas;
- d. Categorização dos radicais da lista do *corpus BNC* em '1' para os primeiros 1.000 radicais e '2' para os próximos 1.000 radicais;
- e. Remoção de radicais duplicados e armazenamento em um dicionário;
- f. Adição dos radicais da lista de palavras de textos acadêmicos ao dicionário com a categorização '3';
- g. Armazenamento do dicionário para ser utilizado nas classificações.

No total, o dicionário final recebeu 1.950 radicais únicos. Com a criação do dicionário, um algoritmo de classificação foi implementado, conforme segue, para cada palavra do texto:

- a. utiliza-se o método de *stemming* para extrair o radical da palavra;
- b. verifica-se a existência do radical no dicionário;
- c. se o radical é encontrado, retorna-se a classificação correspondente do dicionário;
- d. se o radical não é encontrado, retorna-se a classificação '0'.

Em suma, a classificação permite, de forma similar ao *Vocabprofile*, a identificação dos índices *K1*, *K2*, *AWL* e *OFF* por meio das respectivas classificações '1', '2', '3' e '0'.

3.3.8 Identificação de palavras cognatas

Para a identificação automática de palavras cognatas, considerou-se a seguinte definição de palavra cognata: palavra em língua inglesa que possui grande similaridade com qualquer palavra da língua portuguesa. Especificamente a palavra em inglês deve compartilhar uma base de tamanho definido com uma palavra da língua portuguesa.

Para definir a heurística de identificação de palavras cognatas, realizaram-se experimentos com dados que pudessem auxiliar na comparação ortográfica das palavras da língua inglesa e portuguesa: um conjunto de palavras manualmente analisadas dos *corpora* de estudo e referência, lista de bases²⁷ de palavras da língua portuguesa e inglesa a partir de um dicionário bilíngue *online*.

A geração da lista de bases de palavras de língua portuguesa foi conseguida por meio do processamento dos *corpora MacMorpho* e *Floresta Sintáctica* do *NLTK*. O conjunto de palavras utilizado somou 1.381.947 itens e 71.601 formas. No processamento, foi gerada uma lista de frequência dos *corpora*, e um filtro criado removeu as palavras consideradas não produtivas para identificação de palavras cognatas, por possivelmente levarem a identificações inadequadas. O filtro utilizou as seguintes restrições:

- a. a palavra não pode ocorrer na lista de *stop words*²⁸ portuguesa do *NLTK*;
- b. a palavra não pode conter pontuação;
- c. a palavra não pode conter números;
- d. o tamanho da palavra deve ser maior que cinco caracteres;
- e. a palavra não pode ocorrer na lista de palavras do *corpus BNC*;
- f. a palavra não pode conter os caracteres 'k', 'w' e 'y';
- g. a palavra deve possuir frequência maior que 1;

Em relação às restrições de palavras que contêm os caracteres 'k', 'w' e 'y', a decisão de não incluí-las está relacionada à disposição de geralmente indicarem palavras estrangeiras e, também, devido aos resultados obtidos em uma comparação dos usos das letras do alfabeto em palavras do *corpus* de língua portuguesa e do *corpus BNC*:

²⁷ Considera-se como base os cinco primeiros caracteres de uma palavra.

²⁸ Uma lista de palavras gramaticais (mais comuns) utilizada como filtro.

Figura 27 - Gráfico do uso de letras em palavras de língua portuguesa e inglesa

Fonte: Produção do próprio autor

No gráfico, é possível verificar que há muitas diferenças em relação ao uso das letras do alfabeto em palavras da língua portuguesa e inglesa. O eixo vertical (y) refere-se à contagem das letras. Em confirmação à decisão de não incluir palavras com as letras 'k', 'w' e 'y', a proporção de uso de tais letras em palavras de língua inglesa no *corpus BNC*, comparada ao uso de palavras da língua portuguesa dos *corpora* MacMorpho e Floresta Sintáctica, é maior. Foi considerado que, com a restrição, a chance de incluir uma palavra estrangeira seria menor.

Após a filtragem, foram extraídas as bases de palavras de tamanho cinco para cada palavra na lista, as quais foram contadas e eliminadas as bases duplicadas e de frequência igual a um, resultando um número de 539 bases. Para cada uma dessas bases, foi realizada uma busca no *corpus BNC* para remover bases não produtivas, isto é, aquelas que não encontraram nenhuma palavra no *corpus BNC*. Ao final, 465 bases foram consideradas produtivas. Na tabela a seguir, apresentam-se as 100 bases mais frequentes do conjunto:

Tabela 4 - As 100 bases de palavras mais frequentes identificadas

N	base	N	base	N	base	N	base
1	vice-	26	olimp	51	catol	76	rigid
2	auto-	27	grafi	52	mecan	77	jurid
3	sul-a	28	mitic	53	tatic	78	drama
4	pos-m	29	magic	54	econo	79	basic
5	meio-	30	sovie	55	ironi	80	sinte
6	asiat	31	plast	56	simbo	81	ultim
7	histo	32	prima	57	propr	82	preca
8	ex-co	33	brita	58	multi	83	eroti
9	camar	34	rapid	59	intim	84	optic
10	anti-	35	publi	60	organ	85	inedi
11	ex-pr	36	poeti	61	class	86	biolo
12	proxi	37	tecni	62	criti	87	cardi
13	fisic	38	teori	63	valid	88	polit
14	eletr	39	anoni	64	previ	89	genet
15	polic	40	music	65	artis	90	egipc
16	salar	41	tragi	66	turis	91	auten
17	estet	42	exoti	67	pmdb-	92	islam
18	ameri	43	prati	68	minim	93	comic
19	medic	44	coreo	69	bosni	94	etnic
20	servi	45	banca	70	magne	95	legit
21	diari	46	polem	71	clini	96	ident
22	semi-	47	temat	72	conti	97	quimi
23	espir	48	acade	73	bibli	98	tipic
24	agenc	49	croni	74	ilici	99	termi
25	recem	50	meia-	75	mater	100	domes

Fonte: Produção do próprio autor

Outra opção de recurso para identificação de palavras cognatas foi a geração de possíveis palavras candidatas a partir de um dicionário bilíngue. A ideia é a de que pares de tradução com semelhança ortográfica possam indicar possíveis palavras cognatas.

Para tanto, foram utilizados os dados de tradução do dicionário Michaelis (inglês/português), disponível *online* para buscas no sítio: <<http://michaelis.uol.com.br/moderno/ingles>>:

Figura 28 - Interface de busca do Dicionário Michaelis *online* - inglês/português

Fonte: Produção do próprio autor

Como não é viável consultar manualmente palavra por palavra na interface disponibilizada, devido ao tempo e esforço da tarefa, um programa foi criado para baixar as palavras do dicionário para a comparação.

Assim, utilizaram-se os módulos `'urllib'` e `'urllib2'` da linguagem *Python*, os quais permitem o acesso e a leitura de páginas e endereços na *web*, para descarregar os resultados de tradução para cada palavra do dicionário.

Inicialmente, tentou-se acessar as palavras por meio da *url* de busca do sítio, substituindo-se a palavra de busca por uma palavra da lista de frequência do *corpus BNC*. Por exemplo, na *url* `'http://michaelis.uol.com.br/moderno/ingles/index.php?lingua=ingles-portugues&palavra=good'`, apenas seria substituída a palavra `'good'` por outra palavra da lista de frequência para acessar os resultados.

Tal iniciativa não foi considerada bem-sucedida, devido à dependência da existência da palavra na lista de frequência e no banco de dados de palavras do dicionário e, principalmente, pelo fato de que, para determinadas palavras, uma página intermediária era retornada em vez dos resultados.

Por consequência, tal estratégia foi abandonada para utilizar os *links* da listagem alfabética do próprio *site*, a qual compreende o conjunto de palavras disponíveis para busca. Primeiro, foram gerados os índices de cada letra da lista alfabética, de 'A' a 'Z'. Para cada letra, o índice indica o número de páginas de palavras a serem acessadas. Cada página possui um determinado número de palavras para busca. No trecho de código, ilustra-se o uso dos índices para baixar cada página de palavras.

```

1  for letter in letters:
2      urlbase = 'http://michaelis.uol.com.br/moderno/ingles/indice/ingles-portugues/' + letter[0]
3      end = letter[1]
4      for i in range(1,end):
5          myurl = urlbase + str(i) + '.html'
6          print myurl
7          response = urllib2.urlopen(myurl)
8          html = str(response.read())
9          start = html.find('<!-- conteudo -->')
10         end = html.find('<!-- /conteudo -->')
11         content = html[start:end]
```

Figura 29 - Parte da função que baixa os endereços de cada palavra

Fonte: Produção do próprio autor

No código, uma *url* é criada a partir de cada letra e de cada índice dentro do limite de páginas da letra corrente. Em seguida, a página da *url* é baixada para a variável '*html*'. No código *HTML* da página baixada, extrai-se apenas a parte que contém a palavra e sua descrição, localizando os comentários do código da página '<!-- conteúdo -->' e '<!-- conteúdo -->', e armazena-se o conteúdo na variável '*content*'. Por último, uma série de transformações a partir de substituições para remoção de caracteres e códigos indesejados é realizada na variável '*content*'.

No final, apenas o código *HTML*, com a palavra e sua descrição, é salvo em um arquivo. A figura a seguir apresenta um exemplo de termo baixado:

Visualização do código html

```
<p id="contents">
<p><span class='palavra'>rasure </span><br /><span
class='palavraComPontos'>ra.sure </span><br /><span class='descricao'><i>
<b>n</b></i> <b>1</b> raspagem, raspadura. <b>2</b> rasura, emenda. <b>3</b>
eliminação, supressão.</span><br /></p>
```

Visualização normal no navegador

```
rasure
ra.sure
n 1 raspagem, raspadura. 2 rasura, emenda. 3 eliminação, supressão.
```

Figura 30 - Exemplo de termo baixado

Fonte: (Michaelis, 2014)

No total, foram baixadas 79.677 palavras do dicionário. As palavras foram salvas em um banco de dados para o processamento. O passo seguinte foi a identificação de possíveis palavras cognatas a partir da descrição de cada termo. Uma função faz a itemização das palavras na descrição, remove os itens padronizados para todas as descrições de palavras ('n','Bot','abbr','adj','vt','Brit','sl','adv' etc.), exclui os itens que são encontrados na lista de palavras do *corpus BNC* para evitar a inclusão da própria palavra e retorna uma lista com os possíveis candidatos à tradução.

Para cada candidato na lista de possíveis traduções da palavra, uma função, com a fórmula '*Dice*', é utilizada. A função compara a palavra inglesa com as palavras em português de sua descrição a partir de uma medida de similaridade. Se o índice de similaridade de uma das

comparações é maior que 0.75 (definido pelo pesquisador), a palavra inglesa é considerada uma possível palavra cognata. O código da função utilizada é:

```

1  def __dice(self,word1,word2):
2      lenw1 = len(word1)
3      lenw2 = len(word2)
4      biw1 = []
5      biw2 = []
6      for i in range(0, lenw1-1):
7          biw1.append(word1[i] + word1[i+1])
8      for i in range(0, lenw2-1):
9          biw2.append(word2[i] + word2[i+1])
10     nbw1 = len(biw1)
11     nbw2 = len(biw2)
12     nm = 0
13     for bi1 in biw1:
14         for bi2 in biw2:
15             if bi1 == bi2:
16                 nm+=1
17     D = float(2 * nm ) / float(nbw1 + nbw2)
18     return D

```

Figura 31 - Implementação da fórmula *Dice* em *Python*

Fonte: Produção do próprio autor

Na função, o índice é conseguido pela divisão do número de bigramas, letras, compartilhado pelo número total de bigramas em ambas as palavras. Por exemplo, a comparação entre as palavras '*different*' e '*diferente*' retorna um índice de 0.87.

Um dos problemas encontrados foi a comparação de palavras compostas. Para esses casos, uma análise manual foi realizada para sua exclusão, a partir da identificação de caracteres em branco e da palavra ' ou '.

Ao final, das 79.677 palavras iniciais do dicionário foram identificadas 9.415 palavras como possíveis palavras cognatas. A partir dessas palavras identificadas, também foi criada uma lista com 465 bases únicas de tamanho padrão de cinco caracteres.

Para o treinamento e teste dos experimentos, realizou-se uma análise manual das 2000 palavras mais frequentes do *corpus* BNC, 350 palavras do *corpus* de anúncios de emprego e 500 palavras do *corpus* de artigos de divulgação científica do *corpus* de estudo. Os resultados da análise foram combinados para formar um único conjunto de 2189 palavras anotadas, salvas em um arquivo. Na figura a seguir, apresenta-se a forma de anotação:

```

wanted 0
created 0
days 0
priorities 1
silicon 1
moving 1
onto 0
user 0
industrial 1
features 0

```

Figura 32 - Exemplo de palavras cognatas anotadas para treinamento e teste

Fonte: Produção do próprio autor

Na anotação, como mostra a figura, cada palavra recebe uma etiqueta binária (0,1), em que '1' indica que a palavra pode ser considerada cognata e '0' indica que a palavra não pode ser considerada cognata.

Tendo reunido os dados necessários para os experimentos, optou-se por estabelecer como divisão do *corpus* anotado a proporção de 89% para treinamento (1948 palavras) e 11% para teste (240 palavras). Dada a limitação dos dados, o método de validação cruzada (*k-fold*) foi utilizado para avaliar o desempenho dos algoritmos em cada experimento. O valor de *k* definido foi de 100 iterações, em que a cada iteração os dados de treinamento e teste são embaralhados pela função '*random.shuffle*' da linguagem *Python*. Ao final, a média dos valores de acurácia, precisão, abrangência e *f-measure* é retornada.

Nos experimentos, utilizou-se como padrão para todos os algoritmos uma lista de palavras negativas, palavras que não podem ser consideradas cognatas, compilada dos dados de treinamento a cada iteração. Como atributos, além das palavras negativas, utilizaram-se comparações binárias entre palavras e bases de palavras. Informações de contexto não foram levadas em consideração. O quadro abaixo apresenta os principais experimentos realizados a partir dos dados reunidos:

Quadro 6 - Descrição dos algoritmos para identificação de palavras cognatas

Teste	Recursos	Algoritmo
Teste 1	Bases de palavras em português + palavras negativas	<i>Para cada palavra de teste:</i> <i>verifique se é uma palavra negativa</i> <i>se verdadeiro, atribua 0 e termine</i> <i>se falso, verifique se possui base portuguesa</i> <i>se verdadeiro, atribua 1 e termine</i> <i>se falso, atribua 0 e termine</i>

continua

continuação

Teste	Recursos	Algoritmo
Teste 2	Palavras do dicionário bilíngue identificadas como possíveis cognatas + palavras negativas	<i>Para cada palavra de teste: verifique se é uma palavra negativa se verdadeiro, atribua 0 e termine se falso, verifique se está no dicionário se verdadeiro, atribua 1 e termine se falso, atribua 0 e termine</i>
Teste 3	Bases de palavras do dicionário bilíngue identificadas como possíveis cognatas + palavras negativas	<i>Para cada palavra de teste: verifique se é uma palavra negativa se verdadeiro, atribua 0 e termine se falso, verifique se possui base na lista se verdadeiro, atribua 1 e termine se falso, atribua 0 e termine</i>
Teste 4	Bases de palavras em português + palavras do dicionário bilíngue identificadas como possíveis cognatas (em lista única) + palavras negativas	<i>Para cada palavra de teste: verifique se é uma palavra negativa se verdadeiro, atribua 0 e termine se falso, verifique se possui base na lista se verdadeiro, atribua 1 e termine se falso, atribua 0 e termine</i>

Fonte: Produção do próprio autor

Os resultados finais para cada experimento foram:

Tabela 5 - Resultados dos experimentos para identificação de palavras cognatas

Identificação	Acurácia	Precisão	Abrangência	F-measure
Teste 1	0.680248962656	0.132286462234	0.797036862175	0.225265269828
Teste 2	0.692323651452	0.170354301279	0.820419159827	0.280584628238
Teste 3	0.801410788382	0.650117071823	0.752268362118	0.696342698874
Teste 4	0.80643153527	0.66541757752	0.750680035573	0.704223344755

Fonte: Produção do próprio autor

A partir dos resultados, optou-se por considerar o algoritmo e recursos utilizados no 'Teste 4', melhor desempenho na taxa de acurácia. O mesmo algoritmo treinado e testado com todas as instâncias dos dados anotados obteve a taxa máxima de 88% de acurácia.

Os resultados obtidos com o método podem ser considerados satisfatórios. Na figura a seguir, exibe-se o resultado de uma das análises.

Movies with bigger budgets **tended** to earn more at the box-office, as they were more likely to feature big stars and **massive** promotional marketing. Moreover, PG-13 films had a better **chance** of making more money than R-rated films in part because the former are **accessible** to a wider age-range of moviegoers. But even after **controlling** for **factors** such as **total** budget and film rating, the researchers found that smoking was **associated** with 13 **percent** less money made in ticket sales.

Figura 33 - Exemplo de palavras cognatas identificadas

Fonte: Produção do próprio autor

A figura apresenta a análise do parágrafo de um texto. As palavras em destaque foram identificadas como possíveis palavras cognatas. É possível conseguir melhor abrangência e precisão com a melhoria das listas de comparação. Posteriormente, foram aplicadas as análises a um *corpus* de treinamento para verificar o desempenho de cada base ou palavra, decidindo mantê-la ou descartá-la conforme os resultados.

3.3.9 Identificação de grupos nominais

Para a identificação de grupos nominais (*noun phrases*), utilizou-se uma técnica apresentada por Ramshaw e Marcus (1995), com algumas poucas modificações. A questão é lidar com a tarefa de identificação de grupos nominais como um problema de etiquetagem (NUGUES, 2006, p. 227) em uma abordagem de análise rasa (*partial/shallow parsing*). Outra possibilidade seria realizar a identificação por análises sintáticas completas das sentenças por meio de gramáticas livres de contexto, como no exemplo a seguir:

```
import nltk
grammar = nltk.parse_cfg("""
S -> NP VP
VP -> VBZ NP PP
PP -> IN NP
NP -> NNP | DT JJ NN NN | NN
NNP -> 'Python'
VBZ -> 'is'
DT -> 'a'
JJ -> 'good'
NN -> 'programming' | 'language' | 'research'
IN -> 'for'
""")
parser = nltk.ChartParser(grammar)
sent = ['Python', 'is', 'a', 'good',
        'programming', 'language', 'for', 'research']
parsed = parser.parse(sent)
parsed.draw()
```

Figura 34 - Exemplo de gramática livre de contexto e visualização em árvore

Fonte: Produção do próprio autor

Descartou-se realizar uma análise sintática completa das sentenças pela complexidade e esforço em treinar um classificador com larga cobertura gramatical, embora o *NLTK* forneça dados anotados como o próprio *corpus Penn Treebank*. A identificação de grupos como frases nominais são tarefas mais simples e tendem a ser mais precisas.

Assim, considerando a tarefa como um problema de etiquetagem, a técnica consiste em etiquetar os itens com três etiquetas básicas: *I* (*inside*), *O* (*outside*) e *B* (*begin*). Etiqueta '*I*' indica que o item está dentro de um grupo nominal, '*O*' indica que o item está fora de um grupo nominal, e '*B*' indica que o item está no início de um grupo nominal que imediatamente segue outro grupo nominal. Na técnica, a etiquetagem é feita a partir de regras, semelhante à proposta do etiquetador *Brill* (Brill,1995).

Para a implementação nesta pesquisa, foram etiquetadas e revisadas manualmente 100 sentenças do *corpus Penn Treebank*. Como regra para definição de grupos nominais para esta pesquisa, o grupo nominal pode conter pré-modificadores (determinantes, adjetivos e orações adjetivas) e pós-modificadores (advérbios e orações adjetivas), sem preposições ou conjunções. O esquema de anotação utilizado foi '*I-O*', por apresentar melhor desempenho para as generalizações do algoritmo de aprendizagem, como na sentença de ilustração abaixo:

```
Newsweek/NNP[I-NP] ,/, [O] trying/VBG[O] to/TO[O] keep/VB[O] pace/NN[I-NP] with/IN[O]
rival/JJ[I-NP] Time/NNP[I-NP] magazine/NN[I-NP] ,/, [O] announced/VBD[O] new/JJ[I-NP]
advertising/NN[I-NP] rates/NNS[I-NP] for/IN[O] 1990/CD[I-NP] and/CC[O] said/VBD[O]
it/PRP[I-NP] will/MD[O] introduce/VB[O] a/DT[I-NP] new/JJ[I-NP] incentive/NN[I-NP]
plan/NN[I-NP] for/IN[O] advertisers/NNS[I-NP] ./.[O]
```

Figura 35 - Exemplo de anotação utilizada para os dados de treinamento

Fonte: Produção do próprio autor

As 100 sentenças anotadas formaram o *corpus* de treinamento e teste do etiquetador. A criação do etiquetador foi realizada a partir da combinação sequencial do classificador '*MaxentClassifier*' e dos etiquetadores '*BigramTagger*', '*TrigramTagger*' e '*BrillTagger*'. O classificador '*MaxentClassifier*' utiliza como atributos a palavra corrente e sua etiqueta morfossintática. Basicamente, o funcionamento do etiquetador resume-se a uma primeira etiquetagem de itens (palavras/etiqueta morfossintática), com a etiqueta mais provável, e, em seguida, o refinamento da etiquetagem por meio de regras contextuais.

Como entrada, o etiquetador recebe a sentença em uma lista de itens como `['Python/NNP', 'is/VBZ', 'a/DT', 'good/JJ', 'programming/NN', 'language/NN', 'for/IN', 'research/N', 'N', './.']` e retorna uma lista de tuplas, com a palavra, sua etiqueta morfossintática e a etiqueta referente ao grupo nominal: `[('Python/NNP', 'I-NP'), ('is/VBZ', 'O'), ('a/DT', 'I-NP'), ('good/JJ', 'I-NP'), ('programming/NN', 'I-NP'), ('language/NN', 'I-NP'), ('for/IN', 'O'), ('research/NN', 'I-NP'), ('./.', 'O')]`.

A extração de cada grupo nominal acompanha a sequência de etiquetas 'I-NP' delimitada por etiquetas 'O', o que pode ser realizado com a utilização de um algoritmo simples, como mostra a representação de um autômato para a busca dos grupos nominais:

Figura 36- Exemplo de autômato para leitura de grupos nominais

Fonte: Produção do próprio autor

O autômato reconhece as sequências de itens que fazem parte de grupos nominais. Estando no estado inicial 'q0', a aceitação somente é iniciada com a leitura da etiqueta 'I-NP' e passa para o próximo estado 'q1'. Em 'q1', a aceitação é contínua até a leitura da etiqueta 'O', passando para o estado final 'q2', que encerra a aceitação e indica o final do grupo nominal.

Para a avaliação do etiquetador, dividiu-se o *corpus* anotado na proporção de 75 sentenças para treinamento e 25 para teste. Dada a limitação dos dados, o método de validação cruzada (*k-fold*) foi utilizado para avaliar o desempenho do etiquetador para a taxa de acurácia. O valor de *k* definido foi de 10 iterações, em que a cada iteração os dados de treinamento e teste são embaralhados pela função '*random.shuffle*' da linguagem *Python*. Os resultados são apresentados na tabela a seguir:

Tabela 6 - Resultados da avaliação do etiquetador de grupos nominais simples

Iteração	Acurácia
1	0,977859779
2	0,99034749
3	0,98943662
4	0,991624791
5	0,992907801
6	0,9808
7	0,973498233
8	0,991539763
9	0,979890311
10	0,981191223
Média	0,984909601

Fonte: Produção do próprio autor

Embora a taxa de acurácia seja alta (98%), o desempenho efetivo nos *corpora* de estudo pode variar, tendo em vista que a quantidade de dados de treinamento e de teste utilizados foi relativamente pequena e pode estar relacionada a um domínio específico de texto. Mais dados de treinamento tendem a aumentar a confiança das análises.

3.3.10 Identificação de grupos verbais

Para identificação de grupos verbais, primeiro, é necessária a análise de identificação de grupos nominais. Após a análise, o módulo de identificação de grupos verbais pode ser aplicado. A identificação de grupos verbais no módulo é realizada por um conjunto de regras simbólicas escritas manualmente:

Quadro 7 - Regras simbólicas para identificação de grupos verbais

O I-VG P VB	O I-VG PP1 MDVBZ	O I-VG PT_1T1 RBI-VGI-VG
O I-VG P VBD	O I-VG PP1 MDVBP	O I-VG PP_1 RPVB
O I-VG P VBG	O I-VG PP1 TOVB	O I-VG PP_1 RPVBD
O I-VG P VBN	O I-VG PT_1T1 RBI-NPI-VG	O I-VG PP_1 RPVBG
O I-VG P VBZ	O I-VG PT_1P1 RBI-NPVB	O I-VG PP_1 RPVBN
O I-VG P VBP	O I-VG PT_1P1 RBI-NPVBD	O I-VG PP_1 RPVBZ
O I-VG PP1 MDVB	O I-VG PT_1P1 RBI-NPVBG	O I-VG PP_1 RPVBP
O I-VG PP1 MDVBD	O I-VG PT_1P1 RBI-NPVBN	O I-VG PP1 EXVBZ
O I-VG PP1 MDVBG	O I-VG PT_1P1 RBI-NPVBZ	O I-VG PP1 EXVBP
O I-VG PP1 MDVBN	O I-VG PT_1P1 RBI-NPVB	

Fonte: Produção do próprio autor

No conjunto, os espaços em branco delimitam os argumentos da regra. Por exemplo, na primeira regra 'O I-VG P VB', 'O' é a etiqueta de grupo foco, 'I-VG' é a etiqueta alvo, 'P' é o

modelo da condição e 'VB' é o valor da condição. A tradução da regra é: substitua 'O' por 'I-VG' se a etiqueta morfossintática corrente ('P') for igual a 'VB'. Os modelos de condição são formados por 'P' para indicar a etiqueta morfossintática e 'T' para indicar a etiqueta de grupo (verbal ou nominal). Para indicar o contexto das etiquetas, utiliza-se '_1' para a etiqueta anterior e '+1' para a etiqueta posterior.

O conjunto de regras apresentado retorna resultados satisfatórios para a identificação de grupos verbais.

3.3.11 Identificação de entidades nomeadas

A identificação de entidades nomeadas é a tarefa de localização e classificação de elementos no texto a partir de categorias pré-definidas. Em PLN, as categorias mais utilizadas referem-se a nomes de pessoas (*PERSON*), organizações (*ORGANIZATION*), locais (*LOCATION*), datas (*DATE*), tempo (*TIME*), moedas (*MONEY*), entre outras.

Para identificação de entidades nomeadas, utilizou-se a função interna '*nltk.chunk.ne_chunk()*' do próprio *NLTK*, que toma como argumento uma sentença etiquetada.

3.3.12 Identificação de sufixos

Para a identificação de sufixos, criou-se um arquivo de regras simbólicas escritas manualmente, '*suffix_rules.txt*'. As regras levam em consideração a etiqueta morfossintática da palavra, o seu radical e a terminação. A figura a seguir exibe alguns exemplos de regras utilizadas na identificação de palavras com sufixos:

Quadro 8 - Exemplos de regras para a identificação de palavras com sufixos

SUBSTANTIVOS	ADJETIVOS	VERBOS
NN END ation ation	JJ END y y	VB END ify ify
NN END al al	JJ END able able	VB END ize ize
NN END ance ance	JJ END ible ible	VB END ise ise
NN END ence ence	JJ END al al	VB END en en

continuação

SUBSTANTIVOS	ADJETIVOS	VERBOS
NN END ee ee	JJ END ful ful	VBD END ed ed
NN END er er	JJ END ous ous	VBN END ed ed
NN END or or	JJ END ic ic	VBD END d ed
NN END ion ion	JJ END ed ed	VBN END ing ing
NN END ation ation	JJ END ing ing	VBG END ing ing
NN END ition ition	JJ STEMLASTCHAR/END i,y y	VBZ END s s
NN END ision ision	JJR END er er	

Fonte: Produção do próprio autor

Para evidenciar o significado das regras, toma-se, como exemplo, a regra '*NN END ation ation*', que identifica substantivos que possuem a terminação '*ation*'. Outro exemplo é a regra '*JJ STEMLASTCHAR/END i,y y*', que identifica adjetivos em que o radical termina em '*i*' e a palavra termina em '*y*'. O último argumento das regras é o valor a ser retornado como classificação.

O uso da lista de formas processadas pelo método de *stemming* funciona como um fator de restrição para melhorar a precisão em relação à abrangência.

3.3.13 Estrutura de dados para armazenar as análises

XML (eXtensible Markup Language) é uma linguagem de representação, que pode ser usada para representar textos anotados e recursos lexicais (BIRD; KLEIN; LOPER, 2009, p.425). A linguagem permite a criação de representações que se assemelham às de banco de dados, sem especificação de estrutura fixa inicial, além de possibilitar a criação de *tags*, como as de *HTML*, definidas pelo próprio usuário.

O exemplo a seguir mostra a estrutura criada para armazenar dados estatísticos gerais do texto analisado:

```

<statistics>
  <tokens>44</tokens>
  <types>33</types>
  <typetokenratio>0.75</typetokenratio>
  <paragraphs>3</paragraphs>
  <sentences>3</sentences>
  <words>41</words>
  <averagewordspersentence>13.6666666667</averagewordspersentence>
  <numberofcharacters>216</numberofcharacters>
  <syllables>68</syllables>
  <averagesyllablesperword>1.65853658537</averagesyllablesperword>
  <fleschreadingease>108.363333333</fleschreadingease>
  <fleschkincaid>1.54</fleschkincaid>
  <coctagger>36</coctagger>
  <cognatesratio>0.878048780488</cognatesratio>
</statistics>

```

Figura 37 - Exemplo de estrutura XML para armazenar dados

Fonte: Produção do próprio autor

Além disso, os dados representados podem ser acessados e manipulados facilmente por qualquer linguagem de programação. Em *Python*, há várias bibliotecas para manipulação de *XML*. Decidiu-se pela utilização do módulo '*xml.etree.cElementTree*'.

Assim, *XML* constitui uma escolha natural para a pesquisa dada sua funcionalidade e flexibilidade. O papel da linguagem *XML* no módulo é representar as análises de maneira organizada para permitir novas análises e extração de itens lexicais sem a alteração de todo o código, quando houver a necessidade de alterações.

Para tanto, a seguinte estrutura interna em níveis foi projetada:

Quadro 9 - Estrutura em *XML* para representar textos analisados

continua


```

    <statistics>
      <types></types>
      <tokens></tokens>
      <typetokenratio></typetokenratio>
      <sentences></sentences>
      <paragraphs></paragraphs>
    <cognates></cognates>
  </statistics>
  <title>
    <paragraph>
      <sentence>
        <token></token>
      </sentence>
    </paragraph>
  </title>
  <text>
    <paragraph>
      <sentence>
        <token></token>
      </sentence>
    </paragraph>
  </text>
</xmltext>

```

Fonte: Produção do próprio autor

Conforme as análises do texto são realizadas, em paralelo, os resultados das análises são utilizados para o preenchimento da estrutura apresentada acima.

A estrutura preenchida²⁹ com a análise é a seguinte:

²⁹ Na imagem apresenta-se a estrutura com um número limitado de análises para privilegiar a visualização.

```

<paragraph id="2">
  <sentence id="2">
    <token cog="n" freq="1" group="nI" id="38" key="0.0" pos="PP$">Their</token>
    <token cog="n" freq="1" group="nI" id="39" key="3.0" pos="NN">child</token>
    <token cog="n" freq="27" group="O" id="40" key="3.0" pos=",">,</token>
    <token cog="n" freq="8" group="nI" id="41" key="0.0" pos="DT">a</token>
    <token cog="n" freq="2" group="nI" id="42" key="13.0" pos="NN">baby</token>
    <token cog="n" freq="1" group="nI" id="43" key="4.0" pos="NN">boy</token>
    <token cog="n" freq="27" group="O" id="44" key="3.0" pos=",">,</token>
    <token cog="n" freq="11" group="O" id="45" key="11.0" pos="VBD">was</token>
    <token cog="n" freq="1" group="O" id="46" key="7.0" pos="VBN">delivered</token>
    <token cog="n" freq="4" group="O" id="47" key="11.0" pos="IN">after</token>
    <token cog="n" freq="2" group="nI" id="48" key="0.0" pos="PP$">his</token>
    <token cog="n" freq="1" group="nI" id="49" key="3.0" pos="NN">mother</token>
    <token cog="n" freq="2" group="nB" id="50" key="0.0" pos="POS">'s</token>
    <token cog="n" freq="1" group="nI" id="51" key="3.0" pos="NN">death</token>
    <token cog="n" freq="11" group="O" id="52" key="0.0" pos="CC">and</token>
    <token cog="n" freq="1" group="O" id="53" key="7.0" pos="VBD">survived</token>
    <token cog="n" freq="27" group="O" id="54" key="3.0" pos=",">,</token>
    <token cog="n" freq="36" group="nI" id="55" key="7.0" pos="DT">the</token>
    <token cog="y" freq="6" group="nI" id="56" key="38.0" pos="NN">police</token>
    <token cog="n" freq="11" group="O" id="57" key="37.0" pos="VBD">said</token>
    <token cog="n" freq="18" group="O" id="58" key="0.0" pos="SENT">.</token>
  </sentence>
</paragraph>

```

Figura 38 - Exemplo de estrutura XML preenchida com dados de análises

Fonte: Produção do próprio autor

Dada a estrutura de anotação das análises do texto em XML, a adição de novas análises pode ser feita sem a necessidade de grandes alterações no código e no funcionamento das análises já implementadas.

3.4 Aplicação dos módulos de análise aos *corpora* de estudo

Nesta seção, descreve-se a criação do módulo para análise dos *corpora* de estudo e sua aplicação.

Os dois corpora de estudo utilizados para o estabelecimento dos módulos de análise e criação de atividades - anúncios de emprego e textos de divulgação científica das revistas *Scientific American* e *NewScientist* - serviram como bancos de dados de exemplos da língua em uso e ajudaram a responder a questões importantes para a melhoria dos módulos de análise, assim como os movimentos metodológicos da análise da padronização evidenciaram possíveis caminhos para criação de algoritmos para análises automáticas e possibilidades de criação de exercícios.

Para tanto, houve a necessidade de criação de um módulo específico para reunir as análises de textos criadas para análises de *corpora*. Tal módulo é descrito na seção a seguir.

3.4.1 Criação de um analisador de *corpus*

Para auxiliar nas análises dos *corpora* de estudo, foi desenvolvido o módulo '*corpusanalyzer*', que, por meio dos módulos de análise de textos criados, extrai informações de conjuntos de textos.

O módulo possui a função '*add_from_folder*' para a adição de textos a fim de formar um *corpus* que é adicionado à pasta '*corpora*' do sistema de análise de textos. A função exige dois argumentos: '*path*' e '*name*'. O argumento '*path*' é o caminho do diretório onde os textos são armazenados. O argumento '*name*' é o identificador para o *corpus* a ser criado. Na execução, cada texto do diretório de entrada é analisado e salvo em arquivos *XML*.

Para acessar o *corpus* adicionado, usa-se a função '*use*' com o nome do *corpus* como argumento. As principais funções de análise são:

- a. retorno das principais informações do *corpus* (nome, gênero, número total de arquivo, número de itens, número de formas, razão forma item etc.);
- b. criação de listas de frequência de palavras do *corpus*;
- c. criação de listas de palavras-chave;
- d. criação de listas de etiquetas morfossintáticas;
- e. extração de grupos nominais e verbais;
- f. criação de agrupamentos;
- g. extração das principais análises de texto (sufixos, palavras cognatas etc.);
- h. análise de concordâncias;
- i. extração de n-gramas;
- j. extração de colocados de palavras.

Além de auxiliar nas análises do *corpus* de estudo, no módulo, foram programadas funções específicas para extração de informações para criação de atividades. A análise dos *corpora* de estudo contribuíram para a consolidação de suas funções em um movimento de constante análise e desenvolvimento. As seções a seguir, além de fornecerem a descrição dos *corpora* de estudo, também evidenciam tal processo. É dada maior ênfase à análise do *corpus* de

anúncio de emprego, dado que foi o *corpus* que mais contribuiu para a construção das funções, por ter sido o primeiro a receber as análises dos módulos criados. Para o *corpus* de textos de divulgação científica é fornecida apenas a sua composição e palavras-chave.

3.4.2 Análise do *corpus* de anúncios de emprego em inglês

Um anúncio de emprego pode ser definido como uma mensagem em um jornal, *site* de empresa ou em um *site* especializado em anúncios, publicado para conseguir possíveis candidatos para uma vaga disponibilizada por um contratante, de acordo com sua necessidade de determinado profissional.

O principal público-alvo de anúncios de emprego são candidatos à procura de emprego, que estejam adequados aos requisitos e necessidades informadas pelo contratante. O principal propósito comunicativo é fazer com que os candidatos adequados se candidatem à vaga em aberto.

Conforme Behnam e Behnam (2012, p. 94), que investigaram as similaridades e diferenças entre as estruturas esquemáticas de anúncios de emprego em língua inglesa e persa, em 20 textos do jornal eletrônico *Washington Post* e 20 textos do jornal eletrônico *The Times*, os seguintes movimentos podem ser identificados no gênero:

- a. *job title*: nome da vaga que o contratante precisa preencher;
- b. *employer name*: nome da empresa ou contratante;
- c. *reader attraction*: diversas sentenças, na forma de perguntas, geralmente antes do corpo real do anúncio, com o objetivo de atrair a atenção do leitor. É comum o uso de recursos tipográficos para dar o enfoque;
- d. *establishing credentials*: apresenta informações positivas da empresa/contratante por meio de prêmios e resultados atingidos pela empresa, a fim de convencer possíveis candidatos sobre a reputação da empresa/contratante no mercado;
- e. *introducing job offer*: apresenta a vaga disponível oferecida;
- f. *offering incentives*: apresenta os incentivos oferecidos aos candidatos;
- g. *pressure tactic*: tenta convencer de forma imediata aqueles que já estão inclinados a se candidatar;

- h. *employer motto*: refere-se a uma pequena sentença ou algumas palavras que constituem o princípio guia do empregador, geralmente no final do texto;
- i. *soliciting response*: atitude da empresa/contratante em iniciar novas relações e receber pedidos de emprego;
- j. *P.S. move*: uma ou mais sentenças curtas que aparecem após o movimento 'soliciting response', com uma informação que não foi possível ser mencionada no corpo do anúncio;
- k. *Company profile*: é um parágrafo ou mais, que aparece no final do anúncio, sobre a história e longa experiência do contratante no mercado e sua reputação, com descrições detalhadas para um leitor mais inclinado a se candidatar.

Alguns dos movimentos apresentados possuem uma série de submovimentos. Por exemplo, o movimento '*introducing job offer*' pode conter submovimentos relacionados às responsabilidades da vaga, requisitos e qualificações necessárias.

O *corpus* de anúncios de emprego possui a seguinte composição:

Quadro 10 - Composição do *corpus* de anúncios de emprego

Arquivos	Itens	Formas	Razão item/forma	Hápx	Parágrafos	Sentenças
135	37.700	5.075	13%	2.566	765	1.982

Fonte: Produção do próprio autor

O total de itens no *corpus*, com 135 textos, é de 37.700, com um total de 5.075 formas e razão forma/item de 13%. Por essa composição, pode-se considerar que há muita repetição das formas, sendo que os *hápax legomena*³⁰ somam 2.566 formas.

Em análise da frequência dos itens do *corpus*, a dispersão da lista de frequência é apresentada:

³⁰Formas que ocorrem apenas uma vez no *corpus*.

Figura 39 - Gráfico de dispersão da lista de frequência do *corpus* de anúncios de emprego
 Fonte: Produção do próprio autor

No gráfico, é possível verificar que, em geral, há pouca variação nas faixas de frequência. A maioria dos itens possui baixa frequência, comparada à frequência de alguns itens no topo da lista ordenada do item mais frequente para o menos frequente. Em grande parte da lista, a frequência dos itens é uniforme.

Os dados foram conseguidos a partir da contagem da frequência de cada item no *corpus*, a qual resulta em uma lista de frequência (*wordlist_to_file(filename)*). Parte dos dados é mostrada no quadro a seguir:

Quadro 11 - Itens mais frequentes do *corpus* de anúncios de emprego

N ITENS FREQ	126 department 38	252 stock 20	378 under 14
1 and 1971	127 employee 38	253 they 20	379 employees 14
2 , 1788	128 strategic 37	254 user 20	380 theatre 14
3 . 1470	129 industry 37	255 post 20	381 order 14
4 the 1062	130 seeking 37	256 excel 20	382 personal 14
5 to 971	131 software 37	257 current 20	383 primary 14
6 of 835	132 @ 36	258 * 20	384 city 14
7 a 644	133 competitive 36	259 goals 20	385 individuals 14
8 in 620	134 account 36	260 delivery 20	386 standards 14
9 with 474	135 quality 36	261 plans 20	387 shift 14
10 for 443	136 key 36	262 up 20	388 should 14
11 : 409	137 looking 35	263 university 20	389 qualified 14
12 is 281	138 ! 35	264 demonstrated 20	390 partner 14
13 experience 247	139 assist 35	265 purchase 19	391 where 14
14 or 222	140 products 35	266 harvard 19	392 open 14
15 as 214	141 customer 35	267 expertise 19	393 hours 14
16 will 213	142 able 35	268 us 19	394 direction 14
17 -- 186	143 direct 34	269 ideal 19	395 external 14
18 an 167	144 needs 34	270 administrative 19	396 media 14
19 (161	145 about 34	271 any 19	397 status 14
20) 161	146 times 34	272 drive 19	398 positions 14

continua

continuação

21 work 160	147 activities 33	273 exciting 19	399 communicate 14
22 are 156	148 lead 33	274 programme 19	400 prepare 14
23 be 156	149 through 33	275 advertising 19	401 materials 14
24 skills 155	150 applications 33	276 comprehensive 19	402 technologies 14
25 's 150	151 understanding 33	277 producer 19	403 insurance 14
26 on 147	152 apply 33	278 day 19	404 what 14
27 management 132	153 has 32	279 employer 19	405 committed 14
28 you 130	154 people 32	280 national 19	406 maintenance 14
29 new 126	155 assistant 32	281 2005 19	407 clinical 14
30 business 124	156 technology 32	282 s/he 18	408 build 14
31 team 123	157 opportunities 32	283 laboratory 18	409 some 14
32 development 123	158 description 31	284 content 18	410 appropriate 14
33 we 121	159 process 31	285 report 18	411 family 14
34 our 121	160 minimum 31	286 child 18	412 administrator 14
35 have 118	161 create 31	287 closing 18	413 need 14
36 sales 115	162 candidates 31	288 following 18	414 url 14
37 ability 113	163 email 31	289 may 18	415 handle 14
38 years 108	164 duties 30	290 finance 18	416 party 14
39 that 105	165 complex 30	291 success 18	417 works 14
40 this 105	166 plus 30	292 title 18	418 accountant 14
41 ; 104	167 date 30	293 contract 18	419 demonstrate 14
42 including 100	168 field 30	294 # 18	420 compensation 13
43 at 91	169 its 30	295 use 18	421 presentation 13
44 all 85	170 planning 30	296 organization 18	422 then 13
45 strong 82	171 both 30	297 reporting 18	423 bank 13
46 by 82	172 highly 29	298 ad 18	424 approach 13
47 requirements 80	173 it 29	299 additional 18	425 website 13
48 company 77	174 time 29	300 production 17	426 compliance 13
49 working 76	175 3 29	301 qualifications 17	427 perform 13
50 project 76	176 teams 29	302 testing 17	428 advanced 13
51 support 75	177 more 29	303 improve 17	429 partners 13
52 - 74	178 relationships 28	304 providing 17	430 professionals 13
53 degree 73	179 maintain 28	305 corporate 17	431 track 13
54 office 70	180 boston 28	306 background 17	432 between 13
55 position 70	181 programs 28	307 no 17	433 results 13
56 must 68	182 can 28	308 word 17	434 nurse 13
57 data 68	183 join 27	309 schedule 17	435 faculty 13
58 other 67	184 if 27	310 such 17	436 major 13
59 required 66	185 senior 27	311 january 17	437 monthly 13
60 please 66	186 level 27	312 effective 17	438 strategy 13
61 services 65	187 good 27	313 college 17	439 bonus 13
62 responsibilities 65	188 5 27	314 unit 17	440 chairman 13
63 your 64	189 site 27	315 leadership 17	441 international 13
64 knowledge 64	190 human 26	316 functions 17	442 impact 13
65 manager 63	191 multiple 26	317 requires 17	443 areas 13
66 ? 63	192 across 26	318 range 17	444 etc 13
67 job 63	193 children 26	319 growing 17	445 regulatory 13
68 marketing 60	194 ma 26	320 experienced 17	446 hr 13
69 from 60	195 bachelor 26	321 2014 17	447 match 13
70 school 59	196 group 25	322 administration 17	448 familiarity 13
71 client 57	197 offers 25	323 proficiency 17	449 preparation 12
72 environment 57	198 care 25	324 risk 16	450 limited 12
73 executive 56	199 which 25	325 them 16	451 changes 12
74 york 55	200 interpersonal 25	326 associate 16	452 third 12
75 excellent 53	201 individual 25	327 solutions 16	453 schools 12
76 research 53	202 computer 25	328 identify 16	454 each 12
77 manage 53	203 but 25	329 / 16	455 directly 12
78 product 53	204 one 24	330 offer 16	456 priorities 12
79 well 53	205 written 24	331 interested 16	457 manner 12
80 staff 52	206 and/or 24	332 assistance 16	458 participate 12
81 communication 52	207 paid 24	333 market 16	459 do 12
82 health 51	208 equivalent 24	334 london 16	460 interest 12
83 projects 51	209 internal 24	335 review 16	461 specific 12
84 candidate 50	210 community 24	336 provides 16	462 dean 12
85 role 50	211 application 24	337 building 16	463 head 12
86 web 50	212 contact 24	338 members 16	464 removed 12
87 medical 49	213 401 23	339 strategies 16	465 proven 12
88 information 49	214 meetings 23	340 levels 16	466 cost 12
89 online 48	215 news 23	341 customers 16	467 ms 12
90 plan 48	216 k 23	342 communications 16	468 exceptional 12
91 analysis 48	217 creative 23	343 problems 16	469 organisation 12

continua

continuação

92 about.com 47	218 not 23	344 send 16	470 million 12
93 design 47	219 performance 23	345 staffing 16	471 j 12
94 include 47	220 large 23	346 career 16	472 vision 12
95 service 45	221 full 23	347 relevant 16	473 actively 12
96 systems 45	222 based 23	348 available 16	474 powerpoint 12
97 resume 45	223 1 23	349 validation 16	475 html 12
98 salary 45	224 essential 23	350 full-time 15	476 verbal 12
99 who 45	225 successful 22	351 top 15	477 coverage 12
100 program 44	226 fax 22	352 implementation 15	478 form 12
101 responsible 44	227 travel 22	353 assigned 15	479 ten 12
102 education 44	228 processes 22	354 managers 15	480 two 12
103 preferred 43	229 revenue 22	355 possess 15	481 help 12
104 accounting 43	230 director 22	356 newspaper 15	482 member 12
105 ensure 43	231 events 22	357 coordinate 15	483 most 12
106 related 43	232 issues 22	358 area 15	484 various 12
107 managing 43	233 effectively 22	359 calls 15	485 [12
108 financial 42	234 necessary 22	360 resources 15	486 4 12
109 their 42	235 2 21	361 initiatives 15	487 execution 12
110 tax 42	236 flexible 21	362 attention 15	488 trends 12
111 within 41	237 make 21	363 significant 15	489 per 12
112 also 41	238 training 21	364 closely 15	490 access 12
113 develop 40	239 learning 21	365 database 15	491 via 12
114 benefits 40	240 developing 21	366 using 15	492 facilitate 12
115 high 40	241 general 21	367 engineering 15	493 currently 12
116 provide 39	242 these 21	368 organizational 15	494 center 12
117 operations 39	243 leading 21	369 existing 15	495 been 12
118 nytimes.com 39	244 \$ 21	370 multimedia 15	496 implement 12
119 best 39	245 location 21	371 detail 15	497 6 11
120 opportunity 39	246 into 21	372 package 15	498 phone 11
121 technical 39	247 tuition 21	373 located 15	499 over 11
122 reports 38	248 meet 21	374 system 15	500 engineer 11
123 clients 38	249 ' 21	375 ensuring 15	
124 professional 38	250 procedures 21	376 accounts 15	
125 part 38	251 year 21	377 special 15	

Fonte: Produção do próprio autor

Os primeiros itens no topo da lista são palavras gramaticais (artigos, preposições e conjunções) e caracteres de pontuação. Em *corpora* de língua geral do inglês, é muito comum palavras como ‘*the*’ ou ‘*of*’ serem as primeiras da lista, como mostra a tabela com a frequência parcial do *corpus* de referência utilizado nesta pesquisa:

Quadro 12 - Os 100 itens mais frequentes do *corpus* de referência

N ITENS FREQ	26 have 444632	52 so 237498	78 than 143397
1 the 5999032	27 but 443370	53 would 228004	79 could 142730
2 5957695	28 not 427129	54 what 223648	80 well 141308
3 , 4987668	29 from 421736	55 can 209026	81 ! 140368
4 . 4705951	30 had 410926	56 when 208066	82 now 138356
5 of 3019256	31 his 406769	57 no 207835	83 your 133218
6 and 2596326	32) 405925	58 more 207758	84 me 131000
7 to 2576920	33 (399900	59 up 206172	85 over 130008
8 a 2150774	34 ? 385519	60 ; 202635	86 may 126388
9 in 1928407	35 they 373392	61 out 195789	87 it's 125611
10 that 1043929	36 or 364242	62 said 194944	88 just 125394

continua

continuação

11 is 962315	37 which 362416	63 who 191204	89 new 123316
12 it 914705	38 an 334130	64 about 190152	90 also 122545
13 for 872515	39 she 322799	65 do 175482	91 these 122220
14 was 858390	40 were 306358	66 them 165987	92 any 121141
15 on 725086	41 her 301401	67 some 165905	93 first 119763
16 i 720894	42 we 298248	68 its 159099	94 very 118538
17 with 653516	43 one 287739	69 into 156550	95 know 118147
18 as 648664	44 there 283127	70 then 153537	96 people 115043
19 be 645599	45 all 274904	71 time 153276	97 see 114491
20 he 589166	46 : 265999	72 him 152690	98 after 113240
21 you 583875	47 been 258432	73 other 151498	99 such 106181
22 at 519686	48 their 252081	74 two 151424	100 should 104079
23 by 508481	49 if 251661	75 only 147434	
24 are 452852	50 has 249938	76 like 146766	
25 this 449701	51 will 249175	77 my 145727	

Fonte: Produção do próprio autor

Entretanto, a palavra ‘*and*’ é a mais frequente no *corpus* em estudo. O fato pode ser considerado um indício da importância dessa palavra para o gênero em questão. A primeira palavra de conteúdo a surgir é ‘*experience*’, na 13ª posição da lista. Em seguida, conforme se alcança a 24ª posição, o número de formas gramaticais diminui, e a ocorrência de palavras lexicais é mais comum. As palavras de conteúdo ‘*skills*’, ‘*sales*’, ‘*work*’, ‘*years*’, ‘*business*’, ‘*team*’ e ‘*strong*’ podem ser consideradas características do conteúdo abordado pelo gênero dos textos do *corpus* em análise, anúncios de emprego, cujos contextos também são merecedores de investigação.

Em análises em que o objetivo esteja voltado para o *ESP*, é comum privilegiar as palavras de conteúdo, pois são consideradas mais representativas do conjunto de textos e úteis para os aprendizes. Para questões de representatividade, além da frequência bruta, é importante também que a palavra ocorra em vários textos.

Por meio da função ‘*wordsperfile_to_file*’, uma lista de tais informações foi gerada. Por essa função, é possível verificar a ocorrência dos itens nos textos do *corpus*. A título de exemplificação, o Quadro 16 relaciona itens que ocorrem em mais de 9% dos textos do *corpus* de anúncios de emprego.

Quadro 13 - Itens que ocorrem em mais de 9% dos textos do *corpus* de anúncios de emprego

N ITENS FREQ %	77 also 33 24%	153 full 22 16%	229 they 16 11%
1 , 134 96%	78 your 32 23%	154 ma 22 16%	230 2 16 11%
2 . 134 96%	79 services 32 23%	155 research 22 16%	231 nytimes.com 16 11%
3 and 130 93%	80 provide 31 22%	156 planning 22 16%	232 background 16 11%
4 to 129 92%	81 office 31 22%	157 product 22 16%	233 expertise 16 11%
5 of 122 87%	82 responsible 31 22%	158 times 22 16%	234 word 16 11%
6 in 121 86%	83 competitive 31 22%	159 computer 22 16%	235 assistance 16 11%
7 the 120 86%	84 sales 31 22%	160 it 21 15%	236 schedule 16 11%
8 for 120 86%	85 within 31 22%	161 complex 21 15%	237 following 16 11%
9 a 119 85%	86 manage 31 22%	162 people 21 15%	238 group 16 11%
10 : 113 81%	87 description 30 21%	163 and/or 21 15%	239 processes 16 11%
11 with 111 79%	88 preferred 30 21%	164 can 21 15%	240 developing 16 11%
12 experience 108 77%	89 professional 30 21%	165 needs 21 15%	241 good 16 11%
13 is 101 72%	90 project 30 21%	166 not 21 15%	242 building 16 11%
14 or 101 72%	91 benefits 30 21%	167 teams 21 15%	243 these 16 11%
15 skills 83 59%	92 part 30 21%	168 equivalent 21 15%	244 ! 16 11%
16 as 83 59%	93 projects 30 21%	169 assist 21 15%	245 members 16 11%
17 (83 59%	94 web 30 21%	170 its 21 15%	246 send 16 11%
18) 83 59%	95 opportunity 29 21%	171 tuition 21 15%	247 effectively 16 11%
19 work 79 56%	96 develop 29 21%	172 stock 20 14%	248 ' 16 11%
20 's 78 56%	97 who 29 21%	173 relationships 20 14%	249 proficiency 16 11%
21 an 76 54%	98 looking 28 20%	174 reports 20 14%	250 demonstrated 16 11%
22 have 75 54%	99 through 28 20%	175 financial 20 14%	251 year 16 11%
23 be 73 52%	100 understanding 28 20%	176 client 20 14%	252 full-time 15 11%
24 years 72 51%	101 their 27 19%	177 strategic 20 14%	253 qualifications 15 11%
25 on 72 51%	102 health 27 19%	178 senior 20 14%	254 human 15 11%
26 will 72 51%	103 apply 27 19%	179 multiple 20 14%	255 marketing 15 11%
27 are 70 50%	104 email 27 19%	180 across 20 14%	256 us 15 11%
28 management 67 48%	105 operations 26 19%	181 process 20 14%	257 interested 15 11%
29 that 66 47%	106 has 26 19%	182 field 20 14%	258 performance 15 11%
30 team 65 46%	107 staff 26 19%	183 design 20 14%	259 review 15 11%
31 ability 64 46%	108 date 26 19%	184 location 20 14%	260 provides 15 11%
32 including 62 44%	109 able 26 19%	185 customer 20 14%	261 goals 15 11%
33 degree 58 41%	110 key 26 19%	186 meet 20 14%	262 community 15 11%
34 at 57 41%	111 program 25 18%	187 based 20 14%	263 products 15 11%
35 strong 56 40%	112 service 25 18%	188 but 20 14%	264 \$ 15 11%
36 this 56 40%	113 offers 25 18%	189 successful 19 14%	265 range 15 11%
37 our 56 40%	114 york 25 18%	190 purchase 19 14%	266 up 15 11%
38 new 55 39%	115 minimum 25 18%	191 one 19 14%	267 existing 15 11%
39 by 54 39%	116 applications 25 18%	192 if 19 14%	268 ? 15 11%
40 we 53 38%	117 bachelor 25 18%	193 clients 19 14%	269 day 15 11%
41 requirements 51 36%	118 department 25 18%	194 exciting 19 14%	270 experienced 15 11%
42 development 50 36%	119 quality 25 18%	195 leading 19 14%	271 2014 15 11%
43 other 48 34%	120 candidates 25 18%	196 internal 19 14%	272 available 15 11%
44 support 48 34%	121 employee 25 18%	197 comprehensive 19 14%	273 necessary 15 11%
45 business 47 34%	122 education 25 18%	198 opportunities 19 14%	274 production 14 10%
46 all 47 34%	123 direct 24 17%	199 1 19 14%	275 top 14 10%
47 responsibilities 47 34%	124 highly 24 17%	200 2005 19 14%	276 improve 14 10%
48 position 47 34%	125 join 24 17%	201 maintain 18 13%	277 success 14 10%
49 please 46 33%	126 medical 24 17%	202 boston 18 13%	278 programs 14 10%
50 knowledge 46 33%	127 activities 24 17%	203 flexible 18 13%	279 identify 14 10%
51 must 45 32%	128 online 24 17%	204 closing 18 13%	280 / 14 10%
52 working 45 32%	129 plus 24 17%	205 excel 18 13%	281 offer 14 10%
53 required 44 31%	130 data 24 17%	206 level 18 13%	282 such 14 10%
54 salary 44 31%	131 technical 24 17%	207 ideal 18 13%	283 january 14 10%
55 well 44 31%	132 both 24 17%	208 which 18 13%	284 effective 14 10%
56 excellent 43 31%	133 create 24 17%	209 title 18 13%	285 any 14 10%
57 from 43 31%	134 analysis 24 17%	210 individual 18 13%	286 paid 14 10%
58 communication 43 31%	135 about 24 17%	211 # 18 13%	287 contract 14 10%
59 resume 41 29%	136 managing 24 17%	212 application 18 13%	288 media 14 10%
60 ; 41 29%	137 401 23 16%	213 issues 18 13%	289 positions 14 10%
61 job 40 29%	138 best 23 16%	214 site 18 13%	290 delivery 14 10%
62 environment 39 28%	139 k 23 16%	215 employer 18 13%	291 plans 14 10%
63 company 38 27%	140 lead 23 16%	216 executive 18 13%	292 attention 14 10%
64 - 37 26%	141 time 23 16%	217 essential 18 13%	293 use 14 10%
65 include 37 26%	142 5 23 16%	218 fax 17 12%	294 levels 14 10%
66 role 36 26%	143 interpersonal 23 16%	219 report 17 12%	295 reporting 14 10%
67 information 35 25%	144 more 23 16%	220 make 17 12%	296 growing 14 10%
68 plan 35 25%	145 systems 23 16%	221 creative 17 12%	297 detail 14 10%
69 high 35 25%	146 contact 23 16%	222 current 17 12%	298 account 14 10%

continua

continuação

70 you 35 25%	147 school 22 16%	223 training 17 12%	299 located 14 10%
71 @ 34 24%	148 written 22 16%	224 general 17 12%	300 handle 14 10%
72 manager 34 24%	149 duties 22 16%	225 technology 17 12%	301 procedures 14 10%
73 seeking 34 24%	150 3 22 16%	226 into 17 12%	302 national 14 10%
74 related 34 24%	151 industry 22 16%	227 software 17 12%	303 relevant 14 10%
75 candidate 33 24%	152 large 22 16%	228 travel 16 11%	304 additional 14 10%
76 ensure 33 24%			

Fonte: Produção do próprio autor

Os itens que ocorrem em apenas um texto do *corpus* somam 2.856 ocorrências. Também como esperado, as palavras gramaticais ficam no topo da lista, e as de conteúdo são listadas logo a seguir, conforme a frequência diminui. A palavra de conteúdo com maior ocorrência é novamente '*experience*'. Outras palavras de conteúdo representativas, identificadas anteriormente, também aparecem.

A partir da lista, podem-se identificar agrupamentos de frequência. Por exemplo, a palavra '*experience*', que ocorreu em 77% dos textos, está inserida em um grupo de palavras de alta ocorrência, junto com palavras gramaticais como '*the*', '*a*', '*for*' e '*with*'. Já a palavra '*salary*', que ocorre em 31% dos textos, parece estar em outra faixa ou agrupamento bem mais abaixo.

Para ilustrar a observação, apresenta-se o gráfico de dispersão com as possíveis faixas de divisão dos agrupamentos:

Figura 40 - Agrupamentos na dispersão da frequência de itens por texto

Fonte: Produção do próprio autor

No gráfico, há a delimitação de quatro agrupamentos de frequência até a curva para a base. No primeiro grupo, encontram-se as palavras que ocorrem em mais de 70% dos textos, em sua maioria palavras gramaticais, com a ocorrência de apenas '*experience*'. No segundo grupo,

estão as palavras que ocorrem em mais de 40% dos textos, com uma ocorrência maior de palavras lexicais, como '*skills*', '*years*', '*management*', '*team*' e '*work*', que tanto podem ser utilizadas como verbo ou substantivo. Há também a presença de auxiliares ou modais, como '*have*' e '*will*'. No terceiro grupo, há um número maior de palavras, em sua grande maioria, lexicais, com a prevalência de substantivos ('*position*', '*job*', '*candidate*' etc.), mas há a ocorrência de adjetivos como '*strong*', '*new*', '*excellent*', '*high*', '*responsible*' e '*competitive*', além de verbos, como '*seeking*', '*working*', '*include*' e '*provide*', entre outras classes de palavras. No último grupo, está o restante das palavras que ocorrem em menos de 29% dos textos.

A identificação e o conhecimento de tais faixas e agrupamentos podem privilegiar a tarefa de seleção de itens lexicais para análises mais detalhadas, a elaboração de recursos para criação de atividades, além de informar possíveis algoritmos e heurísticas para realizar a seleção automaticamente. Como observado, em determinadas faixas, é possível extrair um número maior de palavras lexicais ou gramaticais que podem ser comuns do *corpus* ou gênero textual de pesquisa. Tais dados podem também auxiliar na tarefa de identificação de palavras que fazem parte da estrutura do gênero e de palavras que estão mais relacionadas ao tópico ou assunto dos textos.

Para ter uma ideia da ocorrência de palavras gramaticais e de palavras de conteúdo, foi realizada uma contagem das etiquetas de cada item no *corpus*. O resultado pode ser visualizado na seguinte tabela:

Tabela 7 - Frequência das etiquetas morfossintáticas no *corpus*

N	ITENS	FREQ	%	N	ITENS	FREQ	%	N	ITENS	FREQ	%
1	NN	7537	20%	17	VBZ	564	1%	33	WRB	33	0%
2	IN	3425	9%	18	VBP	452	1%	34	\$	31	0%
3	NNS	3334	9%	19	PRP	409	1%	35	"	30	0%
4	JJ	3013	8%	20	MD	352	1%	36	RP	22	0%
5	NNP	2597	7%	21	PRP\$	271	1%	37	``	20	0%
6	DT	2313	6%	22	POS	176	0%	38	PDT	20	0%
7	CC	2218	6%	23	NNPS	134	0%	39	PP	14	0%
8	,	2038	5%	24	WDT	90	0%	40	RBR	10	0%
9	.	1892	5%	25	)	88	0%	41	WP\$	10	0%
10	VB	1593	4%	26	JJS	84	0%	42	RBS	9	0%

continua

11	VBG	1038	3%	27	VBD	59	0%	continuação			
12	TO	976	3%	28	WP	58	0%	43	#	7	0%
13	:	781	2%	29	@	54	0%	44	UH	7	0%
14	VDN	623	2%	30	(	48	0%	45	FW	4	0%
15	CD	598	2%	31	JJR	45	0%	46	EX	4	0%
16	RB	583	2%	32	LS	36	0%				

Fonte: Produção do próprio autor

A ocorrência de palavras de conteúdo no texto, como substantivos, adjetivos e verbos, é bem maior em relação à de palavras gramaticais:

Etiquetas morfossintáticas em categorias

Figura 41 - Proporção de etiquetas morfossintáticas em categorias no *corpus* de estudo

Fonte: Produção do próprio autor

O gráfico exibe a proporção de palavras gramaticais, de conteúdo e pontuação no *corpus*: os itens gramaticais ocorrem 10.539; os itens lexicais, 22.136; numerais, pontuação e símbolos, 5.025. Com o resultado, é possível sugerir que os textos possuem uma alta densidade lexical (59%), o que exigiria do aprendiz um conhecimento maior de vocabulário para leitura.

Para observação em maior detalhe das palavras mais representativas do gênero, fez-se a comparação entre a lista de frequência do *corpus* de estudo e a lista de frequência do *corpus* de referência (*BNC*) por meio da função '*corpus.keywords_to_file*', que gerou um valor de chavicidade para cada item. O quadro a seguir exibe o resultado da comparação:

Quadro 14 - Resultado parcial da extração de palavras-chave do *corpus* de estudo

N ITENS CHAVICIDADE	126 description 185	252 presentations 108	378 from 81
1 -- 4101	127 ' 184	253 recruiting 107	379 discussions.requirements 81
2 experience 1864	128 opportunities 182	254 proven 107	380 crm 81
3 's 1773	129 applications 182	255 one 107	381 sagent 81
4 and 1509	130 opportunity 181	256 modeling 107	382 equivalent. 81
5 skills 1335	131 cross-functional 179	257 mba 106	383 demeanor 81
6 about.com 1085	132 software 174	258 ensuring 105	384 http 81
7 nytimes.com 901	133 validation 172	259 communicate 105	385 * 81
8 : 889	134 endeca 171	260 industry 105	386 critical.this 81
9 ability 876	135 javascript 171	261 initiatives 105	387 month.responsibilities 81
10 sales 852	136 coordinate 171	262 bsa/aml 104	388 credentialing 81
11 @ 825	137 health/vision/dental 171	263 required. 104	389 news. 81
12 management 760	138 requirements: 171	264 aerotek 104	390 hrs 80
13 the 751	139 'll 171	265 dhtml 104	391 qualified 80
14 team 740	140 staff 170	266 faculty 104	392 leadership 80
15 was 686	141 internet 169	267 technologies 104	393 database 80
16 web 672	142 new 167	268 biodiversity 104	394 professionals 80
17 - 657	143 creative 167	269 include: 104	395 preschool 80
18 requirements 638	144 professional 166	270 plus. 104	396 contact 80
19 online 634	145 reports 164	271 environment. 104	397 cpa 80
20 i 631	146 systems 162	272 boston.com 104	398 ms 79
21 responsibilities 624	147 apply 161	273 site 104	399 post-sale 79
22 email 591	148 staffing 157	274 coordination 104	400 requests 79
23 it 588	149 which 157	275 lendkey 104	401 date 79
24 resume 562	150 health 156	276 646-459-2177 104	402 timelines 79
25 development 555	151 maintain 155	277 intown 104	403 vacation 79
26 he 531	152 're 155	278 pharmaceutical 103	404 clinical 79
27 business 530	153 reimbursement 154	279 procedures 103	405 developer 79
28 \$ 489	154 2004 154	280 supervise 103	406 admin 79
29 including 474	155 research 152	281 me 102	407 p/t 79
30 degree 464	156 flexible 151	282 provide 102	408 eno 78
31 marketing 457	157 financial 150	283 for 102	409 recruiters 78
32 strong 439	158 products 149	284 what 102	410 editorial 78
33 salary 433	159 tax 149	285 compliance 102	411 csc 78
34 manage 427	160 producer 149	286 piping 102	412 utilizing 78
35 client 422	161 relationships 148	287 goals 102	413 teaming 78
36 # 420	162 processplus 148	288 solutions 101	414 by 77
37 401 413	163 edgesuite 148	289 facilitate 101	415 priorities 77
38 work 410	164 xml 148	290 stakeholders 101	416 metrics 77
39 candidate 402	165 2+ 148	291 service 100	417 no 77
40 project 401	166 ba/bs 148	292 or 100	418 assistance 77
41 preferred 355	167 understanding 147	293 aarp 100	419 incoming 77
42 projects 354	168 so 145	294 advertising 99	420 rada 77
43 excellent 353	169 03/mai 145	295 full-time 99	421 prepare 77
44 communication 353	170 create 144	296 bonus 99	422 54 75
45 executive 351	171 would 143	297 purchase 98	423 content 75
46 managing 342	172 school 142	298 perl 98	424 spt 75
47 2014 342	173 plus 142	299 strategies 98	425 trends 75
48 interpersonal 340	174 key 141	300 proficient 97	426 requires 75
49 please 340	175 prioritize 140	301 microbiological 96	427 applicants 75
50 bachelor 338	176 administrator 140	302 nca 96	428 demonstrable 75
51 accounting 333	177 k 139	303 billing 96	429 functions 75
52 that 331	178 been 139	304 testing 95	430 do 74
53 with 330	179 self-starter 138	305 developing 94	431 cincinnati 74
54 program 330	180 familiarity 138	306 f/t 94	432 organization 74
55 url 329	181 equivalent 137	307 2003 94	433 02/mar 74
56 manager 322	182 organizational 137	308 located 94	434 dental 73
57 his 318	183 multimedia 137	309 travel 92	435 stock 73
58 knowledge 318	184 associate 136	310 information 92	436 experienced 73
59 york 316	185 demonstrated 134	311 deliverables 92	437 because 73
60 boston 311	186 direct 133	312 bachelors 91	438 coverage 73
61 required 311	187 deadlines 133	313 planning 91	439 if 72
62 employee 309	188 analyze 131	314 physician 90	440 manages 72
63 website 306	189 req 130	315 user 90	441 looking 72
64 data 306	190 accountant 130	316 execution 90	442 envisioning 72
65 strategic 291	191 activities 129	317 could 90	443 executives 72
66 assist 290	192 collaboratively 129	318 photoshop 89	444 identify 71
67 j 284	193 collaborate 127	319 field 89	445 gaap 71
68 html 284	194 offers 127	320 must 88	446 learning 71
69 2005 284	195 revenue 127	321 confidential 88	447 associates 71

continua

continuação

70 [284	196 java 127	322 e-mail 88	448 events 71
71 s/he 283	197 complex 126	323 implementation 87	449 supervisory 71
72 environment 281	198 qualifications: 126	324 analytical 87	450 liaison 71
73 position 278	199 expertise 126	325 ideal 87	451 microsoft 71
74 excel 277	200 gpv-mrm 126	326 actively 87	452 developmentally 71
75 product 275	201 gynemed 126	327 needs 87	453 like 71
76 medical 271	202 akamai 126	328 exp 87	454 forums 70
77 working 271	203 employer 126	329 essential 87	455 innovative 70
78 office 262	204 ncc-wch 126	330 demonstrate 87	456 affirmative 70
79 competitive 262	205 etl 126	331 after 86	457 computer 70
80 her 260	206 center 126	332 verbal 86	458 diversity 70
81 clients 258	207 action/equal 126	333 baylis 86	459 caritas 69
82 support 252	208 ceop 126	334 communications 86	460 handle 69
83 tuition 252	209 join 125	335 've 85	461 mid-atlantic 69
84 seeking 247	210 assigned 125	336 etc 85	462 01/fev 69
85 programs 244	211 " 125	337 gym 85	463 pharmacovigilance 69
86 proficiency 244	212 highly 123	338 monthly 85	464 successful 68
87 there 243	213 technologist 123	339 implement 85	465 nurse 68
88 counseling 238	214 department 121	340 assignments 85	466 perform 68
89 5+ 238	215 technology 120	341 than 84	467 background 68
90 operations 237	216 assists 120	342 payroll 84	468 sql 68
91 job 237	217 location 119	343 self-motivated 83	469 hiring 68
92 proactively 236	218 coordinating 119	344 demonstrates 83	470 compensation 68
93 customer 235	219 meetings 117	345 1997 82	471 improve 67
94 powerpoint 235	220 analyzes 117	346 participate 82	472 baccalaureate 67
95 company 234	221 oversee 117	347 corporate 82	473 presentation 67
96 services 233	222 closing 117	348 only 82	474 shift 66
97 related 230	223 reporting 117	349 motivated 82	475 engineer 66
98 responsible 228	224 delivery 117	350 application 82	476 newspaper 66
99 assistant 226	225 laboratory 116	351 quarterly 82	477 cv 66
100 technical 224	226 account 116	352 performs 82	478 maintenance 66
101 our 222	227 out 116	353 exceptional 82	479 positions 66
102 benefits 219	228 quality 116	354 cgrmps 81	480 executing 66
103 but 219	229 possess 115	355 eoe 81	481 humor 65
104 harvard 216	230 senior 115	356 independently. 81	482 diagrams 65
105 `` 216	231 qualifications 114	357 nytd 81	483 at 65
106 not 216	232 when 114	358 knldg 81	484 requisition 65
107 ma 216	233 exciting 114	359 oriented 81	485 solving 65
108 analysis 215	234 bs 113	360 w/ 81	486 ark 65
109 ensure 210	235 fast-paced 113	361 associations.requirements 81	487 up 65
110 design 210	236 effectively 113	362 just 81	488 accuracy 65
111 will 208	237 ad 112	363 slcc 81	489 written 65
112 develop 207	238 schedule 111	364 dean 81	490 administration 65
113 candidates 207	239 recruiter 111	365 617-541-0203 81	491 additional 65
114 hr 207	240 education 111	366 copywriter/marketing 81	492 much 64
115 duties 202	241 administrative 110	367 must.candidate 81	493 post 64
116 fax 200	242 internal 110	368 environment.- 81	494 pediatric 64
117 plan 200	243 openings 109	369 cbar 81	495 finance 64
118 multiple 197	244 comprehensive 109	370 news.- 81	496 very 64
119 and/or 193	245 timely 109	371 category.the 81	497 cost/benefit 64
120 include 190	246 healthcare 109	372 i.e 81	498 vendors 64
121 they 189	247 processes 109	373 pl/sql 81	499 engineering 64
122 teams 189	248 lead 109	374 401k 81	500 process 64
123 years 188	249 regulatory 108	375 e.g 81	
124 role 187	250 provider 108	376 mysql 81	
125 minimum 186	251 css 108	377 3+ 81	

Fonte: Produção do próprio autor

As palavras-chave na lista aparecem em ordem de chavicidade. Tendo em vista que apenas se realizou a comparação das listas do *corpus* de referência e do *corpus* de estudo, sem o uso de uma *stoplist*, alguns caracteres de pontuação, símbolos e numerais ('-', '@', '-', ':' etc.) ainda aparecem nos resultados, os quais podem ser ignorados dados os objetivos da análise.

Além desses casos, é interessante mencionar que a ocorrência de 'it', na maioria das vezes, refere-se à abreviação 'IT' (*Information Technology*) nos textos. O pronome 'it' possui baixa frequência no *corpus*.

Mesmo assim, em sua composição geral, a lista traz as palavras de conteúdo representativas da estrutura e do tópico do gênero anúncios de emprego em um agrupamento privilegiado em relação à lista de frequência do *corpus*.

Para maximizar a representatividade das palavras-chave do *corpus*, realizou-se um experimento em que a lista de palavras por frequência nos arquivos de texto foi combinada com a lista de palavras-chave gerada. No experimento, criaram-se as funções 'keywords_get_keys_times_files' e 'keywords_get_keys_plus_files'. A primeira multiplica os valores de chavacidade da lista de palavras-chave pelo número de ocorrências dos itens nos arquivos de texto do *corpus*. A segunda soma os valores de chavacidade da lista de palavras-chave com o número de ocorrências dos itens nos arquivos de texto do *corpus*. Em seguida, comparam-se as três listas de palavras-chave. A tabela a seguir ilustra parte dos dados das três tabelas em comparação:

N	ITENS	CHAVICIDADE	ITENS	CH*FPAT	ITENS	CH+FPAT
1	--	4101	experience	201312	--	4110
2	experience	1864	and	196170	experience	1972
3	's	1773	's	138294	's	1851
4	and	1509	skills	110805	and	1639
5	skills	1335	:	100457	skills	1418
6	about.com	1085	the	90120	about.com	1097
7	nytimes.com	901	ability	56064	:	1002
8	:	889	management	50920	ability	940
9	ability	876	team	48100	nytimes.com	917
10	sales	852	--	36909	sales	883
11	@	825	with	36630	the	871
12	management	760	requirements	32538	@	859
13	the	751	work	32390	management	827
14	team	740	including	29388	team	805
15	was	686	responsibilities	29328	web	702
16	web	672	@	28050	was	696
17	-	657	development	27750	-	694
18	requirements	638	degree	26912	requirements	689
19	online	634	sales	26412	responsibilities	671
20	i	631	business	24910	online	658
21	responsibilities	624	strong	24584	i	634
22	email	591	-	24309	email	618
23	it	588	resume	23042	it	609
24	resume	562	that	21846	development	605
25	development	555	web	20160	resume	603

Figura 42 - Comparação de resultados de extração de palavras-chave

Fonte: Produção do próprio autor

Na figura, é apresentada a comparação a partir de uma amostra dos 25 primeiros itens com maior valor de chavicidade nas três listas geradas. Em sequência, da esquerda para a direita, a primeira lista possui os valores de chavicidade originais da função '*__loglikelihood*' do módulo '*keywordsextractor*'. A segunda lista traz os valores de chavicidade multiplicados pela frequência dos itens nos arquivos de textos ($CH \cdot FPAT$). A terceira lista apresenta os valores de chavicidade somados à frequência dos itens nos arquivos de texto ($CH + FPAT$). Os traços fazem a conexão dos itens encontrados nas três listas a partir da primeira. As palavras circuladas não são encontradas na segunda lista.

Na comparação, a segunda lista pareceu ser aquela que melhor maximiza a representatividade das palavras-chave no *corpus*. As palavras mais características do *corpus* tiveram sua ascensão para o topo da lista ('*experience*', '*skills*', '*ability*', '*team*', '*resume*' etc.). Palavras que podem ser consideradas específicas de determinado tópico/conteúdo dos textos ('*sales*', '*web*', '*nytimes.com*', '*it*' etc.) caíram de posição ou não apareceram na amostragem da segunda lista. A partir desses resultados, decidiu-se criar uma nova função ('*keywords_get_keys_filtered*') para extração de palavras-chave, em que palavras gramaticais, caracteres de pontuação e símbolos são removidos.

Na figura a seguir, apresenta-se um gráfico de nuvem criado a partir das 100 palavras com maior valor de chavicidade da lista filtrada:

Figura 43 - Resultado parcial de palavras-chave em gráfico de nuvem

Fonte: Produção do próprio autor

No gráfico, apenas as palavras de conteúdo são destacadas. O uso dessa função é útil para identificação do tópico ou assunto de textos.

A identificação das palavras cognatas no *corpus* foi realizada a partir da análise feita em cada texto, extraindo-se a informação da tag 'cog', com a função 'cognates_to_file'. Foram identificadas 1.805 formas como possíveis palavras cognatas. No quadro a seguir, apresentam-se as 100 palavras cognatas mais frequentes no *corpus*:

Quadro 15 - As 100 palavras cognatas mais frequentes do *corpus* de estudo

N PALAVRA FREQ	26 benefits 40	52 computer 25	78 functions 17
1 experience 247	27 operations 39	53 contact 24	79 experienced 17
2 including 100	28 opportunity 39	54 equivalent 24	80 administration 17
3 requirements 80	29 technical 39	55 internal 24	81 proficiency 17
4 company 77	30 professional 38	56 community 24	82 associate 16
5 project 76	31 department 38	57 performance 23	83 solutions 16
6 position 70	32 clients 38	58 successful 22	84 identify 16
7 services 65	33 industry 37	59 events 22	85 offer 16
8 responsibilities 65	34 strategic 37	60 director 22	86 interested 16
9 client 57	35 competitive 36	61 processes 22	87 assistance 16
10 executive 56	36 quality 36	62 flexible 21	88 strategies 16
11 excellent 53	37 direct 34	63 general 21	89 communications 16
12 communication 52	38 applications 33	64 university 20	90 problems 16
13 projects 51	39 assistant 32	65 demonstrated 20	91 relevant 16
14 candidate 50	40 technology 32	66 administrative 19	92 validation 16
15 medical 49	41 opportunities 32	67 expertise 19	93 implementation 15
16 information 49	42 description 31	68 ideal 19	94 possess 15
17 include 47	43 minimum 31	69 programme 19	95 coordinate 15
18 systems 45	44 process 31	70 advertising 19	96 initiatives 15
19 salary 45	45 candidates 31	71 national 19	97 significant 15
20 responsible 44	46 complex 30	72 contract 18	98 organizational 15
21 program 44	47 programs 28	73 organization 18	99 existing 15
22 education 44	48 human 26	74 success 18	100 multimedia 15
23 preferred 43	49 multiple 26	75 additional 18	
24 related 43	50 interpersonal 25	76 testing 17	
25 financial 42	51 individual 25	77 qualifications 17	

Fonte: Produção do próprio autor

No total, foram identificadas 35,56% formas cognatas do *corpus*. A lista pode ser analisada a fim de remover as palavras não identificadas corretamente e incluí-las nos dados do etiquetador para melhorar o desempenho das análises automáticas.

Os sufixos identificados nas palavras do *corpus*, a partir da função 'suffix_count_to_file', foram:

Tabela 8 - Sufixos identificados no *corpus* de anúncios de empregos

N	SUFIXOS	FREQ	N	SUFIXOS	FREQ
1	s	2874	16	ic	118
2	ing	1493	17	ance	115
3	ed	671	18	ee	107
4	al	581	19	ness	102
5	ation	422	20	able	94
6	ment	368	21	ition	90
7	ity	312	22	ible	71
8	ence	295	23	ive	42
9	ly	282	24	ous	40
10	y	198	25	ful	36
11	er	196	26	ize	19
12	er+s	196	27	ise	17
13	ies	179	28	ify	15
14	ion	165	29	or	13
15	ment+s	160	30	ision	10

Fonte: Produção do próprio autor

Os grupos nominais identificados foram extraídos com a função '*groups_count_groups_to_file*', passando o argumento '*I-NP*'. No quadro abaixo, listam-se os 100 grupos mais frequentes do *corpus*:

Quadro 16 - Os 100 grupos nominais mais frequentes do *corpus* de anúncios de emprego

N GRUPOS FREQ	26 sales 17	52 reports 11	78 Minimum 10
1 you 80	27 your resume 17	53 professional 11	79 services 10
2 We 77	28 Requirements 16	54 understanding 11	80 Title 10
3 experience 58	29 Work 16	55 information 11	81 communication 10
4 Experience 47	30 employee stock purchase plan 16	56 plan 11	82 New York City 10
5 Ability 36	31 development 16	57 relationships 11	83 research 9
6 a 35	32 one 15	58 online 11	84 The successful candidate 9
7 You 33	33 clients 15	59 comprehensive health coverage 11	85 they 9
8 About.com 33	34 work 15	60 Candidate 11	86 marketing 9
9 responsible 30	35 procedures 15	61 business 11	87 a top ten Web property 9
10 Responsibilities 27	36 staff 15	62 Boston 11	88 Location 9
11 the ability 25	37 this position 15	63 a company match 11	89 3 9
12 we 24	38 them 14	64 necessary 11	90 equivalent 9
13 able 23	39 the development 14	65 analysis 11	91 systems 9
14 NYTimes.com 23	40 design 14	66 Excel 11	92 products 9
15 401 23	41 MA 13	67 requirements 11	93 processes 9
16 ability 23	42 a team 13	68 activities 11	94 skills 9
17 k 22	43 The ideal candidate 13	69 Word 10	95 the Web 9
18 applications 22			96 Education 9

continua

continuação

19 knowledge 21	44 Bachelor 's degree 13	70 Responsible 10	97 benefits 9
20 the 20	45 Knowledge 12	71 data 10	98 2005 9
21 Description 19	46 support 12	72 tuition assistance 10	99 Marketing 9
22 part 19	47 us 12	73 HTML 10	100 Job Requirements 9
23 projects 18	48 communication skills 12	74 etc 10	
24 The New York Times	49 Duties 12	75 order 10	
Company 18	50 Sales 12	76 position 10	
25 management 18	51 children 12	77 it 10	

Fonte: Produção do próprio autor

No total, 6.643 grupos nominais foram identificados. Apenas 1.402 grupos ocorreram mais de uma vez. Os grupos verbais identificados foram extraídos com a função '*groups_count_groups_to_file*', passando o argumento '*I-VG*'. No quadro abaixo, listam-se os 100 grupos verbais mais frequentes do *corpus*:

Quadro 17 - Os 100 grupos verbais mais frequentes do *corpus* de anúncios de emprego

N GRUPOS FREQ	26 accounting 15	52 Must be 9	78 to meet 7
1 is 168	27 planning 15	53 develop 9	79 Collaborate 7
2 including 91	28 be 14	54 calls 9	80 requires 7
3 are 61	29 is seeking 14	55 is looking 9	81 Demonstrates 7
4 working 48	30 are looking 13	56 to support 9	82 solving 7
5 have 47	31 Develop 12	57 creating 9	83 seeking 7
6 to work 45	32 are seeking 12	58 limited 8	84 needed 7
7 managing 33	33 maintain 12	59 negotiating 8	85 associated 7
8 preferred 32	34 assist 12	60 to identify 8	86 coordinating 7
9 required 30	35 growing 12	61 assigned 8	87 email 7
10 will be 25	36 ensuring 12	62 Ensure 8	88 Be 7
11 include 24	37 using 11	63 Requires 8	89 will lead 6
12 offers 23	38 existing 11	64 is located 8	90 Maintain 6
13 to ensure 21	39 provide 10	65 to improve 8	91 Perform 6
14 to join 21	40 is required 10	66 includes 8	92 oriented 6
15 written 21	41 Manage 10	67 was acquired 8	93 to make 6
16 related 20	42 offer 10	68 will manage 8	94 ensure 6
17 paid 19	43 's 10	69 implement 8	95 awards 6
18 based 18	44 Accounting 10	70 apply 8	96 provides 6
19 has 17	45 to manage 10	71 to handle 8	97 prepare 6
20 Must have 17	46 must have 10	72 has been 7	98 breaking 6
21 developing 17	47 to detail 10	73 make 7	99 must be 6
22 manage 17	48 to communicate 9	74 to facilitate 7	100 build 6
23 will have 16	49 to develop 9	75 performing 7	
24 leading 16	50 Required 9	76 Founded 7	
25 providing 16	51 to provide 9	77 following 7	

Fonte: Produção do próprio autor

No total, 6.643 grupos verbais foram identificados. Apenas 1.402 grupos ocorreram mais de uma vez. Na contagem, também se optou por não converter as palavras em minúsculas ou maiúsculas, deixando o texto original.

As etiquetas morfossintáticas dos grupos verbais foram identificadas a partir da função '*groups_count_POS_to_file*'. O gráfico exibe as etiquetas dos 15 grupos verbais mais frequentes do *corpus* de estudo:

Figura 44 - Etiquetas morfossintáticas dos 25 grupos verbais mais frequentes

Fonte: Produção do próprio autor

No gráfico, evidencia-se que há sete categorias com grandes taxas de frequência. A maior parte dos grupos utiliza a forma nominal do verbo ('VBG', 'TO'+ 'VB' e 'VBN'), além de verbos no presente ('VBP' e 'VBZ') e coligações com modais ('MD'+ 'VB').

A partir da análise das frequências das palavras e de uma reflexão sobre os objetivos comunicativos do gênero anúncio de emprego, considerou-se a possibilidade de identificar os principais movimentos de sua estrutura interna a partir de palavras-chave. Ao analisar as palavras com maior valor de chavicidade, destacam-se: '*experience*', '*ability*', '*required*', '*position*', '*requirements*', '*candidate*', '*resume*', '*benefits*', '*salary*', '*apply*' e '*skills*'. Tais palavras podem funcionar como índices para a identificação e localização de informações no gênero.

Em relação aos objetivos comunicativos do gênero em análise, do ponto de vista do leitor, algumas informações devem ser identificadas:

- a. quem/qual empresa oferece a vaga;
- b. que tipo de vaga é oferecida;
- c. quais são as exigências e qualificações necessárias para a vaga;
- d. qual é o salário e os benefícios oferecidos;
- e. como proceder para se candidatar à vaga.

O estudo da padronização dos itens e também de seus contextos pode auxiliar na identificação de palavras e estruturas padronizadas, além de seus dados servirem para a criação de exemplos para exercícios.

Ao analisar as linhas de concordância da palavra ‘*seeking*’, como mostra a figura a seguir, percebeu-se que seu contexto traz informações sobre o tipo de profissional requerido (ex.: ‘*a talented multimedia producer*’). A palavra ‘*seeking*’ funciona como um índice para esse tipo de informação e pode ser um recurso interessante para a criação de exercícios de localização de informações.

New York Times on the Web (NYTimes.com) is	seeking	an ad trafficker/implementation specialist to join our
er) and numerous POYi and SNDies awards, is	seeking	a talented multimedia producer to create cutting-edge
er of Best News Site (2005, 2004, 2003) is	seeking	a Producer to join our online newsroom. The positions
5, 2004, 2003 (Editor & Publisher, NAA) is	seeking	a Sports Producer to join our online newsroom. The pro
ch 1.7 million worldwide visitors a day, is	seeking	a software engineer. We are looking for an enthusiast
ch 1.7 million worldwide visitors a day, is	seeking	Technical Producers to join the Web Development team,
ews Site awards in 2005, 2004 and 2003, is	seeking	a Copywriter/Marketing Manager to join our marketing t
We are	seeking	a high-performance sales executive for NYTimes.com's
ews Site awards in 2005, 2004 and 2003, is	seeking	a Client Services Director in the Ad Sales department.
We are	seeking	a high-performance sales executive for NYTimes.com's
: is part of The New York Times Company, is	seeking	a smart self-starter who is able to handle multiple t
of The New York Times Company, is actively	seeking	a Product Manager to work primarily with our sales and
of The New York Times Company, is actively	seeking	Client Services Campaign Specialist (Post-Sale).Key R
We are	seeking	:Part-time NurseGroup interviews Thursday at 2:00 pm22
l growth, CSC Consulting Group is currently	seeking	Revenue Accountants for its corp location. The Revenu
We are	seeking	an individual holding a BS/MS degree with 5 years or
ovative people and ideas. We are currently	seeking	the following executives to join our new Copley Place
PREVENTION/ STORE OPERATIONS MANAGERWe are	seeking	a highly motivated, forward thinking executive with th
loyees at higher compensation rates. We are	seeking	ambitious, sales driven professionals to join our wall
office supply, and energy industries. We are	seeking	potential candidates to represent these clients in an
rapidly growing national insurance company	seeking	a full-time Tax Accountant to assist Management with

Figura 45 - Linhas de concordância da palavra ‘*seeking*’

Fonte: Produção do próprio autor

A partir da análise da dispersão da palavra ‘*seeking*’ nos textos, é possível identificar em qual posição ela geralmente pode ser encontrada. A figura a seguir mostra o gráfico da dispersão da palavra:

Figura 46 - Dispersão da palavra ‘*seeking*’

Fonte: Produção do próprio autor

Na figura, os retângulos representam cada texto em que a palavra ocorre. As linhas verticais que cortam os retângulos representam a ocorrência da palavra. No sentido da esquerda para a direita, indica-se a posição de início e o término do texto. Os gráficos mostram que a palavra ‘*seeking*’ ocorre tipicamente logo no início do texto.

O mesmo tipo de análise pode ser executado para cada palavra considerada chave no *corpus*. Outro exemplo é a palavra ‘*salary*’, que funciona como um índice para a identificação e localização das informações sobre o salário e os benefícios oferecidos, como mostram as linhas de concordância na figura:

equivalent. We offer a highly competitive salary and an impressive array of benefits, including: comprehensive journalism. Compensation: Competitive salary commensurate with experience, bonus, 401(k), and comprehensive benefits. This position offers base salary plus sales incentive. NYTimes.com also provides comprehensive requirements to: work@about-inc.com. Local candidates and benefit package. Candidates are asked to submit a letter of interest. Send resume: 134 South Ave., Weston MA 02493 or Fax: 617-541-0203. Grade 053 Union HUETW Eligible for Overtime Date Posted: 01/10/2003 and excellent benefits. Interested candidate, please send resume and benefits package, and full reimbursement of graduate tuition. , bonus commission and benefits* A comprehensive training program is available. The hiring range is \$46,318 - \$54,492. Primary responsibilities include: sales, commission, bonus, 100% employer paid medical and dental insurance, and benefits package...and a place where you can grow and develop. Bachelor's degree is preferred. Competitive salary and commission, 401(k), co. paid employee health, life insurance, and full benefit package. Please fax resume w/salary requirements to: Bilt-Rite Construction, Inc. FAX 617-541-0203. We offer a competitive salary and full benefit package. Please fax resume w/salary requirements to: Bilt-Rite Construction, Inc. FAX 617-541-0203. We offer a competitive salary and full benefit package. Please fax resume w/salary requirements to: Bilt-Rite Construction, Inc. FAX 617-541-0203.

Figura 47 - Linhas de concordância da palavra ‘salary’

Fonte: Produção do próprio autor

Em relação à dispersão da palavra ‘salary’, diferentemente de ‘seeking’, sua ocorrência é marcada no final do texto, como mostram os gráficos de dispersão na figura abaixo:

Figura 48 - Dispersão da palavra ‘salary’

Fonte: Produção do próprio autor

Para o módulo de análise, inclui-se a informação da localização de cada item do texto a partir de uma *tag* ('loc'), como mostra a figura parcial do arquivo *XML* de um texto analisado:

```

key="2" loc="0.65" pos="PRP" size="2" stem="We" suf="-">We</token>
key="6" loc="0.97" pos="VBP" size="3" stem="are" suf="-">are</token>
key="24" loc="1.3" pos="VBG" size="7" stem="seek" suf="ing">seeking</token>
key="2" loc="1.62" pos="DT" size="1" stem="a" suf="-">a</token>
key="42" loc="1.95" pos="JJ" size="16" stem="high-perform" suf="-">high-performance</token>
key="124" loc="2.27" pos="NNS" size="5" stem="sale" suf="s">sales</token>
key="34" loc="2.6" pos="NN" size="9" stem="execut" suf="ive">executive</token>
key="0" loc="2.92" pos="IN" size="3" stem="for" suf="-">for</token>
key="137" loc="3.25" pos="NNP" size="11" stem="NYTimes.com" suf="-">NYTimes.com</token>
key="43" loc="3.57" pos="POS" size="2" stem="'s" suf="-">'s</token>
key="21" loc="3.9" pos="NNP" size="6" stem="Studio" suf="-">Studio</token>
key="46" loc="4.22" pos="NNP" size="13" stem="Entertain" suf="-">Entertainment</token>
key="59" loc="4.55" pos="NN" size="8" stem="categori" suf="-">category</token>

```

Figura 49 - Exemplo de anotação para registro da posição de itens

Fonte: Produção do próprio autor

Tendo em vista a observação de que as palavras possuem preferências para ocorrer em determinadas partes do texto e de que algumas podem estar diretamente relacionadas aos objetivos comunicativos do gênero textual do *corpus* de estudo em análise, decidiu-se criar uma função para agrupar tais palavras a fim de verificar se os agrupamentos revelam os movimentos e submovimentos do gênero.

Para executar tal tarefa, primeiro, criou-se uma função que mapeia as ocorrências das palavras-chave por faixas no texto. Tendo como entrada uma determinada palavra/item do *corpus*, a função funciona da seguinte forma: a. cria uma lista com a posição de cada ocorrência da palavra no *corpus*; b. conta as posições da lista a partir de um filtro que divide o espaço do texto em 10 faixas e classifica as posições, criando uma lista de faixas de frequência das posições; c. transforma a frequência das faixas em porcentagem a partir do número total de ocorrências da palavra. A tabela a seguir exhibe os resultados para as 15 palavras-chave do *corpus*:

Tabela 9 - Dispersão de 15 palavras do *corpus* de estudo em faixas

N	ITENS	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
1	experience	1,62	2,83	2,02	5,26	8,5	10,53	19,03	21,05	13,36	6,88
2	and	5,28	8,83	9,23	10,55	11,62	10,2	9,74	9,79	7,41	6,34
3	's	18	9,33	9,33	9,33	11,33	6	8	8,67	8,67	2,67
4	skills	2,58	0,65	2,58	3,87	7,74	12,26	10,32	14,19	18,71	13,55
5	:	23,23	8,56	5,62	4,89	4,89	7,82	4,89	6,85	10,02	15,4
6	the	11,86	12,9	13,75	10,17	8,29	8,38	6,97	6,21	5,84	6,21
7	ability	0,88	0,88	2,65	2,65	4,42	11,5	15,93	22,12	23,01	6,19
8	management	6,06	5,3	5,3	10,61	13,64	13,64	9,09	12,12	9,09	9,85
9	team	8,94	10,57	13,01	8,13	9,76	8,94	8,13	8,94	6,5	8,13
10	--	0	0	3,76	19,35	3,76	4,3	24,19	7,53	8,06	20,43
11	with	5,7	8,44	9,07	12,87	10,55	12,03	11,39	9,7	5,91	6,12
12	requirements	2,5	12,5	2,5	11,25	10	11,25	11,25	12,5	5	7,5
13	work	3,75	8,75	7,5	6,88	11,25	13,75	9,38	7,5	14,37	6,25
14	including	3	6	10	12	15	12	6	6	8	13
15	responsibilities	16,92	13,85	21,54	7,69	10,77	9,23	1,54	3,08	1,54	1,54

Fonte: Produção do próprio autor

Na tabela, a dispersão das frequências das palavras é dada em 10 faixas. Os valores significam a porcentagem de ocorrência da palavra em cada faixa. A partir dos dados, é possível verificar em qual posição do texto as palavras tipicamente ocorrem. Por exemplo, a palavra 'experience' tem picos de frequência entre as faixas 'F6' e 'F9', o que sugere uma preferência de ocorrência do meio para o final do texto, sendo pouco frequente no início do texto, no gênero anúncios de emprego do *corpus* de estudo.

Os dados de dispersão, como os da tabela, também possibilitam a comparação entre palavras, como mostra o gráfico:

Figura 50 - Dispersão de palavras em faixas para comparação

Fonte: Produção do próprio autor

No gráfico, como exemplo, a dispersão das palavras '*salary*' e '*seeking*' evidencia o contraste das ocorrências, em que a primeira ocorre notadamente no início e no final do texto, enquanto que a segunda ocorre muito mais no início. Pode-se presumir que a informação sobre o salário é tipicamente encontrada ou no início, junto com as especificações da vaga, ou no final, com as informações de benefícios e contato. Já o verbo '*seeking*' mostra a predominância do que se procura logo no início do anúncio. Tais informações são interessantes para habilidades de leitura, na localização de informações, e para habilidades de escrita.

Tendo a dispersão das palavras mapeada, o passo seguinte foi a criação de uma função que as agrupa de acordo com os dados das faixas. Utilizou-se o algoritmo de aprendizado não supervisionado do classificador '*KMeansClusterer*', disponível no *NLTK*, para realizar os agrupamentos. Primeiro, calcula-se a dispersão para as palavras do *corpus*. Em seguida, os dados das faixas (atributos utilizados) são transformados em vetores. Por último, passam-se os vetores para o algoritmo de agrupamento. O valor de '*k*' definido foi de 6, um a mais em relação aos objetivos comunicativos do gênero citados.

No experimento, com a função '*dispersion_group_to_file*', utilizaram-se as 2.500 palavras com maior valor de chavicidade para o processamento. O algoritmo agrupou as palavras com uma numeração de zero a cinco. O quadro a seguir mostra o resultado parcial da saída da função:

Quadro 18 - Resultado de agrupamento a partir da dispersão das palavras

N ITENS GRUPO	26 required 1	52 related 1	78 2014 5
1 experience 1	27 years 1	53 employee 5	79 excel 1
2 's 1	28 candidate 1	54 services 1	80 professional 1
3 skills 1	29 manage 1	55 data 1	81 clients 1
4 ability 1	30 position 1	56 responsible 1	82 ma 5
5 management 1	31 about.com 1	57 include 1	83 customer 1
6 team 1	32 working 1	58 plan 5	84 minimum 1
7 requirements 1	33 support 1	59 ensure 1	85 applications 1
8 work 1	34 project 1	60 marketing 1	86 duties 1
9 including 1	35 environment 1	61 role 1	87 staff 1
10 responsibilities 1	36 manager 0	62 benefits 1	88 apply 1
11 development 1	37 preferred 1	63 medical 1	89 accounting 1
12 degree 1	38 projects 1	64 executive 0	90 health 1

continua

continuação

13 sales 1	39 401 5	65 operations 1	91 design 1
14 business 1	40 job 1	66 assist 1	92 understanding 1
15 strong 1	41 new 1	67 product 1	93 and/or 1
16 resume 5	42 company 1	68 develop 1	94 teams 1
17 web 1	43 bachelor 1	69 strategic 1	95 must 1
18 salary 1	44 client 1	70 boston 1	96 multiple 1
19 email 5	45 seeking 0	71 description 0	97 proficiency 1
20 please 5	46 program 1	72 2005 1	98 systems 1
21 online 1	47 managing 1	73 technical 1	99 website 1
22 excellent 1	48 office 1	74 tuition 5	100 key 1
23 communication 1	49 competitive 1	75 opportunity 1	
24 knowledge 1	50 york 1	76 candidates 1	
25 nytimes.com 1	51 interpersonal 1	77 analysis 1	

Fonte: Produção do próprio autor

O quadro mostra o resultado para as 100 palavras com maior valor de chavacidade do *corpus*. Conforme os dados, para obter os agrupamentos, foi necessário apenas juntar as palavras com valores semelhantes.

Para ilustrar os agrupamentos, criou-se um gráfico de nuvem para cada grupo a partir do valor de chavacidade das palavras. Os gráficos são apresentados em miniatura na figura a seguir:

Figura 51 - Gráficos de nuvem para cada agrupamento de palavras

Fonte: Produção do próprio autor

Nos gráficos, em visão geral, ficaram evidentes alguns agrupamentos. Por exemplo, no grupo '0', palavras relacionadas à empresa, que tipo de vaga é oferecida e que tipo de profissional se busca. No grupo '1', palavras relacionadas à experiência do profissional. Os grupos '2', '3' e '4'

evidenciam as habilidades e competências requeridas. O grupo '5' apresenta palavras relacionadas aos benefícios e a como proceder para se candidatar à vaga.

Esses resultados indicaram a viabilidade de criar um classificador para identificar os movimentos do gênero a partir dos dados de agrupamento. Para textos em que o gênero não possui análises no sistema, é possível o uso dos agrupamentos para que o próprio usuário os classifique.

As análises do uso das palavras do *corpus* foi realizada a partir da extração de colocados e n-gramas, a fim de verificar a padronização de itens e a extração de informações para criação de exercícios. Para tanto, utilizaram-se fórmulas de associação entre palavras, como *t-score* e *mutual information* (informação mútua). As análises levaram em consideração as 500 palavras mais frequentes do *corpus* de estudo. Os resultados de colocação dessas palavras foram salvos para compor exercícios para o ensino da padronização das palavras.

3.4.3 Composição do *corpus* de textos de divulgação científica

Quadro 19 - Composição do *corpus* de textos de divulgação científica

Arquivos	Itens	formas	Razão item/forma	Hápax	Parágrafos	Sentenças
771	575.713	30.328	5%	12.879	8.219	23.907

Fonte: Produção do próprio autor

No quadro abaixo, apresentam-se os 100 itens do *corpus* com maior valor de chavicidade:

Quadro 20 - Os 100 itens com maior valor de chavicidade do *corpus* de divulgação científica

N ITENS CHAVICIDADE	26 published 153560	52 lab 93196	78 including 54934
1 's 35701155	27 online 150183	53 behavior 89964	79 'll 53838
2 says 4615680	28 california 147465	54 gas 88350	80 journal 53445
3 n't 4053450	29 climate 146839	55 planets 88220	81 immune 53152
4 researchers 2443819	30 2010 145824	56 kilometers 84901	82 medicine 51119
5 university 1903396	31 according 144265	57 cancer 84632	83 effects 50864
6 u.s. 1668864	32 2011 138450	58 findings 83974	84 2007 50103
7 percent 990380	33 years 137973	59 scientific 82611	85 likely 49848
8 study 777218	34 've 136350	60 power 81855	86 electricity 49300
9 scientists 714432	35 planet 135135	61 drug 76540	87 patients 48168

continua

continuação

10 're 711490	36 found 133722	62 medical 69296	88 fuel 47594
11 research 623784	37 could 132132	63 carbon 69190	89 help 46592
12 brain 566679	38 also 129792	64 using 67626	90 levels 46375
13 center 513992	39 studies 129429	65 meters 67270	91 wind 46248
14 science 480564	40 system 128763	66 know 66564	92 professor 46170
15 cells 477520	41 team 126654	67 scientist 65805	93 chemical 45849
16 energy 395012	42 space 123684	68 global 65007	94 american 45760
17 solar 381358	43 known 122010	69 roughly 64842	95 body 45752
18 new 379776	44 nasa 120060	70 genetic 64824	96 particles 45414
19 technology 323741	45 humans 114296	71 even 63705	97 surface 45200
20 colleagues 247863	46 cell 107635	72 light 63560	98 risk 44928
21 earth 201344	47 disease 105455	73 said 63180	99 world 44000
22 2009 195930	48 water 101943	74 might 61953	100 planetary 43952
23 institute 191287	49 adds 100875	75 potential 60420	
24 human 188139	50 health 97750	76 2008 59902	
25 data 180868	51 internet 95392	77 notes 57288	

Fonte: Produção do próprio autor

No quadro, é possível notar que grande parte dos itens lexicais estão relacionados às temáticas dos textos. Nas análises, percebeu-se que há maior facilidade em identificar os itens lexicais que estão específicos do gênero no *corpus* de anúncios a partir das palavras-chave. Tais diferenças podem levar a diferentes estratégias de análise em relação aos *corpora* de estudo.

3.5 Sistema de criação automática de atividades

Nesta seção, descreve-se a arquitetura do sistema de criação automática de atividades, as principais funções de seus módulos e a interface com o usuário.

3.5.1 Componentes para o desenvolvimento do sistema

Para o desenvolvimento do sistema de criação automática de atividades, foi necessário juntar os seguintes principais componentes e recursos:

- um módulo que reúne um conjunto de módulos de análises de texto;
- um módulo de análise de *corpora* a partir dos módulos de análises de texto;
- um módulo leitor e extrator de informações de textos analisados em formato *XML*;
- funções de extração de informações de *corpora* analisados;

- e. um módulo interpretador de *scripts* de uma metalinguagem criada para montagem de atividades a partir de informações extraídas dos textos analisados;
- f. um conjunto de módulos para criação de visualizações de informações de análises de texto;
- g. um conjunto de *scripts* em linguagem *PHP*, *HTML*, *CSS* e *JavaScript* para a criação da interface com o usuário e comunicação com os módulos em Python via *Asynchronous Javascript and XML (AJAX)*;
- h. um criador gratuito de *Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA)* para evitar abusos e ataques à página do *site* de criação de atividades;
- i. uma plataforma de editor de textos e *HTML* gratuita para realizar a edição dos resultados de geração de atividades;
- j. um conversor gratuito de *HTML* para *PDF* de terceiros, escrito em *PHP*, para a publicação final das atividades criadas.

Os componentes e recursos listados compreendem o conjunto necessário para a implementação do sistema e disponibilidade *online* para uso dos primeiros usuários em sua avaliação.

A descrição do funcionamento de tais componentes e recursos e a sua relação com o sistema construído são fornecidas nas seções seguintes.

3.5.2 Módulo readler

O módulo readler foi criado para reunir todos os módulos de análise de texto e *corpora*, a fim de propiciar o acesso fácil às diversas funções do sistema de criação. As duas principais funções exclusivas do módulo são: *'text_analyze(raw_text_in_xml, language, genre)'* e *'xml_text_save(xmltext,path)'*.

A função *'text_analyze'* é responsável por realizar as análises de texto. Os argumentos passados são: o texto em formato *XML* (com a divisão de título, subtítulo, autor/fonte e corpo do texto), o idioma referente ao texto de entrada e a especificação do gênero do texto a ser analisado. O processamento do texto é realizado a partir da instanciação de todos os módulos de análise disponíveis no diretório *'analysis'* e sua aplicação em diversos estágios de análise e

montagem da estrutura das anotações em *XML*. A estrutura de anotação em *XML*, em seu primeiro nível, possui as seções: *genre*, *statistics*, *text_title*, *text_subtitle*, *text_author* e *text_body*.

A estrutura para a informação de identificação de gênero é dada pela tag '*<genre>*'. A estrutura para as informações estatísticas do texto é dada pela tag '*<statistics>*' e compreende as tags filhas:

- a. *<tokens>* - o número de itens do texto;
- b. *<types>* - o número de formas do texto;
- c. *<typetokenratio>* - a razão forma/item;
- d. *<paragraphs>* - o número de parágrafos do texto;
- e. *<sentences>* - o número de sentenças do texto;
- f. *<words>* - o número de palavras do texto (não inclui pontuações ou símbolos);
- g. *<averagewordspersentence>* - o número médio de palavras por sentença;
- h. *<numberofcharacters>* - o número total de caracteres do texto;
- i. *<syllables>* - o número total de sílabas do texto³¹;
- j. *<averagesyllablesperword>* - o número médio de sílabas por palavras;
- k. *<fleschreadingease>* - índice de legibilidade da fórmula de *Flesch*³²;
- l. *<fleschkincaid>* - índice de legibilidade da fórmula de *Flesch-Kincaid*;
- m. *<cognates>* - número total de possíveis palavras cognatas do texto;
- n. *<cognatesratio>* - porcentagem de possíveis palavras cognatas do texto.

As estruturas para as análises do corpo do texto são as tags: '*<paragraph>*', '*<sentence>*' e '*<token>*'. A tag '*<token>*' possui as seguintes propriedades:

- a. *cog* - indica se a palavra é cognata (valor '1') ou não (valor '0');
- b. *freq* - indica a frequência do item no texto;
- c. *grp* - indica a que grupo o item pertence: 'O' (fora e classificação), 'I-NP' (grupo nominal) e 'I-VG' (grupo verbal);
- d. *id* - indica o índice da ordenação do item no texto;
- e. *key* - indica o valor de chavidade do item no texto;
- f. *loc* - indica o valor em porcentagem da posição do item no texto (dispersão);

³¹ O número de sílabas é calculado a partir de uma função (contribuições em lista de discussão) para o *NLTK*.

³² Teste utilizado para mensurar a dificuldade de entendimento de um texto em língua inglesa.

- g. *pos* - indica a etiqueta morfossintática do item;
- h. *size* - indica o tamanho do item em caracteres;
- i. *stem* - indica a forma estemizada do item;
- j. *suf* - indica quais sufixos a palavra possui;

O texto do item é representado entre as *tags* como no exemplo de item anotado: `<token cog="0" freq="2" grp="I-NP" id="3" key="19" loc="2.7" pos="NNP" size="3" stem="New" suf="-">New</token>`.

Após o término das análises, a função '*text_analyze*' retorna um objeto '*ElementTree*'. A função '*xml_text_save*', então, é utilizada para converter o objeto em *string* e armazenar o texto analisado em *XML* no banco de dados para futuros processamentos.

3.5.3 Módulo *xmltextreader*

O módulo '*xmltextreader*' foi criado para fazer a leitura dos arquivos *XML* de textos analisados e retornar informações requeridas.

As principais funções do módulo são:

- a. *read_from_file(filename)* - faz a leitura do código *XML* a partir de um arquivo. A função cria internamente duas variáveis para armazenar o objeto '*ElementTree*()' e sua raiz: '*self.__tree*' e '*self.__root*'. Tem como argumento o caminho do arquivo.
- b. *read_from_string(s)* - realiza a mesma tarefa da função '*read_from_file*' a partir de uma *string* com código *XML*;
- c. *read_from_root(root)* - realiza a mesma tarefa da função '*read_from_file*' a partir de um objeto
- d. *statistics_number_of_tokens()* - retorna o número de itens do texto analisado;
- e. *statistics_number_of_types()* - retorna o número de formas;
- f. *statistics_type_token_ratio()* - retorna o número da razão forma/item;
- g. *statistics_number_of_paragraphs()* - retorna o número de parágrafos do texto;
- h. *statistics_number_of_sentences()* - retorna o número de sentenças do texto;
- i. *statistics_number_of_words()* - retorna o número efetivo de palavras do texto;

- j. *statistics_average_words_per_sentence()* - retorna o número médio de itens por sentença;
- k. *statistics_flesch_reading_ease()* - retorna o número do índice de legibilidade da fórmula de *Flesch*;
- l. *statistics_flesch_reading_ease_label()* - retorna a interpretação do índice de legibilidade da fórmula de *Flesch* em: 'Very confusing', 'Difficult', 'Fairly Difficult', 'Standard', 'Fairly easy', 'Easy' e 'Very easy';
- m. *statistics_number_of_cognates()* - retorna o número de palavras cognatas identificadas;
- n. *statistics_lexical_density()* - retorna a densidade lexical do texto;
- o. *genre_category_get()* - retorna o rótulo de gênero do texto;
- p. *title_get()* - retorna o título do texto;
- q. *subtitle_get()* - retorna o subtítulo do texto;
- r. *author_get()* - retorna o autor ou a fonte do texto;
- s. *filter_tokens(args,token_history=[])* - retorna um conjunto de itens do texto a partir de uma série de condições passadas por argumentos:
 - limit* - limita o número de itens a ser retornado;
 - unique* - determina se os itens retornados podem ou não ser duplicados a partir dos valores 'yes' ou 'no';
 - minsize* - determina o tamanho mínimo do item em caracteres;
 - maxsize* - determina o tamanho máximo do item em caracteres;
 - stoplist* - uma lista de itens que não podem ser incluídos no conjunto a ser retornado;
 - pos* - uma lista que determina quais etiquetas morfossintáticas de itens são aceitas para inclusão no conjunto a ser retornado;
 - stoppos* - uma lista que determina quais etiquetas morfossintáticas de itens não podem ser aceitas para a inclusão no conjunto a ser retornado;
 - punct* - determina se caracteres de pontuação podem ser retornados como itens a partir dos valores 'yes' ou 'no';
 - cognate* - determina a inclusão de palavras cognatas a partir dos valores 'yes' ou 'no';
 - minfreq* - determina a frequência mínima do item a ser incluído;
 - maxfreq* - determina a frequência máxima do item a ser incluído;
 - regex* - determina um padrão de expressão regular para a aceitação de itens a serem retornados;

- suffix* - determina o sufixo do item a ser validado para inclusão de itens a serem retornados;
- min_position* - determina a posição mínima do item no texto para a aceitação de itens a serem retornados;
- max_position* - determina a posição máxima do item no texto para a aceitação de itens a serem retornados;
- repeat* - determina se itens já retornados em execuções anteriores da mesma função podem se repetir a partir dos valores 'yes' ou 'no';
- word* - determina a inclusão de uma palavra específica;
- order* - determina a ordem em que os itens são retornados: *keyness*, *keyness-asc*, *freq-asc*, *freq-desc*, *word-asc*, *word-desc*, *position-asc*, *position-desc* e *random* ou *shuffle*.

3.5.4 Módulo *xcorpusanalyser*

O módulo '*xcorpusanalyser*' foi criado para fazer a leitura de um conjunto de arquivos *XML* de textos analisados e extrair informações.

Para a criação de atividades, o módulo possui funções para acesso aos dados dos *corpora* do sistema.

Uma das principais funções do módulo é a função '*concordance_get(word, pos_template, trigram, limit, blank)*'. A função faz uma busca por uma determinada palavra no *corpus* selecionado e retorna as sentenças em que ela ocorre conforme as condições dos argumentos. Os argumentos passados são:

- a. *word* - é palavra de busca ou nóculo da concordância;
- b. *pos_template* - é uma restrição da busca para retornar uma determinada coligação em que o nóculo está inserido. A restrição considera um horizonte de duas etiquetas anteriores e duas etiquetas posteriores à palavra de busca (*T₂*, *T₁*, *T*, *T₁* e *T₂*). O formato do argumento deve ser passado com o modelo de três posições do horizonte, seguido de uma barra '/' e as etiquetas do modelo especificado (Ex.: '*T₁ITT1/DTJJNN*'). Para ignorar o argumento, passa-se o valor '*None*';

- c. *trigram* - é uma restrição de busca para retornar apenas as ocorrências de um trigrama especificado. A restrição considera três modelos/casos de trigramas, em que a posição da palavra de busca ou nóculo se alterna: *NWW*, *WNW* e *WWN*. O formato do argumento é o modelo do trigrama, seguido de uma barra '/' e as palavras do trigrama especificado (Ex.: '*WNW/ofexperiencein*'). Para ignorar o argumento, passa-se o valor '*None*';
- d. *limit* - é o número máximo de sentenças a ser retornado;
- e. *blank* - se ativado com o valor '1', retorna a sentença sem a palavra de busca, isto é, converte a palavra de busca em uma lacuna '____'.

Para exemplificar o funcionamento da função, apresentam-se os seguintes resultados a partir de buscas com a palavra '*experience*', realizadas no *corpus* de estudo:

Quadro 21 - Exemplos de resultados da função *concordance_get*

Argumentos	Resultado
word = 'experience' pos_template = None trigram = 'WNW/ofexperiencein' limit = 1 blank = 0	<p><p>Candidates with 2+ years of experience in a similar position are encouraged to apply .</p></p>
word = 'experience' pos_template = None trigram = 'WNW/ofexperiencein' limit = 1 blank = 1	<p><p>Candidates with 2+ years of ____ in a similar position are encouraged to apply .</p></p>
word = 'experience' pos_template = 'T_2T_1T/NNSINNN' trigram = None limit = 1 blank = 0	<p><p>Candidates with 2+ years of experience in a similar position are encouraged to apply .</p></p>
word = 'experience' pos_template = 'T_2T_1T/NNSINNN' trigram = None limit = 1 blank = 1	<p><p>Candidates with 2+ years of ____ in a similar position are encouraged to apply .</p></p>

Fonte: Produção do próprio autor

Os resultados são retornados com a formatação de códigos *HTML*, uma vez que farão parte da criação de atividades.

3.5.5 Tags em XML para a criação de modelos de atividades

Para permitir a criação de modelos de atividades, foram criadas *tags* personalizadas em XML. O uso de *tags* personalizadas facilita a criação dos modelos, além de conferir flexibilidade e reuso de funções, sem haver a necessidade de alterar o código a cada modificação ou inclusão de modelos de atividades. As principais *tags* criadas são apresentadas no quadro a seguir:

Quadro 22 - *Tags* disponíveis para criação de modelos de atividades

Tag	Propriedades	Descrição
<code><line></code>	Nenhuma.	Definida para representar uma nova linha. É substituída por ' <code>
</code> ' em HTML.
<code><label></code>	Nenhuma.	Definida para imprimir um texto normal que não necessite de processamento ou item extraído do texto analisado. É geralmente utilizada para imprimir as instruções nos modelos de atividades.
<code><tokens></code>	<i>unique</i> (yes ou no), <i>minsize</i> , <i>maxsize</i> , <i>stoplist</i> , <i>pos</i> , <i>stoppos</i> , <i>punct</i> (yes/no), <i>cognate</i> (yes ou no), <i>minfreq</i> , <i>maxfreq</i> , <i>regexp</i> , <i>suffix</i> , <i>min_position</i> , <i>max_position</i> , <i>repeat</i> (yes ou no), <i>word</i> , <i>order</i> (<i>keyness</i> , <i>keyness-asc</i> , <i>freq-asc</i> , <i>freq-desc</i> , <i>word-asc</i> , <i>word-desc</i> , <i>position-asc</i> , <i>position-desc</i> e <i>random</i> ou <i>shuffle</i>).	Definida para retornar um conjunto de itens do texto analisado a partir de uma série de condições, que podem ser configuradas a partir das propriedades da <i>tag</i> .

continua

continuação

Tag	Propriedades	Descrição
<code><wordcloud_frequency></code>	Nenhuma.	Retorna uma nuvem de palavras feita a partir da lista de frequência das palavras do texto. Cada palavra na nuvem possui o tamanho proporcional a sua frequência no texto.
<code><wordcloud_keywords></code>	Nenhuma.	Retorna uma nuvem de palavras feita a partir da lista de palavras-chave do texto. Cada palavra na nuvem possui o tamanho proporcional a sua chavacidade no texto.
<code><paragraphs_keywords></code>	<i>pos, identpar (yes ou no), group, group_tag e delimiter.</i>	Retorna a palavra-chave com maior valor de chavacidade de cada parágrafo. Se a propriedade <i>group</i> for especificada, serão retornados os grupos nominais ou verbais das palavras-chave.
<code><concordance></code>	<i>word, pos_template, trigram, limit, blank</i>	Retorna a(s) sentença(s) em que a palavra de busca ou nóculo ocorre no <i>corpus</i> selecionado, conforme as condições dos argumentos.
<code><groups></code>	<i>group_tag, limit, unique, maxsize, minsize, delimiter, min_position, max_position</i>	Retorna um grupo verbal ou nominal conforme as condições dos argumentos.
<code><line></code>	Nenhuma.	Definida para representar uma nova linha. É substituída por ' <code>
</code> ' em <i>HTML</i> .

Fonte: Produção do próprio autor

As *tags* são combinadas para formar diversos modelos de atividades, que são armazenados no diretório '*activities/templates*' do sistema. Um conjunto de modelos de atividades pode formar uma unidade didática. A interpretação dos modelos de atividades é feita por um módulo específico: '*xinterpreter*'.

3.5.6 Módulo *xinterpreter*

Para a interpretação dos modelos de atividades foi criado o módulo '*xinterpreter*'. O módulo trabalha em conjunto com os módulos '*xmltextreader*' e '*xcorpusanalyser*'. Sua principal função é '*translate(xmltext,xmltemplate)*', que converte um modelo de atividade em *XML* em texto com códigos *HTML*. A figura a seguir apresenta um exemplo de modelo de atividade:

```
<activity>
  <label>Verifique o contexto gramatical das seguintes palavras no texto: </label>
  <label>[i]</label>
  <tokens limit='5' pos='JJ' minsize='5' case='lower' repeat='no' delimiter='; ' ></tokens>
  <label>[/i]</label>
  <line></line><line></line>
  <label>Responda conforme sua observação:</label>
  <line></line>
  <label>a. Qual a classe gramatical das palavras observadas?</label><line></line>
  <label>b. Geralmente, elas estão acompanhadas de que tipo de palavras?</label><line></line>
  <label>c. Quais outras palavras do mesmo tipo você consegue identificar no texto?</label>
  <line></line>
  <line></line>
</activity>
```

Figura 52 - Exemplo de modelo de atividade em *XML*

Fonte: Produção do próprio autor

Os modelos de atividades podem apresentar textos simples, chamados de rótulos ('<label>'), *tags* relacionadas a códigos de formatação de texto *HTML* ([i],[b],[br] etc.) e *tags* que extraem informações dos textos analisados ('<token>').

No exemplo, a linha de código '<tokens limit='5' pos='JJ' minsize='5' case='lower' repeat='no' delimiter='; ' ></tokens>' possui a seguinte instrução: retorne um limite de cinco itens que sejam adjetivos, com tamanho mínimo de cinco caracteres, em caixa baixa, sem repetições, delimitados pelo caractere ';'.

Para realizar a interpretação ou tradução do modelo, é necessário passar como argumentos o texto analisado em formato *XML* e o modelo de atividade.

Para tanto, o módulo '*xinterpreter*' faz a leitura de todas as *tags* do modelo de atividade e, para cada uma, executa a instrução específica relacionada à *tag*, conforme suas propriedades. Por exemplo, quando o interpretador reconhece a *tag* '<token>', identifica seus argumentos e extrai as informações a partir de outro módulo a que pertence tal *tag*, no caso, '*xmltextreader*'.

A figura a seguir exibe a tradução do modelo de atividade em *XML* da figura 73:

Verifique o contexto gramatical das seguintes palavras no texto: *mounting; vulnerable; negative; cocaine; daily;*

Responda conforme sua observação:

- Qual a classe gramatical das palavras observadas?
- Geralmente, elas estão acompanhadas de que tipo de palavras?
- Quais outras palavras do mesmo tipo você consegue identificar no texto?

Figura 53 - Exemplo de tradução de modelo de atividade

Fonte: Produção do próprio autor

No exemplo, apresenta-se o modelo de atividade já traduzido e formatado em visualização de página *HTML*.

O módulo '*xinterpreter*' é capaz de interpretar vários modelos passados de uma só vez, contanto que cada modelo esteja bem delimitado, com as *tags* adequadas, caso contrário erros podem ser gerados. Por tal motivo, a escrita dos modelos de atividades teve de ser realizada com bastante atenção para que erros fossem evitados durante a execução. Cada modelo foi salvo em um arquivo diferente no diretório '*templates*'.

3.5.7 Modelos de atividades

Nesta seção, apresentam-se alguns dos modelos de atividades elaborados para o sistema de criação de atividades que extraem informações do texto de entrada. Modelos que não extraem informações de texto e *corpus* também foram elaborados. Tais modelos geralmente referem-se a perguntas gerais sobre o texto. A elaboração dos modelos de atividades de leitura é baseada em livros de ensino de leitura instrumental e experiência do pesquisador enquanto professor de língua estrangeira. Os códigos de todos os modelos são encontrados nos anexos do trabalho.

Os exemplos apresentados a seguir ilustram o funcionamento das *tags* criadas para o desenvolvimento dos modelos de atividades.

Modelo de atividade *pt_affixes_a*:

Quadro 23 - Código XML do modelo de atividade *pt_affixes_a*

```
<activity>
  <label>Quais palavras são compostas por uma raiz mais afixo (prefixo ou sufixo):</label>
  <line></line> <line></line>
  <label>[i]</label>
  <tokens suffix='ed' limit='1' pos='VB VBP VBZ VBN' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='ise' limit='1' pos='VB VBP VBZ VBN' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='y' limit='1' pos='JJ' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='ly' limit='2' pos='RB' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='ing' limit='1' pos='VBN JJ NN' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='ation' limit='1' pos='NN' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='al' limit='1' pos='NN' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='tion' limit='1' pos='NN' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='ation' limit='1' pos='NN' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='ment' limit='1' pos='NN' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='ship' limit='1' pos='NN' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='able' limit='1' pos='JJ' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='ful' limit='1' pos='JJ' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <tokens suffix='al' limit='1' pos='JJ' minsize='5' case='lower' repeat='no' delimiter='[br]' ></tokens>
  <label>[/i]</label>
  <line></line> <line></line>
</activity>
```

Fonte: Produção do próprio autor

No modelo *pt_affixes_a*, utilizou-se a tag '*tokens*' para extrair os itens do texto. Em suas propriedades, definiu-se '*limit*' para retornar apenas um item de cada vez, '*pos*' para variar as classes de palavras retornadas, '*repeat*' para não repetir itens, '*case*' para converter todos os itens em caixa baixa e '*delimiter*' para que cada item esteja em uma linha.

Modelo de atividade *pt_cognates_identification_a*:

Quadro 24 - Código XML do modelo de atividade *pt_cognates_identification_a*

```
<activity>
  <label>Qual conjunto possui o maior número de palavras cognatas do texto?</label>
  <line></line><line></line>
  <label>[i]</label>
  <label>( )</label><tokens limit='8' cognate='yes' pos='NN' minsize='5' case='lower' repeat='no' delimiter=';' >
```

continua

continuação

```

</tokens><line></line>
  <label>( )</label><tokens limit='8' stoppos='NN NNS' cognate='no' maxsize='7' case='lower' order='shuffle' repeat='no'
delimiter=';' '></tokens><line></line>
  <label>( )</label><tokens limit='8' stoppos='NN NNS' cognate='no' maxsize='6' case='lower' order='shuffle' repeat='no'
delimiter=';' '></tokens><line></line>
  <label>( )</label><tokens limit='8' stoppos='NN NNS' cognate='no' maxsize='7' case='lower' order='shuffle' repeat='no'
delimiter=';' '></tokens><line></line>
  <label>[/i]</label>
  <line></line>
</activity>

```

Fonte: Produção do próprio autor

No modelo *pt_cognates_identification_a*, utilizou-se a tag '*tokens*' para extrair os itens do texto. Para cada conjunto de itens, as propriedades '*repeat*', '*case*', '*pos*' e '*delimiter*' foram definidas para não retornarem itens duplicados, deixar todos os itens em caixa baixa, restringir os itens a substantivos e delimitá-los com o caractere ';'. Para o conjunto que possui o maior número de palavras cognatas, definiu-se '*yes*' para a propriedade '*cognate*' e o valor de '5' para '*minsize*', definindo tamanho mínimo da palavra candidata a cognata a ser retornada. Para os conjuntos negativos, definiu-se a propriedade '*stoppos*' para não retornar substantivos e o valor máximo de '6' ou '7' caracteres para os itens. A propriedade '*order*' foi definida como '*shuffle*' para embaralhar a ordem dos itens retornados.

Modelo de atividade *pt_grammar_pos_a*:

Quadro 25 - Código XML do modelo de atividade *pt_grammar_pos_a*

```

<activity>
  <label>Busque as palavras abaixo no texto e as classifique em: adjetivo, substantivo ou verbo.</label>
  <line></line><line></line>
  <label>a) </label><tokens limit='1' pos='NN NNS' minsize='5' case='lower' repeat='no' delimiter=';' '></tokens>
  <line></line>
  <label>b) </label><tokens limit='1' pos='JJ' minsize='5' case='lower' repeat='no' delimiter=';' '></tokens>
  <line></line>
  <label>c) </label><tokens limit='1' pos='VB VBP VBZ' minsize='5' case='lower' repeat='no' delimiter=';' '></tokens>
  <line></line>
  <label>d) </label><tokens limit='1' pos='JJ NN VB' minsize='5' case='lower' repeat='no' delimiter=';' '></tokens>
  <line></line>
  <label>e) </label><tokens limit='1' pos='VB VBN' minsize='5' case='lower' repeat='no' delimiter=';' '></tokens>
  <line></line>
  <line></line>
</activity>

```

Fonte: Produção do próprio autor

No modelo, as *tags* 'tokens' são utilizadas para retornar os itens das diferentes classes de palavras da atividade. As propriedades 'limit', 'pos', 'minsize', 'case', 'repeat' e 'delimiter' são responsáveis por: retornar apenas um item, definir qual classe de palavra é aceita, restringir o tamanho mínimo das palavras, converter em caixa baixa, evitar a repetição de itens e delimitar cada item com o caractere ';'.

Modelo de atividade *pt_grammar_pos_a*:

Quadro 26 - Código XML do modelo de atividade *pt_grammar_pos_a*

```
<activity>
  <label>Identifique os grupos com [b]V[/b] (grupo verbal) ou [b]N[/b] (grupo nominal):</label>
  <line></line> <line></line>
  <label>[i]</i></label>
  <label>( ) </label><groups group_tag = 'I-VG' delimiter='[br]' limit='1' min_position='0' max_position='30' minsize='2' ></groups>
  <label>( ) </label><groups group_tag = 'I-NP' delimiter='[br]' limit='1' min_position='0' max_position='30' minsize='2' ></groups>
  <label>( ) </label><groups group_tag = 'I-VG' delimiter='[br]' limit='1' min_position='31' max_position='60' minsize='2'
></groups>
  <label>( ) </label><groups group_tag = 'I-NP' delimiter='[br]' limit='1' min_position='31' max_position='60' minsize='2' ></groups>
  <label>( ) </label><groups group_tag = 'I-VG' delimiter='[br]' limit='1' min_position='61' max_position='90' minsize='2' ></groups>
  <label>( ) </label><groups group_tag = 'I-NP' delimiter='[br]' limit='1' min_position='61' max_position='90' minsize='2' ></groups>
  <label>( ) </label><groups group_tag = 'I-VG' delimiter='[br]' limit='1' min_position='91' max_position='100'
minsize='2'></groups>
  <label>( ) </label><groups group_tag = 'I-NP' delimiter='[br]' limit='1' min_position='91' max_position='100'
minsize='2'></groups>
  <label>[i]</i></label>
  <line></line> <line></line>
</activity>
```

Fonte: Produção do próprio autor

No modelo, utilizou-se a *tag* 'groups' para retornar os grupos verbais e nominais do texto. A propriedade 'group_tag' foi utilizada para indicar o tipo de grupo a ser retornado. As propriedades 'limit', 'min_position', 'max_position', 'minsize' e 'delimiter' são responsáveis por: limitar o número de grupos retornados, restringir a posição mínima do grupo (dispersão no texto), restringir a posição máxima do grupo (também dispersão no texto), restringir o tamanho mínimo do grupo a mais de um item ou palavra e delimitar com uma linha em branco.

Modelo de atividade *pt_paragraphs_keywords_b*:

Quadro 27 - Código XML do modelo de atividade *pt_paragraphs_keywords_b*

```

<activity>
  <label>Volte ao texto e responda a que se referem as seguintes informações em cada parágrafo:</label>
  <line></line><line></line>
  <paragraphs_keywords identpar='yes' group='yes' group_tag='I-NP' delimiter='[br]'></paragraphs_keywords>
  <line></line>
</activity>

```

Fonte: Produção do próprio autor

No modelo, utilizou-se a tag '*paragraphs_keywords*', que extrai as palavras com maior valor de chavidade de cada parágrafo. A propriedade '*identpar*' é definida para identificar o número do parágrafo de onde as palavras foram retiradas. As propriedades '*group*' e '*group_tag*' determinam o retorno do grupo nominal em que a palavra-chave se encontra. Por fim, a propriedade '*delimiter*' define uma linha em branco para delimitar os grupos.

Modelo de atividade *pt_references_a*:

Quadro 28 - Código XML do modelo de atividade *pt_references_a*

```

<activity>
  <label>Volte ao texto e identifique os referentes das seguintes palavras:</label>
  <line></line> <line></line>
  <label>[i]</label>
  <tokens limit='5' pos='PRP PRP$ WP WP$' case='lower' repeat='no' delimiter='_____ [br]'></tokens>
  <label>[/i]</label>
  <line></line> <line></line>
</activity>

```

Fonte: Produção do próprio autor

No modelo, utilizou-se a tag '*tokens*' para retornar os pronomes que compõem o exercício de referência pronominal, por meio das propriedades '*pos*', '*case*', '*repeat*' e '*delimiter*', que são responsáveis por: restringir as etiquetas morfossintáticas a pronomes, converter as palavras em caixa baixa, evitar a repetição de palavras e delimitar os itens retornados com um traço e uma linha em branco.

Modelo de atividade *pt_previous_knowledge_b*:

Quadro 29 - Código XML do modelo de atividade *pt_previous_knowledge_b*

```

<activity>
  <label>Antes de ler o texto, com base nas palavras-chave abaixo, descreva em poucas palavras o seu provável assunto.</label>
  <line></line><line></line>
  <wordcloud_keywords></wordcloud_keywords>
  <label>_____</label><line></line>
  <label>_____</label><line></line>
  <line></line><line></line>
</activity>

```

Fonte: Produção do próprio autor

No modelo, utilizou-se a tag '*wordcloud_keywords*', que extrai um número determinado de palavras-chave do texto e, por meio de códigos *HTML* de formatação, retorna cada palavra formatada com tamanho proporcional ao valor de chavicidade.

Modelo de atividade *pt_typographic_marks_a*:

Quadro 30 - Código XML do modelo de atividade *pt_typographic_marks_a*

```

<activity>
  <label>Qual a relação dos seguintes itens no texto:</label>
  <line></line><line></line>
  <label>[i]</label><tokens limit='5' pos='CD' delimiter='; ' ></tokens><label>[/i]</label>
  <line></line><line></line>
  <label>Tais itens facilitam sua compreensão do texto?</label>
  <line></line>
  <label>_____</label><line></line>
  <label>_____</label><line></line>
  <line></line><line></line>
</activity>

```

Fonte: Produção do próprio autor

No modelo, utilizou-se a tag '*tokens*' para retornar os itens que possuem a etiqueta morfossintática '*CD*' a partir da propriedade '*pos*'. As demais propriedades são '*limit*' e '*delimiter*', que limitam o retorno de cinco itens e definem como delimitador o caractere ';'.

3.5.8 Módulo *xbuilder*

O módulo *xbuilder* foi desenvolvido para gerenciar e executar a criação automática das atividades ou unidades de ensino. Unidade de ensino, no escopo do sistema, tem como definição a seleção de um conjunto de modelos de atividades.

Faz-se menção a unidades de ensino devido ao fato de o sistema proporcionar dois níveis de criação automática de atividades: um está relacionado à interpretação dos modelos de atividade; o outro, à seleção desses modelos para formar uma unidade didática a ser interpretada.

A principal função do módulo é '*build_from_string(xmltext)*', que possui como argumento o texto analisado pelos módulos de análise em tipo *string*. A função, conforme as informações de gênero textual e nível do aprendiz (*basic*, *intermediate*, *advanced*), instancia um conjunto de regras para a seleção de modelos de atividade para o texto.

A ideia é a de que um conjunto de regras de seleção pode conferir uma melhor adequação das atividades ao texto de entrada. Tendo em vista a gama de possibilidades de modelos disponíveis, não é recomendado criar todas as atividades para o texto, uma vez que determinados modelos não seriam adequados. Por exemplo, a seleção de um modelo de atividade para criar um exercício de referência pronominal se o texto não possui pronomes para a criação da atividade.

Algumas das condições para a criação de regras para a seleção de modelos de atividades levantadas nesta pesquisa são:

- a. caracterização do gênero textual do texto de entrada;
- b. nível dos aprendizes;
- c. quantidade e tipos de itens encontrados no texto de entrada que podem ou não contemplar os modelos de atividades disponíveis;
- d. tipo de abordagem ou ênfase em determinados objetivos de ensino (foco em: estratégias de leitura, ensino de gramática etc.);
- e. preferências e opções do usuário.

O conjunto evidencia a necessidade de uma análise mais profunda para que um sistema possa lidar com variáveis tão complexas. Para o presente trabalho, considera-se que as análises e recursos disponibilizados privilegiam a exploração de variáveis linguísticas para a criação de atividades de ensino.

A pesquisa, no entanto, prevê tal necessidade e formaliza os seguintes níveis de processamento para um sistema de criação de atividades:

Figura 54 - Níveis do sistema para criação automática de atividades

Fonte: Produção do próprio autor

Por meio do módulo *xbuilder*, tenta-se minimamente realizar o gerenciamento dos níveis de processamento necessários para a criação automática de atividades. Tal gerenciamento mínimo está relacionado ao tratamento das opções do usuário ao definir o texto de entrada, realizar a análise linguística por meio dos módulos de análise e selecionar os modelos de atividades.

Na primeira versão de avaliação do sistema, apenas a seleção de um modelo de atividade para gêneros textuais e regras de inclusão direta de modelos foram implantadas. Mesmo assim, a criação das atividades não foi prejudicada, pois foram elaborados conjuntos de modelos visando minimizar tal problema.

O módulo *xbuilder* é somente utilizado se o usuário do sistema ativar a opção de criação automática. Caso contrário, a criação automática de atividades dá-se apenas na interpretação dos modelos de atividade selecionados pelo usuário.

3.5.9 Arquitetura do sistema

O sistema recebe um texto como entrada e tem como objetivo realizar análises linguísticas automáticas que, em um nível, permitem sua exploração para extração de informações relevantes para a análise pedagógica e seleção de modelos de atividade pelo usuário e, em outro, a criação automática de um conjunto de atividades a partir de uma seleção previamente programada a partir de regras.

Descreve-se, a seguir, a arquitetura conforme o esquema:

Figura 55 - Arquitetura do sistema

Fonte: Produção do próprio autor

De acordo com o esquema proposto, as etapas práticas de utilização do sistema envolvem:

1. o usuário insere o texto de entrada, que será o foco para a criação das atividades de leitura, define opções como o gênero textual e o nível dos aprendizes, e ativa/desativa a opção que cria automaticamente uma unidade didática a partir do texto de entrada;
2. os módulos de análise linguística processam o texto e geram um conjunto de dados de análise em *XML* do texto;

3. se o usuário ativou a opção de criação automática, o programa realiza- a seleção dos modelos de atividade e traz uma unidade pronta como sugestão. Caso contrário, o usuário faz a análise pedagógica a partir dos dados de análise disponibilizados e seleciona os modelos de atividade desejados para formar a unidade didática;
4. o sistema retorna a atividade pronta.

3.5.10 Interface do sistema de criação de atividades

Para possibilitar o uso prático do sistema pelo usuário e sua avaliação, foi criada inicialmente uma interface para a criação das atividades, conforme a descrição do sistema. A interface criada foi disponibilizada em um sítio e domínio adquiridos especificamente para os propósitos da pesquisa.

A aquisição de uma hospedagem e servidor possibilitou o funcionamento pleno dos módulos programados em linguagem *Python*. Em hospedagens compartilhadas não foi possível instalar a maioria das principais bibliotecas utilizadas no trabalho, tal como *NumPy* e o *NLTK*, uma vez que há acesso restrito a determinados recursos.

Além da linguagem *Python*, a linguagem de programação *PHP* e *JavaScript* foram utilizadas para o gerenciamento das páginas do *site*, realizando a comunicação com os módulos em *Python*. Os códigos *HMTL* e *CSS* ficaram responsáveis pela formação das páginas.

Para a interface, optou-se por criar um ambiente simples, com funções mínimas necessárias para a realização das tarefas de criação de atividades. Assim, foi necessária a inclusão de componentes gratuitos de terceiros: um criador de *CAPTCHA*, uma plataforma de editor de textos em *HTML* e um conversor de *HTML* para PDF (*Portable Document Format*).

O criador de *CAPTCHA* utilizado foi o *cool-php-captcha* versão 3.1, disponível em <<https://code.google.com/p/cool-php-captcha>>. A plataforma de editor de texto em *HTML* utilizado foi a *TinyMCE*, versão 4.1.5, disponível no site <<http://www.tinymce.com>>. O conversor de *HTML* para *PDF* utilizado foi o *HTML2PDF*, versão 4.3, disponível no *site* <<http://html2pdf.fr/en/default>>.

A interface foi basicamente composta por um conjunto mínimo de páginas: página inicial (*home*), página de busca de atividades criadas e salvas por usuários (*search*), página sobre o programa (*about*), página para a entrada do texto e opções do usuário (*create*), página de edição de atividades (*edit*) e página de visualização de atividades salvas (*saved*).

A página inicial criada foi:

Figura 56 - Página inicial do Readler

Fonte: Produção do próprio autor

Na página, há uma pequena descrição da ferramenta, os *links* para a página inicial, página de busca, página sobre o programa, além do botão '*Create...*', que leva o usuário à página para a entrada do texto e opções do usuário.

Ao clicar no botão '*Create...*', o usuário é direcionado para a página que contém o seguinte formulário:

Figura 57 - Formulário para entrada de texto e opções do usuário

Fonte: Produção do próprio autor

No formulário, o usuário deve preencher os campos obrigatórios com as informações do título do texto ('*Title*'), o corpo do texto ('*Text*'), além do campo opcional '*Subtitle*', e selecionar as opções também importantes para a preparação do material, como o gênero textual ('*Genre*'), o nível dos aprendizes ('*Level*') e a ativação do recurso para criação automática de uma unidade didática a partir do texto de entrada. Há também um campo adicional obrigatório, que é o preenchimento do código *CAPTCHA* para que o formulário seja enviado e as informações, processadas.

Após o usuário preencher corretamente o formulário e clicar no botão '*Create...*', a página de edição de atividades aparece com o seguinte editor:

Figura 58 - Editor de atividades do Readler

Fonte: Produção do próprio autor

O editor possui as seguintes quatro abas: *Editor*, *Text*, *Templates* e *Save*. As abas foram projetadas para a realização das tarefas de edição, análise pedagógica do texto, seleção de modelos de atividades e publicação das atividades em unidades didáticas (texto mais atividades).

A aba *Editor* é utilizada para editar as atividades criadas por meio de edição simples de texto e formatação.

A aba *Text* é utilizada para visualizar o texto de entrada (como referência) e as análises disponíveis para o texto de entrada: dados estatísticos sobre a frequência das palavras do texto, visualização das palavras cognatas no texto, visualização das palavras-chave no texto, visualização dos grupos nominais e verbais no texto, lista de frequência do texto, afixos nas palavras do texto, lista de palavras-chave, frequência das classes morfossintáticas e informações de legibilidade do texto. A figura a seguir ilustra um dos tipos de visualização da aba *Text*:

Figura 59 - Exemplo de visualização de informações do texto na aba *Text*

Fonte: Produção do próprio autor

No exemplo, os grupos verbais são visualizados no texto, em destaque. A seleção do tipo de informação a ser exibida é feita por uma lista suspensa como em '*Verb groups*'.

A aba *Templates* é utilizada para a seleção dos modelos de atividades para adicioná-los ao *Editor*. A figura a seguir mostra um dos passos de utilização da aba:

Figura 60 - Exemplo de seleção de modelos de atividades na aba Templates

Fonte: Produção do próprio autor

O exemplo mostra a seleção de um dos modelos de atividade. Na caixa 'Items', ficam os modelos selecionáveis. Na caixa 'Selection', são inseridos os modelos selecionados pelo usuário. Quando o usuário seleciona em um dos modelos, uma pré-visualização do modelo de atividade interpretado em tempo real é exibida. O recurso facilita a decisão de inclusão dos modelos de atividade. A inserção e a remoção de modelos são feitas pelos botões que ficam entre as caixas de seleção. O usuário pode inserir os modelos de atividade selecionados a partir do botão 'Add selection...'. Ao clicar, os modelos selecionados são interpretados e inseridos na caixa de edição da aba 'Editor'. O procedimento pode ser realizado quantas vezes o usuário necessitar.

A aba *Save* é utilizada para publicar a atividade quando o usuário termina suas edições e a considera pronta. A figura abaixo ilustra as opções da aba:

Figura 61 - Opções da aba Save

Fonte: Produção do próprio autor

Para realizar a publicação da unidade didática, o usuário deve preencher os campos '*Title*' (título da atividade), '*Author*' (identificação de quem está publicando) e '*Tags*' (palavras para facilitar a busca da atividade). Após o preenchimento, clica-se no botão '*Save...*'. A página com o *link* para a atividade publicada em *PDF* surge, como mostra a figura a seguir:

Figura 62 - Mensagem da página após a publicação da atividade

Fonte: Produção do próprio autor

Na página, uma mensagem de sucesso é exibida juntamente com o *link* para acesso à atividade publicada. Caso o usuário mude de ideia e não queira a atividade publicada, um *link* para excluir a atividade é disponibilizado. Tal etapa encerra a utilização básica da interface para criação de atividades.

Na próxima seção, apresentam-se os resultados e a sua discussão.

4 RESULTADOS E DISCUSSÃO

Nesta seção, apresentam-se os resultados referentes à construção dos módulos de análise de texto e *corpora*, aos modelos de atividades interpretados e utilizados na avaliação do sistema, ao modelo de unidade didática criada automaticamente, às atividades elaboradas sem o uso do sistema e à avaliação do sistema por usuários.

4.1 Módulos de análise de textos

Na pesquisa, foram criados diversos módulos para a análise de textos. No quadro a seguir, apresentam-se os módulos, junto com a descrição de sua função e a avaliação realizada pelo pesquisador em relação ao seu desempenho em tarefas. Na avaliação, foi considerado o nível de acurácia para os módulos que possibilitaram as etapas de treinamento e de teste de seus algoritmos; nos demais casos, foi levado em conta o desempenho dos módulos em subsidiar a criação das atividades.

Quadro 31 - Relação de módulos de análise programados

Nome do módulo	Função	Avaliação
<i>segmenter</i>	Realizar a segmentação do texto de entrada em sentenças e parágrafos.	Satisfatória, conforme as características do texto de entrada.
<i>tokenizer</i>	Realizar a itemização do texto de entrada.	Satisfatória, conforme as características do texto de entrada.
<i>stemmer</i>	Extrair os radicais das palavras do texto de entrada.	Satisfatória.
<i>postagger</i>	Realizar a etiquetagem morfosintática das sentenças.	Satisfatória. Obteve-se a taxa de 97% de acurácia com os dados de treinamento e 92% com textos do <i>corpus</i> de estudo de anúncios de emprego.
<i>counter</i>	Realizar a contagem de itens.	Satisfatória.

continua

continuação

Nome do módulo	Função	Avaliação
<i>keywordsextractor</i>	Realizar a comparação da lista de frequência do texto de entrada com a lista de frequência do <i>corpus</i> de referência para retornar o valor de chavicidade para cada palavra da lista do texto de entrada.	Satisfatória.
<i>vocabpro</i>	Realizar a identificação de faixas de frequência lexical do texto com base no funcionamento do programa <i>Vocabprofile</i> .	Satisfatória.
<i>cogtagger</i>	Realizar a identificação de palavras cognatas no texto de entrada.	Satisfatória. Obteve-se a taxa de 88% de acurácia com os dados de treinamento.
<i>nptagger</i>	Realizar a identificação de grupos nominais simples no texto de entrada.	Satisfatória. Obteve-se a taxa de 98% de acurácia com os dados de treinamento.
<i>vgtagger</i>	Realizar a identificação de grupos verbais no texto.	Satisfatória.
<i>netagger</i>	Realizar a identificação de entidades nomeadas no texto de entrada.	Precisa de melhorias.
<i>affixes</i>	Realizar a identificação de afixos nas palavras do texto de entrada.	Satisfatória para sufixos. Precisa de melhorias.
<i>tutagger</i>	Realizar a identificação de unidades textuais (movimentos) a partir da classificação de sentenças.	Precisa de melhorias.

Fonte: Produção do próprio autor

No quadro, são apresentados apenas os módulos formalizados como independentes do sistema. A questão da independência está relacionada ao modo como foram programados e organizados, tal qual permitem a alteração e adaptação de suas propriedades e métodos para possíveis melhorias, sem a necessidade de alteração nas demais partes do sistema.

A relação de módulos fornece uma série de outras funções de análise que não foram programadas em um módulo independente específico. Um exemplo é a tarefa de identificação de pronomes e possíveis referentes para atividades de referência pronominal, por meio dos módulos '*postagger*' e '*nptagger*', em algoritmo que, ao identificar o pronome, recupera os grupos nominais antecedentes mais próximos.

4.2 Módulo de análise de *corpora*

As análises do módulo '*corpusanalyzer*', que extrai informações de um *corpus* por meio dos módulos de análise de textos criados, permitiram a visualização de caminhos mais atrativos para a seleção de itens lexicais para a criação das atividades e para a elaboração de currículos de ensino. Para ilustrar tais possibilidades, apresentam-se os dados a seguir:

Tabela 10 - Exemplo de dados para a seleção de itens lexicais para criação de atividades

N	PALAVRA	RADICAL	CHAVICIDADE	FREQ	ARQ	VOC	COG
1	Experience	experi	202284	248	248	1	1
2	And	and	195780	1971	1971	1	0
3	's	's	138216	150	150	0	0
4	Skills	skill	110722	155	155	0	0
5	The	the	90360	1062	1062	1	0
6	Ability	abil	56064	113	113	2	0
7	Management	manag	51456	133	133	1	0
8	Team	team	48100	123	123	1	0
9	with	with	36519	474	474	1	0
10	requirements	requir	32487	80	80	0	1
11	work	work	32390	160	160	0	0
12	including	includ	29388	100	100	1	1
13	responsibilities	respons	29328	65	65	0	1
14	development	develop	27700	123	123	1	0
15	degree	degre	26912	73	73	0	0
16	sales	sale	26412	115	115	1	0
17	business	busi	24910	124	124	1	0
18	strong	strong	24584	82	82	0	0
19	resume	resum	23042	45	45	0	2
20	that	that	21912	105	105	1	0
21	web	web	20640	51	51	0	0
22	salary	salari	19052	45	45	0	1
23	email	email	15930	31	31	0	0
24	please	pleas	15640	66	66	0	0
25	online	onlin	15216	48	48	0	0

Fonte: Produção do próprio autor

Na tabela, são apresentadas as 25 palavras com maior valor de chavicidade do *corpus* de anúncios de emprego. As colunas da esquerda para a direita são: a. o número da ordem; b. as palavras; c. os radicais das palavras; d. o valor de chavicidade; e. a frequência no *corpus*; a frequência nos arquivos do *corpus*; f. a faixa lexical; g. o valor de identificação de palavras cognatas. Todas essas informações, em uma combinação, proporcionam uma gama de possibilidades de seleção de itens lexicais para ensino, de forma a atender melhor a critérios pedagógicos.

Pode-se imaginar uma série de cenários de escolhas pedagógicas a partir dos dados. Por exemplo, as faixas lexicais podem auxiliar na escolha de itens para aprendizes de diferentes níveis. Para os aprendizes iniciantes, os itens da faixa '1' podem ser mais adequados. Para aprendizes mais experientes os itens das faixas '2' e '0' são opções interessantes. Com a combinação da identificação de palavras cognatas, pode-se incluir ou excluir os itens cognatos se a intenção for adicionar mais desafio a algum tipo de atividade ou ensino de vocabulário. A frequência dos itens nos arquivos combinada aos valores de chavidade pode assegurar o ensino dos itens específicos do gênero textual.

Tais dados podem figurar em dois tipos de utilização: manual ou automática. Em uma utilização manual, os dados são salvos em arquivos e analisados com algum tipo de filtro, como o recurso disponível no Microsoft Excel, para a seleção dos itens. Em uma utilização automática, os dados são filtrados a partir de funções programadas, de acordo com os possíveis cenários para atender aos propósitos pedagógicos. Em ambos os tipos de utilização, o conjunto de dados mostra-se mais produtivo se comparado apenas aos dados de frequência absoluta ou somente a um tipo de análise, como a extração de palavras-chave.

Para uma comparação, apresentam-se os resultados parciais da extração de palavras-chave do programa *AntConc*, um dos mais utilizados para análise de *corpora*, com os textos do *corpus* de estudo e os textos do *corpus* BNC como referência:

Figura 63 - Resultado parcial da extração de palavras-chave no program *AntConc*

Fonte: Produção do próprio autor

Os resultados do programa trazem alguns itens semelhantes aos do módulo de análise da pesquisa. No entanto, exige-se ainda um trabalho de análise maior e possibilidades menores de cruzamentos de informações para atender a objetivos pedagógicos mais elaborados.

Outro caminho apontado é a utilização de valores de associação entre palavras para a análise da padronização de itens e o fornecimento de contexto para a criação de atividades. O módulo de análise de *corpora* permite a busca por uma determinada palavra no *corpus* e calcula o valor de associação das palavras de seu contexto em um horizonte de cinco palavras da esquerda e cinco da direita. A tabela a seguir apresenta os candidatos a colocados na posição 'R1', primeira à direita, da palavra de busca 'experience' no *corpus* de anúncios de emprego:

Tabela 11 - Candidatos a colocados da palavra 'experience' na posição R1

N	HORIZONTE	ITEM	T-SCORE
1	R1	in	5,286344945
2	R1	with	3,159479182
3	R1	working	2,372462525
4	R1	preferred	2,099219411
5	R1	managing	1,872584229
6	R1	moving	1,344499865
7	R1	creating	1,205072471
8	R1	using	1,065645076
9	R1	levels	1,042407177

Fonte: Produção do próprio autor

Na tabela, os dados estão ordenados pelo valor de maior associação da coluna '*T-SCORE*'. É comum considerar como corte índices com valor maior que dois. Conforme os índices, os candidatos mais bem colocados são: 'in', 'with', 'working' e 'preferred'. Os critérios para definição de colocados podem levar em conta também a frequência do item e outros tipos de informações.

No programa *AntConc*, a identificação de candidatos a colocados pode ser feita a partir de duas fórmulas: *Mutual information* ou *T-score*. Ambas retornam resultados diferentes. Para a comparação com os resultados do módulo de análise de *corpus*, apresentam-se os resultados com a mesma formula:

Total No. of Collocate Types: 77		Total No. of Collocate Tokens: 508			
Rank	Freq	Freq(L)	Freq(R)	Stat	Collocate
1	62	0	62	7.26028	in
2	39	0	39	5.66154	with
3	12	0	12	3.29028	working
4	9	0	9	2.87391	preferred
5	7	0	7	2.41525	required
6	6	0	6	2.31397	managing
7	5	0	5	1.89083	including
8	13	0	13	1.83919	of
9	2	0	2	1.39238	moving
10	2	0	2	1.36509	creating
11	2	0	2	1.33233	using
12	2	0	2	1.33233	levels
13	2	0	2	1.26683	bachelor
14	2	0	2	1.24500	minimum
15	2	0	2	1.19041	within
16	4	0	4	1.18172	as
17	2	0	2	1.04849	knowledge
18	1	0	1	0.99228	yr

Figura 64 - Candidatos a colocados da palavra 'experience' no programa *AntConc*

Fonte: Produção do próprio autor

Os resultados das primeiras palavras são semelhantes aos do módulo da pesquisa. Nesse caso, a vantagem ainda reside na possibilidade de manipulação programada e na combinação de outras informações. Por exemplo, há casos em que os índices dos candidatos a colocados são muito inferiores a um valor de corte aceitável. Para esses casos, é possível evidenciar um padrão de colocação por meio da consideração de uma classe mais genérica do que a própria consideração do vocábulo. A classe genérica poderia ser a etiqueta morfossintática dos candidatos. O possível padrão evidenciado poderia estar na coligação '*NN +experience*', em que '*NN*' poderia incluir a ocorrência de '*minimun*', '*education*' ou '*management*', por exemplo. Ao definir os padrões, a extração dos contextos pode ser realizada por meio de outras funções do módulo, como a extração de concordâncias.

Desse modo, a possibilidade de criação de funções e métodos para tarefas específicas, aliada à combinação de informações provenientes dos módulos de análise criados na pesquisa, mostra-se bastante produtiva e com grande potencial para um processamento mais qualitativo e propenso a atender a necessidades pedagógicas mais complexas nas tarefas de seleção de itens lexicais, extração de informações e criação de atividades.

4.3 Modelos de atividades interpretados

Nesta seção, apresentam-se os exemplos de modelos de atividades criados para a avaliação do sistema. É importante salientar que os apresentados não esgotam as possibilidades de criações de outros modelos de atividades permitidos pelo sistema.

Para tanto, utilizaram-se os textos do ANEXO E e ANEXO F para criar os exemplos. A maioria dos modelos foi criada com o primeiro. Apenas os dois últimos modelos foram criados a partir do texto do ANEXO F (modelos 26 e 27), uma vez que tais modelos não eram aplicáveis ao texto do ANEXO E. Seguem os modelos interpretados:

Quais palavras são compostas por uma raiz mais afixo (prefixo ou sufixo):

needed
timely
actively
closely
marketing
preparation
participation
management
useful
individual

Figura 65 - Modelo interpretado de atividade 1

<Fonte: www.ilexis.org/readler>

Qual conjunto possui o maior número de palavras cognatas do texto?

() *property; company; super-organized; office; person; industry; contact; executive;*
 () *york; ten; is; the; of; that; new; a;*
 () *to; run; for; and; times; san; in; la;*
 () *svp; be; also; will; oversee; ceo; other; chicago;*

Figura 66 - Modelo interpretado de atividade 2

<Fonte: www.ilexis.org/readler>

Traduza as palavras que são semelhantes a palavras da língua portuguesa (cognatas):

Figura 67 - Modelo interpretado de atividade 3

<Fonte: www.ilexis.org/readler>

Sublinhe as palavras semelhantes a palavras da língua portuguesa;

Figura 68 - Modelo interpretado de atividade 4

<Fonte: www.ilexis.org/readler>

Retire do texto palavras semelhantes a palavras da língua portuguesa.
(Exemplos:).

Figura 69 - Modelo interpretado de atividade 5

<Fonte: www.ilexis.org/readler>

Quais itens nos permitem identificar o gênero do texto?

☐ O formato (*layout*)

☐ Os recursos tipográficos (negrito, itálico etc.)

☐ As palavras características do gênero

☐ O tipo de letra

☐ As figuras

Figura 70 - Modelo interpretado de atividade 6

<Fonte: www.ilexis.org/readler>

Busque as palavras abaixo no texto e as classifique em: adjetivo, substantivo ou verbo.

a) property;

b) energetic;

c) oversee;

d) company;

e) organize;

Figura 71 - Modelo interpretado de atividade 7

<Fonte: www.ilexis.org/readler>

Verifique o contexto gramatical das seguintes palavras no texto: *energetic; individual; administrative; responsible; other*;

Responda conforme sua observação:

a. Qual a classe gramatical das palavras observadas?

b. Geralmente, elas estão acompanhadas de que tipo de palavras?

c. Quais outras palavras do mesmo tipo você consegue identificar no texto?

Figura 72 - Modelo interpretado de atividade 8

<Fonte: www.ilexis.org/readler>

Verifique o contexto gramatical das seguintes palavras no texto: *property; company; super-organized; office; operations;*

Responda conforme sua observação:

- Qual a classe gramatical das palavras observadas?
- Geralmente, elas estão acompanhadas de que tipo de palavras?
- Quais outras palavras do mesmo tipo você consegue identificar no texto?

Figura 73 - Modelo interpretado de atividade 9

<Fonte: www.ilexis.org/readler>

Verifique o contexto gramatical das seguintes palavras no texto: *actively; closely; constantly; forward;*

Responda conforme sua observação:

- Qual a classe gramatical das palavras observadas?
- Geralmente, elas estão acompanhadas de que tipo de palavras?
- Quais outras palavras do mesmo tipo você consegue identificar no texto?

Figura 74 - Modelo interpretado de atividade 10

<Fonte: www.ilexis.org/readler>

Retire do texto palavras que você já conhecia o significado;

Figura 75 - Modelo interpretado de atividade 11

<Fonte: www.ilexis.org/readler>

Sublinhe o núcleo dos grupos nominais abaixo e traduza-os:

a top ten Web property
The New York Times Company
2+ Guide events
About.com 's presence
meeting coordination
Finance support roles a plus
tuition reimbursement
your background

Figura 76 - Modelo interpretado de atividade 12

<Fonte: www.ilexis.org/readler>

Com base apenas no título do texto, descreva em poucas palavras o provável assunto.

Figura 77 - Modelo interpretado de atividade 13

<Fonte: www.ilexis.org/readler>

Antes de ler o texto, com base nas palavras-chave abaixo, descreva em poucas palavras o provável assunto.

Please senior organize CEO including Organize person activities Serve travel
administrative your various schedules San Powerpoint office sales leading experience
executives property serve Ensure **New York** Web offices work Experience
please ten Chicago information executive you Times team COO top Coordinate Francisco
Online will **Company** News SVP events other plus NYT industry company
About.com operations online coordinating new

Figura 78 - Modelo interpretado de atividade 14

<Fonte: www.ilexis.org/readler>

Antes de ler o texto, com base nas palavras-chave abaixo, descreva em poucas palavras o provável assunto.

offices property experience team Online including **New** COO Coordinate Powerpoint
News travel serve Ensure executive please Experience **new** CEO leading senior organize
Times office Francisco **company** plus administrative schedules events Serve
About.com your industry Web sales online NYT Please you executives
Company top SVP work other Chicago coordinating various ten will Organize operations
person **York** activities San information

Figura 79 - Modelo interpretado de atividade 15

<Fonte: www.ilexis.org/readler>

Volte ao texto e identifique os referentes das seguintes palavras:

hr _____
aaja _____
your _____
you _____
we _____

Figura 80 - Modelo interpretado de atividade 16

<Fonte: www.ilexis.org/readler>

Qual a relação dos seguintes itens no texto:

1-2; 2+; 5; 1997; 2005;

Tais itens facilitam sua compreensão do texto?

Figura 81 - Modelo interpretado de atividade 17

<Fonte: www.ilexis.org/readler>

Sublinhe o núcleo dos grupos verbais abaixo e traduza-os:

to run
will work
negotiate misc
setting up
how to work
was acquired
to bring
to work

Figura 82 - Modelo interpretado de atividade 18

<Fonte: www.ilexis.org/readler>

Identifique os grupos com **V** (grupo verbal) ou **N** (grupo nominal):

() *to run*
 () *a top ten Web property*
 () *negotiate misc*
 () *2+ Guide events*
 () *how to work*
 () *meeting coordination*
 () *to bring*
 () *tuition reimbursement*

Figura 83 - Modelo interpretado de atividade 19

<Fonte: www.ilexis.org/readler>

Escreva exemplos de palavras do texto que você conseguiu inferir a partir do contexto?

Figura 84 - Modelo interpretado de atividade 20

<Fonte: www.ilexis.org/readler>

Em poucas linhas, faça um resumo do texto:

Figura 85 - Modelo interpretado de atividade 21

<Fonte: www.ilexis.org/readler>

Qual a relação ou utilidade das informações do texto para o seu contexto?

Figura 86 - Modelo interpretado de atividade 22

<Fonte: www.ilexis.org/readler>

Contextualização

- a) Qual a relação entre a imagem e o título do texto?
- b) Qual é o gênero do texto?
- c) Onde é possível encontrar esse texto?
- c) Qual o possível público-alvo do texto?

Figura 87 - Modelo interpretado de atividade 23

<Fonte: www.ilexis.org/readler>

Volte ao texto e tente encontrar as seguintes informações:

- a. quem/qual empresa oferece a vaga;
- b. que tipo de vaga é oferecida;
- c. quais são as exigências e qualificações necessárias para a vaga;
- d. qual é o salário e os benefícios oferecidos.
- e. como proceder para se candidatar à vaga.

Figura 88 - Modelo interpretado de atividade 24

<Fonte: www.ilexis.org/readler>

Análise os três conjuntos de palavras que geralmente aparecem neste gênero textual:

1. *manager, seeking, executive, description, assistant, location, site, administrator, producer, actively, title, recruiting*
2. *experience, 's, skills, ability, management, team, requirements, work, including, responsibilities, development, degree, sales, business, strong, salary*
3. *resume, email, please, employee, plan, MA, fax, url, employer, comprehensive, bonus, assistance, health/vision/dental, coverage, send*

Responda:

- a. Que tipo de informações cada conjunto parece trazer?
- b. Em que partes do texto essas palavras poderiam aparecer?

Figura 89 - Modelo interpretado de atividade 25

<Fonte: www.illexis.org/readler>

Volte ao texto e responda a que se referem as seguintes informações em cada parágrafo:

1. depression
2. mice
3. region
4. brain
5. team
6. abuse

Figura 90 - Modelo interpretado de atividade 26

<Fonte: www.illexis.org/readler>

Volte ao texto e responda a que se referem as seguintes informações em cada parágrafo:

1. depression
2. mice
3. a reward region
4. that brain region
5. course. The team
6. drug abuse

Figura 91 - Modelo interpretado de atividade 27

<Fonte: www.illexis.org/readler>

4.4 Exemplo de unidade didática gerada automaticamente

Apresenta-se aqui um exemplo de unidade didática gerada automaticamente, quando o usuário da atividade seleciona a opção 'Auto' da interface desenhada. A unidade foi gerada com o texto do ANEXO F.

Quadro 32 - Exemplo de unidade didática gerada automaticamente

Com base apenas no título do texto, descreva em poucas palavras o provável assunto.

Qual a relação dos seguintes itens no texto:

one;

Tais itens facilitam sua compreensão do texto?

Qual conjunto possui o maior número de palavras cognatas do texto?

() evidence; substance; stress; neuroscience; depression; simulation; intimidating; molecular;

() yet; to; known; mood; that; such; are; as;

() more; the; also; eric; j.; makes; and; of;

() sinai; mount; at; new; school; his; nestler; he;

Quais itens nos permitem identificar o gênero do texto?

() O formato (layout)

() Os recursos tipográficos (negrito, itálico etc.)

() As palavras características do gênero

() O tipo de letra

() As figuras

Que tipo de texto é este?

Retire do texto palavras cujo significado você já conhecia;

Escreva exemplos de palavras do texto cujo significado você conseguiu inferir a partir do contexto?

Quais palavras são compostas por uma raiz mais afixo (prefixo ou sufixo):

subjected

daily

commonly

artificially

mounting

vulnerable

stressful

shorter-than-usual

continua

continuação

Busque as palavras abaixo no texto e as classifique em: adjetivo, substantivo ou verbo.

- a) disorders;
- b) negative;
- c) increase;
- d) abuse;
- e) given;

Verifique o contexto gramatical das seguintes palavras no texto: *cocaine; chronic; likely; drug-free; aggressive;*

Responda conforme sua observação:

- a. Qual a classe gramatical das palavras observadas?
- b. Geralmente, elas estão acompanhadas de que tipo de palavras?
- c. Quais outras palavras do mesmo tipo você consegue identificar no texto?

Verifique o contexto gramatical das seguintes palavras no texto: *people; effects; chair; details; humans;*

Responda conforme sua observação:

- a. Qual a classe gramatical das palavras observadas?
- b. Geralmente, elas estão acompanhadas de que tipo de palavras?
- c. Quais outras palavras do mesmo tipo você consegue identificar no texto?

Verifique o contexto gramatical das seguintes palavras no texto: *trickier; sometimes; naturally;*

Responda conforme sua observação:

- a. Qual a classe gramatical das palavras observadas?
- b. Geralmente, elas estão acompanhadas de que tipo de palavras?
- c. Quais outras palavras do mesmo tipo você consegue identificar no texto?

Identifique os grupos com **V** (grupo verbal) ou **N** (grupo nominal):

- () *are known to increase*
- () *Mood disorders*
- () *to interact*
- () *intimidating mouse*
- () *to produce*
- () *this regulatory molecule*
- () *is believed to ratchet up*
- () *drug abuse*

Sublinhe o núcleo dos grupos nominais abaixo e traduza-os:

Mood disorders
drug abuse risk
intimidating mouse
The drug-treated mice
this regulatory molecule
the nucleus accumbens
drug abuse
drug-induced depression

continua

continuação

Volte ao texto e responda a que se referem as seguintes informações em cada parágrafo:

1. depression
2. mice
3. a reward region
4. that brain region
5. course. The team
6. drug abuse

Volte ao texto e identifique os referentes das seguintes palavras:

their _____
they _____

Em poucas linhas, faça um resumo do texto:

Qual a relação ou utilidade das informações do texto para o seu contexto?

<Fonte: www.ilexis.org/readler>

4.5 Atividades elaboradas sem o uso do sistema

Para realizar a comparação, em termos de abrangência, das possibilidades de criação de atividades pelo sistema com a elaboração de atividades realizadas por possíveis usuários da ferramenta, foram coletadas 14 atividades elaboradas por sete duplas de professores (ANEXO G a ANEXO T), em uma orientação técnica sobre criação de atividades de leitura.

Na orientação técnica, foram disponibilizados dois textos em inglês para a criação de atividades de leitura: um texto de anúncio de emprego e outro de divulgação científica (ANEXO E e ANEXO F). Os professores ficaram livres para a elaboração das atividades, em que puderam definir o público-alvo e os objetivos da atividade.

O material elaborado pelos professores foi analisado a fim de verificar quais tipos de atividades foram elaboradas, muito embora, se tenham identificado também algumas inadequações de uso da língua e inadequações conceituais. A análise de categorização feita pelo pesquisador considerou 58 atividades, das quais resultaram 27 categorias. O quadro a seguir traz uma relação dos tipos de atividades em ordem alfabética e sua frequência:

Quadro 33 - Tipos de atividades elaboradas por professores

N	Tipo de atividade	Freq.
1	Ativação de conhecimentos prévios	3
2	Atividade de escrita com base nas informações do texto	1
3	Atividade de predição (a partir das palavras mais frequentes em gráfico)	1
4	Atividade de reescrita das partes mais importantes do texto	1
5	Atividade de tradução (do título)	1
6	Atividade que considera o texto como um todo	2
7	Comparação de linguagem verbal e não-verbal	1
8	Conexão de sentenças partidas e embaralhadas do próprio texto	1
9	Gramática - formação do plural de substantivos	1
10	Gramática - identificação de classes de palavras	1
11	Gramática - identificação do uso de verbos no imperativo	1
12	Identificação de falsos cognatos	1
13	Identificação de marcas tipográficas	1
14	Identificação de palavras cognatas	7
15	Identificação de palavras conhecidas	1
16	Identificação de palavras-chave	2
17	Identificação de pontos principais do texto (por parágrafos)	2
18	Identificação de siglas e nomes de cidades	1
19	Identificação do gênero do texto (a partir do título)	6
20	Inferência do significado de uma palavra	1
21	Leitura do texto (primeira)	3
22	Pergunta direta para identificação do gênero do texto	1
23	Perguntas de localização de informações explícitas	9
24	Perguntas para contextualização do gênero	6
25	Preenchimento de formulário com dados pessoais e qualificações	1
26	Reflexão sobre dificuldades de leitura do texto	1
27	Vocabulário - levantamento de sinônimos	1

Fonte: Produção do próprio autor

Para uma melhor visualização das informações, apresenta-se o gráfico a seguir:

Figura 92 - Gráfico de frequência dos tipos de atividades elaboradas por professores
 Fonte: Produção do próprio autor

No gráfico, pode-se verificar que a maioria dos tipos de atividade (18) ocorre apenas uma vez e corresponde a 67% dos tipos identificados. As atividades que ocorrem mais de uma vez são nove, ou seja, 33% dos tipos identificados. O gráfico a seguir mostra a proporção das atividades que ocorrem mais de uma vez:

Figura 93 - Gráfico dos tipos de atividades que ocorrem mais de uma vez
 Fonte: Produção do próprio autor

O gráfico mostra uma grande repetição de determinados tipos de atividades, como perguntas de localização de informações explícitas, identificação de palavras cognatas, perguntas para contextualização do gênero textual e sua identificação.

Os números apresentados indicam uma variedade considerável de tipos de atividades dada a quantidade de amostras (14 atividades), mas uma concentração em determinados tipos. Considerou-se que tal variedade abrange, em maior ou menor quantidade, aspectos importantes do ensino de leitura: níveis de compreensão da leitura, estratégias de leitura e aspectos léxico-gramaticais. Pôde-se considerar também que talvez haja uma preocupação maior dos professores em relação ao ensino do gênero textual, à condição do aluno em relação ao vocabulário para a compreensão do texto (por meio de palavras cognatas, identificação de marcas tipográficas, inferência etc.), à compreensão dos pontos principais do texto e à localização de informações explícitas no texto.

Uma observação interessante em relação às atividades elaboradas pelos professores é o subuso de elementos linguísticos do próprio texto para a elaboração das atividades. Por exemplo, há um grande número de perguntas diretas para questões de identificação e contextualização do gênero textual. A caracterização linguística do texto é pouco explorada nas atividades.

4.6 Avaliação do sistema por usuários

Para a avaliação do sistema implementado, criou-se um questionário *online* com 16 perguntas, utilizando-se os recursos do *Google Drive*, do buscador *Google*, para criação de formulários a partir de uma conta de e-mail do pesquisador no *Google*. A criação de formulários no *Google Drive* é um recurso gratuito que permite criar e disponibilizar questionários por meio de um *link* para acesso público, além de ferramentas para monitorar as respostas e realizar a tabulação dos resultados.

Dentre as 16 perguntas, foram elaboradas duas questões abertas e as restantes fechadas para os usuários fazerem a marcação em uma escala de 1 a 5. A figura a seguir ilustra a interface do formulário publicado:

Pesquisa

Avaliação de uma ferramenta computacional para auxiliar na elaboração de atividades de leitura em língua inglesa.

***Obrigatório**

É difícil aprender a usar a ferramenta? *

1 2 3 4 5

● ● ● ● ●

É fácil a utilização? *

1 2 3 4 5

● ● ● ● ●

A ferramenta é eficiente no que deve fazer? *

1 2 3 4 5

● ● ● ● ●

Figura 94 - Formulário para avaliação do sistema de criação de atividades

Fonte: Google Drive

O questionário elaborado foi publicado e disponibilizado para os usuários no sítio <https://docs.google.com/forms/d/1WxRNPGCFOaSIOMt_zGLMQsg_l-_u_BK4u4_Htjbp7rc/viewform>. O quadro abaixo apresenta as questões elaboradas:

Quadro 34 - Questões do formulário de avaliação

Número	Questão	Tipo
1	É difícil aprender a usar a ferramenta?	fechada
2	É fácil a utilização?	fechada
3	A ferramenta é eficiente no que deve fazer?	fechada
4	A ferramenta permite a elaboração de atividades de forma produtiva?	fechada
5	O processamento das tarefas é realizado de forma rápida?	fechada
6	Há erros no processamento das informações?	fechada
7	A ferramenta é flexível para realizar as tarefas de criação de atividades?	fechada

continua

continuação

Número	Questão	Tipo
8	Qual seu grau de satisfação em relação às tarefas realizadas pela ferramenta?	fechada
9	Há itens ou partes da ferramenta que você não entendeu ou teve dificuldade de entender?	fechada
10	Os modelos e atividades disponíveis são pedagogicamente significativos para aprendizes da língua inglesa?	fechada
11	Como você avalia a criação automática das atividades pela ferramenta?	fechada
12	Qual o grau de utilidade da ferramenta para professores de língua inglesa?	fechada
13	As informações geradas pela ferramenta são confiáveis?	fechada
14	As análises são pertinentes e auxiliam no entendimento melhor do texto?	fechada
15	O que você sugere para melhorar a ferramenta?	aberta
16	Há alguma outra ferramenta que você utiliza para preparar atividades? Qual?	aberta

Fonte: Produção do próprio autor

As questões visaram à avaliação das análises automáticas, ao conjunto de modelos de atividades disponíveis e à interface do sistema.

Os usuários convidados a utilizar e a avaliar a ferramenta foram: 20 professores de inglês da rede estadual de ensino, da Secretaria da Educação do Estado de São Paulo (SEE-SP), que participaram de uma orientação técnica sobre criação de atividades de leitura na Diretoria de Ensino da Região Leste 3, ministrada pelo pesquisador e, também, os professores coordenadores de Língua Estrangeira Moderna (LEM), formadores de docentes, de 91 Diretorias de Ensino do Estado de São Paulo, por meio de uma lista de e-mails. Embora todos trabalhem na rede estadual de ensino, é sabido que muitos professores e coordenadores ministram aulas em diversos contextos educacionais, em rede particular ou pública, o que favorece a não restrição dos resultados a profissionais de apenas um contexto educacional.

Para que os usuários pudessem realizar a avaliação, um tutorial escrito em *PDF* e um vídeo de utilização básica foram enviados, juntamente com o *link* para a utilização da ferramenta e o *link* do formulário de avaliação.

Ao todo, obteve-se retorno de oito avaliações a partir do formulário. As respostas às perguntas fechadas foram:

Tabela 12 - Respostas às perguntas fechadas

USR	RP1	RP2	RP3	RP4	RP5	RP6	RP7	RP8	RP9	RP10	RP11	RP12	RP13	RP14
1	1	5	5	5	5	1	5	5	1	5	5	5	5	5
2	3	3	3	3	3	3	3	3	4	4	3	3	3	4
3	1	1	4	4	4	1	4	4	1	4	4	4	4	4
4	3	3	5	5	5	3	4	4	1	2	4	4	4	4
5	1	5	4	4	5	1	4	5	1	5	5	5	5	5
6	5	4	5	5	5	1	5	5	3	5	5	3	5	5
7	1	5	5	5	5	1	5	5	1	5	5	5	4	5
8	1	1	5	1	5	1	5	1	1	4	5	5	5	5

Fonte: *Google Drive*

Na primeira coluna da tabela, há a identificação numérica do usuário e, nas colunas restantes, as respostas em uma escala de 1 a 5.

Para melhor visualizar os resultados das respostas às perguntas fechadas, foi criado o gráfico a seguir:

Figura 95 - Gráfico das respostas fechadas

Fonte: *Google Drive*

As respostas às perguntas abertas foram:

Quadro 35 - Respostas às perguntas abertas

USR	O que você sugere para melhorar a ferramenta?	Há alguma outra ferramenta que você utiliza para preparar atividades? Qual?
1	Sem sugestões.	Currículo+
2	talvez seria interessante habilitar um dispositivo de audio interativo conforme fossemos desenvolvendo os exercícios	não
3	Acredito que a ferramenta já está bastante completa e consegue contemplar o objetivo proposto.	Não.
4	quando aparece o erro, saber porque aconteceu.	www.ego4u.com
5	como nunca havia utilizado esse tipo de ferramenta, achei todo resultado muito bom.	não nunca experimentei um tipo de ferramenta assim, mas gostei muito.
6	Nada	Bolgs,sites de inglês,etc.
7	Sem sugestão	Sem sugestão
8	atividade com linkings	não

Fonte: Google Drive

Na primeira coluna do quadro, há identificação do usuário e, nas seguintes, cópia das respostas às perguntas abertas.

Além das avaliações do formulário, obtiveram-se os seguintes *feedbacks* por e-mail:

Feedback do e-mail 1:

Quadro 36 - Mensagem 1

<p><i>"Olá, José!</i></p> <p><i>Também respondi e já enviei o formulário, achei a ferramenta extraordinária.</i></p> <p><i>By!!!"</i></p>

Fonte: E-mail de jlopes.sp@uol.com.br

Feedback do e-mail 2:

Quadro 37 - Mensagem 2

*"Olá Jean!
Acessei a ferramenta sugerida, usei e gostei.
Respondi o questionário e enviei. O que mais é preciso para lhe ajudar com a pesquisa?
Era só isso?
Abraços,
CLÔ"*

Fonte: E-mail de jlopes.sp@uol.com.br

Feedback do e-mail 2:

Quadro 38 - Mensagem 3

*"Bom dia, José Lopes!
Acabei de entrar no link referente a essa ferramenta e achei bastante interessante e completa. Gostaria de ter mais informações sobre ela: os professores de língua inglesa da rede pública já têm informações de como usá-la? Isso já foi divulgado pelos PCs das escolas?
Abraços"*

Fonte: E-mail de jlopes.sp@uol.com.br

Os *feedbacks* obtidos reforçam os resultados obtidos por meio da avaliação e o potencial da ferramenta para os propósitos

4.7 Adaptações do sistema para uso de textos em língua portuguesa

Embora os módulos do sistema tenham sido projetados para análise de textos em língua inglesa, sua criação foi desenhada para permitir a análise de textos de outras línguas por meio de adaptações e inclusão de recursos lexicais específicos da língua que se pretende utilizar.

Para cada módulo, há uma opção interna de seleção de língua. O uso do módulo para uma outra língua pode ser conseguido, acrescentando-se um novo conjunto de regras, léxico, classificador, ou outro tipo de recurso, conforme a especificidade da análise. A figura a seguir exhibe a organização dos diretórios para possibilitar o armazenamento de dados específicos para uma determinada língua:

Figura 96 - Exemplo de organização de diretórios e código

Fonte: Produção do próprio autor

Por exemplo, no módulo de extração de palavras-chave, há o diretório '*data/en*' e, dentro, um arquivo de referência '*reflist.txt*'. Para extração de palavras-chave em português, bastaria adicionar um novo diretório e arquivo de referência ('*data/br/reflist.txt*') e, no código, na instanciação da classe, passar o argumento '*br*' como língua de análise.

Embora as adaptações para o uso de textos em língua portuguesa não tenha sido realizado nesta pesquisa, a arquitetura e organização de módulos e códigos criam os ambientes propícios para tal finalidade.

4.8 Possíveis melhorias ao sistema

Descreve-se, a seguir, a arquitetura planejada para a implementação de possíveis melhorias ao sistema, tendo em vista o conceito de adaptatividade de Neto (2004). A Tecnologia Adaptativa está relacionada a técnicas, métodos e disciplinas que estudam as aplicações da adaptatividade, que pode ser entendida como uma propriedade que um determinado modelo tem de modificar espontaneamente seu próprio comportamento em resposta direta a uma entrada, sem auxílio externo. Um sistema adaptativo é aquele que possui a propriedade de se auto modificar a partir de determinada entrada, sem a necessidade de um agente externo. A figura a seguir ilustra o funcionamento de tal arquitetura:

Figura 97 - Arquitetura para melhoria do sistema

Fonte: Produção do próprio autor

A arquitetura apresentada pode ser descrita a partir dos seguintes passos:

- o usuário insere o texto de entrada, que será o foco para a criação das atividades de leitura;
- os módulos de análise linguística processam o texto e geram um conjunto de itens de análise automática para validação do usuário;
- o usuário valida e/ou corrige os itens de análise automática;
- os itens validados e/ou corrigidos são armazenados em um banco de dados;
- um módulo de aprendizagem e validação analisa o banco de dados de validação do usuário e gera novas regras, léxicos e/ou dados de aprendizagem;
- os módulos de análise linguística são atualizados com os dados gerados pelo módulo de aprendizagem e validação;
- os módulos de análise linguística geram um arquivo XML com a análise do texto de entrada;
- modelos de atividades escritos em XML são instanciados para o texto analisado (selecionados pelo usuário ou requeridos a partir das características do texto analisado);
- um interpretador traduz cada instrução dos modelos de atividade e, por meio de um módulo de leitura e extração de informações, os modelos são preenchidos para gerar as atividades;
- o sistema retorna a atividade pronta.

Embora não especificados em detalhes na figura, os modelos de atividades escritos em *XML* poderiam ser gerados no momento de execução conforme as características do texto analisado. Tal funcionalidade estaria relacionada a uma análise pedagógica automática, em que seriam determinados quais tipos de exercícios e itens seriam mais adequados à análise do texto de entrada.

Entende-se que um texto pode ser explorado de diversas formas. Por exemplo, um professor pode preferir explorar as características gramaticais do texto em relação a sua organização discursiva. Nesse caso, modelos de atividades gramaticais seriam instanciados preferencialmente em relação a outros, conforme as características do texto de entrada.

Uma forma de implementação seria a criação de perfis conforme as possibilidades de exploração pedagógica de um texto. Para cada perfil, um conjunto de modelos de atividades estaria disponível. A seleção de um determinado perfil implicaria a instanciação de determinados modelos de atividades que seriam adicionados de acordo com regras definidas, cujos argumentos seriam satisfeitos com base nas análises automáticas do texto de entrada. Após as adições dos modelos de atividades, todo o conjunto seria submetido ao interpretador para gerar as atividades.

Como já citado, são propostas de possíveis melhorias. Considera-se que tais modificações contribuam para conferir uma característica adaptativa ao sistema, o que promoveria sua constante evolução e aumento de suas funcionalidades para a criação de atividades.

4.9 Avanços do sistema desenvolvido

Os resultados obtidos com a implementação do sistema mostram avanços em relação ao primeiro protótipo construído, *RCB*, e a programas para análise e exploração de *corpora* como o *Kitconc*. Alguns pontos podem ser evidenciados:

- a. a facilidade de uso da ferramenta em relação a programas de análise de *corpora* e sua disponibilização *online*;
- b. a automatização de análises e exploração de *corpora* para a elaboração de materiais didáticos;

- c. o emprego de técnicas e métodos de análise da área de Processamento de Línguas Naturais e da Aprendizagem de Máquina para possibilitar a diminuição de erros de análise;
- d. o aumento das possibilidades de criação de atividades para a variação das atividades por meio de funções parametrizadas e uma metalinguagem, que conferem maior flexibilidade ao sistema;
- e. a arquitetura propícia para a adequação do conjunto de atividades de acordo com o texto de entrada;
- f. o fornecimento de recursos para a caracterização linguística do texto, com potencial para a exploração em atividades;
- g. a possibilidade de melhoria do sistema sem a necessidade de reprogramação de todo o código;

Tais avanços corroboram a possibilidade e viabilidade do sistema proposto nesta pesquisa, assim como as premissas adotadas para o seu desenvolvimento.

5 CONCLUSÃO

Nesta tese, por meio de todo o seu desenvolvimento, procurou-se comprovar a viabilidade do emprego de uma ferramenta computacional que aproveita os avanços científicos das áreas da Linguística e da Computação para a criação automática de atividades de leitura que privilegiam a língua em uso, com foco no ensino-aprendizagem de língua estrangeira.

Partiu-se do pressuposto de que a preparação de materiais no ensino de leitura em língua inglesa pode ser assistida e enriquecida pelo auxílio de recursos computacionais projetados para tal fim, que contribuam para a otimização dos estudos linguísticos em larga escala, tanto em relação ao tempo e esforço, quanto em qualidade de seus resultados, por meio da automatização de análises de textos e exploração de *corpora* e, também, do aproveitamento dos dados de análise para a criação de atividades.

Para tanto e com base no estudo e investigação de abordagens e ferramentas de análise de *corpora* para a criação de materiais de ensino na área de Linguística de Corpus, bem como em técnicas, algoritmos e recursos de áreas da Computação, como o Processamento de Línguas Naturais e o Aprendizado de Máquina, foi proposta a criação de um Sistema de Processamento de Língua Natural capaz de analisar textos para gerar, automaticamente, atividades de leitura e ensino de padrões (léxico-gramática) em língua inglesa a partir de um texto e *corpus*.

Para o desenvolvimento do sistema, a escolha da linguagem de programação *Python*, com a biblioteca *Natural Language Toolkit (NLTK)*, na construção de módulos de análise de texto e *corpora* mostrou-se adequada, uma vez que forneceu as ferramentas necessárias para a criação de etiquetadores e classificadores para as análises linguísticas. As classes de etiquetadores e classificadores do *NLTK* possibilitaram o armazenamento dos etiquetadores e classificadores treinados em arquivos, por meio do módulo '*pickle*', o que contribuiu para a organização dos diretórios e códigos para possíveis adaptações e uso de outras línguas - novos etiquetadores podem ser treinados e armazenados em diretórios específicos, para serem carregados e executados sem alterações no código, que já prevê tal tipo de utilização ao receber o argumento '*language*' na classe de cada módulo de análise.

Os *corpora* da pesquisa - o *corpus* de referência *British National Corpus (BNC)*, o *corpus* de referência Floresta Sinta(c)tica do *NLTK*, o *corpus* de referência MacMorpho do *NLTK*, o *corpus* de estudo/treinamento com 135 textos de anúncios de emprego em inglês da Internet, o *corpus* de estudo/treinamento com 771 textos de divulgação científica das revistas eletrônicas *Scientific American* e *NewScientist* - possibilitaram o material linguístico para o desenvolvimento de todo o sistema. O *corpus* de referência *BNC* foi utilizado na complementação dos dados do etiquetador morfossintático treinado e para gerar uma lista de frequência de palavras que possibilitou a criação do módulo de extração de palavras-chave, que funciona a partir da comparação da lista de frequência do *corpus* de língua geral, *BNC*, com a lista de frequência das palavras de um texto ou *corpus*. Os *corpora* de referência de língua portuguesa foram importantes para o desenvolvimento do módulo de identificação de palavras cognatas - os radicais das formas em língua portuguesa foram extraídos a fim de serem comparados com os radicais de palavras em língua inglesa para determinar candidatos a palavras cognatas. Os *corpora* de estudo serviram para o treinamento, desenvolvimento e testes dos principais módulos de análise, além de oferecerem recursos linguísticos para a criação de atividades.

Os modelos em *XML* criados a partir das análises automáticas programadas, os quais extraem informações de texto e *corpus* para a criação de atividades, permitem o reuso de funções que extraem dados linguísticos dos textos. Os modelos funcionam com uma metalinguagem para a extração de tais dados linguísticos e evitam o trabalho de criar códigos em programação *Python* para cada modelo de atividade. A partir desses recursos, a criação de novos modelos de atividades é realizada facilmente com enorme flexibilidade para possibilidades de extração de informações de dados linguísticos e alterações. Os modelos com funções parametrizadas também podem favorecer a automodificação dos próprios modelos já criados a partir das informações de um texto de entrada, a fim de conferir características adaptativas ao sistema.

A interface gráfica desenhada para a implementação do sistema, por meio das linguagens *PHP*, *JavaScript*, *HTML* e *CSS*, e disponibilizada *online* para a avaliação de possíveis usuários finais, partiu do propósito de, ao mesmo tempo, fornecer sugestões de unidades didáticas de forma automática, de acordo com a vontade do usuário, uma vez que há a opção para habilitar tal recurso, e possibilitar a escolha do usuário em relação aos modelos disponíveis. Em um dos recursos da interface, há a possibilidade de o usuário visualizar análises automáticas do texto de entrada, o que fornece a oportunidade de realizar uma análise

pedagógica do potencial do texto para a criação das atividades. Outro elemento importante é a inclusão de um editor *HTML*, um *plugin* gratuito de terceiros, para que não seja preciso utilizar outro programa para realizar a elaboração das atividades. A funcionalidade de publicação das atividades foi implementada também por meio de um complemento gratuito de terceiros, um conversor de *HTML* para *PDF*. Desse modo, as atividades podem ser salvas no computador do usuário e armazenadas no servidor da ferramenta para disponibilidade em buscas.

A avaliação do sistema em relação aos possíveis usuários finais, nos seus dois procedimentos que visaram a verificar se os modelos de atividades criados atendiam às necessidades de elaboração de atividades, se as análises e informações geradas pelo programa poderiam ser consideradas confiáveis, e se a interface implementada atendia a critérios mínimos de usabilidade, apontou o potencial da ferramenta criada e direções para a sua ampliação e aprimoramento. Os resultados do primeiro procedimento, em que se compararam as possibilidades de criação de atividades pelo sistema com a elaboração de atividades realizadas sem o auxílio da ferramenta, indicaram a viabilidade de a instrumentação computacional do sistema abranger os tipos de atividades mais utilizados pelos professores. A análise das avaliações do segundo procedimento, que envolveu o preenchimento por possíveis usuários finais de um questionário elaborado pelo pesquisador, mostrou-se positiva tanto em relação a informações geradas pelo sistema, quanto a aspectos da interface. O pesquisador também recebeu mensagens em seu e-mail com *feedbacks* positivos em relação ao uso da ferramenta.

Além das informações relevantes apontadas pelas avaliações realizadas por possíveis usuários finais, destaca-se o desempenho dos módulos de análise de textos e *corpora*, que, em sua maioria, realizam satisfatoriamente as funções para as quais foram programados e também apontam caminhos mais atrativos para a seleção de itens lexicais para a criação das atividades e para a elaboração de currículos de ensino, dada a possibilidade de manipulação programada e a combinação de dados de análise.

Os módulos de análise criados para a pesquisa podem ser aplicados a diversos objetivos de análise. Mesmo no ensino de línguas, a utilidade não se restringe à criação de atividades sugeridas neste trabalho, mas se estende à análise de textos para desenvolvimento de currículos de ensino, análise de materiais didáticos, entre outras tarefas relacionadas. Os

módulos também são aplicáveis a áreas próximas, como a Lexicografia, Terminologia, em diversos fins.

Considera-se que a construção de cada módulo de análise, assim como as relações entre o texto e o *corpus*, poderiam configurar pesquisas mais aprofundadas, com resultados ainda mais sólidos. Contudo, tendo em vista o caráter e objetivos desta tese, as análises realizadas foram suficientes para demonstrar as possibilidades e benefícios do sistema criado para o ensino de línguas em geral.

A arquitetura do sistema também se mostrou adequada e flexível para adaptações e melhorias futuras, como a possibilidade de uso de textos em outras línguas e de uma arquitetura que permita a constante otimização do sistema a partir do aproveitamento das experiências de uso dos usuários. Neste último caso, a exploração de outros recursos das áreas de saber consideradas na pesquisa, por exemplo e em especial, na área da Computação, o emprego de técnicas adaptativas será certamente apropriado para a produção de um sistema que seja automodificável em função de variáveis relevantes no processo de ensino-aprendizagem de leitura: gêneros dos textos, usuários finais, pressupostos didáticos e pedagógicos.

Os resultados da pesquisa não se restringem ao sistema desenvolvido, uma vez que permitem uma ampliação dos estudos a partir da possibilidade de incorporação de novas metas, como trabalhos futuros: modelagem do aprendizado da classe (do aluno), acompanhamento da evolução desses aprendizados, geração de material de apoio ao docente, como gabaritos, listas de notas, correções automáticas de testes, indicação de tópicos em que os alunos precisam de reforço, levantamento de modelos de aproveitamento para a classe e por aluno, expondo os pontos fracos do aprendizado e indicando caminhos para sua superação, trabalho com outras línguas, trabalhos com tradução, trabalhos com ampliação de vocabulário e com estruturas linguísticas avançadas, formalização de língua, aprofundamento em gramática etc. Como exemplo prático, há a possibilidade de geração automática de gabaritos para questionários gerados também automaticamente pelo sistema. Seria um primeiro passo para automatizar a tarefa do professor de supervisionar o aproveitamento dos alunos. Em um segundo momento, a correção automática das resoluções dos exercícios e respectiva avaliação do aproveitamento poderiam auxiliar ainda mais o docente nesta tarefa.

Em geral, os resultados obtidos no trabalho são significativos, pois sugerem que o sistema proposto, que permite a automatização de análises de texto e *corpora* para a criação automática de atividades didáticas de leitura e ensino de léxico-gramática em língua inglesa, apresenta um diferencial em relação a ferramentas disponíveis para análise de textos: fornece análises mais apuradas para a tarefa de elaboração de atividades didáticas, quando comparado, por exemplo, a programas como concordanciadores.

Acredita-se que a tese abre caminho para diversos outros trabalhos. A ampliação das áreas de interesse entre a Linguística de *Corpus* e o Ensino de Línguas pode ser concretizada a partir do estudo e criação de ferramentas computacionais para a automatização de análises linguísticas para o propósito de elaboração de materiais de ensino, como um possível campo de atuação. O não esgotamento da pesquisa aponta uma série de requisitos para seu estabelecimento tal como a necessidade de *corpora* especificamente anotados para a criação automática de atividades.

Dentre as contribuições do trabalho, destacam-se o percurso do desenvolvimento do sistema como parte integrante da pesquisa, o diálogo entre as Humanidades – a Linguística e a Língua Inglesa – e as Ciências Exatas – a Computação, com o Processamento de Línguas Naturais e o Aprendizado de Máquina –, e a automatização de tarefas de análise de textos para fins de criação de materiais pedagógicos para o ensino de línguas.

REFERÊNCIAS

- ALONSO, M. C. G. **Corpus linguístico e a aquisição de falsos cognatos em espanhol como língua estrangeira**. 2006. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2006.
- AMARANTE, R. P. **Uma contribuição da linguística de corpus para a fonologia: Um estudo de colocações e aspectos segmentais das vogais da língua inglesa**. 2006. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2006.
- ANTHONY, L. A critical look at software tools in corpus linguistics. **Linguistic Research**, v. 2, n. 30, p. 141-161, 2013.
- _____. Laurence Anthony's Website. **Laurence Anthony's Website**, 2012. Disponível em: <<http://www.antlab.sci.waseda.ac.jp/>>. Acesso em: 01 janeiro 2014.
- ASTON, G. Corpora and language teaching. In: BURNARD, L.; MCENERY, A. M. **Rethinking language pedagogy from a corpus perspective**. Hamburg: Peter Lang, v. II, 2000. Cap. 1, p. 7-17.
- BARBOSA, M. **Material didático para o ensino de inglês instrumental online: uma abordagem experiencial baseada em corpus, gênero e tarefa**. 2004. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2004.
- BEHNAM, B.; BEHNAM, N. Schematic Structure of Job Advertisements in English and Persian: A Comparative Study and Preliminary Framework. **Journal of Language, Culture, and Translation (LCT)**, v. 1, p. 85–103, 2012.
- BERBER SARDINHA, T. **Linguística de Corpus**. Barueri: SP: Manole, 2004. 410 p.
- BÉRTOLI-DUTRA, P. **Explorando a Linguística de Corpus e letras de música na produção de atividades pedagógicas**. 2002. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2002.
- BIRD, S.; KLEIN, E.; LOPER, E. **Natural Language Processing with Python**. [S.l.]: O'Reilly Media Inc, 2009.
- BRILL, E. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. **Computational Linguistics**, v. 21, n. 4, p. 543–565, 1995.
- CONDI DE SOUZA, R. **Dois corpora, uma tarefa. O percurso de coleta, análise e utilização de corpora eletrônicos na elaboração de uma tarefa para ensino de inglês como Língua Estrangeira**. 2005. 191 p. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2005.
- DAVIES, M. **CORPUS OF CONTEMPORARY AMERICAN ENGLISH**. 2012. Disponível em: <<http://www.corpus.byu.edu/coca>>. Acesso em: 01 janeiro 2014.

DIAS DA SILVA, B. C. **A FACE TECNOLÓGICA DOS ESTUDOS DA LINGUAGEM: o processamento automático das línguas naturais**. Araraquara. 1996. 272 f. Tese (Doutorado em Letras) - Faculdade de Ciências e Letras, Universidade Estadual Paulista, 1996.

DIAS DA SILVA, B.C.; Montilha, G.; Rino, L.H.M.; Specia, L.; Nunes, M.G.V.; Oliveira Jr., O.N.; Martins, R.T.; Pardo, T.A.S. (2007). **Introdução ao Processamento das Línguas Naturais e Algumas Aplicações**. Série de Relatórios do NILC de Corpus. NILC-TR-07-10. São Carlos-SP, Agosto, 121 p.

DOMINGUES, M. L. C. S. **Abordagem para o desenvolvimento de um etiquetador de alta acurácia para o Português do Brasil**. Belém: 2011. 154 f. Tese (Doutorado em Engenharia Elétrica) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, 2011.

FERRARI, J. **ESP, Linguística de Corpus e Sócio-Interacionismo na elaboração de uma unidade de um material didático para comércio exterior**. 2004. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2004.

FLOWERDEW, J. Concordancing as a tool in course design. **System**, v. 21, n. 2, p. 231-244, 1993.

FOX, G. Using corpus data in the classroom. In: TOMLINSON, B. (org) **Materials development in language teaching**. Cambridge: Cambridge University Press, 1998. p. 25-43.

GREFENSTETTE, G.; TAPANAINEN, P. **What is a Word, What is a Sentence? Problems of Tokenization**. Proceedings of the 3rd Conference on Computational Lexicography (COMPLEX'94), Budapest, p.7-10, 1994.

GRELLET, F. **Developing Reading Skills**. Cambridge: Cambridge University Press, 1981.

HARRINGTON, P. **Machine learning in action**. Connecticut: Manning Publications Co., 2012.

HOLDEN, S. **O Ensino de Língua Inglesa nos Dias Atuais**. 1ª Edição. ed. São Paulo: SBS, 2009.

HOLMES, J. **Working Papers nº 4: Stages, Strategies and Activities**. PUC. São Paulo, p. 20. 1982.

HUNSTON, S. **Corpora in Applied Linguistics**. Cambridge: Cambridge University Press, 2002.

JACOBI, C. C. B. D. **Linguística de Corpus e ensino de espanhol a brasileiros: Descrição de padrões e preparação de atividades didáticas (decir/hablar; mismo, mientras/en cuanto/aunque)**. 2001. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2001.

JOHNS, T. From printout to handout: grammar and vocabulary teaching in the context of data-driven learning. **ELR Journal**, Birmingham: University of Birmingham, v. 4, p. 1-16, 1991.

JURAFSKY, D.; MANNING, C. **Some Methods and Results on Sequence Models for POS Tagging**. 2012. Disponível em: < <https://www.youtube.com/watch?v=RIYQD8zF2e0>>. Acesso em: 04 dezembro 2014.

KENNEDY, G. D. **An introduction to corpus linguistics**. Nova York, Longman, 1998.

LUGER, G. F. **Inteligência Artificial**. São Paulo: Pearson Education do Brasil, 2013.

MARSLAND, S. **Machine learning: an algorithmic perspective**. Chapman & Hall/CRC Machine Learning and Pattern Recognition Series. CRC, Boca Raton, 2009.

MOREIRA FILHO, J. L. **Desenvolvimento de um software para preparação semiautomática de atividades de leitura em inglês**. 2007. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2007.

_____. **Kitconc 4.0**. 2008. Disponível em: <<http://www.fflch.usp.br/dl/li/>>. Acesso em: 01 janeiro 2014.

_____. **Kitconc - concordanciador desktop gratuito em português**. 2011. Disponível em: < <http://www.fflch.usp.br/dl/li/x/?p=435>>. Acesso em: 01 janeiro 2014.

NETO, J. J. **Tecnologia Adaptativa**. São Paulo: LTA - Laboratório de Linguagens e Tecnologias Adaptativas, 2004. Disponível em: <http://www.pcs.usp.br/~lta/roteiro_estudo/index.html> Acesso em 1 de novembro de 2010.

NUGUES, P. M. **An introduction to language processing with Perl and Prolog**. Berlin: Springer-Verlag, 2006.

NUNES, M.G.V. **O Processamento de Línguas Naturais: para quê e para quem?** Notas Didáticas do ICMC, N. 73, Instituto de Ciências Matemáticas e de Computação. São Carlos-SP, Maio, 2008. 12 p.

PUSTEJOVSKY, J.; STUBBS, A. **Natural Language Annotation for Machine Learning**. Cambridge: O'Reilly, 2012.

RAMOS, R. C. G. GÊNEROS TEXTUAIS: UMA PROPOSTA DE APLICAÇÃO EM CURSOS DE INGLÊS PARA FINS ESPECÍFICOS. **the ESPECIALIST**, São Paulo, v. 25, n. 2, p. 107-129, 2004.

RAMSHAW, L.; MARCUS, M. P. **Text chunking using transformation-based learning**. In: Yarowsky, D. and Church, K., editors, *Proceedings of the Third Workshop on Very Large Corpora*, Cambridge, Massachusetts, p. 82-94, 1995.

RAYSON, P.; GARSIDE, R. **Comparing corpora using frequency profiling**. In: proceedings of the workshop on Comparing Corpora, held in conjunction with the 38th annual

meeting of the Association for Computational Linguistics (ACL 2000), Hong Kong, p. 1-6, October 2000.

RICHARDS, J. C.; RENANDYA, W. A. **Methodology in Language Teaching: An Anthology of Current Practice**. 13ª Edição. ed. New York: Cambridge University Press, 2002.

SCOTT, M.; CARIONI, L.; ZANATTA, M.; BAYER, E.; QUINTANILHA, T. Using a Standard Exercise in Teaching Reading Comprehension. **English Language Teaching Journal**, v. 38, n. 2, p. 114-20.

SCOTT, M. **WordSmith Tools**. 1996. Disponível em: < <http://www.lexically.net/software> >. Acesso em: 01 janeiro 2014.

_____. **Focusing on the Text and Its Key Words**. In: TALC 98 Proceedings, ed. C. Stephens, Oxford: Humanities Computing Unit, Oxford University, p. 152-164, 1998.

SILVA, E. B. VocabProfile: uma ferramenta linguístico-estatística para a aula de língua inglesa. **Domínios de Linguagem**, v. V, n. 1, p. 144-159, 2011.

SILVA E TEIXEIRA, R.B. **Termos de (Onco)mastologia: uma abordagem mediada por corpus**. 2010. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2010.

SOLÉ, I. **Estratégias de Leitura**. 6ª Edição. ed. Porto Alegre: Artmed, 1998.

THE ACADEMIC WORD LIST. Apresenta recursos lexicais para professores e aprendizes. Disponível em: <<http://www.victoria.ac.nz/lals/resources/academicwordlist>>. Acesso em: 27 novembro 2014.

THURSTUN, J.; CANDLIN, C. Concordancing and the teaching of the vocabulary of Academic English. **English for Specific Purposes**, v. 3, n. 17, p. 267-280, 1998.

TRIBBLE, C.; JONES, G. **Concordances in the classroom: a resource book for teachers**. London: Longman, 1990.

VICENTINI, G. **A Linguística de Corpus e o seriado Friends como base para o ensino de chunks em sala de aula de língua inglesa**. 2006. Dissertação de Mestrado em Linguística Aplicada - LAEL, PUC, São Paulo, 2006.

VISL - Visual Interactive Syntax Learning. Apresenta recursos e ferramentas para análise linguística. Disponível em: < <http://beta.visl.sdu.dk/> >. Acesso em: 8 de jan. 2011.

ZAPPAROLI, Z. M. **O método matemático-estatístico-computacional de análise de textos de André Camlong e o programa Stablex**. 2010. Disponível em: <<http://www.fflch.usp.br/dl/li/x/?p=201>>. Acesso em: 10 de set. 2014.

ANEXO A - Questões da atividade-padrão

- 1 Read only the title of your text. Predict and write down at least five vocabulary items—key words—which you expect to see in the text. Use a dictionary if necessary. The key words can be noted down in English or in Portuguese.
 - 2 Skim the text quickly (maximum one minute), looking for key words in the text. Use all the typographical indications, your previous knowledge, cognates, and repeated words. Now write down, in no more than fifteen words, the main theme of the text.
- Re-read the text as often as necessary to answer the following questions:
- 3 What seems to be the author's main intention: to *persuade* you or just to *inform* you?
 - 4 Write down any words which look important in the text (key words) which you did not know before reading it. Beside each one, write down your idea of what it probably means.
 - 5 Write down the main idea of each paragraph, using only one sentence for each main idea. If the text consists of more than seven paragraphs, write down the main idea of each main section. Avoid translating, and try not to mention insignificant details.
 - 6 Divide the text into sections. Is there an introduction? If so, where does it end? Is there a conclusion? If so, where does it start? Explain your answer.
 - 7 Write one sentence reporting something which you learned from the text.
 - 8 Critical reaction: whose interests does this text reflect? Which country, which social class, or which institution? Who would find the publication of this text desirable? Is the information in this text applicable to your own situation?
 - 9 Indicate your interest in this text, using a scale from 1 to 5 (5 = very interesting, 1 = very boring).
 - 10 How many times did you need to use a dictionary to answer the questions so far?
 - 11 Write down the number of each paragraph which you feel you couldn't understand properly, or aren't sure you understood.
 - 12 Try to work out why you found the paragraphs you listed in the last question so difficult. What was the main reason?—
 - a. lack of previous knowledge of the topic
 - b. a grammatical problem (which one?)
 - c. inefficient reading strategies
 - d. difficulty in separating main points from details
 - e. difficulty in identifying the introduction or conclusion
 - etc.
 - 13 Now estimate your comprehension of the text (e.g. 50 per cent, 80 per cent).

Figure 1. Standard Exercise (translated from Portuguese)

ANEXO B - Interface do primeiro protótipo de RCB

ANEXO C - Exemplo de atividade preparada pelo RCB

Unit I

READING STRATEGIES

Before reading the text, pay attention to the most frequent words and try to answer the questions:

Lexical Words			Grammatical Words		
1. OTHER	4	(01,78%)	1. AND	9	(04,00%)
2. AVERAGE	3	(01,33%)	2. TO	6	(02,67%)
3. IS	3	(01,33%)	3. THAT	4	(01,78%)
4. BE	3	(01,33%)	4. MORE	4	(01,78%)
5. INCREASE	3	(01,33%)	5. ON	3	(01,33%)
6. YEARS	2	(00,89%)	6. THE	21	(09,33%)
7. CLIMATE	2	(00,89%)	7. A	2	(00,89%)
8. EFFECTS	2	(00,89%)	8. IN	2	(00,89%)
9. °C	2	(00,89%)	9. BETWEEN	2	(00,89%)
10. TEMPERATURE	2	(00,89%)	10. OVER	2	(00,89%)

1. Can you predict the topic of the text?
2. What kind of text would show the words above?

Skim the text and try to answer the following questions:

3. Are there pictures, symbols, graphs or other clues which may help you understand the text?
4. Read the list of words given below and mark the ones you believe are cognates. Can you find more cognates in the text? If so, what are their translations?

- CLIMATE - TEMPERATURE - GLOBAL - EFFECTS - CLIMATIC - OCEANS - POLAR - SEVERITY - GASES - INTERACTIVE -

5. These are some important words of the text. Which ones are you familiar with? Try to group them according to a pattern of meaning.

- AVERAGE - INCREASE - CLIMATE - WARMING - TEMPERATURE - GLOBAL - BELIEVED - EFFECTS - OTHER - CLIMATIC -

6. What kind of text is this?
7. What is the text layout like and its importance to this kind of text?
8. What is the aim of the text?
9. Who could be interested in reading this text?

Read the text and answer

10. What is the general idea of the text? Compare with your answer in question 3.
11. What are the main ideas?
12. The computer believes that 55,56% are repeated words, 38,4% are cognates and it classifies the text as fairly easy. Do you agree? Why?

LEXICO-GRAMMAR

1. Check if the words below have prefixes or suffixes. List them.

INTERACTIVE INCLUDING INFORMATION INCLUDE RESULTING

2. What is the function of the following words in the text? Try to identify their grammatical class.

	Meaning	Grammatical Category
WHILE		
AND		
THAN		
THAT		
AS		

4. Look at the examples below and try to figure out the meaning for each centralized word. The meanings of the words below are the same as in the text?

1. a regrouping of forces might allow the average voter a better pull at the right lever
2. and drove in 86 runs on a .255 freshman average, completes the Birds' spring squad at 4
3. x hits in 46 at-bats for a .130 batting average in the first 12 games. He took a midnig
4. re was 30-30. Willie's lifetime batting average of .318 is 11 points beyond Mickey's. T
5. Monday's losses in the Associated Press average and making the largest gain in about tw
6. around 675 in the Dow Jones industrial average. It was a level at which some of the in
7. r lower taxes. The Dow Jones industrial average advanced 7.19 to 687.87. Of 1,253 issue
8. s. On \$1.1 billion of 90-day bills, the average yield was 2.325%. The rate a week ago w
9. two dealers are experiencing are above average for their business, farm equipment sale
10. icially forecast at 129% of the 1947-49 average, three points higher than the July 1 es

1. f Friday in voting itself a \$10 per day increase in expense allowances. Pelham said Sund
2. ne on whether the "quickie" vote on the increase can be repealed outright or whether not
3. in the event Congress does provide this increase in federal funds", the State Board of E
4. allas and Sen Parkhouse. The bill would increase from \$5,000,000 to \$15,000,000 the maxi
5. er social security costs. OUTLAYS WOULD INCREASE Officials estimated the annual tax boos
6. esident said he will recommend later an increase in funds for programs under the childre
7. President said he will ask Congress to increase grants to states for vocational rehabil
8. rch he asked a 20 million dollar a year increase, from 30 to 50 millions, in matching gr
9. ntenance of the nuclear deterrent. This increase in the "threshold", as the conventional
10. oubled since 1948, each year showing an increase in offenders. Among arrests reported by

CRITICAL READING

1. What's your opinion about the text? Do you agree with it?

TEXT:

All About Global Warming

Global warming is the term used to describe a gradual increase in the average temperature of the Earth's atmosphere and its oceans, a change that is believed to be permanently changing the Earth's climate forever.

While many view the effects of global warming to be more substantial and more rapidly occurring than others do, the scientific consensus on climatic changes related to global warming is that the average temperature of the Earth has risen between 0.4 and 0.8 °C over the past 100 years. The increased volumes of carbon dioxide and other greenhouse gases released by the burning of fossil

ANEXO D - Lista de etiquetas morfossintáticas do corpus Penn Treebank

1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential <i>there</i>
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NP	Proper noun, singular
15.	NPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PP	Personal pronoun
19.	PP\$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	<i>to</i>
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VCN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP\$	Possessive wh-pronoun
36.	WRB	Wh-adverb

ANEXO E - Texto de anúncio de emprego em inglês utilizado em orientação técnica

Job advertisement

About.com, a top ten Web property that is part of The New York Times Company, is actively recruiting for a super-organized, energetic individual to run the administrative office operations in the New York office and oversee activities in the LA, San Francisco and Chicago offices.

S/he will also be responsible for the coordinating schedules and travel for the CEO, COO and SVP of sales and other senior executives.

This person will work closely with HR, sales and marketing executives to organize and implement various industry events.

Responsibilities:

- Ensure the smooth day-to-day administrative operations of the 90-person office in New York and serve as contact person for the small 1-2 person offices in LA, San Francisco and Chicago.
 - Coordinate schedules and travel as needed for CEO, COO, SVP of sales and other members of senior executive team. Assist executive team with preparation of Powerpoint slides for NYTD and NYT Co. presentations and events.
 - Organize various company and industry events including 2+ Guide events per year, About.com's presence with The New York Times Company at NABJ, NAHJ, AAJA and other journalism events, and participation and sponsorship of annual Online News Association (ONA) and related conferences.
 - Coordinate and negotiate misc. billing and vendor activities, including recruitment job postings. Serve as liaison to the building facilities management (Primedia) in New York.
 - Ensure timely turnaround of new hires, from scheduling interviews with hiring managers and travel arrangements to setting up desks, phones and laptops for new employees.
 - Manage About.com intranet, including updating of useful About and NYTCo. information to facilitate "self-service" for employees.
- Requirements:
- Minimum of 5 years of experience in managing small- to medium- sized offices and/or supporting senior-level executives.
 - Dedicated and hard-working. Ability to constantly prioritize and re-prioritize incoming requests; organization skills are critical.
 - Experience coordinating multiple calendars for executives.
 - Experience in event planning and meeting coordination.
 - Experience in Human Resources or Finance support roles a plus.
 - Demonstrates an awareness of how to work with sensitive and confidential information and situations.

- High energy and sense of humor a must.
- Expertise with Word and Powerpoint required. Excel a plus.
- Bachelor's degree or equivalent.

Founded in 1997, About.com is the leading online source for original consumer information and advice and was acquired in March 2005 by The New York Times Company (NYSE: NYT - News), a leading media company with 2004 revenues of \$3.3 billion.

As a top ten Web property, About.com holds a valued role in the lives of more than 22 million Internet users.

About.com offers an exciting and informal atmosphere, competitive compensation package, full health and welfare benefits, 401(k) with a company match, employee stock purchase plan and tuition reimbursement.

If your background fits the above description and you thrive in a fast-paced environment, we want you to bring your talents to work for us.

Please forward your resume along with cover letter and salary requirements to: work@about-inc.com. Local candidates only please.

ANEXO F - Texto de divulgação científica utilizado em orientação técnica

Drug Addiction May Make Users More Vulnerable to Stress

Depression and substance abuse modify the same brain circuit

By Andrea Anderson | January 30, 2012 | 1

Mood disorders such as depression are known to increase drug abuse risk. Yet mounting evidence suggests that substance abuse also makes people more vulnerable to depression and the negative effects of stress, according to Eric J. Nestler, chair of neuroscience at the Mount Sinai School of Medicine. He and his team reported new details about the link between depression and drug abuse in *Neuron* in August.

The team found that mice given cocaine daily for a week a simulation of chronic drug abuse in humans were more likely than their drug-free counterparts to display behaviors reminiscent of depression after being subjected to socially stressful situations involving an aggressive and intimidating mouse. The drug-treated mice became lethargic and reluctant to interact with other mice following a shorter-than-usual bout of this “social defeat” stress, which is commonly used to study depression in mice.

Most striking, the researchers found that the cocaine use led to the same molecular changes in the nucleus accumbens, a reward region, as are found in mice prone to stress and depression. The mice had lower levels of a molecule that polices the activity of certain genes and keeps at least one signaling circuit in check.

When the researchers artificially dialed down or up the levels of this regulatory molecule in the nucleus accumbens, they were able to produce or protect against depression in mice. This effect suggests that shifts in that brain region can cause and are not just a side effect of depression.

Testing for such changes in the human brain is trickier, of course. The team did find low levels of some of the same gene-regulating components in postmortem tissue samples from the nucleus accumbens of people diagnosed with depression, hinting that humans with the disorder might experience altered signaling in this brain region, too.

If so, the findings may provide clues about why cases of drug abuse and depression sometimes spiral out of control, given that drug-induced depression is believed to ratchet up the chances of subsequent abuse in the same way that naturally occurring depression can.

ANEXO G - Atividade preparada por professores 1

Atividade. TEXTO I - “Job Advertisement”

Público Alvo: Ensino Médio

Objetivo: Competência Leitora por meio da inferência.

- 1) Tente ler o texto.
- 2) Repita a leitura. Destaque quais expressões você conhece.
- 3) Identifique as palavras que são parecidas com o nosso idioma (palavras chave)
- 4) Identifique as expressões com números e escreva abaixo:
- 5) Identifique as siglas do texto e separe o nome das cidades.
- 6) Dentre os temas abaixo, escolha um no qual identifique o assunto do texto.
 - a) Esportes
 - b) Entretenimento
 - c) Família/Escola
 - d) Emprego/Trabalho
- 7) Justifique a escolha de sua alternativa.

ANEXO H - Atividade preparada por professores 2

Atividade TEXTO II -

- 1)Tente ler todo o texto.
- 2)Faça a tradução do título
- 3)Identifique o nome do local da pesquisa e seu responsável.
- 4)Identifique as palavras parecidas com o nosso idioma.
- 5)Quais os sintomas após o uso da droga citada no texto?
- 6)Localize no segundo parágrafo do texto, como os pesquisadores conseguiram identificar esses sintomas?
- 7)Como você transmitiria essa informação para outra pessoa?

ANEXO I - Atividade preparada por professores 3

Público alvo: 3.ano E. M

Objetivo Geral: identificar o gênero textual anúncio de emprego.

Objetivo Específico: reconhecer os elementos textuais para compreensão.

Before Reading:

- 1- Have you ever read a job ads? If yes, where?
- 2 – Did you have difficulties to understand the text? Which one (s)?

During the reading:

- 1- Identify the employee qualities to get a job.
- 2 – Could you identify the use of the imperative in the text? Give the examples.
- 3 – Underline the cognates and the false cognates, and report the differences between them.
- 4 – Talking about synonym, what another word could replace “resumé”?

Let's check all the comprehension in relation the genre:

- 5 – Rewrite the job ad and select the most important parts that you could notice.

ANEXO J - Atividade preparada por professores 4**Artigo Científico**

Público alvo: 8º's e 9º's Anos.

Objetivo Geral: Reconhecer o gênero textual artigo científico e conscientizar sobre os problemas do uso de drogas.

Objetivo Específico: Compreensão e análise do texto.

Before reading:

1 – What's the genre of the text? Justify your answer.

- a) News
- b) Short story
- c) Scientific article

2- The title of the text is about:

- a) The problems involving drugs
- b) The new product from Apple
- c) The sinopse of a movie

How did you notice it?

3 – What's the animal is used to do the experience?

4 – According to the text, what's the behave is caused because of the use of the substance?

- a) Problems of reading in general texts.
- b) Depression and stress to deal with other people.
- c) Stress of the animals.

5- Do you believe the substance can cause health problems? Explain showing your point of view and compare your opinion with the text.

ANEXO K - Atividade preparada por professores 5**Público Alvo:** 9º ano (8ª série)**Objetivo Geral**

Identificar as palavras já conhecidas, a fim de assimilar o contexto, ou seja, inserir o aluno à mensagem principal do texto.

Objetivo Específico

Reconhecer o gênero textual.

Estratégia

Preenchimento de um formulário de emprego, a fim de reconhecer as dificuldades do educando.

About.com

Name: _____

Date of Birth: ____/____/____

Nationality: _____

Address: _____ zip code: _____ - _____

e-mail: _____

Habilities: _____

Describe your experiences:

Describe yourself:

Salary requirements: _____

ANEXO L - Atividade preparada por professores 6

Atividade II

Público Alvo: 9º ano (8ª série)

Objetivo Geral

Identificar as palavras principais do texto, com o intuito de facilitar a leitura e compreensão da mensagem transmitida.

Objetivo Específico

Reconhecer o gênero textual;

Uso adequado dos plurais;

Estratégia

Questionário

Questions

Who wrote the text?

When was it written?

What kind of text is it?

Do you know anybody with depression? Who?

Write the correct plural form of the words below:

Case: _____ Disorder: _____ Detail:

Mouse: _____ Accumben: _____

Level: _____

Match the collums:

a) He and his team reported new
[...]

() molecule in the nucleus accumbens

b) The drug-treated mice became

() spiral out of control [...]

c) When the researchers artificially

() lethargic and reluctant to interact with

dialed down or up the levels of this regulatory

d)The findings may provide clues depression about why cases of drug abuse and depression sometimes

other mice [...]

() details about the link between and drug abuse in Neuron in August.

ANEXO M - Atividade preparada por professores 7**Texto 1**

- Ensino médio 3 ano;
- Objetivo: Apresentar gênero anúncio de emprego.
- O que é pedido no anúncio de emprego;
- Palavras chaves que indica o gênero;
- Qual função está sendo disponibilizada;
- Palavras cognatas;
- Competências e habilidades requeridas.
- Experiência profissional
- Responsabilidades

ANEXO N - Atividade preparada por professores 8**Texto 2**

- Ensino médio 3 ano;
- Objetivo: Apresentar o gênero científico;
- Identificar os danos cerebrais causados pelo uso de drogas;
- Identificar as palavras chaves que levam a essa conclusão e suas cognatas;
- Perguntas que o levem a identificar as causas e os efeitos do uso de drogas;

ANEXO O - Atividade preparada por professores 9

Atividade 1 – Job Advertisement

Público alvo: alunos do ensino médio

Objetivo: identificar o gênero textual e sua finalidade

Estratégias de leitura

entregar o texto para o aluno

a partir do título identificar qual o provável assunto do texto e o gênero textual

Conhecimentos prévios sobre o mercado de trabalho e expectativas de emprego

Leitura do texto e identificação de palavras cognatas

identificar no texto palavras relacionadas a *job*

identificar o nome da empresa contratante

localizar quais cargos estão oferecendo e o nome das cidades onde estão as vagas

o que a pessoa contratada terá que fazer

quais são os pré-requisitos para a pessoa trabalhar com eles

como a pessoa faz para se candidatar a vaga

ANEXO P - Atividade preparada por professores 10**Atividade 2 – Drug Addiction May Make Users More Vulnerable to Stress**

Público alvo: alunos do ensino fundamental (anos finais) e ensino médio

Objetivo: identificar o gênero textual e sua finalidade

Estratégias de leitura

entregar o texto para o aluno

a partir do título identificar qual o provável assunto do texto e o gênero textual

quem é a autora do artigo e quando ele foi publicado

qual o nome do pesquisador, o nome do local onde ele trabalha e o que ele está pesquisando

você sabe o que é drug?/qual é a palavra no texto para drug?

Qual o resultado obtido nos ratos que usaram a cocaína diariamente?

Esse tipo de situação é semelhante em seres humanos?

A pesquisa em ratos demonstrou que eles podem ter depressão? Justifique a sua resposta.

Os seres humanos apresentam as mesmas características?

De acordo com a pesquisa as pessoas que usam drogas ficam mais vulneráveis ao stress?

ANEXO Q - Atividade preparada por professores 11

Texto 1:

Público alvo: alunos do 3º ano do E.M

Objetivos : que o aluno possa, a partir do texto, elaborar um formulário com os requisitos necessários para preenchimento de vaga na área de vendas em uma empresa de grande porte.

1-identifique a que gênero literário pertence esse texto.

2-como você chegou a essa conclusão?

3-Aonde é veiculado esse tipo de texto?

4- Localize no texto palavras que comprovem a sua resposta.

5-Construa um formulário de solicitação de emprego em determinada empresa, de acordo com os requisitos apresentados no texto.

ANEXO R - Atividade preparada por professores 12

Público-alvo : alunos do E.M

Objetivo : Através do texto apresentado o aluno consiga identificar os efeitos das substâncias químicas que levam os usuários a situações de depressão e estresse prejudicando sua saúde física e mental.

- 1- Identifique a que gênero literário pertence esse texto.
- 2-Identifique palavras cognatas que facilitaram seu entendimento do texto.
- 3-Identifique em cada parágrafo qual o principal tema abordado.
- 4- Transcreva o parágrafo em que se chega à conclusão dos malefícios do uso de droga.
- 5-Identifique gramaticalmente o uso da palavra mouse e mice.

ANEXO S - Atividade preparada por professores 13

Público Alvo: Ensino Médio

A qual gênero pertence esse texto? O que possibilitou o seu reconhecimento? De que forma ele está dividido?

Qual o cargo oferecido?

Quais são alguns dos requisitos para ele? Localize aqueles que dizem respeito à “experiência”.

São exigidas habilidades de informática? Em quais programas?

Caso você tenha interesse na vaga, como você entraria em contato com a empresa?

Qual a empresa que está oferecendo a vaga?

APÊNDICE A - Recursos da linguagem Python para a pesquisa linguística

O apêndice inclui os principais recursos que justificam a escolha da linguagem de programação *Python* para a pesquisa linguística, além de conceitos e convenções utilizadas nas descrições de códigos e algoritmos da pesquisa:

a. Facilidade de escrever programas

Python é uma linguagem de programação fácil de aprender. O código é de fácil leitura se comparado a outras linguagens, como *Perl*. Há uma grande variedade de bibliotecas disponíveis.

Um programa pode basicamente ser definido como o conjunto de instruções sequenciais para execução de algo. Vejamos um exemplo de programa simples que conta e retorna o número total de palavras de uma frase como ‘Aprender a programar para pesquisa linguística é muito fácil.’:

```

1  def contar_palavras (frase):
2      itens = frase.split(' ')
3      contador = 0
4      for item in itens:
5          if item not in ['.', ',', ';', ':', '?', '!']:
6              contador+=1
7      return contador

```

Figura 98 - Programa simples para contar palavras

Fonte: Produção do próprio autor

No exemplo, com apenas sete linhas de código, uma frase deve ser passada como entrada para a função 'contar_palavras' e, ao final da execução, o programa retorna o número de palavras. Os detalhes do processamento são:

Linha 1 – define a função ‘contar_palavras’ e a requisição de um argumento de entrada 'frase'.

Linha 2 – separa cada item delimitado por um espaço da variável ‘frase’ e armazena em outra variável ‘itens’ como uma lista;

Linha 3 – inicializa uma variável ‘contador’ para receber a contagem das palavras;

Linha 4 –passa o valor de cada item da lista, do início ao fim, à variável 'item';

Linha 5 – verifica se o item não é um caractere de pontuação na lista;

Linha 6 – adiciona 1 ao valor da variável ‘contador’;

Linha 7 – retorna o valor da variável 'contador' ao fim da execução.

Após a escrita do programa, pode-se executá-lo no *IDLE*, chamando a função e passando o argumento. Veja o exemplo:

A screenshot of a Python 2.7.6 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Windows', and 'Help'. The main text area shows the following content:

```
Python 2.7.6 (default, Nov 10 2013, 19:24:24) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
>>> contar_palavras('Programar em Python é legal.')
5
>>> |
```

The status bar at the bottom right indicates 'Ln: 7 Col: 4'.

Figura 99 - Execução de um programa em *Python*

Fonte: Produção do próprio autor

Após o processamento da função, o programa imprime na tela o número de palavras da frase de entrada. O programa escrito deve funcionar bem com sentenças em que não haja espaços duplicados e outros tipos de pontuação. Alguns ajustes devem ser feitos para que o programa possa lidar com outras situações previstas, mas o exemplo é suficiente para mostrar a simplicidade e facilidade de escrever e executar programas na linguagem *Python*.

b) Janela interativa do IDLE

A mesma janela interativa da *IDE*, que imprime os resultados do programa na tela, é útil para visualizar e inspecionar elementos dos programas enquanto são desenvolvidos, além de possibilitar a interferência em sua execução, validando ou adicionando novas informações ou dados em tempo de execução. Por exemplo, ao desenvolver um programa que classifica palavras a partir de dados de anotação, é possível adicionar uma instrução que solicita a intervenção do usuário quando o programa tem poucas condições de classificar uma palavra.

c) Operadores e variáveis

Em *Python*, é possível fazer cálculos que geralmente são feitos em calculadoras. Para isso, utilizam-se operadores. Os operadores auxiliam em operações de multiplicação, adição, subtração e divisão, entre outras mais complexas.

Os símbolos básicos são:

Quadro 39 - Operadores da linguagem *Python*

Símbolo	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão

Fonte: Produção do próprio autor

Para fazer um teste de uso dos operadores apresentados, seguem os seguintes passos:

1. Abre-se a janela de comandos do *IDLE*.
2. Digitam-se as seguintes expressões, pressionando a tecla 'Enter' ao final de cada

uma:

```
>>> 10+10
20
>>> 10-10
0
>>> 10*10
100
>>> 10/10
1
>>>
```

Conforme os comandos, os resultados são impressos na tela.

A ordem das operações pode ser alterada com o uso de parênteses. Digitando as expressões abaixo, são obtidos resultados diferentes:

```
>>> 10+10*10
110
>>> (10+10)*10
200
>>>
```

Nos exemplos, os resultados são impressos na tela. Os resultados também podem ser armazenados em variáveis para uso em outras expressões.

Em linguagem de programação, uma variável pode ser descrita como um local na memória para armazenar informações, tal como números, texto, listas de números e textos, entre outros tipos.

Em *Python*, uma variável é criada ao atribuir um valor a ela, utilizando o sinal de igual '='. Por exemplo, para criar uma variável que armazena o ano de nascimento de uma pessoa, digita-se o nome de identificação da variável, o sinal de igual e o valor a ser atribuído à variável.

```
>>> nascimento = 1982
>>> print nascimento
1982
>>>
```

No exemplo, foi criada a variável:

```
>>> ano_atual = 2014
>>> idade = ano_atual - nascimento
>>> print idade
32
>>>
```

d) Manipulação de *strings*

Uma *string* pode ser definida como uma sequência de caracteres. Na linguagem *Python*, indicamos uma sequência de caracteres como tipo *string* utilizando aspas simples ou duplas. *Python* permite uma série de operações de manipulação de *strings*: concatenação, repetição, contagem de caracteres, substituição, cortes. A seguir, exemplificamos os recursos mais comuns utilizados na pesquisa.

Concatenação - operador '+' pode ser usado para concatenar *strings*, ou seja, juntar:

```
>>> primeira = 'especial'
>>> segunda = 'mente'
>>> terceira = primeira + segunda
>>> print terceira
especialmente
>>>
```

Repetição - operador '*' pode ser usado para repetir *strings*:

```
>>> a = 'ba'
>>> b = 'na'
>>> c = a + b*2
>>> print c
banana
>>>
```

Contagem de caracteres - a função 'len' é usada para retornar o número de caracteres de uma *string*:

```
>>> palavra = 'programação'
>>> len(palavra)
11
>>>
```

Acesso a *substrings* - o operador '[' permite o acesso a um caractere por vez. No exemplo abaixo, é acessado o segundo caractere da palavra 'girafa':

```
>>> animal = 'girafa'
>>> animal[1]
'i'
>>>
```

Cada caractere é acessado por um índice, que se inicia em 0:

g(0) i(1) r(2) a(3) f(4) a(5)

Assim, para retornar o primeiro e último caractere da palavra ‘girafa’, usam-se os índices 0 e 5. Exemplo:

```
>>> animal[0]
'g'
>>> animal[5]
'a'
>>>
```

Pode ser usado o laço de repetição ‘*for*’ para acessar todos os caracteres e imprimir um de cada vez. Em ‘range’, especifica-se o índice inicial e final:

```
>>> for i in range(0,5):
    print animal[i]

g
i
r
a
f
>>>
```

Se não se sabe a princípio o tamanho da *string*, pode-se usar a função ‘*len*’. O mesmo resultado é obtido:

```
>>> for i in range(0, len(animal)):
    print i, animal[i]

0 g
1 i
2 r
3 a
4 f
5 a
>>>
```

No exemplo, imprime-se também o valor de ‘i’, usado como índice.

Podem ser selecionadas partes específicas de uma *string*. No exemplo abaixo, foram selecionados os três primeiros caracteres da palavra ‘girafa’:

```
>>> animal[0:3]
'gir'
>>>
```

Para separar as sílabas da palavra ‘girafa’, pode-se usar:

```
>>> animal[0:2]
'gi'
>>> animal[2:4]
'ra'
>>> animal[4:6]
'fa'
>>>
```

O corte deve ser feito no índice seguinte ao último a ser retornado. Para selecionar a primeira sílaba ‘gi’, o índice inicial é 0 (g), e o índice final é 1.

Se o primeiro índice for omitido, o corte começa no início da *string*, retornando os primeiros caracteres da *string*:

```
>>> animal[:3]
'gir'
>>>
```

Se o último índice é omitido, o corte inicia-se no final da *string*, retornando os últimos caracteres da *string*:

```
>>> animal[3:]
'afa'
>>>
```

Em *Python*, as variáveis do tipo string também possuem funções/métodos para sua manipulação. No quadro abaixo, apresentam-se as funções mais comuns utilizadas no trabalho:

Quadro 40 - Principais funções de manipulação de *strings*

Método	Descrição
<i>Endswith</i>	Verifica uma determinada terminação de uma <i>string</i> e retorna falso ou verdadeiro.
<i>Find</i>	Busca um ou mais caracteres na string e retorna o índice da primeira ocorrência caso encontre.
<i>Isdigit</i>	Verifica se a <i>string</i> tem formato número e retorna falso ou verdadeiro.
<i>Join</i>	Concatena <i>strings</i> em uma sequência.
<i>Lower</i>	Retorna a cópia de uma <i>string</i> convertida em minúsculas.
<i>Replace</i>	Retorna a cópia de uma <i>string</i> com a substituição de um ou mais caracteres por outros.
<i>Split</i>	Retorna uma lista das palavras na <i>string</i> a partir de um delimitador.
<i>Strip</i>	Retorna uma cópia da <i>strings</i> sem os espaços em branco de ambas as extremidades da <i>string</i> .

continua

continuação

<i>Startswith</i>	Verifica um determinado prefixo de uma <i>string</i> e retorna falso ou verdadeiro.
<i>Upper</i>	Retorna a cópia de uma <i>string</i> convertida em maiúsculas.

Fonte: Produção do próprio autor

As funções apresentadas podem ser combinadas para realizar uma série de tarefas. O código na figura a seguir é um exemplo:

```
import string
texto = '''
Dinosaur tracks recently discovered on the Arabian Peninsula are not only
the first of their kind in the region, but they also reveal more about
the herding behavior of the prehistoric beasts.
'''
for p in string.punctuation:
    texto = texto.replace(p, ' ' + p + ' ')
texto = texto.lower()
texto = texto.strip()
palavras = texto.split(' ')
selecao = []
for palavra in palavras:
    if palavra.endswith('y'):
        selecao.append(palavra + '; ')
print ''.join(selecao)
```

Figura 100 - Exemplo de uso de funções de manipulação de *strings*

Fonte: Produção do próprio autor

No código, as funções '*replace*', '*lower*', '*strip*', '*split*', '*endswith*' e '*join*' são utilizadas para itemizar o texto, selecionar as palavras que terminam com 'y' e imprimir na tela. O resultado é a impressão das palavras: '*recently; only; they;*'.

Como se pode verificar, as possibilidades de manipulação de *strings* são enormes. No trabalho de desenvolvimento, tais recursos foram constantemente utilizados.

e) Expressões regulares

Expressões regulares são padrões, em uma linguagem formal, que podem ser formados por caracteres literais ou metacaracteres para identificação de cadeias específicas de

interesse (endereços de e-mails, sites, nomes próprios, datas, moeda etc.). No quadro abaixo, seguem alguns exemplos:

Quadro 41 - Exemplo de cadeias de caracteres e expressões regulares

Cadeias de caracteres	Expressões regulares
Senhor Senhores senhor senhores	'[s S]enhore?s?'
Av. Paulista, 123	'Av\.[A-Z][a-z]+.[0-9]+'
02/01/1982	'[0-9]+/[0-9]+/[0-9]{4}'
jlopes@usp.br	'[a-z]+@[a-z]+\.[a-z]+'

Fonte: Produção do próprio autor

As expressões regulares são ferramentas básicas e fundamentais para o processamento de textos. Praticamente todas as linguagens de programação possuem algum tipo de biblioteca ou módulo para lidar com expressões regulares.

Os metacaracteres mais comuns são: '.', '^', '\$', '*', '+', '?', '{}', '\', '[' e '|'. Há também classes de metacaracteres como '\A', '\b', '\B', '\d', '\D', '\s', '\S', '\w', '\W' e '\Z'. Por exemplo, em uma busca por uma sequência de números/dígitos, é possível utilizar a expressão '[0-9]+' ou simplesmente '\d', que são equivalentes. As classes de caracteres simplificam a escrita de padrões que poderiam requerer um número grande de combinações de metacaracteres.

Em *Python*, as operações com expressões regulares são fornecidas pelo módulo '*re*'. A figura a seguir mostra um exemplo de utilização de expressões regulares:

```
>>> import re
>>> texto = "O livro custa R$ 50,99 reais, mas só tenho R$32,50."
>>> padrao = re.compile("R\$ *[0-9]+,[0-9]+")
>>> padrao.findall(texto)
['R$ 50,99', 'R$32,50']
>>>
```

Figura 101 - Exemplo de uso de expressões regulares

Fonte: Produção do próprio autor

No exemplo, para a identificação de moeda, uma expressão regular foi compilada e aplicada pelo método '*findall*'. Os resultados da busca no texto são retornados em uma

lista. É possível identificar que o padrão utilizado retornou corretamente todos os valores encontrados no texto.

Nem sempre as expressões regulares retornam corretamente o que se deseja. Dois tipos de erros são possíveis: identificação de caracteres indesejados (falsos positivos) ou a não identificação de caracteres desejados (falsos negativos). Em Processamento de Línguas Naturais, é comum lidar com tais tipos de erros. Nesse sentido, a redução da taxa de erro de uma aplicação acarreta dois esforços antagônicos: o aumento da acurácia ou precisão (minimizando falsos positivos) e o aumento da cobertura ou abrangência (minimizando falsos negativos) (PUSTEJOVSKY e STUBBS, 2012, p. 27).

Mesmo assim, expressões regulares têm um papel importante no processamento de textos. Sequências sofisticadas de expressões regulares constituem a primeira escolha para qualquer processamento de texto. Para outras tarefas de grande complexidade, classificadores de aprendizado de máquina são desejáveis. Além disso, podem ser muito úteis para capturar generalizações.

Nesta pesquisa, expressões regulares são utilizadas nos diversos módulos de análise linguística, tanto no pré-processamento de textos quanto na extração de atributos para algoritmos de aprendizado de máquina.

f) Listas, dicionários e tuplas

Em *Python*, uma lista é uma estrutura que armazena uma sequência de valores. Esses valores podem ser de qualquer tipo, *strings*, números inteiros, números com pontos flutuantes, funções, classes e objetos em geral. Inclusive, uma lista pode armazenar outras listas do mesmo tipo, como mostra a figura a seguir:

```

>>> lista = ['amoroso', ['amor', 'oso'], 'ADJ', 7, 0.10]
>>> for i in range(0, len(lista)):
    print i, lista[i], type(lista[i])

0 amoroso <type 'str'>
1 ['amor', 'oso'] <type 'list'>
2 ADJ <type 'str'>
3 7 <type 'int'>
4 0.1 <type 'float'>
>>>

```

Figura 102- Exemplo de criação de lista e acesso a seus elementos

Fonte: Produção do próprio autor

No código, utiliza-se '[' e vírgulas para criar a lista. No exemplo, cada elemento da lista possui um tipo diferente. Os elementos podem ser acessados por meio de índices, assim como o acesso a caracteres em *strings*. Métodos específicos da linguagem permitem o corte, concatenação, adição e exclusão de elementos, uma vez que o tipo lista é mutável.

Uma tupla é uma lista de valores separados por vírgulas. Na declaração de uma tupla, é comum o uso de parênteses. As tuplas são similares às listas, exceto por seus valores serem imutáveis. As tuplas podem ser mais eficientes em relação à velocidade de processamento se comparadas às listas.

Os dicionários têm estrutura semelhante à das listas, uma vez que são mutáveis e podem armazenar dados de qualquer tipo. Um dicionário difere de uma lista na maneira como os elementos são indexados, isto é, por meio de índices que podem ser tanto valores numéricos como outros tipos de dados, criando uma relação de mapeamento em um par de chave e valor (*key, value*):

```

>>> dicionario = {'O': 'artigo', 'o': 'artigo', 'pegou': 'verbo', 'gato': 'substantivo',
    'rato': 'substantivo', '.': '.'}
>>> frase = 'O gato pegou o rato .'
>>> for item in frase.split(' '):
    print item, '->', dicionario[item]

O -> artigo
gato -> substantivo
pegou -> verbo
o -> artigo
rato -> substantivo
. -> .
>>>

```

Figura 103 - Exemplo de criação e uso de um dicionário

Fonte: Produção do próprio autor

No exemplo, utilizam-se '{}', dois pontos e vírgulas para criar um dicionário. O dicionário criado simula um pequeno léxico para classificar itens de uma frase com categorias morfológicas. Os itens são classificados a partir de sua busca no dicionário, em que a chave é o próprio item, e o valor retornado é a classificação.

Os dicionários também podem ser utilizados para contagem de itens. Por exemplo, é possível utilizá-los para contar o número de ocorrências de palavras, caracteres, entre outras necessidades, como é feito na pesquisa. Segue um exemplo simples:

```
>>> dicionario = {}
>>> for item in frase.split(' '):
    if item.lower() in dicionario:
        dicionario[item.lower()] += 1
    else:
        dicionario[item.lower()] = 1

>>> print dicionario
{'rato': 1, 'gato': 1, 'pegou': 1, 'o': 2, '.': 1}
>>>
```

Figura 104 - Exemplo de uso de um dicionário para contar as ocorrências de palavras

Fonte: Produção do próprio autor

No exemplo, foi criado um dicionário vazio para armazenar os itens e sua contagem. Para cada item da frase em caixa baixa, o programa verifica se o item já ocorreu no dicionário. Caso seja a primeira vez, o item é armazenado como chave no dicionário, e o valor '1' é atribuído à chave. Se o item já estiver armazenado como chave no dicionário, acrescenta-se '1' ao valor corrente. Os dados do dicionário impresso mostram a frequência de cada item. A partir de funções específicas, é possível classificar os dados em ordem crescente ou decrescente do dicionário, seja por chave ou valor.

As listas, dicionários e tuplas são as estruturas nativas da linguagem *Python*, utilizadas extensivamente na pesquisa para armazenar listas de palavras, léxicos, entre outras necessidades.

g) Criação de gráficos com o módulo matplotlib

O *matplotlib* é um módulo *Python*, criado por John Hunter, disponível no *site* <<http://matplotlib.org>>, o qual permite a fácil criação de gráficos 2D em vários formatos. A sua documentação *online* disponibiliza uma galeria de exemplos de gráficos juntamente com códigos comentados, sendo fácil a adaptação para as necessidades requeridas.

O módulo foi utilizado para auxiliar na visualização de resultados dos algoritmos de aprendizado de máquina e na análise de dados de *corpora*. Embora os recursos do *Microsoft Excel* sejam ótimos para a criação de gráficos e manipulação de dados extraídos de *corpora*, uma vez que algumas tarefas podem ser realizadas sem o constante esforço de programação (exploração de diferentes filtros e tipos de gráfico), há situações em que é necessária a programação de componentes personalizados: quando algum recurso não está disponível ou a utilização repetida demandaria mais esforço do que a tarefa de programação.

Um exemplo é a criação de gráficos para visualização da dispersão de palavras em um texto, um recurso que não está presente no *Microsoft Excel*. Para a programação de uma função para exibir a dispersão de palavras, utilizou-se o módulo *matplotlib* para desenhar um retângulo que representa o texto e linhas verticais dentro do retângulo para representar as ocorrências. Segue um exemplo hipotético em que o texto é:

"Dinosaur tracks recently discovered on the Arabian Peninsula are not only the first of their kind in the region, but they also reveal more about the herding behavior of the prehistoric beasts."

Fonte: (Scientific American, 2014)

O texto de exemplo possui um total de 34 itens. Para visualizar a dispersão das palavras '*the*' (cinco ocorrências) e '*of*' (duas ocorrências), divide-se a ordem de aparecimento de cada ocorrência das palavras selecionadas no texto pelo número total de itens, multiplicam-se os resultados por 100 e passam-se os valores como argumentos para a função '*plot_barcode*'.

```
>>> myplot = plotting()
>>> myplot.plot_barcode (title='Example', labels=['the', 'of'], positions=([17.64,
35.29, 52.94, 79.41, 91.17], [41.17, 88.23]))
```


Figura 105 - Exemplo de código para criar um gráfico de dispersão

Fonte: Produção do próprio autor

Como resultado da função, uma janela com os gráficos é exibida:

Figura 106 - Exemplo de gráfico de dispersão criado com o módulo *matplotlib*

Fonte: Produção do próprio autor

Outro exemplo é a criação de gráficos para análise de padrões estatísticos:

Figura 107 - Gráfico para visualização de probabilidades criado com o módulo *matplotlib*

Fonte: Produção do próprio autor

No gráfico, é mostrada a probabilidade de cada instância ser uma palavra cognata em relação ao seu tamanho em caracteres. Tal tipo de visualização permite o estudo e refinamento de possíveis atributos em algoritmos de aprendizagem. Por exemplo, a partir dos dados, é possível verificar que palavras com tamanho maior que seis caracteres têm probabilidade maior de serem cognatas. Desse modo, o tamanho da palavra em caracteres pode ser um atributo interessante a ser explorado na tarefa de identificação de palavras cognatas.

Além dos gráficos criados com o *matplotlib*, outros tipos de visualização foram possíveis de serem gerados a partir do módulo *NLTK*.

h) Acesso a páginas da web com o módulo *urllib*

A linguagem *Python* possui o módulo *urllib*, que fornece acesso a páginas da Internet. Com o *urllib* é possível acessar páginas de conteúdo interessante para pesquisa linguística, tal como dicionários *online*, páginas de buscadores, páginas de conteúdo da Wikipédia etc. Pode-se utilizá-lo também para automatizar a coleta de textos na Internet, além de acessar ferramentas de análise disponíveis como os analisadores do projeto *VISL*.

Como exemplo prático, apresenta-se a utilização de duas funções programadas durante o início do desenvolvimento da pesquisa para etiquetar textos com o etiquetador *VISL* para a língua inglesa. Uma função é responsável por acessar a página do etiquetador, passar os parâmetros necessários, como o texto a ser etiquetado, e retornar os resultados: uma página com códigos em *HTML* com a etiquetagem realizada. Outra função é responsável por extrair os resultados da etiquetagem, eliminando os códigos *HTML*. A função que limpa os códigos dos resultados é:

```
def clean_html(html):
    html = html.replace("\n", "").replace("</dl>", "\n").replace("<dt>", "\n<dt>")
    lines = html.split('\n')
    htmlpart = []
    for line in lines:
        if line.startswith("<dt>") == True:
            htmlpart.append(line + "\n")
    html = ''.join(htmlpart)
    html = html.replace("<dt>", "").replace("<dl>", "")
    html = html.replace('<font color="maroon">', "")
    html = html.replace('<font color="maroon">', "")
    html = html.replace('<font color="green">', "")
    html = html.replace('<font color="blue">', "")
    html = html.replace('<font color="blue">', "")
    html = html.replace('<font color="darkgreen">', "")
    html = html.replace('<font color="orange">', "")
    html = html.replace("</font>", "")
    html = html.replace("<b>", "").replace("</b>", "")
    html = html.replace("&lt;", "<").replace("&gt;", ">")
    html = html.replace(" ", " ").replace(" [", "\t[")
    html = html.replace(" ]", "\t]").replace(" @#", "#")
    html = html.replace(" #", "\t#").replace(" @", "\t@")
    html = re.sub('\t+', '\t', html)
    html = re.sub('\n+', '\n', html)
    lines = html.strip().split('\n')
    return lines
```

Figura 108 - Função para eliminar códigos de resultados do etiquetador *VISL*

Fonte: Produção do próprio autor

Para realizar a limpeza, a função possui uma série de instruções de substituição para retornar apenas os resultados da etiquetagem. A criação da função é conseguida a partir da análise da página em *HTML*, procedimento necessário para obter a estrutura exata dos resultados.

A função principal que acessa o etiquetador, passa os parâmetros e baixa os resultados é apresentada a seguir:

```
def visl_tag(sentence):
    params = urllib.urlencode({'parser': 'dep-eb', 'visual': 'niceline', 'text': sentence })
    u = urllib.urlopen('http://beta.visl.sdu.dk/visl/en/parsing/automatic/parse.php', params)
    html = u.read()
    tagged = clean_html(html)
    return tagged
```

Figura 109 - Função para acessar o etiquetador *VISL*

Fonte: Produção do próprio autor

A função tem como argumento uma sentença a ser etiquetada. A sentença é passada como parâmetro para o módulo *urllib*, assim como o endereço do etiquetador *online* e as opções de seu formulário. A função retorna a página com os resultados da etiquetagem, após a limpeza dos códigos. Um exemplo de utilização e resultados é exibido na figura:

```
>>> visl_tag("This is just an example.")
['This \t[this]\t<*> <dem> INDP S\t@SUBJ>\t#1->2', 'is \t[be]\t<DA:v\xc3\xa6re>
<mv> V PR 3S\t@FS-STA\t#2->0', 'just \t[just]\t<DA:lige> <atemp> ADV\t@>N\t#3->5
', 'an \t[a]\t<indef> ART S\t@>N\t#4->5', 'example \t[example]\t<DA:eksempel> <s
em-c> <ac-cat> <idf> <nhead> N S NOM\t@<SC\t#5->2', '.\t[.]\tPU\t@PU\t#6->0']
>>>
```

Figura 110 - Função para acessar o etiquetador *VISL*

Fonte: Produção do próprio autor

No exemplo, uma sentença de teste é passada à função '*visl_tag*' e os resultados são retornados e impressos na tela. A função programada pode perder sua validade a partir de possíveis alterações que o administrador do *site* do etiquetador *VISL* possa realizar na página.

Para esta pesquisa, o módulo *urllib* foi especificamente necessário para baixar recursos de um dicionário para a criação do módulo de análise de palavras cognatas.

i) A biblioteca Natural Language Toolkit (NLTK)

Na pesquisa, utilizou-se o *Natural Language Toolkit* para uma série de processamentos, conforme as necessidades de desenvolvimento do trabalho. A versão utilizada foi a 2.7.6 (final), incluindo os *corpora* disponíveis para *download*.

O *NLTK* é uma plataforma que permite a criação de programas para o Processamento de Línguas Naturais. Uma série de bibliotecas e módulos estão disponíveis, que permitem realizar tarefas, como: análise em *corpora* (frequência, concordâncias, extração de colocados, n-gramas, dispersão de palavras), funções de alinhamento, pré-processamento básico de textos (segmentação, itemização, lematização e etiquetagem), suporte para criação de gramáticas, léxicos e representações semânticas, classificadores e funções de aprendizado de máquina, funções de métrica para avaliação, além de recursos linguísticos e *corpora*. Embora as ferramentas do *NLTK* possam ser utilizadas para uma variedade de línguas diferentes, grande parte dos recursos linguísticos, como *corpora* e léxicos, está disponível apenas para o processamento da língua inglesa.

O *NLTK* está disponível para *download* no sítio <www.nltk.org>. No sítio, há toda a documentação dividida por módulos e tutoriais. É possível também fazer o *download* gratuito do livro *Natural Language Processing with Python* (BIRD, KLEIN e LOPER, 2009). O livro pode ser considerado uma introdução ao PLN por meio do *NLTK*.

Tendo em vista a gama de ferramentas e possibilidades disponíveis no *NLTK*, comentam-se os principais recursos que foram utilizados na pesquisa. Para tanto, apresentam-se alguns exemplos criados para melhor compreensão do suporte que o *NLTK* fornece às análises de *corpora*, Processamento de Línguas Naturais e Aprendizado de Máquina.

Com o *NLTK*, não é muito difícil realizar tarefas como itemização, etiquetagem e até classificação de entidades nomeadas com poucas linhas de código. Há uma série de recursos e ferramentas já prontas para uso, como mostra o código na figura a seguir:

```
import nltk
texto = '''
President Barack Obama is in New York City for a meeting with Vladimir Putin.
'''
texto_etiquetado = nltk.pos_tag(nltk.word_tokenize(texto))
analise = nltk.ne_chunk(texto_etiquetado)
analise.draw()
```

Figura 111 - Exemplo de código para análise de uma sentença com o *NLTK*

Fonte: Produção do próprio autor

No código, as funções '*word_tokenize*' e '*pos_tag*' fazem a etiquetagem da sentença, a função '*ne_chunk*' classifica as entidades nomeadas, e a função '*draw*' abre uma janela com as análises. O resultado é a representação em árvore da sentença analisada:

Figura 112 - Representação em árvore de sentença analisada

Fonte: Produção do próprio autor

No gráfico, estão em destaque as entidades nomeadas identificadas: '*Barack Obama*' (pessoa), '*New York City*' (localização) e '*Vladmir Putin*' (pessoa). As classes de entidades mais comuns utilizadas em sistemas são: *ORGANIZATION*, *PERSON*, *LOCATION*, *DATE*, *TIME*, *MONEY*, *PERCENT*, *FACILITY* e *GPE*.

Nem sempre os recursos são completos, compatíveis ou adequados para as soluções desejadas. Por exemplo, o classificador de entidades nomeadas disponível pode ser limitado para a identificação de certas entidades em determinados gêneros ou registros de textos. Muitas vezes, é necessário complementar, personalizar ou construir etiquetadores e classificadores para as necessidades requeridas.

O *NLTK* possui diversas opções para a criação de classificadores e etiquetadores de palavras: *DefaultTagger*, *RegexTagger*, *UnigramTagger*, *BigramTagger*, *TrigramTagger*, *BrillTagger*, além de outros classificadores.

A criação de etiquetadores requer dados de treinamento no formato de sentenças já etiquetadas, em listas de tuplas para o *NLTK*. A partir dos dados, o algoritmo é treinado e como resultado um etiquetador é retornado, o qual pode ser armazenado para usos futuros, utilizando o módulo '*pickle*'³³, uma vez que o treinamento do algoritmo leva um tempo considerável e geralmente não é preciso treinar um etiquetador para cada uso.

Para ilustrar o processo de criação de etiquetadores com o *NLTK*, em um exemplo criado, apresenta-se o código comentado a seguir:

³³O módulo '*pickle*' fornece funções para armazenar e carregar qualquer tipo de objeto ou dados na linguagem *Python*, sem qualquer tipo de conversão para tanto.

```

# sentenças para treinamento
sents = []
sents.append([('I', 'PRP'), ('like', 'VBP'), ('dogs', 'NNS'), ('.', '.')])
sents.append([('My', 'PRP$'), ('friend', 'NN'), ('has', 'VBZ'), ('a', 'DT'),
              ('little', 'RB'), ('dog', 'JJ'), ('.', '.')])
sents.append([('Her', 'PRP$'), ('dog', 'NN'), ('is', 'VBZ'), ('very', 'RB'),
              ('beautiful', 'JJ'), ('.', '.')])
sents.append([('The', 'DT'), ('dog', 'NN'), ('is', 'VBZ'), ('like', 'IN'),
              ('a', 'DT'), ('wolf', 'NN'), ('.', '.')])
# treinamento dos etiquetadores
default_etiq = nltk.tag.DefaultTagger('NN')
regexp_etiq = nltk.RegexpTagger([(r'.*ed$', 'VBD'), (r'.*s$', 'NNS'), (r'.*ing$', 'VBG')], backoff=default_etiq)
unigram_etiq = nltk.UnigramTagger(sents, backoff=regexp_etiq)
bigram_etiq = nltk.BigramTagger(sents, backoff=unigram_etiq)
# teste com cada etiquetador
sent_teste = ('Her', 'beautiful', 'cat', 'is', 'like', 'a', 'little', 'tiger', '.')
print default_etiq.tag(sent_teste)
print
print regexp_etiq.tag(sent_teste)
print
print unigram_etiq.tag(sent_teste)
print
print bigram_etiq.tag(sent_teste)

```

Figura 113 - Exemplo de criação de etiquetadores em sequência

Fonte: Produção do próprio autor

No código, inicialmente, foram utilizadas quatro sentenças simples como dados de treinamento, armazenadas na lista 'sents'. Em seguida, treinados quatro etiquetadores diferentes, em sequência (*DefaultTagger*, *RegexpTagger*, *UnigramTagger* e *BigramTagger*). O etiquetador '*DefaultTagger*' atribui uma etiqueta padrão ('NN') para todas as palavras. O etiquetador '*RegexpTagger*' utiliza expressões regulares fornecidas em uma lista para etiquetar as palavras. O etiquetador '*UnigramTagger*', treinado com as sentenças, atribui a etiqueta mais provável para a palavra a partir de um dicionário criado internamente. O etiquetador '*BigramTagger*', também treinado com as sentenças etiquetadas, atribui a etiqueta mais provável para a palavra com base na etiqueta anterior (hipótese de *Markov*). A combinação dos etiquetadores é feita sequencialmente por meio do argumento '*backoff*'. Por último, foi testado cada etiquetador com a sentença '*Her beautiful cat is like a little tiger.*', na qual apenas as palavras '*cat*' e '*tiger*' são desconhecidas dos dados de treinamento. O resultado da anotação é:

```

[('Her', 'NN'), ('beautiful', 'NN'), ('cat', 'NN'), ('is', 'NN'), ('like', 'NN'), ('a', 'NN'), ('little', 'NN'), ('tiger', 'NN'), ('.', 'NN')]

[('Her', 'NN'), ('beautiful', 'NN'), ('cat', 'NN'), ('is', 'NNS'), ('like', 'NN'), ('a', 'NN'), ('little', 'NN'), ('tiger', 'NN'), ('.', 'NN')]

[('Her', 'PRP$'), ('beautiful', 'JJ'), ('cat', 'NN'), ('is', 'VBZ'), ('like', 'VBP'), ('a', 'DT'), ('little', 'RB'), ('tiger', 'NN'), ('.', '.')]

[('Her', 'PRP$'), ('beautiful', 'JJ'), ('cat', 'NN'), ('is', 'VBZ'), ('like', 'IN'), ('a', 'DT'), ('little', 'RB'), ('tiger', 'NN'), ('.', '.')]
>>>

```

Figura 114 - Exemplo de sentenças anotadas com etiquetadores

Fonte: Produção do próprio autor

Na etiquetagem realizada pelos quatro etiquetadores treinados, é possível perceber os diferentes estágios de adequação. Na primeira etiquetagem, realizada pelo etiquetador '*DefaultTagger*', todas as palavras recebem a mesma etiqueta ('*NN*'), uma vez que a taxa de acertos para palavras desconhecidas é estatisticamente significativa. Na segunda, com a combinação dos etiquetadores '*DefaultTagger*' e '*RegexTagger*', a palavra '*is*', terminada em '*s*', recebe a etiqueta '*NNS*', etiqueta inadequada. Na terceira, com a combinação dos dois primeiros etiquetadores e o etiquetador '*UnigramTagger*', a etiquetagem é adequada para quase todas as palavras, com exceção da palavra '*like*', classificada como '*VBP*'. Na última etiquetagem, com a combinação de todos os etiquetadores, a palavra '*like*' é etiquetada corretamente como '*IN*', dada a consideração da etiqueta anterior para sua definição.

Embora os exemplos práticos apresentados sejam simples e limitados, é possível perceber a facilidade em treinar etiquetadores. É importante salientar que os recursos ilustrados não estão limitados apenas à etiquetagem morfossintática, mas aptos a possibilitar uma série de funções de classificação, assim como a extensão e personalização das opções de algoritmos são possíveis. O *NLTK* fornece classes para a transformação de classificadores utilizados em aprendizado supervisionado em etiquetadores que podem ser combinados.

A criação de classificadores é feita a partir de dados de treinamento. Para exemplificar, consideram-se os dados classificados na tabela a seguir:

Tabela 13 - Exemplo de dados classificados e sua conversão em números

#	T	T_1	T1	Classificação	T	T_1	T1	Classificação
1	DT	<s>	JJ	B-NP	3	11	4	1
2	JJ	DT	NN	I-NP	4	3	5	2
3	NN	JJ	VBD	I-NP	5	4	17	2
4	VB	TO	NN	O	7	10	5	0
5	IN	NNS	NN	O	8	5	5	0
6	JJ	<s>	NN	B-NP	4	11	5	1
7	NN	NN	NN	I-NP	5	5	5	2
8	VBN	VBZ	TO	O	9	12	10	0
9	VBN	NN	IN	O	9	5	8	0
10	VBN	JJ	NN	I-NP	9	4	5	2
11	VBN	VBG	IN	O	9	13	8	0
12	VBN	JJ	CD	I-NP	9	4	18	2
13	VBN	IN	NN	B-NP	9	8	5	1
14	VBN	RB	IN	O	9	14	8	0
15	JJ	CC	.	O	4	15	19	0
16	JJ	IN	JJ	B-NP	4	8	4	1
17	JJ	PRP\$	NN	I-NP	4	16	5	2
18	TO	VBD	JJ	O	10	17	4	0
19	DT	VB	JJ	B-NP	3	7	4	1
20	DT	IN	NN	B-NP	3	8	5	1

Fonte: Produção do próprio autor

A tabela da esquerda apresenta 20 instâncias de dados classificados para identificação de grupos nominais simples (*simple noun phrases*) a partir de anotação morfosintática, em que a coluna 'T' refere-se à etiqueta foco, a coluna 'T_1' refere-se à primeira etiqueta à esquerda, a coluna 'T1' refere-se à primeira etiqueta à direita, e a coluna 'Classificação' indica se a etiqueta foco inicia um grupo nominal (B-NP), está dentro de um grupo nominal (I-NP), ou fora (O).

A tabela da direita apresenta os mesmos dados em valores numéricos. Muitas vezes, para o processamento computacional, é necessário transformar dados de texto em valores numéricos por questões de otimização, visto que o computador lida melhor com números. No *NLTK*, há funções e algoritmos que requerem a conversão de textos em valores numéricos. Para o exemplo, foi considerada a tabela que possui os dados em tipo texto.

Tendo os dados anotados juntamente com seus atributos (T, T_1 e T1), que são propriedades individuais dos dados observados para auxiliar na determinação da classificação, pode-se selecionar entre vários tipos de classificadores para treinamento. O código a seguir mostra o treinamento e avaliação de três diferentes classificadores

para a mesma tarefa, disponíveis no *NLTK*: *DecisionTreeClassifier*, *NaiveBayesClassifier* e *MaxentClassifier*.

```
# carrega dados de treinamento
dados = carrega_dados(tabela01)
# formata os dados
treinamento = []
for d in dados:
    treinamento.append((dict(T=d[0],T_1=d[1],Tl=d[2]), d[3]))
# treina classificadores
dt_classifier = nltk.classify.DecisionTreeClassifier.train(treinamento)
bayes_classifier = nltk.classify.NaiveBayesClassifier.train(treinamento)
max_classifier = nltk.classify.MaxentClassifier.train(treinamento)
# carrega dados para teste
teste = []
teste.append((dict(T='VBN',T_1='JJ',Tl='NN'),'I-NP') )
teste.append((dict(T='TO',T_1='IN',Tl='NN'),'O'))
teste.append((dict(T='DT',T_1='CC',Tl='NN'),'B-NP'))
teste.append((dict(T='JJ',T_1='NN',Tl='NN'),'I-NP'))
teste.append((dict(T='VBN',T_1='PRPS',Tl='NN'),'I-NP'))
# verifica acurácia de cada classificador
print 'Decision tree: ', nltk.classify.accuracy(dt_classifier,teste)
print 'Naive Bayes: ',nltk.classify.accuracy(bayes_classifier,teste)
print 'Maxent: ',nltk.classify.accuracy(max_classifier,teste)
print dt_classifier
```

Figura 115 - Exemplo de criação de classificadores

Fonte: Produção do próprio autor

No código, primeiro, os dados da tabela são carregados para uma variável do tipo lista, em que cada instância é um item da lista. Em seguida, as instâncias são colocadas em um formato específico para o treinamento nos algoritmos do *NLTK*, armazenadas na variável tipo lista 'treinamento'. O formato requer uma tupla com os rótulos e valores dos atributos em um dicionário e a classificação das instâncias em separado. Depois, os classificadores são treinados a partir desses dados. Por último, criam-se cinco instâncias de dados não conhecidos para testar a acurácia de cada classificador. Os resultados impressos de cada classificador juntamente com os dados da árvore de decisão criada são exibidos na figura a seguir:

```

Decision tree:  0.4
Naive Bayes:   0.8
Maxent:        0.8
T_1=<s>? ..... B-NP
T_1=CC? ..... O
T_1=DT? ..... I-NP
T_1=IN? ..... B-NP
T_1=JJ? ..... I-NP
T_1=NN? ..... O
  T=NN? ..... I-NP
  T=VBN? ..... O
T_1=NNS? ..... O
T_1=PRP$? ..... I-NP
T_1=RB? ..... O
T_1=TO? ..... O
T_1=VB? ..... B-NP
T_1=VBD? ..... O
T_1=VBG? ..... O
T_1=VBZ? ..... O

```

Figura 116 - Avaliação de classificações e dados de árvore de decisão em texto

Fonte: Produção do próprio autor

O resultados mostraram que os classificadores *NaiveBayesClassifier* (80%) e *MaxentClassifier* (80%) tiveram um desempenho muito melhor que o classificador *DecisionTreeClassifier* (40%). Abaixo da avaliação, está impressa em texto a árvore de decisão criada a partir do treinamento. Essa é uma vantagem da utilização do classificador *DecisionTreeClassifier*, é possível facilmente acessar o conhecimento adquirido e até modificá-lo, o que é inviável para alguns tipos de algoritmos de aprendizado de máquina. A decisão sobre qual classificador utilizar depende de uma série de fatores, como o tipo de tarefa, quais dados de treinamento estão disponíveis, o número e tipos de atributos.

Uma tarefa diferente das realizadas pelos classificadores apresentados, que, a partir de dados já classificados, conseguem fazer predições sobre a classificação de dados não conhecidos (aprendizado supervisionado), é o agrupamento de dados quando não existem dados de treinamento (*clustering*). Basicamente, como não há uma classificação prévia, agrupam-se os dados que são similares em categorias. Esse agrupamento é um tipo de aprendizado não supervisionado. No *NLTK*, o algoritmo disponível é o *k-means*, em que *k* é o número de agrupamentos que se pretende encontrar.

Como exemplo de uso do algoritmo *k-means* do *NLTK*, considerando os dados numéricos de treinamento para identificação de grupos nominais, apresenta-se o código abaixo:

```
dados = carrega_dados(tabela02)
vetores = []
for d in dados:
    vetores.append([int(d[0]),int(d[1]),int(d[2])])
vetores = numpy.array(vetores)

clusterer = nltk.cluster.kmeans.KMeansClusterer(3,nltk.euclidean_distance, initial_means)
clusters = clusterer.cluster(vetores, True, trace=True)
print 'Clustered:', vetores
print 'As:', clusters
print 'Means:', clusterer.means()
print ''
```

Figura 117 - Exemplo de utilização do algoritmo *KMeansClusterer* do *NLTK*

Fonte: Produção do próprio autor

No código, primeiro, os dados da tabela de valores numéricos, pois o algoritmo aceita apenas números, são carregados para uma lista de vetores. Como o objetivo é demonstrar o agrupamento feito pelo algoritmo, apenas os atributos são incluídos (T, T_1 e T1). Em seguida, a lista de vetores é transformada no formato requerido pelo *NLTK*, isto é, convertida para o tipo de vetor do módulo '*NumPy*'. O vetor convertido é então utilizado para treinar o algoritmo '*KMeansClusterer*'. Por último, os resultados de agrupamento são impressos, como mostra a figura:

```
Clustered: [[ 3 11  4]
 [ 4  3  5]
 [ 5  4 17]
 [ 7 10  5]
 [ 8  5  5]
 [ 4 11  5]
 [ 5  5  5]
 [ 9 12 10]
 [ 9  5  8]
 [ 9  4  5]
 [ 9 13  8]
 [ 9  4 18]
 [ 9  8  5]
 [ 9 14  8]
 [ 4 15 19]
 [ 4  8  4]
 [ 4 16  5]
 [10 17  4]
 [ 3  7  4]
 [ 3  8  5]]
As: [2, 0, 0, 2, 0, 2, 0, 1, 0, 0, 1, 1, 0, 1, 1, 2, 2, 1, 2, 2]
Means: [array([ 7.          ,  4.85714286,  7.14285714]), array([ 8.33333333, 12.
  5.          , 11.16666667]), array([ 4.          , 10.14285714,  4.57142857])]

>>>
```

Figura 118 - Exemplo de agrupamentos com '*KMeansClusterer*' do *NLTK*

Fonte: Produção do próprio autor

Os resultados apresentam o vetor convertido (*Clustered:*), o agrupamento realizado (*As:*) e as probabilidades (*Means:*). O número de k agrupamentos definido foi de três, tendo em vista que as classificações dos dados (*B-NP*, *I-NP* e *O*) já eram conhecidas. A convenção utilizada no resultado para indicar os agrupamentos são os índices numéricos '0', '1' e '2'.

Para melhor visualização dos resultados dos agrupamentos realizados, apresenta-se a tabela reordenada com os índices de agrupamento:

Tabela 14 - Dados reorganizados a partir do agrupamento realizado

#	T	T_1	T1	Classificação	Agrupamento
2	JJ	DT	NN	I-NP	0
3	NN	JJ	VBD	I-NP	0
5	IN	NNS	NN	O	0
7	NN	NN	NN	I-NP	0
9	VDN	NN	IN	O	0
10	VDN	JJ	NN	I-NP	0
13	VDN	IN	NN	B-NP	0
8	VDN	VBZ	TO	O	1
11	VDN	VBG	IN	O	1
12	VDN	JJ	CD	I-NP	1
14	VDN	RB	IN	O	1
15	JJ	CC	.	O	1
18	TO	VBD	JJ	O	1
1	DT	<s>	JJ	B-NP	2
4	VB	TO	NN	O	2
6	JJ	<s>	NN	B-NP	2
16	JJ	IN	JJ	B-NP	2
17	JJ	PRPS	NN	I-NP	2
19	DT	VB	JJ	B-NP	2
20	DT	IN	NN	B-NP	2

Fonte: Produção do próprio autor

Na tabela, é possível comparar o agrupamento realizado com a classificação dos dados. Como a classificação não foi incluída no treinamento do algoritmo e apenas alguns atributos foram levados em consideração, é compreensível que o agrupamento não seja preciso em agrupar adequadamente as classes '*B-NP*', '*I-NP*' e '*O*'. Contudo, é perceptível o agrupamento de dados similares, em que o índice '0' estaria relacionado mais à classe '*I-NP*'; o índice '1', à classe '*O*'; e o índice '2', à classe '*B-NP*'. O simples exemplo mostra a utilidade do algoritmo para busca e identificação de padrões, uma tarefa essencial para trabalhos com análise de *corpora*.

Além dos módulos de pré-processamento de textos, etiquetadores e classificadores, o trabalho faz uso de uma série de recursos linguísticos do *NLTK*, por exemplo, o uso dos *corpora* Floresta Sinta(c)tica e MacMorpho, descritos na próxima seção, que apresenta os *corpora* utilizados na pesquisa.

APÊNDICE B - Código do módulo affixes

[illegible]

APÊNDICE C - Código do módulo cogtagger

```

import os

class cogtagger:

    def __init__(self, language='en'):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language
        self.__negative_words = self.__load_list(self.__path + '/data/' +
self.language + '/negative_words.txt' )
        self.__positive_roots = self.__load_list(self.__path + '/data/' +
self.language + '/positive_roots.txt' )
        self.__false_friends = self.__load_list(self.__path + '/data/' +
self.language + '/false_friends.txt' )

    def __load_list(self, path):
        items = []
        for line in open(path, 'r').readlines():
            if len(line.strip()) > 0:
                items.append(line.strip())
        return items

    def get_cognates_from_tokens(self, list_of_tokens):
        cognates = {}
        for token in list_of_tokens:
            if token not in cognates:
                cognates[token] = self.tag(token)

        return cognates

    def tag(self, word):
        word = word.lower()
        if word in self.__false_friends:
            tag = '2'
        elif word in self.__negative_words:
            tag = '0'
        elif len(word) < 5:
            tag = '0'
        elif word[0:5] in self.__positive_roots:
            tag = '1'
        else:
            tag = '0'
        return tag

```


APÊNDICE D - Código do módulo counter

```

import os
import math

class counter:

    def __init__(self,language='en'):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language
        self.LOWER = 'lower'
        self.UPPER = 'upper'
        self.NONE = ''

    def count_from_list(self,items, case = ''):
        dic = {}
        number_of_tokens=0
        for item in items:
            number_of_tokens+=1
            if case == '':
                if item in dic:
                    dic[item]+=1
                else:
                    dic[item]=1
            elif case == 'lower':
                if item.lower() in dic:
                    dic[item.lower()]+=1
                else:
                    dic[item.lower()]=1
            elif case == 'upper':
                if item.upper() in dic:
                    dic[item.upper()]+=1
                else:
                    dic[item.upper()]=1
        number_of_types = len(dic)
        try: # division by zero error
            type_token_ratio = float(number_of_types) / float(number_of_tokens)
        except:
            type_token_ratio =0.0

        return dic,number_of_types,number_of_tokens,type_token_ratio

    def count_words_from_tokens(self,list_of_tokens):
        import string
        word_count = 0
        for word in list_of_tokens:
            if word not in string.punctuation:
                word_count+=1

        return word_count

    def count_characters_from_tokens(self,list_of_tokens):
        import string
        char_count = 0
        for word in list_of_tokens:
            char_count+= len(word)

        return char_count

```

APÊNDICE E - Código do módulo keywordsextractor

```

import os
import math

class keywordsextractor:

    def __init__(self,language='en'):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language
        self.__load_reference_list()

    def __load_reference_list(self):
        self.reflist = {}
        self.refntokens = 0
        for line in open(self.__path + '/data/' + self.language +
            '/reflist.txt','r').readlines():
            if len(line.strip()) > 0:
                fields= line.strip().split('\t')
                if len(fields) > 1:
                    self.reflist[fields[0].strip()] = int(fields[1].strip())
                    self.refntokens+= int(fields[1].strip())

    def extract(self, word, word_frequency, number_of_tokens):
        if word.lower() in self.reflist :
            ifreqincr = self.reflist[word.lower()]
        else:
            ifreqincr = 1
        ll = self.__loglikelihood(number_of_tokens, word_frequency,
            self.refntokens, ifreqincr)
        return ll

    def __loglikelihood(self,ntokensinsc,ifreqiinsc, ntokensincr,ifreqincr):
        #convert to float
        nt1 = float(ntokensinsc)+1#number of tokens in study corpus
        nt2 = float(ntokensincr)+1#number of tokens in ref corpus
        f1 = float(ifreqiinsc)+1#freq of item in study corpus
        f2 = float(ifreqincr) +1#freq of item in ref corpus
        #calculate
        e1 = nt1 * ((f1+f2) / (nt1+nt2))
        e2 = nt2 * ((f1+f2) / (nt1+nt2))
        ll = 2 * ( (f1 * math.log( f1/e1,2 ) ) + (f2 * math.log(f2/e2,2) ) )
        #return value
        return int(round(ll))

```

APÊNDICE F - Código do módulo netagger

```

import os
import re
import nltk

class netagger:

    def __init__(self, language='en'):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language
        self.__load_rules()

    def __load_rules(self):
        pass

    def __IO(self, tag):
        tag = re.sub('B-|I-', '', tag)
        return tag

    def tag(self, sentence):
        try:
            tagged = nltk.tree2conlltags(nltk.ne_chunk(sentence))
            entities = []
            for i in range(0, len(tagged)):
                entities.append(self.__IO(tagged[i][2]))
        except:
            entities = []
            for i in range(0, len(sentence)):
                entities.append('0')

        return entities

```

APÊNDICE G - Código do módulo nptagger

```

import os
import nltk
try:
    import cPickle as pickle
except:
    import pickle

class nounphraseclassifier(nltk.TaggerI):

    def __init__(self, backoff=None, language='en'):
        if backoff is None:
            self._taggers = [self]
        else:
            self._taggers = [self] + backoff._taggers

        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language

        try:
            self.__np_classifier = self.__load_classifier( self.__path + '/data/'
+ self.language + '/np_classifier.pkl')
        except:
            self.__np_classifier = None

    def backoff(self):
        """The backoff tagger for this tagger."""
        return self._taggers[1] if len(self._taggers) > 1 else None

    def tag_one(self, tokens, index, history):
        tag = None
        for tagger in self._taggers:
            tag = tagger.choose_tag(tokens, index, history)
            if tag is not None: break
        return tag

    def choose_tag(self, tokens, index, history):
        tagged = self.tag(tokens)
        tag = tagged[index][1]
        return tag

    def batch_tag(self, sentences):
        tagged = []
        for sentence in sentences:
            tagged.append(self.tag(sentence))
        return tagged

    def get_features(self, token):
        arr = token.split('/')
        features = dict(W=arr[0], T=arr[1])
        return features

    def get_features_train(self, token):
        arr = token[0].split('/')
        features = dict(W=arr[0], T=arr[1])
        return features

```

```

def train(self,tagged_sentences):
    features_sets=[]
    for sentence in tagged_sentences:
        for token in sentence:
            features = (self.get_features_train(token),token[1])
            features_sets.append(features)
    classifier =
nltk.classify.MaxentClassifier.train(features_sets,max_iter=100)

    self.__save_classifier(classifier, self.__path + '/data/' + self.language
+ '/np_classifier.pkl')
    print 'Trained!'

def tag(self,tokens):
    tagged =[]
    for i in range(0, len(tokens)):
        features = self.get_features(tokens[i])
        tag = self.__np_classifier.classify(features)
        tagged.append((tokens[i],tag))

    return tagged

def __save_classifier(self,classifier,filename):
    f = open(filename,'wb')
    pickle.dump(classifier,f)
    f.close()

def __load_classifier(self,filename):
    f = open(filename,'rb')
    classifier = pickle.load(f)
    f.close()

    return classifier

```

APÊNDICE H - Código do módulo postagger

```

import os
import nltk
from operator import itemgetter
try:
    import cPickle as pickle
except:
    import pickle
import wfe
import cfe

# -----
# -----
# -----
# -----

class clstagger(nltk.tag.TaggerI):

    def __init__(self, language='en'):

        self.language = language
        self.__path = os.path.dirname(os.path.abspath(__file__))

        self.__load_classifiers()

    def __load_classifiers(self):
        try:
            self.__wordtag_dict = self.__load_classifier(self.__path + '/data/' +
self.language + '/wordtag_dict.pkl')
        except:
            'wordtag_dict.pkl not loaded (Ignore it if training)'
        try:
            self.__unk_classifier = self.__load_classifier(self.__path + '/data/'
+ self.language + '/unknown_classifier.pkl')
        except:
            'unkown_classifier.pkl not loaded (Ignore it if training)'
        try:
            self.__context_classifier = self.__load_classifier(self.__path +
'/data/' + self.language + '/context_classifier.pkl')
        except:
            'context_classifier.pkl not loaded (Ignore it if training)'

    def __load_classifier(self, filename):
        f = open(filename, 'rb')
        classifier = pickle.load(f)
        f.close()
        return classifier

    def tag_one(self, tokens, index, history):
        tag = None
        for tagger in self._taggers:
            tag = tagger.choose_tag(tokens, index, history)

```

```

        if tag is not None: break
    return tag

def choose_tag(self, tokens, index, history):
    tagged = self.tag(tokens)
    tag = tagged[index][1]
    return tag

def batch_tag(self, sentences):
    tagged = []
    for sentence in sentences:
        tagged.append(self.tag(sentence))
    return tagged

def tag(self, tokens):
    tagged = []
    for i in range(0, len(tokens)):
        if tokens[i] in self.__wordtag_dict :
            tag = self.__wordtag_dict[tokens[i]]
        elif tokens[i].lower() in self.__wordtag_dict:
            tag = self.__wordtag_dict[tokens[i].lower()]
        else:
            features = (wfe.get_word_features(tokens[i]))
            tag = self.__unk_classifier.classify(features)
        tagged.append((tokens[i], tag))

    for i in range(0, len(tagged)):
        features = cfe.get_4wordtag_features(tagged, i)
        tag = self.__context_classifier.classify(features)
        tagged[i] = (tagged[i][0], tag)

    return tagged

def firsttag(self, tokens):
    tagged = []
    for i in range(0, len(tokens)):
        if tokens[i] in self.__wordtag_dict :
            tag = self.__wordtag_dict[tokens[i]]
        elif tokens[i].lower() in self.__wordtag_dict:
            tag = self.__wordtag_dict[tokens[i].lower()]
        else:
            features = (wfe.get_word_features(tokens[i]))
            tag = self.__unk_classifier.classify(features)
        tagged.append((tokens[i], tag))
    return tagged

def train(self, trainset):
    if not os.path.exists(self.__path + '/data/' + self.language):
        os.makedirs(self.__path + '/data/' + self.language)
    if self.language == 'en':
        pass
    else:
        self.__make_wordtag_dictionary(trainset)
        self.__make_unknown_classifier(trainset)
        self.__make_3words_classifier(trainset)
        self.__make_context_classifier(trainset)
        self.__load_classifiers()

```

```

def __make_context_classifier(self, trainset):
    sentences = []
    for sentence in trainset:
        tokens = []
        for token in sentence:
            tokens.append(token[0])
        sentences.append(tokens)

    tagged = []
    for sentence in sentences:
        tagged.append(self.firsttag(sentence))

    features_sets = []
    for i in range(0, len(tagged)):
        for j in range(0, len(tagged[i])):
            features =
(cfe.get_4wordtag_features(tagged[i], j), trainset[i][j][1])
            features_sets.append(features)

    classifier = nltk.classify.NaiveBayesClassifier.train(features_sets)

    #classifier = nltk.maxent.train_maxent_classifier_with_gis(features_sets,
max_iter=25)
    self.__save_classifier(classifier, self.__path + '/data/' + self.language +
'/context_classifier.pkl')

def __make_wordtag_dictionary(self, sentences):
    dic = {}
    for i in range(0, len(sentences)):
        for j in range(0, len(sentences[i])):
            word = sentences[i][j][0]
            tag = sentences[i][j][1]
            key = (word, tag)
            if key in dic:
                dic[key] += 1
            else:
                dic[key] = 1
    wordtag = {}
    for k in sorted(dic.iteritems(), key=itemgetter(1), reverse = True):
        word = k[0][0]
        tag = k[0][1]
        if word not in wordtag:
            wordtag[word] = tag
    self.__save_classifier(wordtag, self.__path + '/data/' + self.language +
'/wordtag_dict.pkl')
    print 'Total items: ', len(wordtag)
    print 'Done!'

def __make_unknown_classifier(self, sentences):
    features_set = []
    for i in range(0, len(sentences)):
        for j in range(0, len(sentences[i])):
            features = (dict(wfe.get_word_features(sentences[i][j][0]))
, sentences[i][j][1])
            features_set.append(features)

```



```

        classifier = nltk.NaiveBayesClassifier.train(features_set)
        #classifier = nltk.maxent.train_maxent_classifier_with_gis(features_set,
max_iter=250)
        print nltk.classify.accuracy(classifier, features_set)
        self.__save_classifier(classifier,self.__path + '/data/' + self.language +
'/unknown_classifier.pkl')

    def __make_3words_classifier(self,sentences):

        dic = {}
        for i in range(0,len(sentences)):
            for j in range(0,len(sentences[i])):
                features = cfe.get_3words_tpl(sentences[i], j)
                key = (features[0],features[1],features[2],sentences[i][j][1])
                if key in dic:
                    dic[key]+=1
                else:
                    dic[key]=1
        wordtag = {}
        cutoff = 1
        tridic = {}
        for k in sorted(dic.iteritems(),key=itemgetter(1),reverse = True):
            if k[1] > cutoff:
                key = (k[0][0],k[0][1],k[0][2])
                value = k[0][3]
                if key not in tridic:
                    tridic[key] = value

        self.__save_classifier(tridic,self.__path + '/data/' + self.language +
'/3words_dict.pkl')
        print 'Total',len(tridic)

    def __save_classifier(self,classifier,filename):
        f = open(filename,'wb')
        pickle.dump(classifier,f)
        f.close()

```

APÊNDICE I - Código do módulo segmenter

```

import os
import nltk
import re
try:
    import cPickle as pickle
except:
    import pickle

class segmenter:

    def __init__(self,language='en'):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language
        try:
            url = 'file:%s' % nltk.data.find( self.__path + '/data/' +
self.language + '/tokenizer.pickle')
            self.tok = nltk.data.load(url)
        except:
            self.tok = pickle.load(open(self.__path + '/data/' + self.language +
'/tokenizer.pickle'))

    def segment(self,text):
        text = re.sub('\n+','\n',text)
        paragraphs = text.strip().split('\n')
        segmentation = []
        for paragraph in paragraphs:
            if len(paragraph.strip()) != 0:
                sentences = self.tok.tokenize(paragraph)
                segmentation.append(sentences)
        return segmentation

```

APÊNDICE J - Código do módulo stemmer

```
from nlk.stem.porter import *
from nlk.stem.snowball import SnowballStemmer

class stemmer:

    def __init__(self, language='en'):
        self.language = language
        if self.language == 'en':
            self.mystemmer = PorterStemmer()

    def stem(self, word):
        return self.mystemmer.stem(word)
```

APÊNDICE K - Código do módulo tokenizer

```

import os
import nltk
try:
    import cPickle as pickle
except:
    import pickle

class tokenizer:

    def __init__(self, language='en'):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language
        try:
            if language == 'en':
                self.tok = nltk.TreebankWordTokenizer()
            else:
                url = 'file:%s' % nltk.data.find( self.__path + '/data/' +
self.language + '/tokenizer.pickle')
                self.tok = nltk.data.load(url)
        except:
            self.tok = pickle.load(open(self.__path + '/data/' + self.language +
'/tokenizer.pickle'))

    def tokenize(self, sentence):
        tokenized = self.tok.tokenize (self.__chr(sentence))
        return tokenized

    def __chr(self,content):
        #this function tries to fix xml problems from the input text
        charlist = r" abcdefghijklmnopqrstuvwxyz1234567890!@#%*()-
_+={[]]?;.:/><,|'"
        chars = []
        for c in charlist:
            chars.append(c)
        s=[]
        unwanted = []
        for c in content:
            if c.lower() in chars:
                s.append(c)
            elif c == '\n':
                s.append(c)
            elif c == '"':
                s.append('"')
            elif c == "'":
                s.append("'")
            elif c == '<':
                s.append("<")
            elif c == '>':
                s.append(">")
            elif c == '&':
                s.append('&')
            elif c == '"':
                s.append(""")
            elif c == '<':
                s.append("<")
            elif c == '>':
                s.append(">")
            elif c == '&':
                s.append('&')
            elif c == '"':
                s.append(""")
            else:
                if c not in unwanted:
                    unwanted.append(c)

```

```
#f = open('unwanted.txt','a')
#f.write('').join(unwanted))
#f.close()

content = ''.join(s)
try:
    content = content.replace('"','"')
    content = content.replace("'",'')
    content = content.replace("`",'')
    content = content.replace("`",'')
    content = content.replace(",",'')
    content = content.replace('-', '-')
    content = content.replace('-', '-')
except:
    pass

#f = open('error.txt','w')
#f.write(content)
#f.close()

return content
```

APÊNDICE L - Código do módulo vgtagger

```

import os
import nltk
try:
    import cPickle as pickle
except:
    import pickle

class verbgrouptagger(nltk.TaggerI):

    def __init__(self, language='en'):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language
        self.__load_rules()

    def __load_rules(self):
        self.__rules = []
        for line in open(self.__path + '/data/' + self.language +
            '/rules.txt', 'r').readlines():
            fields = line.strip().split(' ')
            if len(fields) == 4:
                self.__rules.append(fields)

    def batch_tag(self, tagged_sentences, *kwargs):
        tagged_sents = []
        for sentence in tagged_sentences:
            tagged_sents.append(self.tag(sentence))
        return tagged_sents

    def tag(self, sentence):
        for i in range(0, len(sentence)):
            for rule in self.__rules:
                if sentence[i][1] == rule[0]:
                    if self.__get(sentence, i, rule[2]) == rule[3]:
                        sentence[i] = (sentence[i][0], rule[1])
        return sentence

    def show_tree(self, sentence):
        grammar = """
        NP: {<I-NP>+}
        VG: {<I-VG>+}
        O: {<O>+}
        """
        cp = nltk.RegexpParser(grammar)
        result = cp.parse(sentence)
        result.draw()

    def __get(self, sentence, idx, template):
        """Ex.: P=POS T=I-VG|I-NP|O|I-B"""
        s=None
        P = nltk.str2tuple(sentence[idx][0])[1]
        T = sentence[idx][1]
        # - 1
        try:
            P_1 = nltk.str2tuple(sentence[idx-1][0])[1]
            T_1 = sentence[idx-1][1]
        except:

```

```

        P_1 = '<s>'
        T_1 = '<s>'

    # + 1
    try:
        P1 = nltk.str2tuple(sentence[idx+1][0])[1]
        T1 = sentence[idx+1][1]
    except:
        P1 = '</s>'
        T1 = '</s>'

    if template == 'P':
        s = P
    elif template == 'PP1':
        s = P + P1
    elif template == 'PP_1':
        s = P + P_1
    elif template == 'PP_1P1':
        s = P + P_1 + P1
    elif template == 'PT_1':
        s = P + T_1
    elif template == 'PT1':
        s = P + T1
    elif template == 'PT_1T1':
        s = P + T_1 + T1
    elif template == 'PT_1P1':
        s = P + T_1 + P1

    return s

```

APÊNDICE M - Código do módulo vocabpro

```

import os
try:
    import cPickle as pickle
except:
    import pickle

class vocabpro:

    def __init__(self, language='en'):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language
        self.__load_lex()

    def __load_lex(self):
        self.__lex = self.__load_classifier(self.__path + '/data/' + self.language
+ '/vocab.pkl' )

    def classify(self, stem):
        tag = '0'
        if stem in self.__lex:
            tag = str(self.__lex[stem])
        return tag

    def classify_from_word(self, word):
        from nltk.stem import PorterStemmer
        stemmer = PorterStemmer()
        stem = stemmer.stem(word)
        tag = self.classify(stem)
        return tag

    def train(self, filename):
        import re
        from nltk.stem import PorterStemmer
        stemmer = PorterStemmer()

        mostfreq = []
        dic = {}
        alfa = re.compile('[a-z]')
        num = re.compile('[0-9]')
        i=0
        j=0
        for line in open(filename, 'r').readlines():
            j+=1
            cat = 0
            if j <= 1000:
                cat = 1
            else:
                cat = 2

            if i == 2000:
                break
            if len(line.strip()) > 0:
                fields = line.strip().split('\t')
                word = fields[0].strip()
                stem = stemmer.stem(word)

```



```

        if len(word) == 1:
            if len(alfa.findall(word)) != 0:
                if stem not in mostfreq:
                    i+=1
                    mostfreq.append(stem)
                    dic[stem] = cat
            else:
                j-=1
        else:
            if len(alfa.findall(word)) != 0:
                if len(num.findall(word)) == 0:

                    if stem not in mostfreq:
                        i+=1
                        mostfreq.append(stem)
                        dic[stem] = cat
                    else:
                        j-=1
            if j == 2000:
                break

    print j, i, len(mostfreq)

    self.__save_classifier(dic, self.__path + '/data/' + self.language +
'/vocab.pkl')
    self.__awl

    def __awl(self):
        import re
        from nltk.stem import PorterStemmer
        stemmer = PorterStemmer()
        voc = self.__load_classifier(self.__path + '/data/' + self.language +
'/vocab.pkl')

        alfa = re.compile('[a-z]')
        for line in open(self.__path + '/data/en/awl.txt','r').readlines():
            if len(line.strip()) > 0 and len(alfa.findall(line)) != 0:
                word = line.strip()
                stem = stemmer.stem(word)
                if stem not in voc:
                    voc[stem]= 3

        tmp = []
        for k in voc:
            tmp.append( k + '\t' + str(voc[k]) + '\n')

        f = open(self.__path + '/data/' + self.language + ' /voc.txt','w')
        f.write(''.join(tmp))
        f.close()

    self.__save_classifier(voc, self.__path + '/data/' + self.language +
'/vocab.pkl')

    def __save_classifier(self,classifier,filename):
        f = open(filename,'wb')
        pickle.dump(classifier,f)

```

```
f.close()

def __load_classifier(self,filename):
    f = open(filename,'rb')
    classifier = pickle.load(f)
    f.close()
    return classifier
```

APÊNDICE N - Código do módulo corpusanalyser

```

import os
import nltk
import numpy
import math
try:
    import cPickle as pickle
except:
    import pickle
import re
import xml.etree.cElementTree as etree
from operator import itemgetter
import xmltextreader
import analysis.corpora
import readler

class corpusanalyser:

    def __init__(self,language='en'):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.language = language
        self.__corpus_path = None
        self.__corpus_files = None
        self.__corpus_name = None
        self.__corpus_tokens = None
        self.__corpus_types = None
        self.__genre = None

# -----
# -----
# -----
# ADD CORPUS
# -----
# -----
# -----

    def add_from_folder(self,path,name,**kwargs):

        if os.path.exists(path) == True and os.path.isdir(path) == True:
            if len(name) > 0:
                name = re.sub(' ','_',name)
                if os.path.exists(analysis.corpora.__path__[0] + '/' + name) ==
False:
                    os.makedirs(analysis.corpora.__path__[0] + '/' + name)
                    files = os.listdir(path)
                    filetype='xml'
                    if 'filetype' in kwargs:
                        filetype = kwargs['filetype']
                    self.__corpus_genre = 'all'
                    if 'genre' in kwargs:
                        self.__corpus_genre = kwargs['genre']

                    print 'Starting adding...'

```

```

i = 0
total_files = len(files)
for filename in files:
    i+=1
    if filetype != 'xml':
        f = open( path + '/' + filename,'r')
        content = f.read()
        f.close()
        title,subtitle,author_source,text =
self.__text2args(content)
        content = None
        raw_text_in_xml = '<text><title>' + title +
'</title><subtitle>' + subtitle + '</subtitle>' + '<author-source>' +
author_source + '</author-source><text_body>' + text + '</text_body></text>'
        root = readler.text_analyze(raw_text_in_xml,
self.language, self.__corpus_genre)
        filename = filename.replace('.txt','') + '.xml'
        readler.xmltext_save(root, analysis.corpora.__path__[0] +
'/' + name + '/' + filename)
        print filename + ' ( ' + str(i) + ' of ' +
str(total_files) + ' ) '
    else:
        f = open(path + '/' + filename,'r')
        raw_text_in_xml = f.read()
        f.close()
        root = readler.text_analyze(raw_text_in_xml,
self.language, self.__corpus_genre)
        filename = filename.replace('.txt','') + '.xml'
        readler.xmltext_save(root, analysis.corpora.__path__[0] +
'/' + name + '/' + filename)
        print filename + ' ( ' + str(i) + ' of ' +
str(total_files) + ' ) '

self.__corpus_name = name
self.__corpus_files = os.listdir(analysis.corpora.__path__[0] +
'/' + name)
self.__corpus_path = analysis.corpora.__path__[0] + '/' + name

print 'Making data...'

wordlist = {}
poslist = {}
wordperfile = {}
tokens = 0
types = 0
for filename in self.__corpus_files:
    if filename.endswith('.xml')== True:
        tree = etree.parse(self.__corpus_path + '/' + filename)
        root = tree.getroot()
        # make wordlist
        dicwordfile = {}
        for e in root.iter('token'):
            tokens+=1
            word = e.text.lower()
            freq = e.attrib['freq']
            tag = e.attrib['pos']
            #
            if tag in poslist:

```

```

        poslist[tag]+=1
    else:
        poslist[tag]=1

    if word in wordlist:
        wordlist[word]+=1
    else:
        wordlist[word]=1
    if word not in dicwordfile:
        dicwordfile[word]=0
# word per file
for k in dicwordfile:
    if k in wordperfile:
        wordperfile[k]+=1
    else:
        wordperfile[k]=1

types = len(wordlist)
self.__corpus_tokens = tokens
self.__corpus_types = types
self.__corpus_typedtokenratio = float(types)/float(tokens)

# make keywords
from analysis.keywordsextractor.xkeywordsextractor import
keywordsextractor
ke = keywordsextractor()
keywordlist = {}
for k in wordlist:
    keyness = ke.extract(k, wordlist[k], tokens)
    keywordlist[k] = keyness

# corpus info
info = {}
info['name'] = self.__corpus_name
info['files'] = self.__corpus_files
info['path'] = self.__corpus_path
info['tokens'] = self.__corpus_tokens
info['types'] = self.__corpus_types
info['typedtokenratio'] = self.__corpus_typedtokenratio
info['genre'] = self.__corpus_genre
# save
self.__util_pickle_save(info, self.__corpus_path + '/info.pkl')
self.__util_pickle_save(poslist, self.__corpus_path +
'/poslist.pkl')
self.__util_pickle_save(wordlist, self.__corpus_path +
'/wordlist.pkl')
self.__util_pickle_save(keywordlist, self.__corpus_path +
'/keywordlist.pkl')
self.__util_pickle_save(wordperfile, self.__corpus_path +
'/wordperfile.pkl')

print 'Done!'

def __text2args(self,text):
    arr = text.strip().split('\n')
    title = arr[0].strip()
    subtitle = '-'
    author_source = '-'

```

```

        return title, subtitle, author_source, text

# -----
# -----
# -----
# UTIL
# -----
# -----

def __util_readalltext(self, filename):
    f = open(filename, 'r')
    content = f.read()
    f.close()
    return content

def __util_writealltext(self, text, filename):
    f = open(filename, 'w')
    f.write(text)
    f.close()

def __util_get_root_from_file(self, filename):
    root = self.__util_read_from_file(filename)
    return root

def __util_pickle_open(self, filename):
    """Opens an object from a file using pickle."""
    f = open(filename, 'rb')
    content = pickle.load(f)
    f.close()
    return content

def __util_pickle_save(self, obj, filename):
    """Saves an object to a file using pickle."""
    f = open(filename, 'wb')
    pickle.dump(obj, f)
    f.close()

def __util_read_from_file(self, filename):
    tree = etree.parse(filename)
    root = tree.getroot()
    return root

def __util_read_from_string(self, s):
    tree =
etree.fromstring(s.replace('<text_author></text_author>', '<text_author>-
</text_author>'))
    root = tree
    return root

def __util_read_from_root(self, root):
    tree = root

```

```

    root = tree
    return root

def __util_xml_to_string(self,element):
    return etree.tostring(element)

def __util_save_as_xml(self,element,path):
    f = open(path,'w')
    f.write(self.__ident_xml(etree.tostring(element)))
    f.close()

def __util_loglikelihood(self,ntokensinsc,ifreqiinsc, ntokensincr,ifreqincr):
    #convert to float
    nt1 = float(ntokensinsc)#number of tokens in study corpus
    nt2 = float(ntokensincr)#number of tokens in ref corpus
    f1 = float(ifreqiinsc)#freq of item in study corpus
    f2 = float(ifreqincr) #freq of item in ref corpus
    #calculate
    e1 = nt1 * ((f1+f2) / (nt1+nt2))
    e2 = nt2 * ((f1+f2) / (nt1+nt2))
    ll = 2 * ( (f1 * math.log(f1/e1) ) + (f2 * math.log(f2/e2) ) )
    #return value
    return ll

def __util_oe(self,fw1w2,fw1incorpus,fw2incorpus,corpussize,horsize):
    """Returns the observed expected values
    fw1w2: how many times w1 (node) e w2 (colocate) appear together
    fw1incorpus: how many times w1 appear in corpus
    fw2incorpus: how many times w2 appear in corpus
    corpussize: total number of tokens
    horsize: size of horizon ex.a: l1 N r1 = 1
    ex.b: l2 l1 N r1 r2 = 2
    observed value"""
    o = float(fw1w2) / float(corpussize)
    #expected value (probability of the event)
    e = float(float(fw1incorpus)/float(corpussize)) *
float(float(fw2incorpus)/ float(corpussize))
    e = e * horsize
    #how many times observed value is greater than expected value

    return o,e

def
__util_observed_expected(self,fw1w2,fw1incorpus,fw2incorpus,corpussize,horsize):
    """Returns the observed expected values
    fw1w2: how many times w1 (node) e w2 (colocate) appear together
    fw1incorpus: how many times w1 appear in corpus
    fw2incorpus: how many times w2 appear in corpus
    corpussize: total number of tokens
    horsize: size of horizon ex.a: l1 N r1 = 1
    ex.b: l2 l1 N r1 r2 = 2
    observed value"""
    o = float(fw1w2) / float(corpussize)
    #expected value (probability of the event)
    e = float(float(fw1incorpus)/float(corpussize)) *
float(float(fw2incorpus)/ float(corpussize))
    e = e * horsize

```

```

    #how many times observed value is greater than expected value
    OE = o/e
    return OE

def __util_mi(self,o,e):
    I = math.log((o/e),2)
    #I must be > 3
    return I

def __util_mutual_information(self,oe):
    I = math.log(oe,2)
    #I must be > 3
    return I

def __util_t_escore(self,o,e,fw1w2,corpussize):
    T = (o-e) / ( math.sqrt(fw1w2) / corpussize)
    #T must be > 2
    return T

def __util_dice(self,word1,word2):
    lenw1 = len(word1)
    lenw2 = len(word2)
    biw1 = []
    biw2 = []
    for i in range(0, lenw1-1):
        biw1.append(word1[i] + word1[i+1])
    for i in range(0, lenw2-1):
        biw2.append(word2[i] + word2[i+1])
    nbw1 = len(biw1)
    nbw2 = len(biw2)
    nm = 0
    for bi1 in biw1:
        for bi2 in biw2:
            if bi1 == bi2:
                nm+=1
    D = float(2 * nm ) / float(nbw1 + nbw2)
    return D

# -----
# -----
# -----
# -----
# -----
# -----

def __ident_xml(self,xmlstring):
    xmlstring = xmlstring.replace('<title>','\n <title>')
    xmlstring = xmlstring.replace('<text>','\n <text>')
    xmlstring = xmlstring.replace('<paragraph>','\n    <paragraph>')
    xmlstring = xmlstring.replace('<sentence ','\n        <sentence ')
    xmlstring = xmlstring.replace('<token ','\n            <token ')
    xmlstring = xmlstring.replace('</sentence>','\n        </sentence>')
    xmlstring = xmlstring.replace('</paragraph>','\n    </paragraph>')
    xmlstring = xmlstring.replace('</text>','\n </text>')
    xmlstring = xmlstring.replace('</xmltext>','\n</xmltext>')
    xmlstring = xmlstring.replace('</statistics>','\n</statistics>')
    xmlstring = xmlstring.replace('<statistics>','\n<statistics>')

```



```

xmlstring = xmlstring.replace('</genre>','\n</genre>')
xmlstring = xmlstring.replace('<genre>','\n<genre>')
xmlstring = xmlstring.replace('<category>','\n    <category>')
xmlstring = xmlstring.replace('<tokens>','\n    <tokens>')
xmlstring = xmlstring.replace('<types>','\n    <types>')
xmlstring = xmlstring.replace('<typetokenratio>','\n    <typetokenratio>')
xmlstring = xmlstring.replace('<sentences>','\n    <sentences>')
xmlstring = xmlstring.replace('<words>','\n    <words>')
xmlstring = xmlstring.replace('<averagewordspersentence>','\n
<averagewordspersentence>')
xmlstring = xmlstring.replace('<numberofcharacters>','\n
<numberofcharacters>')
xmlstring = xmlstring.replace('<syllables>','\n    <syllables>')
xmlstring = xmlstring.replace('<averagesyllablesperword>','\n
<averagesyllablesperword>')
xmlstring = xmlstring.replace('<fleschreadingease>','\n
<fleschreadingease>')
xmlstring = xmlstring.replace('<fleschkincaid>','\n    <fleschkincaid>')
xmlstring = xmlstring.replace('<cogtagger>','\n    <cogtagger>')
xmlstring = xmlstring.replace('<cognatesratio>','\n    <cognatesratio>')
xmlstring = xmlstring.replace('<text_title>','\n<text_title>')
xmlstring = xmlstring.replace('<text_subtitle>','\n<text_subtitle>')
xmlstring = xmlstring.replace('<text_author>','\n<text_author>')
xmlstring = xmlstring.replace('<text_body>','\n<text_body>')
xmlstring = xmlstring.replace('</text_body>','\n</text_body>')

return xmlstring

```

```

def __chr(self,content):
    #this function tries to fix xml problems from the input text
    charlist = r" abcdefghijklmnopqrstuvwxyz1234567890!@#%*()-
_+={[}]?;.:><|'"
    chars = []
    for c in charlist:
        chars.append(c)
    s=[]
    unwanted = []
    for c in content:
        if c.lower() in chars:
            s.append(c)
        elif c == '\n':
            s.append(c)
        elif c == '':
            s.append('')
        elif c == '':
            s.append('')
        elif c == '':
            s.append('')
        elif c == '':
            s.append('')
        elif c == '&':
            s.append('and')
        elif c == '':
            s.append('')
        else:
            if c not in unwanted:
                unwanted.append(c)

```

```

content = ''.join(s)
try:
    content = content.replace("'", '"')
    content = content.replace('"', '"')
    content = content.replace("`", '"')
    content = content.replace("“", '"')
    content = content.replace("”", '"')
    content = content.replace("„", '"')
    content = content.replace('-', '-')
    content = content.replace('-', '-')
except:
    pass

return content

# -----
# -----
# -----
# USE CORPUS
# -----
# -----
# -----

def use(self, corpus_name):
    if os.path.isdir(analysis.corpora.__path__[0] + '/' + corpus_name)==True:
        info = self.__util_pickle_open(analysis.corpora.__path__[0] + '/' +
corpus_name + '/info.pkl')
        self.__corpus_name = info['name']
        self.__corpus_files = info['files']
        self.__corpus_path = info['path']
        self.__corpus_tokens = info['tokens']
        self.__corpus_types = info['types']
        self.__corpus_typedtokenratio = info['typedtokenratio']
        self.__corpus_genre = info['genre']

# -----
# -----
# -----
# STATISTICS
# -----
# -----
# -----

def info(self):
    paragraphs = 0
    sentences = 0
    words = 0
    for filename in self.__corpus_files:
        if filename.endswith('.xml') == True:
            root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
            for e in root.iter('statistics'):
                for item in e:
                    if item.tag == 'paragraphs':
                        paragraphs+=int(item.text)
                    if item.tag == 'sentences':
                        sentences+=int(item.text)
                    if item.tag == 'words':
                        words+=int(item.text)

```

```

hapax = 0
wordlist = self.__util_pickle_open(self.__corpus_path + '/wordlist.pkl' )
for k in wordlist:
    if wordlist[k] == 1:
        hapax+=1

s = 'Name: \t %s \n' % (self.__corpus_name)
s += 'Genre: \t %s \n' % (self.__corpus_genre)
s += 'Files: \t %s \n' % (len(self.__corpus_files))
s += 'Tokens: \t %s \n' % (self.__corpus_tokens)
s += 'Types: \t %s \n' % (self.__corpus_types)
s += 'Type token ratio: \t %s \n' % (self.__corpus_typedtokenratio)
s += 'Words: \t %s \n' % (words)
s += 'Hapax: \t %s \n' % (hapax)
s += 'Paragraphs: \t %s \n' % (paragraphs)
s += 'Sentences: \t %s \n' % (sentences)

return s

# -----
# -----
# WORDLIST
# -----
# -----
def wordlist_to_file(self,filename):
    wordlist = self.__util_pickle_open(self.__corpus_path + '/wordlist.pkl')
    tmp = []
    for k in sorted(wordlist.iteritems(),key=itemgetter(1),reverse = True):
        per = float(k[1]) / float(self.__corpus_tokens)
        tmp.append(k[0] + '\t' + str(k[1]) + '\t' + str(per).replace('.',','))
+ '\n')

    self.__util_writealltext(''.join(tmp).strip(), filename)

def wordlist_get(self):
    wordlist = self.__util_pickle_open(self.__corpus_path + '/wordlist.pkl')
    return wordlist

# -----
# -----
# WORDS PER FILE
# -----
# -----
def wordsperfile_to_file(self,filename):
    words = self.__util_pickle_open(self.__corpus_path + '/wordperfile.pkl')
    tmp = []
    for k in sorted(words.iteritems(),key=itemgetter(1),reverse = True):

```

```

        per = float(k[1]) / float(len(self.__corpus_files))
        tmp.append(k[0] + '\t' + str(k[1]) + '\t' + str(per).replace('.', ','))
+ '\n')
    self.__util_writealltext(''.join(tmp).strip(), filename)

    def wordsperfile_get(self):
        words = self.__util_pickle_open(self.__corpus_path + '/wordperfile.pkl')
        return

# -----
# -----
# -----
# KEYWORDS
# -----
# -----
# -----

    def keywords_get(self):
        keywordlist = self.__util_pickle_open(self.__corpus_path +
'/keywordlist.pkl')
        return keywordlist

    def keywords_save(self, keywordlist, filename):
        tmp = []
        for k in sorted(keywordlist.iteritems(), key=itemgetter(1), reverse = True):
            tmp.append(k[0] + '\t' + str(k[1]).replace('.', ','))
        self.__util_writealltext(''.join(tmp).strip(), filename)

    def keywords_to_file(self, filename):
        keywordlist = self.keywords_get()
        self.keywords_save(keywordlist, filename)

    def keywords_get_keys_times_files(self):
        keywordlist = self.__util_pickle_open(self.__corpus_path +
'/keywordlist.pkl')
        wordperfile = self.__util_pickle_open(self.__corpus_path +
'/wordperfile.pkl')

        keykeys = {}

        for k in keywordlist:
            key = k
            value = wordperfile[k] * keywordlist[k]
            keykeys[key] = value
        return keykeys

    def keywords_get_keys_times_files_to_file(self, filename):
        keywordlist = self.keywords_get_keys_times_files()
        self.keywords_save(keywordlist, filename)

    def keywords_get_keys_plus_files(self):
        keywordlist = self.__util_pickle_open(self.__corpus_path +
'/keywordlist.pkl')
        wordperfile = self.__util_pickle_open(self.__corpus_path +
'/wordperfile.pkl')

```

```

keykeys = {}

for k in keywordlist:
    key = k
    value = wordperfile[k] + keywordlist[k]
    keykeys[key] = value
return keykeys

def keywords_get_keys_plus_files_to_file(self,filename):
    keywordlist = self.keywords_get_keys_plus_files()
    self.keywords_save(keywordlist, filename)

def keywords_get_keys_filtered(self):
    import string
    stoplist = nltk.corpus.stopwords.words('english')
    stoplist.append('--')
    for c in string.punctuation:
        stoplist.append(c)

    keywordlist = self.__util_pickle_open(self.__corpus_path +
'/keywordlist.pkl')
    wordperfile = self.__util_pickle_open(self.__corpus_path +
'/wordperfile.pkl')

    keykeys = {}

    for k in keywordlist:
        if k not in stoplist:
            key = k
            value = wordperfile[k] * keywordlist[k]
            keykeys[key] = value
    return keykeys

def keywords_get_keys_filtered_to_file(self,filename):
    keywordlist = self.keywords_get_keys_filtered()
    self.keywords_save(keywordlist, filename)

# -----
# -----
# -----
# VOCAB PROFILE
# -----
# -----
# -----

def vocab_profile_count_from_tokens(self):
    dic = {}
    for filename in self.__corpus_files:
        root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
        for e in root.iter('token'):
            rank = e.attrib['voc']
            cat = e.attrib['voc']
            if cat in dic:
                dic[cat]+=1
            else:
                dic[cat]=1

```

```

print 'K1: ', float(dic['1']) / float(self.__corpus_tokens)
print 'K2: ', float(dic['2']) / float(self.__corpus_tokens)
print 'AWL: ', float(dic['3']) / float(self.__corpus_tokens)
print 'OFF: ', float(dic['0']) / float(self.__corpus_tokens)

def vocab_profile_count(self):
    dic = {}
    for filename in self.__corpus_files:
        root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
        for e in root.iter('token'):
            cat = e.attrib['voc']
            if e.text.lower() not in dic:
                dic[e.text.lower()] = cat

    dic_cat = {}

    for k in dic:
        if dic[k] in dic_cat:
            dic_cat[dic[k]] += 1
        else:
            dic_cat[dic[k]] = 1

    print 'K1: ', float(dic_cat['1']) / float(self.__corpus_types)
    print 'K2: ', float(dic_cat['2']) / float(self.__corpus_types)
    print 'AWL: ', float(dic_cat['3']) / float(self.__corpus_types)
    print 'OFF: ', float(dic_cat['0']) / float(self.__corpus_types)

def vocab_compare(self, dest):
    dic = {}
    for filename in self.__corpus_files:
        if filename.endswith('.xmlt'):
            root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)

            for e in root.iter('token'):
                cat = e.attrib['voc']
                if e.text.lower() not in dic:
                    dic[e.text.lower()] = cat

    dic_cat = {}

    for k in dic:
        if dic[k] in dic_cat:
            dic_cat[dic[k]] += 1
        else:
            dic_cat[dic[k]] = 1

    print 'K1: ', float(dic_cat['1']) / float(self.__corpus_types)
    print 'K2: ', float(dic_cat['2']) / float(self.__corpus_types)
    print 'AWL: ', float(dic_cat['3']) / float(self.__corpus_types)
    print 'OFF: ', float(dic_cat['0']) / float(self.__corpus_types)
    print self.__corpus_types, len(dic)

from analysis.stemmer.xstemmer import stemmer
from analysis.cogtagger.xcogtagger import cogtagger

st = stemmer()
cog = cogtagger()

```

```

    alfa = re.compile('[a-z]')
    num = re.compile('[0-9]')
    keywords = self.keywords_get_keys_times_files()
    wordlist = self.wordlist_get()
    perfile = self.wordlist_get()

    tmp = []
    for k in keywords:
        if len(alfa.findall(k)) > 0:
            if len(num.findall(k)) == 0:
                tmp.append(k + '\t' + st.stem(k) + '\t' + str(keywords[k]) +
'\t' + str(wordlist[k]) + '\t' +
str(perfile[k]) + '\t' + str(dic[k]) + '\t' +
str(cog.tag(k)) + '\n')

    f = open(dest, 'w')
    f.write(''.join(tmp))
    f.close()

```

```

# -----
# POS
# -----
def pos_frequency_list_to_file(self, filename):
    pos = self.__util_pickle_open(self.__corpus_path + '/poslist.pkl')
    tmp = []
    for k in sorted(pos.iteritems(), key=itemgetter(1), reverse = True):
        per = float(k[1]) / float(self.__corpus_tokens)
        tmp.append(k[0] + '\t' + str(k[1]) + '\t' + str(per).replace('.', ','))
    + '\n')
    self.__util_writealltext(''.join(tmp).strip(), filename)

```

```

# -----
# COGNATES
# -----
def cognates_get(self):

```

```

        cognates = {}
        for filename in self.__corpus_files:
            root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
            for e in root.iter('token'):
                if e.attrib['cog'] == '1':
                    if e.text.lower() in cognates:
                        cognates[e.text.lower()] += 1
                    else:
                        cognates[e.text.lower()] = 1
            return cognates

    def cognates_save(self, cognates, filename):
        tmp = []
        for k in sorted(cognates.iteritems(), key=itemgetter(1), reverse = True):
            tmp.append(k[0] + '\t' + str(k[1]) + '\n')
        self.__util_writealltext(''.join(tmp).strip(), filename)

    def cognates_to_file(self, filename):
        cognates = self.cognates_get()
        self.cognates_save(cognates, filename)

# -----
# -----
# -----
# GROUPS
# -----
# -----
# -----

    def groups_get_from_sentence(self, e_sentence, group='I-NP'):
        groups = []
        g = []
        for token in e_sentence:
            if token.attrib['grp'] == group:
                g.append( (token.text, token.attrib['pos']) )
            elif token.attrib['grp'] != group:
                if len(g) == 0:
                    pass
                else:
                    groups.append(g)
                    g = []
        return groups

    def groups_get(self, group='I-NP'):
        all_groups = []
        for filename in self.__corpus_files:
            root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
            for e in root.iter('sentence'):
                sent_groups = self.groups_get_from_sentence(e, group)
                if len(sent_groups) > 0:
                    for sent_group in sent_groups:
                        all_groups.append(sent_group)

        return all_groups

    def groups_count_groups(self, group='I-NP'):
        dic_group={}

```



```

all_groups = self.groups_get(group)
for group in all_groups:
    pos=[]
    for token in group:
        pos.append(token[0] + ' ')

    key = ''.join(pos).strip()
    if key in dic_group:
        dic_group[key]+=1
    else:
        dic_group[key]=1

return dic_group

def groups_count_POS(self,group='I-NP'):
    dic_group_POS={}
    all_groups = self.groups_get(group)
    for group in all_groups:
        pos=[]
        for token in group:
            pos.append(token[1] + ' ')

        key = ''.join(pos).strip()
        if key in dic_group_POS:
            dic_group_POS[key]+=1
        else:
            dic_group_POS[key]=1

    return dic_group_POS

def groups_count_save(self,dic_groups,filename):
    tmp = []
    for k in sorted(dic_groups.iteritems(),key=itemgetter(1),reverse = True):
        tmp.append(k[0] + '\t' + str(k[1]) + '\n')

    self.__util_writealltext(''.join(tmp).strip(), filename)

def groups_count_groups_to_file(self,group,filename):
    dic_groups = self.groups_count_groups(group)
    self.groups_count_save(dic_groups, filename)

def groups_count_POS_to_file(self,group,filename):
    dic_groups = self.groups_count_POS(group)
    self.groups_count_save(dic_groups, filename)

# -----
# -----
# -----
# AFFIXES
# -----
# -----

def suffix_count_words(self):
    words = {}
    for filename in self.__corpus_files:
        root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
        for e in root.iter('token'):

```

```

        if e.attrib['suf'] != '-':
            if e.text.lower() in words:
                words[e.text.lower()] += 1
            else:
                words[e.text.lower()] = 1
    return words

def suffix_count(self):
    suffixes = {}
    for filename in self.__corpus_files:
        root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
        for e in root.iter('token'):
            if e.attrib['suf'] != '-':
                if e.attrib['suf'] in suffixes:
                    suffixes[e.attrib['suf']] += 1
                else:
                    suffixes[e.attrib['suf']] = 1
    return suffixes

def suffix_count_save(self, dic_suffix, filename):
    tmp = []
    for k in sorted(dic_suffix.iteritems(), key=itemgetter(1), reverse = True):
        tmp.append(k[0] + '\t' + str(k[1]) + '\n')

    self.__util_writealltext(''.join(tmp).strip(), filename)

def suffix_count_to_file(self, filename):
    suffixes = self.suffix_count()
    self.suffix_count_save(suffixes, filename)

# -----
# -----
# -----
# POSITION
# -----
# -----
# -----

def dispersion_word(self, word):
    positions = []
    word_freq = 0
    for filename in self.__corpus_files:
        root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
        tokens = 0
        for e in root.iter('tokens'):
            if e.tag == 'tokens':
                tokens = int(e.text)

        for e in root.iter('token'):
            if e.text.lower() == word.lower():
                word_freq += 1
                position = int(round((float(e.attrib['id']) / float(tokens)) *
100))
                positions.append(position)

    dic_ranges = dict(r1=0, r2=0, r3=0, r4=0, r5=0, r6=0, r7=0, r8=0, r9=0, r10=0)

```

```

for position in positions:
    if position in range(0,10):
        dic_ranges['r1']+=1
    elif position in range(11,20):
        dic_ranges['r2']+=1
    elif position in range(21,30):
        dic_ranges['r3']+=1
    elif position in range(31,40):
        dic_ranges['r4']+=1
    elif position in range(41,50):
        dic_ranges['r5']+=1
    elif position in range(51,60):
        dic_ranges['r6']+=1
    elif position in range(61,70):
        dic_ranges['r7']+=1
    elif position in range(71,80):
        dic_ranges['r8']+=1
    elif position in range(81,90):
        dic_ranges['r9']+=1
    elif position in range(91,100):
        dic_ranges['r10']+=1

positions = None

dic = {}
for k in dic_ranges:
    p = ((float(dic_ranges[k]) / float(word_freq)) * 100)
    dic[k] = float("{:10.2f}".format(p))

dispersion = (dic['r1'],dic['r2'],dic['r3'],dic['r4'],dic['r5'],
              dic['r6'],dic['r7'],dic['r8'],dic['r9'],dic['r10'])

return dispersion

def dispersion_to_string(self,dispersion):
    tmp = []
    for d in dispersion:
        tmp.append(str(d).replace('.',',') + '\t')
    return ''.join(tmp).strip()

def dispersion_to_file(self,filename,limit=0):
    keywordlist = self.keywords_get_keys_filtered()
    if limit == 0:
        limit = len(keywordlist)
    results = []
    i=0
    for k in sorted(keywordlist.iteritems(),key=itemgetter(1),reverse = True):
        dispersion = self.dispersion_to_string(self.dispersion_word(k[0]))
        print k[0]
        results.append(k[0] + '\t' + dispersion + '\n')
        i+=1
        if i >= limit:
            break

    self.__util_writealltext(''.join(results).strip(), filename)

def dispersion_to_pkl(self,filename,limit=0):
    keywordlist = self.keywords_get_keys_filtered()

```

```

    if limit == 0:
        limit = len(keywordlist)
    results = []
    i=0
    for k in sorted(keywordlist.iteritems(),key=itemgetter(1),reverse = True):
        dispersion = self.dispersion_word(k[0])
        print i,k
        results.append((k[0],dispersion))
        i+=1
        if i >= limit:
            break

    self.__util_pickle_save(results, filename)

def dispersion_open_pkl(self,filename):
    dispersion_results = self.__util_pickle_open(filename)
    return dispersion_results

def dispersion_group(self,ngroups,dispersion_results):
    arr_words = []
    arr = []
    for dispersion in dispersion_results:
        arr_words.append(dispersion[0])
        arr.append(
[dispersion[1][0],dispersion[1][1],dispersion[1][2],dispersion[1][3],dispersion[1]
[4],
dispersion[1][5],dispersion[1][6],dispersion[1][7],dispersion[1][8],dispersion[1][
9]])

        vectors = numpy.array(arr)
        clusterer =
nltk.cluster.kmeans.KMeansClusterer(ngroups,nltk.euclidean_distance,initial_means=
None)
        clusters = clusterer.cluster(vectors, True,trace=True)

        results = []
        for i in range(0,len(clusters)):
            results.append((arr_words[i],clusters[i]))

    return results

def dispersion_group_to_file(self,ngroups,dispersion_results,filename):
    results = self.dispersion_group(ngroups, dispersion_results)
    tmp = []
    for result in results:
        tmp.append(result[0] + '\t' + str(result[1]) + '\n')
    self.__util_writealltext(''.join(tmp), filename)

# -----
# -----
# -----
# TRIGRAMS
# -----
# -----
# -----

def trigrams_get(self):

```

```

        dic = {}
        for filename in self.__corpus_files:
            root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
            for e in root.iter('sentence'):
                sentence=['<s>','<s>','<s>','<s>','<s>']
                for token in e:
                    sentence.append(token.text)
                sentence+=['</s>','</s>','</s>','</s>','</s>']

                for i in range(5,len(sentence)-5):
                    W = sentence[i]
                    W_1 = sentence[i-1]
                    W1 = sentence[i+1]

                    if '<s>' not in [W,W_1,W1] and '</s>' not in [W,W_1,W1]:
                        key = (W_1 + ' ' + W + ' ' + W1)
                        if key in dic:
                            dic[key]+=1
                        else:
                            dic[key]=1

            return dic

    def trigrams_pos_get(self):
        dic = {}
        for filename in self.__corpus_files:
            root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
            for e in root.iter('sentence'):
                sentence=['<s>','<s>','<s>','<s>','<s>']
                for token in e:
                    sentence.append(token.text + '/' + token.attrib['pos'])
                sentence+=['</s>','</s>','</s>','</s>','</s>']

                for i in range(5,len(sentence)-5):
                    W = sentence[i]
                    W_1 = sentence[i-1]
                    W1 = sentence[i+1]

                    if '<s>' not in [W,W_1,W1] and '</s>' not in [W,W_1,W1]:
                        key = (W_1 + ' ' + W + ' ' + W1)
                        if key in dic:
                            dic[key]+=1
                        else:
                            dic[key]=1

            return dic

    def trigrams_save(self,trigrams,filename):
        tmp = []
        for k in sorted(trigrams.iteritems(),key=itemgetter(1),reverse = True):
            tmp.append(k[0] + '\t' + str(k[1]) + '\n')
        self.__util_writealltext(''.join(tmp).strip(), filename)

    def trigrams_to_file(self,filename):
        trigrams = self.trigrams_get()

```

```

        self.keywords_save(trigrams, filename)

    def trigrams_pos_to_file(self,filename):
        trigrams = self.trigrams_pos_get()
        self.keywords_save(trigrams, filename)

# -----
# -----
# -----
# CONCORDANCE
# -----
# -----
# -----

    def concordance_get(self,word,pos_template=None,trigram=None,limit=1,blank=0):
        try:
            pos_template = pos_template.strip().split('/')
        except:
            pos_template = None

        try:
            trigram = trigram.strip().split('/')
        except:
            trigram = None

        concordances = []
        for filename in self.__corpus_files:
            if len(concordances) == limit:
                break

            root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
            for e in root.iter('sentence'):
                if len(concordances) == limit:
                    break

                sentence = []
                sentence.append((' ',' '))
                sentence.append((' ',' '))
                for token in e:
                    sentence.append((token.text,token.attrib['pos']))
                sentence.append((' ',' '))
                sentence.append((' ',' '))

                for i in range(2,len(sentence)-2):
                    if sentence[i][0].lower() == word.lower():
                        W_2 = sentence[i-2][0]
                        W_1 = sentence[i-1][0]
                        W = sentence[i][0]
                        W1 = sentence[i+1][0]
                        W2 = sentence[i+2][0]
                        T_2 = sentence[i-2][1]
                        T_1 = sentence[i-1][1]
                        T = sentence[i][1]
                        T1 = sentence[i+1][1]
                        T2 = sentence[i+2][1]

                        left_hor = []

```

```

for j in range(i-1,0,-1):
    try:
        if sentence[i-j][0] != ' ':
            left_hor.append(sentence[i-j][0] + ' ')
    except:
        pass

right_hor = []
for j in range(1,len(sentence)-i):
    try:
        if sentence[i+j][0] != ' ':
            right_hor.append(sentence[i+j][0] + ' ')
    except:
        pass

if blank == 1:
    NODE = '_____'
else:
    NODE = W

flag_add = True

if pos_template != None:
    flag_add = False
    if pos_template[0] == 'T_1TT1':
        if pos_template[1] == T_1+T+T1:
            flag_add = True
    elif pos_template[0] == 'T_2T_1T':
        if pos_template[1] == T_2+T_1+T:
            flag_add = True

if trigram != None:
    flag_add = False
    if trigram[0] == 'NWW':
        if trigram[1] == str(W+W1+W2).lower():
            flag_add = True

    elif trigram[0] == 'WNW':
        if trigram[1] == str(W_1+W+W1).lower():
            flag_add = True

    elif trigram[0] == 'WWN':
        if trigram[1] == str(W_2+W_1+W).lower():
            flag_add = True

if flag_add == True:
    concordance = '<p>' + ''.join(left_hor).strip() + '
<b>' + NODE.strip() + '</b>' + ''.join(right_hor).strip() + '</p>\n'
    if concordance not in concordances:
        concordances.append(concordance)

return ''.join(concordances)

```

```

# -----
-----
-----

```

```

# COLOCATES
# -----
-----
-----

def collocates_horizon_get(self,word):
    horizon = []
    for filename in self.__corpus_files:
        root = self.__util_get_root_from_file(self.__corpus_path + '/' +
filename)
        for e in root.iter('sentence'):
            sentence = []
            sentence.append((' ',' '))
            sentence.append((' ',' '))
            sentence.append((' ',' '))
            sentence.append((' ',' '))
            sentence.append((' ',' '))
            sentence.append((' ',' '))
            for token in e:
                sentence.append((token.text,token.attrib['pos']))
            sentence.append((' ',' '))
            sentence.append((' ',' '))
            sentence.append((' ',' '))
            sentence.append((' ',' '))
            sentence.append((' ',' '))
            sentence.append((' ',' '))

            for i in range(5,len(sentence)-5):
                if sentence[i][0].lower() == word.lower():
                    W_5 = sentence[i-5][0]
                    W_4 = sentence[i-4][0]
                    W_3 = sentence[i-3][0]
                    W_2 = sentence[i-2][0]
                    W_1 = sentence[i-1][0]
                    W = sentence[i][0]
                    W1 = sentence[i+1][0]
                    W2 = sentence[i+2][0]
                    W3 = sentence[i+3][0]
                    W4 = sentence[i+4][0]
                    W5 = sentence[i+5][0]
                    T_5 = sentence[i-5][1]
                    T_4 = sentence[i-4][1]
                    T_3 = sentence[i-3][1]
                    T_2 = sentence[i-2][1]
                    T_1 = sentence[i-1][1]
                    T = sentence[i][1]
                    T1 = sentence[i+1][1]
                    T2 = sentence[i+2][1]
                    T3 = sentence[i+3][1]
                    T4 = sentence[i+4][1]
                    T5 = sentence[i+5][1]

                    h = [(W_5,T_5),(W_4,T_4),(W_3,T_3),(W_2,T_2), (W_1,T_1),
(W,T), (W1,T1),(W2,T2), (W3,T3),(W4,T4),(W5,T5)]
                    horizon.append(h)

        return horizon

def collocates_patterns_get(self,word,dest):

```



```

from collections import defaultdict
from collections import Counter

horizon = self.colocates_horizon_get(word)
nodefreq = len(horizon)
wordlist = self.wordlist_get()
s = ''
s += 'Node frequency: %s' % (len(horizon))
r1 = Counter()
r2 = Counter()
r3 = Counter()
r4 = Counter()
r5 = Counter()
l1 = Counter()
l2 = Counter()
l3 = Counter()
l4 = Counter()
l5 = Counter()

for h in horizon:
    l5[h[0][0]]+=1
    l4[h[1][0]]+=1
    l3[h[2][0]]+=1
    l2[h[3][0]]+=1
    l1[h[4][0]]+=1
    r1[h[6][0]]+=1
    r2[h[7][0]]+=1
    r3[h[8][0]]+=1
    r4[h[9][0]]+=1
    r5[h[10][0]]+=1

l5_c = {}
for k in l5:
    try:
        o,e = self.__util_oe(l5[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)
        ts = self.__util_t_escore(o, e, l5[k], self.__corpus_tokens)
        l5_c[k]=ts
    except:
        pass

l4_c = {}
for k in l4:
    try:
        o,e = self.__util_oe(l4[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)
        ts = self.__util_t_escore(o, e, l4[k], self.__corpus_tokens)
        l4_c[k]=ts
    except:
        pass

l3_c = {}
for k in l3:
    try:
        o,e = self.__util_oe(l3[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)
        ts = self.__util_t_escore(o, e, l3[k], self.__corpus_tokens)

```

```

        l3_c[k]=ts
    except:
        pass

    l2_c = {}
    for k in l2:
        try:
            o,e = self.__util_oe(l2[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)
            ts = self.__util_t_escore(o, e, l2[k], self.__corpus_tokens)
            l2_c[k]=ts
        except:
            pass

    l1_c = {}
    for k in l1:
        try:
            o,e = self.__util_oe(l1[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)
            ts = self.__util_t_escore(o, e, l1[k], self.__corpus_tokens)
            l1_c[k]=ts
        except:
            pass

    r1_c = {}
    for k in r1:
        try:
            o,e = self.__util_oe(r1[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)
            ts = self.__util_t_escore(o, e, r1[k], self.__corpus_tokens)
            r1_c[k]=ts
        except:
            pass

    r2_c = {}
    for k in r2:
        try:
            o,e = self.__util_oe(r2[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)
            ts = self.__util_t_escore(o, e, r2[k], self.__corpus_tokens)
            r2_c[k]=ts
        except:
            pass

    r3_c = {}
    for k in r3:
        try:
            o,e = self.__util_oe(r3[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)
            ts = self.__util_t_escore(o, e, r3[k], self.__corpus_tokens)
            r3_c[k]=ts
        except:
            pass

    r4_c = {}
    for k in r4:
        try:
            o,e = self.__util_oe(r4[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)

```

```

        ts = self.__util_t_escore(o, e, r4[k], self.__corpus_tokens)
        r4_c[k]=ts
    except:
        pass

    r5_c = {}
    for k in r5:
        try:
            o,e = self.__util_oe(r5[k], nodefreq, wordlist[k.lower()],
self.__corpus_tokens, 5)
            ts = self.__util_t_escore(o, e, r5[k], self.__corpus_tokens)
            r5_c[k]=ts
        except:
            pass

    tmp = []

    for k in l5_c:
        tmp.append('L5' + '\t' + k + '\t' + str(l5_c[k]).replace('.',',') +
'\n')
    for k in l4_c:
        tmp.append('L4' + '\t' + k + '\t' + str(l4_c[k]).replace('.',',') +
'\n')
    for k in l3_c:
        tmp.append('L3' + '\t' + k + '\t' + str(l3_c[k]).replace('.',',') +
'\n')
    for k in l2_c:
        tmp.append('L2' + '\t' + k + '\t' + str(l2_c[k]).replace('.',',') +
'\n')
    for k in l1_c:
        tmp.append('L1' + '\t' + k + '\t' + str(l1_c[k]).replace('.',',') +
'\n')

    for k in r1_c:
        tmp.append('R1' + '\t' + k + '\t' + str(r1_c[k]).replace('.',',') +
'\n')
    for k in r2_c:
        tmp.append('R2' + '\t' + k + '\t' + str(r2_c[k]).replace('.',',') +
'\n')
    for k in r3_c:
        tmp.append('R3' + '\t' + k + '\t' + str(r3_c[k]).replace('.',',') +
'\n')
    for k in r4_c:
        tmp.append('R4' + '\t' + k + '\t' + str(r4_c[k]).replace('.',',') +
'\n')
    for k in r5_c:
        tmp.append('R5' + '\t' + k + '\t' + str(r5_c[k]).replace('.',',') +
'\n')

    f = open(dest,'w')
    f.write(''.join(tmp))
    f.close()

```

APÊNDICE O - Código do módulo interpreter

```

import os
import xml.etree.cElementTree as etree
import xmltextreader

class interpreter:

    def __init__(self):
        self.__path = os.path.dirname(os.path.abspath(__file__))
        self.__load_html_tags()
        self.__reader = xmltextreader.xmltextreader()
        self.__token_history = []
        self.__sentence_history = []

    def __load_html_tags(self):
        self.__html_tags = []
        for line in open( self.__path +
            '/activities/html_tags.txt', 'r').readlines():
            if not line.startswith('#'):
                fields = line.strip().split(' ')
                if len(fields) > 1:
                    self.__html_tags.append((fields[0].strip(), fields[1].strip()))

    def translate(self, xmltext, xmltemplate):
        self.__reader.read_from_string(xmltext)
        translation = self.__parse(xmltemplate)
        return translation

    def __parse(self, xmltemplate):
        tmp = [] # for concatenation
        # read the template
        template_tree = etree.fromstring(xmltemplate)
        template_root = template_tree
        # parse and translate
        for item in template_root:
            if item.tag == 'activity':
                for e in item:
                    if e.tag == 'label':
                        tmp.append(e.text)
                    elif e.tag == 'line':
                        tmp.append('<br>')
                    elif e.tag == 'tokens':
                        tmp.append(self.__tokens(e))
                    elif e.tag == 'sentences':
                        tmp.append(self.__sentences(e))
                    elif e.tag == 'wordcloud_keywords':
                        tmp.append(self.__word_cloud_keywords(e))
                    elif e.tag == 'wordcloud_frequency':
                        tmp.append(self.__word_cloud_frequency(e))
                    elif e.tag == 'paragraphs_keywords':
                        tmp.append(self.__paragraphs_keywords(e))
                    elif e.tag == 'groups':
                        tmp.append(self.__groups(e))

```

```

return self.__html(''.join(tmp))

def __html(self,s):
    for item in self.__html_tags:
        s = s.replace(item[0],item[1])
    return s

# -----
# -----
# -----
# TOKEN
# -----
# -----
def __paragraphs_keywords(self,e):
    args = {}
    if 'pos' in e.attrib:
        args['pos'] = e.attrib['pos']
    else:
        args['pos'] = 'NN'

    if 'delimiter' in e.attrib:
        args['delimiter'] = e.attrib['delimiter']
    else:
        args['delimiter'] = ' '

    if 'identpar' in e.attrib:
        args['identpar'] = e.attrib['identpar']
    else:
        args['identpar'] = 'no'

    if 'group' in e.attrib:
        args['group'] = e.attrib['group']
    else:
        args['group'] = 'no'

    if 'group_tag' in e.attrib:
        args['group_tag'] = e.attrib['group_tag']
    else:
        args['group_tag'] = 'I-NP'

    s = self.__reader.keywords_from_paragraphs(args)
    return s

def __tokens(self,e):
    args = {}

    if 'limit' in e.attrib:
        args['limit'] = int(e.attrib['limit'])
    else:

```

```

    args['limit'] = int(10)

if 'unique' in e.attrib:
    args['unique'] = str(e.attrib['unique']).strip()
else:
    args['unique'] = 'yes'

if 'minsize' in e.attrib:
    args['minsize'] = int(e.attrib['minsize'])

if 'maxsize' in e.attrib:
    args['maxsize'] = int(e.attrib['maxsize'])

if 'stoplist' in e.attrib:
    args['stoplist'] = e.attrib['stoplist'].strip().split(' ')

if 'pos' in e.attrib:
    args['pos'] = e.attrib['pos'].strip().split(' ')

if 'stoppos' in e.attrib:
    args['stoppos'] = e.attrib['stoppos'].strip().split(' ')

if 'punct' in e.attrib:
    args['punct'] = str(e.attrib['punct']).strip()
else:
    args['punct'] = 'no'

if 'cognate' in e.attrib:
    args['cognate'] = str(e.attrib['cognate']).strip()

if 'case' in e.attrib:
    args['case'] = str(e.attrib['case']).strip()
else:
    args['case'] = 'none'

if 'minfreq' in e.attrib:
    args['minfreq'] = int(e.attrib['minfreq'])

if 'maxfreq' in e.attrib:
    args['maxfreq'] = int(e.attrib['maxfreq'])

if 'regexp' in e.attrib:
    args['regexp'] = str(e.attrib['regexp'])

if 'suffix' in e.attrib:
    args['suffix'] = str(e.attrib['suffix']).strip()

if 'min_position' in e.attrib:
    args['min_position'] = int(e.attrib['min_position'])

if 'max_position' in e.attrib:
    args['max_position'] = int(e.attrib['max_position'])

if 'order' in e.attrib:
    args['order'] = str(e.attrib['order']).strip()

if 'repeat' in e.attrib:
    args['repeat'] = str(e.attrib['repeat']).strip()

```

```

        if 'before-token' in e.attrib:
            args['before-token'] = str(e.attrib['before-token']).strip()

        if 'delimiter' in e.attrib:
            args['delimiter'] = str(e.attrib['delimiter'])

        if 'word' in e.attrib:
            args['word'] = str(e.attrib['word']).strip()

        s ,self.__token_history = self.__reader.filter_tokens(args,
self.__token_history)

        return s

# -----
# SENTENCE
# -----
def __sentences(self,e):
    s = ''
    args = {}

    if 'word' in e.attrib:
        args['word'] = int(e.attrib['word'])

    #s, self.__sentence_history = self.myreader.filter_sentences(args,
self.__sentence_history)

    return s

# -----
# WORD CLOUD KEYWORDS
# -----
def __word_cloud_keywords(self,e):
    s = ''
    s = self.__reader.get_word_cloud_from_keywords()
    return s

def __word_cloud_frequency(self,e):
    s = ''
    s = self.__reader.get_word_cloud_from_frequency()
    return s

def __groups(self,e):
    args = {}
    if 'limit' in e.attrib:
        args['limit'] = int(e.attrib['limit'])
    else:
        args['limit'] = 1

    if 'unique' in e.attrib:

```

```

        args['unique'] = str(e.attrib['unique']).strip()
    else:
        args['unique'] = 'yes'

    if 'minsize' in e.attrib:
        args['minsize'] = int(e.attrib['minsize'])
    else:
        args['minsize'] = 1

    if 'maxsize' in e.attrib:
        args['maxsize'] = int(e.attrib['maxsize'])
    else:
        args['maxsize'] = 10

    if 'min_position' in e.attrib:
        args['min_position'] = float(e.attrib['min_position'])
    else:
        args['min_position'] = 0

    if 'max_position' in e.attrib:
        args['max_position'] = float(e.attrib['max_position'])
    else:
        args['max_position'] = 100

    if 'delimiter' in e.attrib:
        args['delimiter'] = str(e.attrib['delimiter'])
    else:
        args['delimiter'] = ' '

    if 'group_tag' in e.attrib:
        args['group_tag'] = str(e.attrib['group_tag'])
    else:
        args['group_tag'] = 'I-NP'

s = self.__reader.act_groups(args)

return s

```


APÊNDICE P - Código do módulo readler

```

import os
import nltk
try:
    import cPickle as pickle
except:
    import pickle
import xml.etree.cElementTree as etree
from analysis.segmenter.xsegmenter import segmenter
from analysis.tokenizer.xtokenizer import tokenizer
from analysis.postagger.xclstagger import clstagger
from analysis.counter.xcounter import counter
from analysis.readability.xreadability import readability
from analysis.cogtagger.xcogtagger import cogtagger
from analysis.keywordsextractor.xkeywordsextractor import keywordsextractor
from analysis.nptagger.xnounphrasetagger import nounphrasetagger
from analysis.vgtagger.xvgtagger import verbgroupstagger
from analysis.stemmer.xstemmer import stemmer
from analysis.affixes.xaffixes import affixes
from analysis.vocabpro.xvocabpro import vocabpro

# -----
# -----
# -----
# TEXT
# -----
# -----

def text_analyze(raw_text_in_xml,language='en',genre='all'):
    root = __build (raw_text_in_xml,language,genre)
    return root

# -----
# -----
# -----
# -----

def __build (raw_text_in_xml,language='en',genre='all'):
    # get args from raw_text_in
    title,subtitle,author_source,text = __text_in_xml_to_args(raw_text_in_xml)
    raw_text_in_xml = None
    # process text
    processed_text = __process(text,language)
    text = None
    # create XML
    # <xmltext>
    root = etree.Element('xmltext')
    # <genre>
    e_genre = etree.SubElement(root,'genre')
    e_genre_category = etree.SubElement(e_genre,'category')
    e_genre_category.text = genre
    # <statistics>

```

```

e_statistics = etree.SubElement(root, 'statistics')
for label in ['tokens', 'types', 'typetokenratio', 'paragraphs', 'sentences',
              'words', 'averagewordspersentence', 'numberofcharacters',
              'syllables', 'averagesyllablesperword', 'fleschreadingease',
              'fleschkincaid', 'cognates', 'cognatesratio']:
    new_e = etree.SubElement(e_statistics, label)
    new_e.text = '*'
# <text_title>
e_text_title = etree.SubElement(root, 'text_title')
e_text_title.text = title
# <subtitle>
e_text_subtitle = etree.SubElement(root, 'text_subtitle')
e_text_subtitle.text = subtitle
# <text_author>
e_text_author = etree.SubElement(root, 'text_author')
e_text_author = author_source
# <text_body>
stm = stemmer(language)
np = nounphrasetag (language)
vg = verbgroup (language)
affix = affixes(language)
vocab = vocabpro()
text_body = etree.SubElement(root, 'text_body')
par_id = 0
sent_id = 0
tkn_id = 0
list_of_words = []
list_of_pos = []
number_of_sentences = 0
number_of_paragraphs = 0
# <paragraph>
for paragraph in processed_text:
    number_of_paragraphs+=1
    par = etree.SubElement(text_body, 'paragraph')
    par.set('id', str(par_id))
    # <sentence>
    for sentence in paragraph:
        number_of_sentences+=1
        sent = etree.SubElement(par, 'sentence')
        sent.set('id', str(sent_id))
        sent_id+=1
        idx = 0
        # noun phrase & verb group
        np_parsed = np.tag(np.tplsentence_to_tokens(sentence))
        np_parsed = vg.tag(np_parsed)
        # <token>
        for token in sentence:
            list_of_words.append(token[0])
            list_of_pos.append(token[1])
            tkn = etree.SubElement(sent, 'token')
            tkn.set('id', str(tkn_id))
            tkn.text = token[0]
            tkn.set('pos', token[1])
            tkn.set('size', str(len(token[0])))
            stem = stm.stem(token[0])
            tkn.set('stem', stem)
            tkn.set('voc', vocab.classify(stem))
            tkn.set('suf', affix.extract_suffixes(token[0], token[1], stem))
            tkn.set('grp', np_parsed[idx][1] )

```

```

        tkn_id+=1
        idx+=1
    par_id+=1
    # release resources
    stm = None
    np = None
    vg = None
    affix = None
    # analysis
    # frequency list
    cfreq = counter()
    wordlist,types,tokens,type_token_ratio = cfreq.count_from_list(list_of_words,
cfreq.LOWER)
    # frequency poslist
    poslist,pos_types,pos_tokens,pos_type_token_ratio =
cfreq.count_from_list(list_of_pos, cfreq.NONE)
    # count words
    total_words = cfreq.count_words_from_tokens(list_of_words)
    # average words per sentence
    avgwordspersent = float(total_words) / float(number_of_sentences)
    # char count
    nchars = cfreq.count_characters_from_tokens(list_of_words)
    cfreq = None
    # readability
    readab = readability()
    syllables = readab.count_syllables_from_tokens (list_of_words)
    syllablesperword = float(syllables) / float(total_words)
    flesch_ease = readab.FleschReadingEase(avgwordspersent, syllables,
total_words)
    flesch_kincaid = readab.FleschKincaidGradeLevel(avgwordspersent, syllables,
total_words)
    readab = None
    # cogtagger
    cog = cogtagger(language)
    dic_cognates = cog.get_cognates_from_tokens(list_of_words)
    numcognates = len(dic_cognates)
    cognates_ratio = float(numcognates)/ float(total_words)
    cog = None
    # keywords
    ke = keywordsextractor(language)
    keywordlist = {}
    for k in wordlist:
        keywordlist[k] = ke.extract(k.lower(), wordlist[k.lower()], tokens)
    ke = None
    # release resources
    list_of_words = None
    list_of_pos = None
    # add analysis
    for e in root.iter('token'):
        e.set('loc', str(float("{:10.2f}".format((float(e.attrib['id'])) /
float(tokens))*100 ))))
        e.set('key',str(keywordlist[e.text.lower()]))
        e.set('cog', str(dic_cognates[e.text]))
        e.set('freq',str(wordlist[e.text.lower()]))
    # release resources
    keywordlist = None
    dic_cognates = None
    # statistics
    for statistics in root.iter('statistics'):

```

```

    for e in statistics:
        if e.tag == 'tokens':
            e.text = str(tokens)
        elif e.tag == 'types':
            e.text = str(types)
        elif e.tag == 'typetokenratio':
            e.text = str(type_token_ratio)
        elif e.tag == 'paragraphs':
            e.text = str(number_of_paragraphs)
        elif e.tag == 'sentences':
            e.text = str(number_of_sentences)
        elif e.tag == 'words':
            e.text = str(total_words)
        elif e.tag == 'averagewordspersentence':
            e.text = str(avgwordspersent)
        elif e.tag == 'numberofcharacters':
            e.text = str(nchars)
        elif e.tag == 'syllables':
            e.text = str(syllables)
        elif e.tag == 'averagesyllablesperword':
            e.text = str(syllablesperword)
        elif e.tag == 'fleschreadingease':
            e.text = str(flesch_ease)
        elif e.tag == 'fleschkincaid':
            e.text = str(flesch_kincaid)
        elif e.tag == 'cognates':
            e.text = str(numcognates)
        elif e.tag == 'cognatesratio':
            e.text = str(cognates_ratio)

    #print __ident_xml(etree.tostring(root))
    return root

def xml_to_string(element):
    return __ident_xml(etree.tostring(element))

def xmltext_save(xmltext,path):
    f = open(path,'w')
    f.write(__ident_xml(etree.tostring(xmltext)))
    f.close()

# -----
# -----
# -----
# UTIL
# -----
# -----
# -----
def __ident_xml(xmlstring):

    xmlstring = xmlstring.replace('<title>','\n <title>')
    xmlstring = xmlstring.replace('<text>','\n <text>')
    xmlstring = xmlstring.replace('<paragraph>','\n     <paragraph>')
    xmlstring = xmlstring.replace('<sentence ','\n         <sentence ')
    xmlstring = xmlstring.replace('<token ','\n             <token ')
    xmlstring = xmlstring.replace('</sentence>','\n         </sentence>')
    xmlstring = xmlstring.replace('</paragraph>','\n     </paragraph>')

```



```

        s.append('')
    elif c == '&':
        s.append('and')
    elif c == '"':
        s.append('')

    else:
        if c not in unwanted:
            unwanted.append(c)

    content = ''.join(s)
except:
    content = '-'
try:
    content = content.replace("'", '"')
    content = content.replace('"', '"')
    content = content.replace("<", "<")
    content = content.replace(">", ">")
    content = content.replace("&", "&")
    content = content.replace(""", """)
    content = content.replace("<img alt=", "<img alt=")
    content = content.replace("<img alt=", "<img alt=")
    content = content.replace("<img alt=", "<img alt=")
except:
    pass

return content

def __text_in_xml_to_args(raw_text_in_xml):
    content = __chr(raw_text_in_xml)
    author_source = '-'
    root = etree.fromstring(content)
    for e in root:
        if e.tag == 'title':
            title = __chr(e.text)
        elif e.tag == 'subtitle':
            subtitle = __chr(e.text)
        elif e.tag == 'authorsource':
            try:
                author_source = __chr(e.text)
            except:
                author_source = '-'
        elif e.tag == 'text_body':
            text = e.text

    return title, subtitle, author_source, text

def __process(raw_text, language='en'):
    seg = segmenter(language)
    tk = tokenizer(language)
    pos = clstagger(language)
    segmented_text = seg.segment(raw_text)
    processed_text = []
    for par in segmented_text:
        paragraph = []
        for sent in par:
            sentence = []
            tokens = tk.tokenize(sent)

```

```
        tagged = pos.tag(tokens)
        for token in tagged:
            sentence.append(token)
        paragraph.append(sentence)
    processed_text.append(paragraph)
return processed_text
```

APÊNDICE Q - Código do módulo builder

```

import os
import nltk
import xmltextreader
import xinterpreter
import xml.etree.cElementTree as etree

def build_from_string(xmltext):
    root = etree.fromstring(xmltext)
    activity = __create(root)
    return activity

def __create(root):
    reader = xmltextreader.xmltextreader()
    reader.read_from_root(root)
    genre = reader.genre_category_get()

    xmltemplate = []
    xmltemplate.append('<?xml version="1.0" encoding="ISO-8859-1"?>\n<activities>\n')

    rules = __load_rules(genre)
    parts = []
    for rule in rules:
        if rule[0] == 'pass':
            if rule[1] == 'pass':
                if rule[2] == 'add':
                    parts.append([rule[2], rule[3], __add(rule[3])])

    for part in parts:
        xmltemplate.append(part[2])

    xmltemplate.append('\n</activities>')
    xmltemplate = ''.join(xmltemplate).strip()

    maker = xinterpreter.interpreter()
    activity = maker.translate(etree.tostring(root), xmltemplate)

    return activity

def __add(template_name):
    f = open('activities/templates/' + template_name + '.xml', 'r')
    content = f.read()
    f.close()
    return content.strip()

def __load_rules(genre):
    try:
        f = open('activities/rules/' + genre + '.txt', 'r')
        content = f.read()
        f.close()
    except:
        f = open('activities/rules/all.txt', 'r')
        content = f.read()

```



```
f.close()

lines = content.strip().split('\n')
rules = []
for line in lines:
    if len(line.strip()) > 0 and line.startswith('#') == False:
        fields = line.strip().split(' ')
        rules.append(fields)
return rules
```

APÊNDICE R - Código do módulo xmltextreader

```

import os
import xml.etree.cElementTree as etree
from operator import itemgetter
import random
import re

class xmltextreader:

    def __init__(self):
        self.__path = os.path.dirname(os.path.abspath(__file__))

# -----
# -----
# OPEN XML
# -----
# -----

    def read_from_file(self,filename):
        self.__tree = etree.parse(filename)
        self.__root = self.__tree.getroot()

    def read_from_string(self,s):
        self.__tree = etree.fromstring(s)
        self.__root = self.__tree

    def read_from_root(self,root):
        self.__tree = root
        self.__root = self.__tree

# -----
# -----
# STATISTICS
# -----
# -----

    def statistics_number_of_tokens(self):
        """Returns the number of tokens as integer"""
        for e in self.__root.iter('tokens'):
            if e.tag == 'tokens':
                return int(e.text)

    def statistics_number_of_types(self):
        """Returns the number of types as integer"""
        for e in self.__root.iter('types'):
            if e.tag == 'types':
                return int(e.text)

    def statistics_type_token_ratio(self):
        """Returns the type token ratio as float"""
        for e in self.__root.iter('typetokenratio'):
            if e.tag == 'typetokenratio':
                return float(e.text)

```

```

def statistics_number_of_paragraphs(self):
    """Returns the number of paragraphs as integer"""
    for e in self.__root.iter('paragraphs'):
        if e.tag == 'paragraphs':
            return int(e.text)

def statistics_number_of_sentences(self):
    """Returns the number of sentences as integer"""
    for e in self.__root.iter('sentences'):
        if e.tag == 'sentences':
            return int(e.text)

def statistics_number_of_words(self):
    """Returns the number of words as integer"""
    for e in self.__root.iter('words'):
        if e.tag == 'words':
            return int(e.text)

def statistics_average_words_per_sentence(self):
    """Returns the average number of words per sentence as float"""
    for e in self.__root.iter('averagewordspersentence'):
        if e.tag == 'averagewordspersentence':
            return float(e.text)

def statistics_flesch_reading_ease(self):
    """Returns the flesch reading ease as float"""
    for e in self.__root.iter('fleschreadingease'):
        if e.tag == 'fleschreadingease':
            return float(e.text)

def statistics_flesch_reading_ease_label(self):
    """Returns the flesch reading ease as string"""
    score = 0
    for e in self.__root.iter('fleschreadingease'):
        if e.tag == 'fleschreadingease':
            score = float(e.text)
    label = ''
    if score > 0 and score < 29:
        label = 'Very confusing'
    elif score > 30 and score < 49:
        label = 'Difficult'
    elif score > 50 and score < 59:
        label = 'Fairly Difficult'
    elif score > 60 and score < 69:
        label = 'Standard'
    elif score > 70 and score < 79:
        label = 'Fairly easy'
    elif score > 80 and score < 89:
        label = 'Easy'
    elif score > 90 and score < 100:
        label = 'Very easy'
    else:
        label = 'Error'
    return label

def statistics_number_of_cognates(self):
    """Returns the number of cogtagger as integer"""
    for e in self.__root.iter('cognates'):

```

```

        if e.tag == 'cognates':
            return int(e.text)

    def statistics_cognates_ratio(self):
        """Returns the cogtagger ratio as float"""
        for e in self.__root.iter('cognatesratio'):
            if e.tag == 'cognatesratio':
                return float(e.text)

    def statistics_number_of_lex_gram_words(self):
        """Returs the number of lexical and grammatical words as two integer
values"""
        import string
        p_lex = re.compile("NN|VB|ADJ")
        l = 0
        g = 0

        for e in self.__root.iter('token'):
            if e.text not in string.punctuation:
                if len(p_lex.findall(e.attrib['pos'])) > 0:
                    l+=1
                else:
                    g+=1

        return l,g

    def statistics_lexical_density(self):
        """ Returns the lexical density as float"""
        import string
        p_lex = re.compile("CD|FW|JJ|NN|RB|VB")
        l = 0
        g = 0

        for e in self.__root.iter('token'):
            if e.text not in string.punctuation:
                if len(p_lex.findall(e.attrib['pos'])) > 0:
                    l+=1
                else:
                    g+=1
        total = l + g
        lexical_density = float(l) / float(total)
        return lexical_density

# -----
# -----
# -----
# GENRE
# -----
# -----
# -----

    def genre_category_get(self):
        """Returns the genre category as string"""
        for genre in self.__root.iter('genre'):
            if genre.tag == 'genre':
                for e in genre:
                    if e.tag == 'category':
                        return str(e.text)

```

```

# -----
# -----
# XML
# -----
# -----

def xml_sentence_by_id(self,sentence_id):
    e_sentence = None
    for e in self.__root.iter('sentence'):
        if int(e.attrib['id']) == sentence_id:
            e_sentence = e
            break
    return e_sentence

def xml_sentence_to_string(self,e_sentence):
    tmp=[]
    for e in e_sentence:
        if e.tag == 'token':
            tmp.append(e.text + ' ')

    sentence = ''.join(tmp).strip()
    return sentence

def xml_sentence_get_all(self):
    sentences = []
    for e in self.__root.iter('sentence'):
        sentences.append(e)
    return sentences

def xml_paragraph_by_id(self,paragraph_id):
    e_paragraph = None
    for e in self.__root.iter('paragraph'):
        if int(e.attrib['id']) == paragraph_id:
            e_paragraph = e
            break

    return e_paragraph

def xml_paragraph_to_string(self,e_paragraph):
    tmp=[]
    for e in e_paragraph:
        if e.tag == 'sentence':
            for token in e:
                if token.tag == 'token':
                    tmp.append(token.text + ' ')

    paragraph = ''.join(tmp).strip()
    return paragraph

def xml_token_by_id(self,token_id):
    e_token = None
    for e in self.__root.iter('token'):
        if int(e.attrib['id']) == token_id:
            e_token = e

```

```

        break
    return e_token

def xml_token_to_string(self,e_token):
    tmp=[]
    token = e_token.text
    return token

def xml_token_get_all(self):
    tokens = []
    for e in self.__root.iter('token'):
        tokens.append(e)
    return tokens

def xml_paragraph_get_all(self):
    paragraphs = []
    for e in self.__root.iter('paragraph'):
        paragraphs.append(e)
    return paragraphs

def xml_root_to_string(self):
    """Returns xml object as string """
    return etree.tostring(self.__root)

# -----
# -----
# -----
# PARTS
# -----
# -----
# -----

def title_get(self):
    """Returns the title of the text as string"""
    for e in self.__root.iter('text_title'):
        if e.tag == 'text_title':
            return e.text

def subtitle_get(self):
    """Returns the subtitle of the text as string"""
    for e in self.__root.iter('text_subtitle'):
        if e.tag == 'text_subtitle':
            return e.text

def author_get(self):
    """Returns the author of the text as string"""
    for e in self.__root.iter('text_author'):
        if e.tag == 'text_author':
            return e.text

# -----
# -----
# -----
# COGNATES
# -----
# -----
# -----

```

```

def get_cognate_words(self):
    """Returns a list of cognate words as a list"""
    cognates = []
    for e in self.__root.iter('token'):
        if e.attrib['cog'] == '1':
            cognates.append(e.text)
    return cognates

def get_cognate_types(self):
    """Returns a list of cognate type words as a list"""
    cognates = []
    for e in self.__root.iter('token'):
        if e.attrib['cog'] == '1':
            if e.text.lower() not in cognates:
                cognates.append(e.text.lower())
    return cognates

def get_cognates(self, limit=10, delimiter = ' ', case=None, minsize=None,
maxsize=None, stoplist=None, pos=None):
    words = []
    stoplist = stoplist.lower().split(' ')
    if pos != None:
        pos = pos.upper().split(' ')
    for e in self.__root.iter('token'):
        if e.attrib['cog'] == '1':
            flag = True
            # check conditions
            if minsize != None:
                if len(e.text) < minsize:
                    flag = False
            if maxsize != None:
                if len(e.text) > maxsize:
                    flag = False
            if stoplist != None:
                if e.text.lower() in stoplist:
                    flag = False
            if pos != None:
                if e.attrib['pos'] not in pos:
                    flag = False
            # add or not
            if flag == True:
                if case == None:
                    words.append(e.text)
                elif case == 'upper':
                    words.append(e.text.upper())
                elif case == 'lower':
                    words.append(e.text.lower())
                else:
                    words.append(e.text)

    # check total words and limit
    total_words = len(words)

    if total_words < limit:
        limit = total_words
        selected = []
        for word in words:
            selected.append(word + delimiter)

```

```

else:
    wordlist = []
    i = 0
    while i != limit:
        idx = random.randrange(0, total_words)
        if words[idx] not in wordlist:
            wordlist.append(words[idx])
            i+=1
    selected = []
    for word in wordlist:
        selected.append(word + delimiter)

    return ''.join(selected)

# -----
# -----
# -----
# FREQUENCY LISTS
# -----
# -----
# -----

def get_frequency_type_list_desc(self):
    """Returns a frequency list ordered by freq desc as a list of tuples"""
    dic = {}
    for e in self.__root.iter('token'):
        if dic.has_key(e.text.lower()) == False:
            dic[e.text.lower()] = int(e.attrib['freq'])

    freqlist = []
    for k in sorted(dic.iteritems(),key=itemgetter(1),reverse = True):
        freqlist.append(( k[0],k[1] ) )

    return freqlist

def get_frequency_type_list_asc(self):
    """Returns a frequency list ordered by freq asc as a list of tuples"""
    dic = {}
    for e in self.__root.iter('token'):
        if dic.has_key(e.text.lower()) == False:
            dic[e.text.lower()] = int(e.attrib['freq'])

    freqlist = []
    for k in sorted(dic.iteritems(),key=itemgetter(1),reverse = False):
        freqlist.append(( k[0],k[1] ) )

    return freqlist

# -----
# -----
# -----
# POS
# -----
# -----
# -----

def get_POS_frequency_list(self):

```



```

        """Returns a POS frequency list ordered by freq desc as a list of
        tuples"""
        dic = {}
        for e in self.__root.iter('token'):
            if e.attrib['pos'] in dic:
                dic[e.attrib['pos']] += 1
            else:
                dic[e.attrib['pos']] = 1

        freqlist = []
        for k in sorted(dic.iteritems(), key=itemgetter(1), reverse = True):
            freqlist.append(( k[0], k[1] ) )

        return freqlist

# -----
# -----
# -----
# KEYWORDS
# -----
# -----
# -----

    def get_keywordlist_filtered(self):
        """Returns a list of keywords order by keynesss desc as a list of words
        filtered"""
        import string

        stop_pos = ['DT', 'IN', 'TO']

        dic = {}
        for e in self.__root.iter('token'):
            if int(e.attrib['freq']) > 1 and e.attrib['pos'] not in stop_pos:
                if e.text not in string.punctuation:
                    if e.text not in dic:
                        dic[e.text] = float(e.attrib['key'])
        keywords = []
        for k in sorted(dic.iteritems(), key=itemgetter(1), reverse = True):
            keywords.append(k[0])

        return keywords

    def get_keywordlist_desc(self):
        """Returns a list of keywords order by keynesss desc as a list"""
        import string

        dic = {}
        for e in self.__root.iter('token'):
            if e.text not in string.punctuation:
                if e.text not in dic:
                    dic[e.text] = float(e.attrib['key'])
        keywords = []
        for k in sorted(dic.iteritems(), key=itemgetter(1), reverse = True):
            keywords.append(( k[0], k[1] ) )

        return keywords

    def get_keywordlist_asc(self):

```

```

"""Returns a list of keywords order by keynesss desc as a list"""
import string

dic = {}
for e in self.__root.iter('token'):
    if e.text not in string.punctuation:
        if e.text not in dic:
            dic[e.text] = float(e.attrib['key'])
keywords = []
for k in sorted(dic.iteritems(),key=itemgetter(1),reverse = False):
    keywords.append(( k[0],k[1] ) )

return keywords

def get_keywords(self, limit=10, delimiter = ' ', case=None, minsize=None,
maxsize=None, stoplist=None,pos=None):
    words = []
    stoplist = stoplist.lower().split(' ')
    if pos != None:
        pos = pos.upper().split(' ')
    for e in self.__root.iter('token'):
        if float(e.attrib['key']) != 0.0:
            flag = True
            # check conditions
            if minsize != None:
                if len(e.text) < minsize:
                    flag = False
            if maxsize != None:
                if len(e.text) > maxsize:
                    flag = False
            if stoplist != None:
                if e.text.lower() in stoplist:
                    flag = False
            if pos != None:
                if e.attrib['pos'] not in pos:
                    flag = False
            # add or not
            if flag == True:
                if case == None:
                    words.append(e.text)
                elif case == 'upper':
                    words.append(e.text.upper())
                elif case == 'lower':
                    words.append(e.text.lower())
                else:
                    words.append(e.text)

    # check total words and limit
    total_words = len(words)

    if total_words < limit:
        limit = total_words
        selected = []
        for word in words:
            selected.append(word + delimiter)
    else:
        wordlist = []
        i = 0
        while i != limit:

```

```

        idx = random.randrange(0, total_words)
        if words[idx] not in wordlist:
            wordlist.append(words[idx])
            i+=1
        selected = []
        for word in wordlist:
            selected.append(word + delimiter)

    return ''.join(selected)

# -----
# -----
# -----
# N-GRAMS
# -----
# -----

def get_ngrams_frequency_list(self,n=3):
    """Returns ngrams from the text as a list of tuples"""
    tokens = []
    for e in self.__root.iter('token'):
        tokens.append(e.text)
    tokens+=['<s> ', '<s>', '<s>', '<s>', '<s>']

    freqlist = {}
    for i in range(0, len(tokens)-5):
        if n == 2:
            item = (tokens[i].lower() , tokens[i+1].lower())
        elif n == 3:
            item = (tokens[i].lower() , tokens[i+1].lower(),
tokens[i+2].lower())
        elif n == 4:
            item = (tokens[i].lower() , tokens[i+1].lower(),
tokens[i+2].lower(), tokens[i+3].lower())
        else:
            item = (tokens[i].lower() , tokens[i+1].lower())

        if '<s>' not in item:
            if item in freqlist:
                freqlist[item]+=1
            else:
                freqlist[item]=1

    ngrams = []
    for k in sorted(freqlist.iteritems(),key=itemgetter(1),reverse = True):
        ngrams.append(( k[0],k[1] ) )

    return ngrams

# -----
# -----
# -----
# RANDOM WORDS
# -----
# -----

```

```

def get_random_word(self):
    """Returns a random word from the text as string"""
    import string

    words = []
    total = 0
    for e in self.__root.iter('token'):
        words.append(e.text)
        total+=1
    word = None
    while word == None:
        idx = random.randrange(0, total)
        if words[idx] not in string.punctuation:
            word = words[idx]

    return word

def get_random_list_of_words(self,limit):
    """Returns a list of random words as list"""
    words = []
    for i in range(0,limit):
        words.append(self.get_random_word())
    return words

def get_random_words(self, limit=10, delimiter = ' ', case=None, minsize=None,
maxsize=None, stoplist=None, pos=None):
    import string
    words = []
    stoplist = stoplist.lower().split(' ')
    if pos != None:
        pos = pos.upper().split(' ')
    for e in self.__root.iter('token'):
        flag = True
        # check conditions
        if e.text in string.punctuation:
            flag = False
        if minsize != None:
            if len(e.text) < minsize:
                flag = False
        if maxsize != None:
            if len(e.text) > maxsize:
                flag = False
        if stoplist != None:
            if e.text.lower() in stoplist:
                flag = False
        if pos != None:
            if e.attrib['pos'] not in pos:
                flag = False

        # add or not
        if flag == True:
            if case == None:
                words.append(e.text)
            elif case == 'upper':
                words.append(e.text.upper())
            elif case == 'lower':
                words.append(e.text.lower())
            else:
                words.append(e.text)

```



```

        else:
            noun_phrases[''.join(np).strip()]=1
            np = []
            i+=1
    freqlist = []
    for k in sorted(noun_phrases.iteritems(),key=itemgetter(1),reverse =
True):
        freqlist.append(( k[0],k[1] ) )

    return freqlist

# -----
# -----
# -----
# wheighted lists
# -----
# -----
# -----

def get_wheighted_frequency_list(self):
    "Returns a wheighted frequency list as a list of tuples"
    tokens = self.get_number_of_tokens()
    wheighted_dic = {}
    for e in self.__root.iter('token'):
        if e.text.lower() not in wheighted_dic:
            wheight = (float( int(e.attrib['freq'])) / float(tokens)) * 100
            wheighted_dic[e.text.lower()] = wheight

    output_list = []
    for k in sorted(wheighted_dic.iteritems(),key=itemgetter(1),reverse =
True):
        output_list.append((k[0],k[1]))

    return output_list

# -----
# -----
# -----
# FILTER TOKENS
# -----
# -----
# -----

def filter_tokens(self,args, token_history=[]):
    import string # for punctuation
    tokens = [] # will the final token list
    total_tokens = self.statistics_number_of_tokens() + 1 # +1 to avoid
division by zero
    i=0
    # iterate through each token in text
    for e in self.__root.iter('token'):
        # add the token as default
        flag = True
        # calc position
        try:
            # +1 to avoid division by zero

```

```

        position = int( (float(int(e.attrib['id']))+1) /
float(total_tokens)) * 100)
    except:
        position = 1

    # CONDITIONS:
    if 'unique' in args:
        if args['unique'] == 'yes':
            for token in tokens:
                if e.text.lower() == token.text.lower():
                    flag = False
                    break

    if 'minsize' in args:
        if len(e.text) < args['minsize']:
            flag = False

    if 'maxsize' in args:
        if len(e.text) > args['maxsize']:
            flag = False

    if 'stoplist' in args:
        if e.text.lower() in args['stoplist']:
            flag = False

    if 'pos' in args:
        if e.attrib['pos'] not in args['pos']:
            flag = False

    if 'stoppos' in args:
        if e.attrib['pos'] in args['stoppos']:
            flag = False

    if 'punct' in args:
        if args['punct'] == 'no':
            if e.text in string.punctuation:
                flag = False

    if 'cognate' in args:
        if args['cognate'] == 'yes':
            if e.attrib['cog'] == '0':
                flag = False

    if 'minfreq' in args:
        if int(e.attrib['freq']) < args['minfreq']:
            flag = False

    if 'maxfreq' in args:
        if int(e.attrib['freq']) > args['maxfreq']:
            flag = False

    if 'regexp' in args:
        try:
            pattern = args['regexp']
            if len(re.findall(pattern, e.text)) == 0:
                flag=False
        except:
            pass

```

```

    if 'suffix' in args:
        if e.attrib['suf'] != args['suffix']:
            flag = False

    if 'min_position' in args:
        if position < args['min_position']:
            flag = False

    if 'max_position' in args:
        if position > args['max_position']:
            flag = False

    if 'repeat' in args:
        if args['repeat'] == 'no':
            if e.text.lower() in token_history:
                flag = False

    if 'word' in args:
        if e.text.lower() != args['word'].lower():
            flag = False

    # add it or not, according to conditions
    if flag == True:
        tokens.append(e)
        i+=1

    if args['limit'] == i:
        break

# SORT

if 'order' in args:
    dic = {}
    newlist = []
    if args['order'] == 'keyness':
        for i in range(0,len(tokens)):
            dic[i] = float(tokens[i].attrib['key'])
        for k in sorted(dic.iteritems(),key=itemgetter(1),reverse = True):
            newlist.append(tokens[k[0]])
        tokens = newlist
    elif args['order'] == 'keyness-asc':
        for i in range(0,len(tokens)):
            dic[i] = float(tokens[i].attrib['key'])
        for k in sorted(dic.iteritems(),key=itemgetter(1),reverse =
False):
            newlist.append(tokens[k[0]])
        tokens = newlist
    elif args['order'] == 'freq-asc':
        for i in range(0,len(tokens)):
            dic[i] = int(tokens[i].attrib['freq'])
        for k in sorted(dic.iteritems(),key=itemgetter(1),reverse =
False):
            newlist.append(tokens[k[0]])
        tokens = newlist
    elif args['order'] == 'freq-desc':
        for i in range(0,len(tokens)):
            dic[i] = int(tokens[i].attrib['freq'])

```



```

        for k in sorted(dic.iteritems(),key=itemgetter(1),reverse = True):
            newlist.append(tokens[k[0]])
        tokens = newlist
    elif args['order'] == 'word-asc':
        for i in range(0,len(tokens)):
            dic[i] = str(tokens[i].text)
        for k in sorted(dic.iteritems(),key=itemgetter(1),reverse =
False):
            newlist.append(tokens[k[0]])
            tokens = newlist
    elif args['order'] == 'word-desc':
        for i in range(0,len(tokens)):
            dic[i] = str(tokens[i].text)
        for k in sorted(dic.iteritems(),key=itemgetter(1),reverse = True):
            newlist.append(tokens[k[0]])
            tokens = newlist
    elif args['order'] == 'position-asc':
        for i in range(0,len(tokens)):
            dic[i] = int(tokens[i].attrib['id'])
        for k in sorted(dic.iteritems(),key=itemgetter(1),reverse =
False):
            newlist.append(tokens[k[0]])
            tokens = newlist
    elif args['order'] == 'position-desc':
        for i in range(0,len(tokens)):
            dic[i] = int(tokens[i].attrib['id'])
        for k in sorted(dic.iteritems(),key=itemgetter(1),reverse = True):
            newlist.append(tokens[k[0]])
            tokens = newlist
    elif args['order'] == 'random' or args['order'] == 'shuffle':
        x = [[i] for i in range(0,len(tokens))]
        random.shuffle(x)
        for i in x:
            newlist.append(tokens[i[0]])
        tokens = newlist

# prepare for return
before_token = ''
if 'before-token' in args:
    before_token = args['before-token']

delimiter = ' '
if 'delimiter' in args:
    delimiter = args['delimiter']

# concatenate everything
s = []

for token in tokens:
    if token.text.lower() not in token_history:
        token_history.append(token.text.lower())

    if args['case'] == 'upper':
        s.append(before_token + token.text.upper() + delimiter)
    elif args['case'] == 'lower':
        s.append(before_token + token.text.lower() + delimiter)
    else:

```

```

        s.append(before_token + token.text + delimiter)
    # return

    return ''.join(s), token_history

# -----
# -----
# FILTER SENTENCES
# -----
# -----
def filter_sentences(self, args, sentence_history=[]):
    sentences = []

    i = 0
    for e in self.__root.iter('sentence'):
        flag = True

        # add it or not, according to conditions
        if flag == True:
            sentences.append(e)
            i+=1
            ...
        if args['limit'] == i:
            break
        ...

    return sentences

# -----
# -----
# WORD CLOUDS
# -----
# -----
def get_word_cloud_from_frequency(self):
    """Returns a word cloud from frequency as string"""
    import string

    stop_pos = ['DT', 'IN', 'TO', 'CC', 'RB']

    dic = {}
    for e in self.__root.iter('token'):
        if int(e.attrib['freq']) > 1 and e.attrib['pos'] not in stop_pos and
len(e.text) > 2:
            if e.text not in string.punctuation:
                if e.text not in dic:
                    dic[e.text] = float(e.attrib['freq'])

    max_value = max(dic.iteritems(), key=itemgetter(1))[1]
    min_value = min(dic.iteritems(), key=itemgetter(1))[1]
    max_size = 40
    min_size = 8

```

```

        colors = {}
        colors[0]= '#000000'
        colors[1]= '#484848'
        colors[2]= '#787878'
        colors[3]= '#A8A8A8'
        colors[4]= '#C0C0C0'

        keywords = []
        for k in dic:
            size = int(round((float( dic[k] ) / max_value) * (max_size - min_size)
+ min_size))
            i = random.randrange(0,4)
            keywords.append((k,str(size), colors[i]))

        dic = None
        random.shuffle(keywords)
        wordcloud = []

        for item in keywords:
            wordcloud.append('<span style="font-size:%spx;color:%s;" >%s</span> '
% (item[1],item[2], item[0]))

        return ''.join(wordcloud)

def get_word_cloud_from_keywords(self):
    """Returns a word cloud from keywords as string"""
    import string

    stop_pos = ['DT','IN','TO','CC','RB']

    dic = {}
    for e in self.__root.iter('token'):
        if int(e.attrib['freq']) > 1 and e.attrib['pos'] not in stop_pos and
len(e.text) > 2:
            if e.text not in string.punctuation:
                if e.text not in dic:
                    dic[e.text] = float(e.attrib['key'])

    max_value = max(dic.iteritems(), key=itemgetter(1))[1]
    min_value = min(dic.iteritems(), key=itemgetter(1))[1]
    max_size = 40
    min_size = 8

    colors = {}
    colors[0]= '#000000'
    colors[1]= '#484848'
    colors[2]= '#787878'
    colors[3]= '#A8A8A8'
    colors[4]= '#C0C0C0'

    keywords = []
    for k in dic:
        size = int(round((float( dic[k] ) / max_value) * (max_size - min_size)
+ min_size))
        i = random.randrange(0,4)
        keywords.append((k,str(size), colors[i]))

    dic = None

```

```

        random.shuffle(keywords)
        wordcloud = []

        for item in keywords:
            wordcloud.append('<span style="font-size:%spx;color:%s;" >%s</span> '
% (item[1],item[2], item[0]))

        return ''.join(wordcloud)

# -----
# -----
# -----
# KEYWORDS FROM PARAGRAPHS
# -----
# -----

def keywords_from_paragraphs(self,args):
    # conditions
    if 'pos' in args:
        pos = args['pos'].strip().split(' ')
    else:
        pos = ['NN']

    if 'delimiter' in args:
        delimiter = args['delimiter']
    else:
        delimiter = ' '

    if 'identpar' in args:
        identpar = args['identpar']
    else:
        identpar = 'no'

    if 'group' in args:
        flag_group = args['group']
    else:
        flag_group = 'no'

    if 'group_tag' in args:
        group_tag = args['group_tag']
    else:
        group_tag = 'I-NP'

    group = group_tag
    groups = []
    keywords = []
    npars = self.statistics_number_of_paragraphs()
    for i in range(0,npars):
        paragraph = self.xml_paragraph_by_id(i)
        keyword = []
        for sentence in paragraph:
            g=[]
            for token in sentence:
                if token.attrib['grp'] == group:
                    g.append(token.text)
                elif token.attrib['grp'] != group:
                    if len(g) == 0:

```

```

        pass
    else:
        groups.append((g,i+1))
        g = []
    if len(keyword) == 0:
        if token.text not in keywords:
            if token.attrib['pos'] in pos:
                keyword = [token.text,int(token.attrib['key'])]

    elif int(token.attrib['key']) > keyword[1]:
        if token.text not in keywords:
            if token.attrib['pos'] in pos:
                keyword = [token.text,int(token.attrib['key'])]

    keywords.append(keyword[0])

if flag_group == 'yes':
    i=0
    keygroups = []

    for word in keywords:
        i+=1
        candidates = []
        for group in groups:
            if group[1] == i:
                candidates.append(group[0])
        flag = True
        for candidate in candidates:
            if flag == True:
                if word in candidate:
                    keygroups.append(candidate)
                    flag = False

    keywords = []
    for group in keygroups:
        s = []
        for word in group:
            s.append(word + ' ')
        keywords.append(''.join(s).strip())

i = 0
s = []
for word in keywords:
    i +=1
    if identpar == 'yes':
        s.append(str(i) + '. ')
    s.append(word + delimiter)

return ''.join(s).strip()

```

```

# -----
# -----
# -----
# GROUPS

```