Maximização de Entropia em Lingüística Computacional para a Língua Portuguesa

Archias Alves de Almeida Filho

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO DE MESTRE

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Marcelo Finger

São Paulo
2003

**Maximização de Entropia
em Lingüística Computacional
para a Língua Portuguesa**

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por Archias Alves de Almeida Filho
e aprovada pela comissão julgadora.

São Paulo, 14 de abril de 2003

Banca examinadora:

- Prof. Dr. Marcelo Finger (orientador) (IME-USP)
- Prof. Dr. Júnior Barrera (IME-USP)
- Profa. Dra. Maria das Graças Volpe Nunes (ICMC-USP)

Resumo

Neste trabalho, estudamos o método estatístico de Maximização de Entropia, exemplificando o seu uso para resolução do problema lingüístico da Análise Morfossintática. Desta forma procuramos tornar o mesmo mais fácil de ser entendido e implementado para outros problemas de aprendizado computacional. Para tanto, o algoritmo de Maximização de Entropia é caracterizado como um método de aprendizado computacional, explicado e matematicamente definido. Em seguida é aplicado ao problema da Análise Morfossintática usando o Corpus Tycho Brahe como base de treinamento; seus resultados são comparados com os obtidos pelo método de Brill já aplicado ao mesmo Corpus e comparados com os resultados obtidos pelo mesmo método, porém aplicado ao Inglês.

Abstract

In this work, we study the Entropy Maximization statistical method, exemplifying it by solving the linguistic problem of Part-of-Speech Tagging. This way we try to make the algorithm easier to understand and to use in other computer learning problems. In order to do so, the Entropy Maximization Algorithm is described as a computer learning method, explained and mathematically defined. The algorithm is applied to the problem of Part-of-Speech Tagging using the Tycho Brahe Corpus as the basis for the training; the results are compared to the ones from Brill's method applied to the same Corpus and are compared to the ones obtained by another implementation of the same algorithm applied to English.

Agradecimentos

Agradeço o auxílio do meu orientador Prof. Dr. Marcelo Finger, cujos conselhos e experiência foram fundamentais para a consecução desta tese; e agradeço meus familiares sem cujo apoio e incentivo a conclusão desta tese não teria sido possível.

Conteúdo

1	Introdução	4
2	Aprendizado Computacional	6
2.1	O Modelo de Aprendizado Computacional	6
2.2	Aprendizado Supervisionado e Não-Supervisionado	7
2.3	Métodos de Aprendizado Computacional	8
2.4	Método de Brill variante Chacur-Finger	9
3	Estimação por Maximização de Entropia	10
3.1	Algoritmo de Maximização de Entropia	11
3.1.1	Exemplo do Algoritmo de Maximização	12
3.2	Definição de Entropia	14
3.3	Problema de Maximização de Entropia	15
3.4	Generalized Iterative Scaling	16
3.5	Dedução do Algoritmo de Maximização	23
4	Aplicação ao Problema da Análise Morfossintática	28
4.1	Resumo da Solução	29
4.2	Arquitetura Teórica da Solução	29
4.3	Conjunto de Treinamento	30
4.4	Definição de Funções Características	31
4.5	Geração Automática de Funções Características	32
4.6	Seleção de Funções Características	34
4.7	Classificação Morfossintática de um Termo	34
4.7.1	Procedimento de Treinamento	35
4.7.2	Procedimento de Ajuste	35
4.7.3	Procedimento de Teste	37
5	Aplicação Prática da Técnica	38
5.1	Arquitetura da Implementação	38
5.2	Detalhes da Implementação e Problemas Enfrentados	39

5.2.1	Processamento Inicial	39
5.2.2	Criação e Seleção de Funções Características	40
5.2.3	Maximização de Entropia	40
5.3	Resultados Obtidos	41
6	Conclusão	45
A	Programa de Exemplo	49
A.1	Onde Obter	49
A.2	Como Instalar	49
A.3	Como Utilizar o Programa	49
A.4	Descrição Sucinta das Classes do Programa	51

Capítulo 1

Introdução

O objetivo desta tese é estudar o método estatístico de aprendizado computacional conhecido como Maximização de Entropia, apresentando o método integralmente, desde os seus princípios matemáticos até estratégias de implementação e sua eficiência. Para tanto, usaremos a sua aplicação ao problema lingüístico da Análise Morfossintática, também conhecido como Etiquetagem, para exemplificar o seu uso e analisar o seu resultado. Este problema é básico para a lingüística e por isso será usado como exemplo.

A lingüística computacional é a parte da ciência da computação que trata dos problemas relacionados com o processamento da linguagem natural (falada ou escrita) por meio de computadores. Um dos problemas fundamentais desta área é justamente o problema da análise morfossintática que, simplificando, consiste em realizar a classificação gramatical das palavras de uma frase em contexto. A resolução deste problema permite a resolução de outros mais complexos e de aplicações mais interessantes como tradução automática de textos, extração de informações de textos, classificação automática de temas de textos e qualquer outro que esteja relacionado com algum tipo de processamento mais sofisticado da linguagem.

A resolução de qualquer problema nesta área não é simples, de forma que técnicas de inteligência artificial, em particular de aprendizado computacional, são usadas para tentar resolver problemas lingüísticos de forma satisfatória. No contexto de aprendizado computacional, as regras usadas para resolver o problema são “aprendidas” pelo computador, ao invés de definidas pelo programador. Esta abordagem é muito útil quando as regras não são claras, ou são muito complexas, ou são em grande quantidade, como é o caso da maioria dos problemas de lingüística computacional.

Abordaremos o aprendizado computacional com mais detalhes no próximo capítulo. No momento basta dizermos que diversas técnicas existem e dentre elas estaremos estudando em mais detalhes a técnica estatística de maximização de entropia. Esta técnica, como outras de aprendizado computacional, precisa de um grande conjunto de dados para ser aplicada. Com base neste conjunto de dados, chamado de corpus, as regras para resolução do problema são derivadas.

Os dados do corpus estão, portanto, muito relacionados com o problema a ser resolvido. No nosso caso, para resolver o problema da análise morfossintática estaremos utilizando o Corpus Tycho Brahe do Português histórico [3].

Esta técnica foi escolhida para ser estudada por causa dos excelentes resultados obtidos por outros pesquisadores como Ratnaparkhi [18], Della Pietra [7] e Rosenfeld [15] na resolução de problemas lingüísticos e por outras qualidades como sua facilidade de reaproveitamento para resolução de outros problemas. Esta técnica será abordada em maior profundidade no capítulo 3.

A adaptação desta técnica para resolver o problema da análise morfo-sintática é relativamente simples e mostrada no capítulo 4. O mesmo método usado para adaptar a este problema pode ser usado para resolver outros. Como ilustração do método foi construído ainda um programa para resolver o problema em questão usando esta técnica. Sua descrição e seus resultados estão mostrados no capítulo 5. No capítulo 6 concluímos a tese, analisando a aplicabilidade da técnica de maximização de entropia em geral, retomando os resultados e dificuldades encontrados na implementação do problema de etiquetagem em particular.

Capítulo 2

Aprendizado Computacional

O Aprendizado Computacional é uma área da ciência da computação que foi concebida há cerca de 40 anos com o ousado objetivo de desenvolver métodos computacionais para implementar diversas formas de aprendizado, em particular mecanismos capazes de induzir “conhecimento” a partir de exemplos ou dados.

Esta indução de conhecimento é particularmente desejada em problemas que não possuem solução algorítmica, são mal definidos, ou apenas enunciados informalmente. Exemplos de tais problemas englobam diagnóstico médico ou técnico, reconhecimento de conceitos visuais, projetos de engenharia, comportamento de materiais, jogos de xadrez, ou detecção de regularidades interessantes em grandes conjuntos de dados. Em particular, na área de lingüística computacional, problemas como classificação morfossintática, análise sintática, classificação de textos (*text categorization*) e colocação de sintagmas preposicionais (*prepositional phrase attachment*) são problemas nos quais a abordagem do Aprendizado Computacional é utilizada com sucesso.

2.1 O Modelo de Aprendizado Computacional

Aprendizado Computacional, como todas as áreas do conhecimento, possui um conjunto de termos próprios que auxiliam na descrição dos problemas e na sua resolução. A noção mais básica e importante é a noção de *conceito*. Conceito é uma abstração que representa um conjunto de objetos que compartilham algumas propriedades que os diferenciam de outros conjuntos de objetos (conceitos). Assim, ‘pássaro’, ‘tigre’, ‘vertebrado’, ‘número primo’, ‘dia chuvoso’, ‘galáxia’, ‘terra fértil’ são conceitos. Note que os limites de um conceito nem sempre são claros.

Matematicamente esta noção é um pouco mais complicada de definir, pois temos inicialmente que restringir o conjunto de coisas sobre o qual estamos falando (universo de discurso). Vamos definir este conjunto como sendo um

subconjunto $X \in \Sigma^*$, onde Σ^* é o conjunto de todas as cadeias não vazias, finitas de elementos de um alfabeto Σ . Desta forma, se no nosso alfabeto tivermos símbolos para representar características como número de pernas, se tem um exoesqueleto ou um endoesqueleto, tamanho, entre outras coisas, cada elemento de X pode descrever um animal ou inseto. Se quisermos então definir o conceito ‘tigre’ basta-nos uma função que nos diga se uma descrição em X corresponde ou não a um tigre (supondo que temos características suficientes em Σ para realizar com precisão esta definição). Seja $c : X \rightarrow \{0, 1\}$ esta função. Chamamos então c de conceito. X é conhecido como *espaço de exemplos* e cada um de seus elementos é chamado de *exemplo*. Quando $c(x) = 1$, dizemos que x é um exemplo positivo e x é um exemplo negativo quando $c(x) = 0$.

A tarefa do Aprendizado Computacional pode ser agora descrita como: a partir de exemplos e conhecimento prévio do domínio encontrar uma descrição satisfatória de um conceito. Com esta descrição é possível perguntar à máquina se um exemplo y é um ‘tigre’ e a máquina poderá nos responder com segurança. Mas infelizmente não é possível, na maioria dos casos, obter a descrição completa do conceito, pois os conjuntos de exemplos reais são muito grandes. O que se pode obter é apenas um subconjunto H do conjunto de conceitos possíveis. Chamaremos H de *espaço de hipóteses* e um seu elemento é conhecido por hipótese. Assim o que realmente podemos obter não é a descrição computacional do conceito mas de uma hipótese que aproxima o conceito. Mesmo assim, isso já conduz a resultados bastante satisfatórios.

Normalmente, para encontrarmos a hipótese, dispomos de um conjunto de exemplos. Vamos definir *amostra* de tamanho m , como sendo um conjunto ordenado de m exemplos. Tal amostra é dita de treinamento se ao invés de ser apenas uma seqüência de m exemplos, for uma seqüência de m pares (*exemplo*, $\{0, 1\}$), onde 0 ou 1 indicam se o exemplo é positivo ou negativo. O método que procura a partir de uma amostra obter a hipótese é conhecido como *algoritmo de aprendizado*.

Este modelo pode ser estudado mais a fundo no livro de Anthony e Biggs [6], de onde foram extraídas as definições acima descritas.

2.2 Aprendizado Supervisionado e Não-Supervisionado

Quando estamos trabalhando com uma amostra de treinamento, dizemos que o algoritmo é de aprendizado supervisionado, pois opera como se tivesse um professor dizendo o que está certo e o que está errado. Este método é muito útil quando podemos obter tal amostra já classificada, ou seja, quando temos muitos exemplos e conhecemos o conceito bem o suficiente para dizer à máquina qual exemplo corresponde ao conceito e qual não corresponde.

Quando a amostra não é de treinamento, temos apenas dados sem saber dizer se fazem parte de um conceito ou não, chamamos de aprendizado não supervisionado. Este tipo de aprendizado aparentemente sem sentido, é muito útil

quando queremos encontrar padrões no meio dos dados, correlações que de outra forma não seriam percebidos. As técnicas de Data Mining são baseadas nestes algoritmos. Não vamos nos preocupar com estes algoritmos, pois o problema que buscaremos resolver é de aprendizado supervisionado.

2.3 Métodos de Aprendizado Computacional

Diversos algoritmos existem para resolver o problema de obtenção de hipóteses. Podemos dividi-los em 4 grandes áreas de concentração: redes neurais, métodos simbólicos, métodos estatísticos e métodos híbridos. Na realidade, no contexto de aprendizado computacional, cada um destes métodos foca em uma forma de representação distinta de uma hipótese enquanto função de classificação.

O método de redes neurais procura usar como base o funcionamento dos neurônios, onde simplificadaamente cada neurônio realiza um processamento específico e passa adiante para outros neurônios o resultado do seu processamento. Desta forma a estrutura do cérebro seria a de uma rede de neurônios, e tal comportamento tenta-se imitar. As redes neurais são bastante populares, mas dependendo do problema a ser resolvido, são lentas e exigem uma grande quantidade de dados, além de ser um método de caixa preta, no qual a representação do conceito não é inteligível.

Os métodos simbólicos englobam as técnicas que usam lógicas, árvores de decisão, regras de transformação e são chamados de simbólicos por se basearem na manipulação de estruturas simbólicas de conceitos. Desta forma, não são métodos de caixa preta. Por trabalharem com a manipulação de estruturas simbólicas costumam ser mais lentos que os métodos de caixa preta, eminentemente numéricos. Apesar disso são muito populares e obtêm resultados muito bons. O método de regras de transformação em particular foi usado para resolver o problema da análise morfosintática por Brill [8] obtendo resultados muito bons.

Os métodos estatísticos são baseados em resultados de probabilidade, estatística e teoria da informação, possuindo sólidas bases matemáticas. Costumam ser de caixa-preta, exigindo malabarismos computacionais para poder representar de forma inteligível o conceito obtido. Dentre os métodos estatísticos destacam-se as Cadeias de Markov Ocultas (*Hidden Markov Models* ou HMMs), o método da Máxima Verossimilhança e o método da Máxima Entropia. Para maiores informações sobre HMMs veja Charniak [10]. Os métodos da Máxima Verossimilhança e da Máxima Entropia são muito parecidos e serão esclarecidos mais adiante, quando explicarmos o método da máxima entropia.

Os métodos Híbridos são os métodos que misturam dois ou mais dos métodos citados acima, normalmente para suprir as deficiências de um com as qualidades de outro.

Estaremos comparando o resultado desta tese com um método de regras de transformação variante do método de Brill [8], proposto e implementado por Chacur e Finger [5]. Esta variante foi feita para o Português e será descrito superficialmente a seguir.

2.4 Método de Brill variante Chacur-Finger

O método de Brill [8] foi originalmente criado para a língua inglesa e não pode ser aplicado diretamente ao Português. Esta impossibilidade se deve ao fato de o Português ser uma língua morfologicamente mais rica que o Inglês, necessitando de um número muitas vezes maior de classificações para realizar a sua marcação. Vamos descrever o método de Brill no contexto do problema de etiquetagem conforme descrito no artigo de Chacur e Finger [5].

O método de Brill é baseado em regras de transformação. Tais regras nada mais são que uma forma de codificação das informações aprendidas pelo etiquetador. Essas regras são geradas a partir de moldes e são compostas de duas partes: as ações e as condições.

As ações são regras que efetivamente atribuem ou alteram a etiqueta de uma palavra, conforme o seguinte exemplo:

Troque a etiqueta de VERBO para SUBSTANTIVO.

As condições descrevem situações que devem ser satisfeitas para que uma determinada ação seja realizada. Um exemplo de regra deste tipo é:

A palavra anterior é um ARTIGO.

Na etiquetagem baseada em transformações, as regras são aprendidas em duas fases. Na primeira, aprende-se as regras para etiquetagem de palavra desconhecidas ao etiquetador. Este possui um léxico, um conjunto de palavras com as suas possíveis etiquetas, quando a palavra não consta deste léxico diz-se que ela é desconhecida ao etiquetador. A segunda fase envolve o aprendizado de regras contextuais. Este aprendizado consiste em avaliar cada uma das regras candidatas e selecionar as melhores para serem usadas pelo etiquetador.

Estas duas fases são praticamente as mesmas do método de Brill, e conforme analisado por Chacur e Finger, tal algoritmo não pode ser usado diretamente no Português devido ao elevado número de etiquetas. Para contornar este problema, as etiquetas foram quebradas em núcleo e demais traços morfossintáticos, o algoritmo de Brill passa a ser usado apenas nos núcleos, cuja quantidade é equivalente ao número de etiquetas do inglês, e para inferir os demais traços é acrescida uma terceira fase ao algoritmo, o refinamento.

Este refinamento funciona da seguinte forma: para cada palavra do texto de entrada, verificar se a etiqueta corrente pode ou não ser refinada. Em caso positivo, investigar se há alguma característica na palavra que permita a determinação do complemento da etiqueta. Caso não seja encontrada nenhuma tal característica, utilizar um complemento padrão (isto se a etiqueta dever ser necessariamente refinada, pois em alguns casos o refinamento não é obrigatório). Se a etiqueta atual não admitir refinamento, nada é feito.

A precisão inicial deste método foi de 85.86% para os núcleos das etiquetas e de 78.28% para as etiquetas completas. Após a aplicação de técnicas de otimização da precisão, este resultado melhorou atingindo 95.45% para etiquetas completas. Sugerimos ao leitor os artigos de Chacur e Finger [5, 4] para maiores detalhes do método descrito acima.

Capítulo 3

Estimação por Maximização de Entropia

O objetivo deste capítulo é apresentar as bases matemáticas do algoritmo de maximização de entropia, bem como a sua dedução. Este conhecimento é importante para o leitor poder compreender o funcionamento do algoritmo e assim poder aplicá-lo a outros problemas diversos do da análise morfossintática. Aproveitamos ainda o ensejo para tentar minorar um outro problema que é a falta de uma referência auto-contida do método. O método de maximização de entropia até o momento foi tratado parcialmente por cada um dos pesquisadores da área, sendo necessária a leitura de pelo menos 3 artigos [12, 11, 18] para se ter quase toda a construção matemática do algoritmo de maximização. Digo ‘quase’ porque na passagem de um artigo para o outro existem algumas lacunas que procuramos igualmente preencher neste capítulo.

Antes de começarmos a descrever a técnica de maximização de entropia, é interessante fazermos a conexão entre ela e o aprendizado computacional. Esta ligação não é direta pois esta técnica é uma forma de obter uma distribuição de probabilidade com a característica de maximizar a entropia no seu domínio.

Retornando ao contexto de aprendizado computacional, imaginemos um conjunto de objetos (exemplos) e um conjunto também finito de conceitos. Imagine-mos ainda que cada objeto pode ser um exemplo de um conceito. Se pudéssemos calcular a probabilidade de cada objeto ser exemplo de um conceito, teríamos algo como 1 quando o par (*exemplo, conceito*) está correto e 0 caso contrário. Poderíamos assim usar a distribuição de probabilidade para definir não apenas um, mas todos os conceitos do conjunto, desde que tenhamos um conjunto de objetos grande o suficiente e os respectivos conceitos que eles representam. No entanto, na prática o conjunto de objetos pode ser infinito, de forma que tal probabilidade não pode ser obtida com precisão. O que se faz é deixar a distribuição perfeita de lado e encontrar uma distribuição aproximada, ou seja, um conjunto de hipóteses no paradigma de aprendizado computacional. É aqui onde se encaixa a técnica de maximização de entropia. Ela serve para tentarmos

obter uma distribuição que aproxime a real com base em dados incompletos. Este também é o modo de funcionamento da maximização da verossimilhança, que em alguns casos fornece resultado idêntico, conforme mostra Ratnaparkhi [18]. A diferença destes dois métodos ocorre apenas no critério de “proximidade” da distribuição de probabilidade.

Vejam nas próximas seções o algoritmo computacional usado para maximizar a entropia (Seção 3.1), e suas bases matemáticas assim separadas: uma definição de Entropia (Seção 3.2), como maximizá-la analiticamente (Seção 3.3), um algoritmo matemático para obter esta maximização (Seção 3.4) e a obtenção do algoritmo computacional a partir do algoritmo matemático (Seção 3.5).

3.1 Algoritmo de Maximização de Entropia

Vamos começar definindo os seguintes termos e variáveis que serão usados no algoritmo:

função característica é uma função que associa um contexto a uma classificação retornando 1 quando o par (classificação, contexto) está correto.

contexto é um exemplo de conceito.

conjunto de treinamento é um conjunto de pares (exemplo, classificação).

- T é o conjunto de treinamento.
- B é o conjunto de contextos observados no conjunto de treinamento.
- b é uma instância de um contexto.
- $\tilde{p}(b)$ é a probabilidade observada do contexto b no conjunto de treinamento.
- k é o número de funções características definidas.
- A é o conjunto de classificações.
- a é uma instância de uma classificação.
- $\mu_s^{(n)}$ é o s -ésimo termo de um vetor μ de multiplicadores em sua n -ésima iteração.
- $E_p b_s$ é a esperança de uma função característica b_s com relação a uma distribuição de probabilidade p .
- $\tilde{p}(a, b)$ é a probabilidade observada no conjunto de treinamento do par (a, b) [classificação, contexto].
- $b_s(a, b)$ é o valor da s -ésima função característica para o par (a, b) .
- N é o tamanho do conjunto de treinamento.
- $p^{(n)}(a|b)$ é probabilidade de a dado b na n -ésima iteração do algoritmo.

- $Z^{(n)}(b)$ é um fator de normalização para cálculo de $p^{(n)}(a|b)$.

Em pseudo-código o algoritmo para maximização da entropia está mostrado na Figura 3.1, que tratamos de desenvolver nas seções seguintes.

Entrada: conjunto de treinamento T .

Saída: distribuição de probabilidade $p^{(n)}(a|b)$.

1. Define-se o conjunto de funções características: $b_1 \dots b_k$.
2. Calcula-se $C = \max_{a \in A, b \in B} \sum_{s=1}^k b_s(a, b)$.
3. Cria-se nova função característica $b_{k+1}(a, b) = C - \sum_{s=1}^k b_s(a, b), \forall(a, b)$ e adiciona-se ao conjunto de funções características.
4. Para todo contexto b no conjunto de treinamento
5. calcula $\tilde{p}(b)$;
6. Para $s=1$ até $k+1$
7. $\mu_s^{(0)} = 1$
8. $E_{\tilde{p}} b_s = \frac{1}{N} \sum_{i=1}^N b_s(a_i, b_i)$
9. Para $n=1$ até Parada
10. $Z^{(n)}(b) = \sum_{a \in A} \prod_{s=1}^{k+1} (\mu_s^{(n)})^{b_s(a, b)}$
11. Para cada par (a, b)
12. $p^{(n)}(a|b) = \frac{1}{Z^{(n)}(b)} \prod_{s=1}^{k+1} (\mu_s^{(n)})^{b_s(a, b)}$
13. Para cada $s=1$ até $k+1$
14. $E_{p^{(n)}} b_s = \sum_{a, b} \tilde{p}(b) p^{(n)}(a|b) b_s(a, b)$
15. $\mu_s^{(n+1)} = \mu_s^{(n)} \left[\frac{E_{\tilde{p}} b_s}{E_{p^{(n)}} b_s} \right]^{\frac{1}{C}}$
16. Atualiza-se critério de parada.

Figura 3.1: Algoritmo para maximização de entropia.

Neste trabalho, estaremos usando $\log$ para representar o logaritmo neperiano.

O critério de parada foi deixado em aberto pois podem ser usados dois métodos distintos: número fixo de iterações ou variação muito pequena da entropia. O primeiro método consiste em definir um número de iterações. Ratnaparkhi utiliza e indica o uso do número 100 para a quantia de iterações do algoritmo. O segundo método, muito mais lento que o primeiro, consiste em calcular $H(p^{(n+1)}) = - \sum_{a, b} \tilde{p}(b) p^{(n+1)}(a|b) \log(p^{(n+1)}(a|b))$ e verificar se $|H(p^{(n+1)}) - H(p^{(n)})| < \epsilon$, onde ϵ é uma medida de precisão previamente definida.

3.1.1 Exemplo do Algoritmo de Maximização

Vamos usar como exemplo do uso do algoritmo o problema da detecção de fim de sentença. Este problema consiste em determinar se um ponto ‘.’ em uma oração

é um ponto final ou está representando uma abreviação. No contexto de aprendizado computacional, vamos propor uma solução para este problema usando um corpus que corresponde a um texto no qual seus pontos estão previamente classificados. O corpus não é a rigor o nosso conjunto de treinamento (T), pois contém mais informação do que apenas os pares (exemplo, classificação), mas a partir dele podemos derivar facilmente o conjunto de treinamento conforme veremos logo a seguir.

Suponhamos para fins didáticos que a classificação de um ponto dependa apenas de seus vizinhos imediatos. Assim, no seguinte extrato de texto: "... para a Sra. Gertrudes. Ontem ela ..." para classificarmos os dois pontos, precisaremos apenas de: "Sra. Gertrudes" e de "Gertrudes. Ontem". Cada um destes trechos de texto corresponde a um contexto de classificação e será neste trabalho doravante denominado apenas "contexto". Extraindo do corpus todos estes contextos e suas classificações criamos facilmente o conjunto de treinamento T.

Para este problema, temos claramente duas classes ou classificações: a_1 onde o ponto é um ponto final e a_2 onde o ponto é de uma abreviação. Assim, nosso conjunto A de classificações é: $A = \{a_1, a_2\}$. A partir do conjunto de treinamento, podemos criar o conjunto de contextos observados B , e com base em B podemos calcular a probabilidade observada ($\tilde{p}$) de cada um dos contextos $b \in B$. Uma vez que os contextos estão previamente classificados no corpus, podemos igualmente calcular a probabilidade observada $\tilde{p}(a, b)$ dos pares (a, b) , $a \in A \wedge b \in B$.

Vejamos agora como funciona o algoritmo. O primeiro passo é a definição do conjunto de funções características. Uma função característica é uma função que recebe como entrada um par (a, b) e retorna como saída 0 se a classe a não corresponde ao contexto b com base em um relacionamento definido internamente na função e retorna 1 caso contrário. Este conjunto de funções costuma ser definido automaticamente a partir do conjunto de treinamento. No entanto, estaremos fornecendo algumas destas funções para fins de ilustração.

Nas funções a seguir, b_{-1} é a palavra anterior ao ponto em um contexto b e b_{+1} a palavra posterior ao ponto em um contexto b . Sejam:

$$\begin{aligned}
 b_1(a, b) &= \begin{cases} 1, & \text{se } ((b_{-1} = \text{"Sr"}) \vee (b_{-1} = \text{"Sra"}) \vee \dots \vee (b_{-1} = \text{"Ilmo"})) \text{ e } a = a_2. \\ 0, & \text{caso contrário.} \end{cases} \\
 b_2(a, b) &= \begin{cases} 1, & \text{se } b_{+1} \text{ é um substantivo próprio e } a = a_2. \\ 0, & \text{caso contrário.} \end{cases} \\
 &\vdots \\
 b_k(a, b) &= \begin{cases} 1, & \text{se } b_{+1} \text{ é um substantivo próprio começando com letra maiúscula e } a = a_1. \\ 0, & \text{caso contrário.} \end{cases}
 \end{aligned}$$

Os passos 2 a 8 do algoritmo preparam as variáveis que serão usadas no loop principal 9. Em 9 estaremos a cada iteração calculando o vetor μ e o termo $Z(b)$ que definem a probabilidade $p(a|b)$ conforme o passo 12.

Ao final do algoritmo, teremos calculado os parâmetros que definem $p(a|b)$, isto é importante, pois podemos assim generalizar o resultado, o que não seria possível se tivéssemos apenas $p(a|b)$, $a \in A \wedge b \in B$, pois estaríamos restritos ao conjunto de treinamento.

Note que o conjunto de funções características depende do problema a ser resolvido, e sua obtenção de forma automática será mostrada no próximo capítulo, juntamente com a utilização deste algoritmo para resolver o problema da classificação morfosintática. Este exemplo foi definido e bem desenvolvido por Ratnaparkhi na sua tese de doutorado [18]. Vejamos a partir da próxima seção como se chega ao algoritmo da Figura 3.1.

3.2 Definição de Entropia

Seja I um conjunto finito e $\mathbf{p} = \{p_i \mid i \in I, p_i \geq 0, \sum_{i \in I} p_i = 1\}$ uma distribuição de probabilidade em I . A entropia da distribuição de probabilidade $\mathbf{p}$ é dada por:

$$H(\mathbf{p}) = - \sum_{i \in I} p_i \log p_i \quad (3.1)$$

Essa quantidade fornece uma medida única e sem ambigüidade para a incerteza representada por uma distribuição de probabilidade $\mathbf{p}$. É sempre positiva, cresce com a incerteza e é aditiva para fontes independentes de incerteza conforme foi provado por Shannon[19]. Esta incerteza a que nos referimos é um conceito qualitativo que pode ser fracamente associado ao desvio padrão da distribuição de probabilidade. Quanto menor a incerteza menor o desvio padrão e quanto maior a incerteza maior o desvio padrão. Poder-se-ia cogitar então em usar o desvio padrão como medida da incerteza. Tal não ocorre devido à não linearidade do desvio padrão.

Note que a função entropia possui valores mais elevados quando a distribuição se aproxima da uniforme, e seu máximo ocorre justamente quando $\mathbf{p}$ é uma distribuição uniforme. Esta última afirmação pode ser verificada a partir da desigualdade de Jensen:

$$\sum_{k=1}^m a_k f(b_k) \leq f\left(\sum_{k=1}^m a_k b_k\right), \text{ se } f \text{ é uma função convexa.} \quad (3.2)$$

fazendo:

$$a_k = \frac{1}{m}, \quad b_k = p_k \quad \text{e} \quad f(x) = -x \log x$$

temos então:

$$\begin{aligned} - \sum_{k=1}^m \frac{1}{m} p_k \log p_k &\leq - \left(\sum_{k=1}^m \frac{1}{m} p_k \right) \log \left(\sum_{k=1}^m \frac{1}{m} p_k \right) \Rightarrow \\ - \frac{1}{m} \sum_{k=1}^m p_k \log p_k &\leq - \frac{1}{m} \left(\sum_{k=1}^m p_k \right) \log \left(\frac{1}{m} \sum_{k=1}^m p_k \right) \Rightarrow \end{aligned}$$

como $\sum_{k=1}^m p_k = 1$ temos:

$$-\sum_{k=1}^m p_k \log p_k \leq -\log\left(\frac{1}{m}\right) = -\sum_{k=1}^m \frac{1}{m} \log \frac{1}{m}.$$

Ora, a distribuição uniforme é a distribuição mais incerta que se pode ter. Se quisermos então, para um dado problema, a distribuição de probabilidade mais incerta, basta maximizarmos $H(\mathbf{p})$ levando em consideração as restrições fornecidas pelo problema (veja [13]). Este é o princípio por trás da maximização de entropia. Vamos tentar encontrar agora tal distribuição de probabilidade.

3.3 Problema de Maximização de Entropia

Suponhamos que um problema de maximização de entropia esteja condicionado às seguintes d restrições:

$$\sum_{i \in I} b_{si} p_i = k_s, \quad s = 1, 2, \dots, d$$

onde $b_{si} = b_s(i) : I \rightarrow \mathbb{R}$ é suposta conhecida. Note que $k_s = E[b_s]$.

O problema de maximização é então:

$$\begin{aligned} & \max_{\mathbf{p}} - \sum_{i \in I} p_i \log p_i \\ & \text{sujeito a:} \\ & p_i > 0, \quad \sum_{i \in I} p_i = 1 \\ & E[b_s] = k_s = \sum_{i \in I} b_{si} p_i, \quad s = 1, 2, \dots, d. \end{aligned}$$

Para resolver este problema, vamos recorrer aos multiplicadores de Lagrange, tal qual Guisan e Shenitzer[12]. Como H é convexa¹ e contínua definida em um conjunto convexo, existe um único máximo global no conjunto de pontos viáveis do problema (os pontos viáveis são os pontos que satisfazem as restrições do problema). Tomando a função de Lagrange:

$$L(\mathbf{p}, \alpha, \beta) = H(\mathbf{p}) - \alpha \left(\sum_{i \in I} p_i - 1 \right) - \sum_{s=1}^d \left(\beta_s \left(\sum_{i \in I} b_{si} p_i - k_s \right) \right)$$

onde $\beta = \{\beta_s \mid s = 1, 2, \dots, d\}$. O ponto de máximo ocorre quando o gradiente

¹ uma função $f(x) : S \rightarrow \mathbb{R}$ é dita convexa se $f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$ para cada $x_1, x_2 \in S$ e $\forall \lambda \in (0, 1)$

de L for nulo, portanto, devemos ter:

$$\frac{\partial L}{\partial p_i} = -\log p_i - 1 - \alpha - \sum_{s=1}^d \beta_s b_{si} = 0, \quad i \in I \quad (3.3a)$$

$$\frac{\partial L}{\partial \alpha} = 1 - \sum_{i \in I} p_i = 0 \quad (3.3b)$$

$$\frac{\partial L}{\partial \beta_s} = k_s - \sum_{i \in I} b_{si} p_i = 0, \quad s = 1, 2, \dots, d \quad (3.3c)$$

de (3.3a) temos:

$$\begin{aligned} p_i &= e^{-(1+\alpha)} \cdot e^{-\sum_{s=1}^d \beta_s b_{si}} \\ &= e^{-(1+\alpha)} \cdot \prod_{s=1}^d e^{-\beta_s b_{si}} \end{aligned} \quad (3.4)$$

substituindo (3.4) em (3.3b), ficamos com:

$$\begin{aligned} \sum_{i \in I} \left(e^{-(1+\alpha)} \cdot \prod_{s=1}^d e^{-\beta_s b_{si}} \right) &= 1 \Rightarrow \\ \Rightarrow e^{-(1+\alpha)} \sum_{i \in I} \prod_{s=1}^d e^{-\beta_s b_{si}} &= 1 \Rightarrow \\ \Rightarrow e^{-(1+\alpha)} &= \frac{1}{\sum_{i \in I} \prod_{s=1}^d e^{-\beta_s b_{si}}} \end{aligned} \quad (3.5)$$

substituindo (3.5) em (3.4), chegamos a:

$$p_i = \frac{\prod_{s=1}^d e^{-\beta_s b_{si}}}{\sum_{i \in I} \prod_{s=1}^d e^{-\beta_s b_{si}}} \quad (3.6)$$

onde β_s é obtido a partir de (3.3c) e (3.3a). Fica então faltando determinar β_s pois a resolução de (3.3c) e (3.3a) não é trivial. A distribuição em (3.6) possui a mesma forma da equação (3.7) mostrada mais adiante. Isso é muito interessante pois Darroch e Ratcliff [11] criaram um algoritmo conhecido por *Generalized Iterative Scaling*(GIS) que encontra justamente os parâmetros β_s dadas as funções b_s .

3.4 Generalized Iterative Scaling

Seja $\mathbf{p}$ uma distribuição de probabilidade $\mathbf{p} = \{p_i \mid i \in I, p_i \geq 0, \sum p_i = 1\}$ da forma:

$$p_i = \pi_i \mu \prod_{s=1}^d \mu_s^{b_{si}} \quad (3.7)$$

com $\pi = \{\pi_i \mid i \in I, \pi_i > 0, \sum_{i \in I} \pi_i \leq 1\}$, a qual satisfaz as restrições

$$\begin{aligned} \sum_{i \in I} b_{si} p_i &= k_s, \quad s = 1, 2, \dots, d \\ \sum_{i \in I} p_i &= 1, \end{aligned} \tag{3.8}$$

sendo que para todo s , existe um $i \in I$ tal que $b_{si} \neq 0$. O algoritmo GIS encontra μ, μ_s dadas as constantes π_i e b_{si} . Antes de mostrar a demonstração do algoritmo, precisamos de alguns lemas e da *função de discriminação de informação* K definida a seguir:

$$K(\mathbf{p}, \pi) = \sum_{i \in I} p_i \log \frac{p_i}{\pi_i}$$

definindo $0 \log 0 = 0$ para garantir a continuidade à direita da função $x \log x$ na origem.

Lema 1 *Seja $\mathbf{p} = \{p_i \mid i \in I, p_i \geq 0, \sum_{i \in I} p_i = 1\}$ uma distribuição de probabilidade em I e $\pi = \{\pi_i \mid i \in I, \pi_i > 0, \sum_{i \in I} \pi_i \leq 1\}$. Então temos:*

$$K(\mathbf{p}, \pi) \geq 0 \quad \text{com} \quad K(\mathbf{p}, \pi) = 0 \Leftrightarrow \mathbf{p} = \pi.$$

PROVA. Quando $\sum_{i \in I} \pi_i = 1$ basta lembrarmos que $\log x \leq x - 1$ com igualdade apenas se $x = 1$:

$$\begin{aligned} K(\mathbf{p}, \pi) &= \sum_{i \in I} p_i \log \frac{p_i}{\pi_i} = - \sum_{i \in I} p_i \log \frac{\pi_i}{p_i} \geq \\ &\geq - \sum_{i \in I} p_i \left(\frac{\pi_i}{p_i} - 1 \right) = - \sum_{i \in I} \pi_i + \sum_{i \in I} p_i = -1 + 1 = 0. \end{aligned}$$

Quando $\sum_{i \in I} \pi_i < 1$ definamos $\sigma = \sum_{i \in I} \pi_i$. Então temos:

$$\begin{aligned} K(\mathbf{p}, \pi) &= \sum_{i \in I} p_i \log \frac{p_i}{\pi_i} = \sum_{i \in I} p_i \log \frac{p_i}{\pi_i} + \log \sigma - \log \sigma = \\ &= \sum_{i \in I} p_i \log \frac{p_i}{\pi_i} + \log \sigma \sum_{i \in I} p_i - \log \sigma = \sum_{i \in I} p_i \log \frac{p_i}{\pi_i} + \sum_{i \in I} p_i \log \sigma - \log \sigma = \\ &= \sum_{i \in I} p_i \left(\log \frac{p_i}{\pi_i} + \log \sigma \right) - \log \sigma = \sum_{i \in I} p_i \log \frac{p_i}{\frac{\pi_i}{\sigma}} - \log \sigma = \\ &= K(\mathbf{p}, \frac{\pi}{\sigma}) - \log \sigma. \end{aligned}$$

Como $K(\mathbf{p}, \frac{\pi}{\sigma}) \geq 0$ e $-\log \sigma > 0$ temos que $K(\mathbf{p}, \pi) > 0$.

Lema 2 *Se uma distribuição de probabilidade $\mathbf{p}$ da forma (3.7) satisfazendo (3.8) existir, então ela minimiza $K(\mathbf{p}, \pi)$ sujeita a (3.8) e é única.*

PROVA. Seja $\mathbf{q}$ uma outra distribuição de probabilidade satisfazendo (3.8). Então:

$$\begin{aligned}
K(\mathbf{p}, \boldsymbol{\pi}) &= \sum_{i \in I} p_i \left[\log \mu + \sum_{s=1}^d b_{si} \log \mu_s \right] = \sum_{i \in I} p_i \log \mu + \sum_{i \in I} p_i \left[\sum_{s=1}^d b_{si} \log \mu_s \right] \\
&= \log \mu \sum_{i \in I} p_i + \sum_{s=1}^d \sum_{i \in I} p_i b_{si} \log \mu_s = \log \mu \sum_{i \in I} q_i + \sum_{s=1}^d \log \mu_s \sum_{i \in I} p_i b_{si} \\
&= \sum_{i \in I} q_i \log \mu + \sum_{s=1}^d \log \mu_s \sum_{i \in I} q_i b_{si} = \sum_{i \in I} q_i \log \mu + \sum_{i \in I} \sum_{s=1}^d q_i b_{si} \log \mu_s \\
&= \sum_{i \in I} q_i \log \mu + \sum_{i \in I} q_i \left[\sum_{s=1}^d b_{si} \log \mu_s \right] = \sum_{i \in I} q_i \left[\log \mu + \sum_{s=1}^d b_{si} \log \mu_s \right] \\
&= \sum_{i \in I} q_i \log \mu \prod_{s=1}^d \mu_s^{b_{si}} = \sum_{i \in I} q_i \log \frac{p_i}{\pi_i}.
\end{aligned}$$

Então, temos que:

$$\begin{aligned}
K(\mathbf{p}, \boldsymbol{\pi}) - K(\mathbf{q}, \boldsymbol{\pi}) &= \sum_{i \in I} q_i \log \frac{p_i}{\pi_i} - \sum_{i \in I} q_i \log \frac{q_i}{\pi_i} \\
&= \sum_{i \in I} q_i \left(\log \frac{p_i}{\pi_i} - \log \frac{q_i}{\pi_i} \right) = \sum_{i \in I} q_i \log \frac{p_i}{q_i} \\
&= - \sum_{i \in I} q_i \log \frac{q_i}{p_i} = -K(\mathbf{q}, \mathbf{p}) \leq 0.
\end{aligned}$$

Logo, $K(\mathbf{p}, \boldsymbol{\pi}) \leq K(\mathbf{q}, \boldsymbol{\pi})$, ou seja $\mathbf{p}$ minimiza $K(\mathbf{p}, \boldsymbol{\pi})$ e é único.

Lema 3 Se uma distribuição de probabilidade $\mathbf{p}$ da forma $p_i = \mu \prod_{s=1}^d \mu_s^{b_{si}}$ satisfazendo (3.8) existir, então ela maximiza a entropia $H(\mathbf{p})$ e é única.

PROVA. Igual à do Lema 2, usando $H(\mathbf{p}) - H(\mathbf{q})$ ao invés de $K(\mathbf{p}, \boldsymbol{\pi}) - K(\mathbf{q}, \boldsymbol{\pi})$.

Note que este último lema corrobora o resultado obtido anteriormente através dos multiplicadores de Lagrange. Basta tomarmos em (3.6):

$$\begin{aligned}
\mu_s &= e^{-\beta_s}, \\
\mu &= \frac{1}{\sum_{i \in I} \prod_{s=1}^d e^{-\beta_s b_{si}}}
\end{aligned}$$

É interessante notar também que embora não apareça π_i na fórmula de $\mathbf{p}$ dada acima, não fizemos $\pi_i = 1$, o que fizemos foi supor π_i constante e o incorporamos a μ .

Definição 1 As restrições (3.8) são ditas consistentes se $\exists \mathbf{q} = \{q_i, i \in I, q_i > 0 : \mathbf{q} \text{ satisfaz (3.8)}\}$

O próximo Lema apenas expressa (3.7) e (3.8) em uma forma mais conveniente para o algoritmo.

Lema 4 *Dado que as restrições (3.8) são consistentes, então (3.7) e (3.8) podem ser expressas como:*

$$p_i = \pi_i \prod_{r=1}^c \lambda_r^{a_{ri}} \quad (3.9)$$

$$\sum_{i \in I} a_{ri} p_i = h_r, \quad r = 1, 2, \dots, c \quad (3.10)$$

com

$$a_{ri} \geq 0, \quad \sum_{r=1}^c a_{ri} = 1, \quad h_r > 0, \quad \sum_{r=1}^c h_r = 1.$$

PROVA. Seja:

$$a_{si} = t_s(u_s + b_{si}), \quad \forall i, \quad h_s = t_s(u_s + k_s), \quad s = 1, 2, \dots, d$$

para $u_s \geq 0, t_s > 0$ escolhidos de forma que:

$$a_{si} \geq 0 \quad \forall i \in I, \quad \text{e} \quad \sum_{s=1}^d a_{si} \leq 1.$$

Note que isso sempre é possível pois escolhendo u_s podemos fazer $a_{si} \geq 0, \forall i \in I$ e escolhendo t_s convenientemente podemos fazer $\sum_{s=1}^d a_{si} \leq 1$. Se $\sum_{s=1}^d a_{si} \leq 1, \forall i \in I$ então definimos $c = d$, senão fazemos $c = d + 1$ e:

$$a_{ci} = 1 - \sum_{s=1}^d a_{si}, \quad \text{para cada } i \in I \text{ não satisfeito e } h_c = 1 - \sum_{s=1}^d h_s.$$

como para ambos os casos $\sum_{r=1}^c a_{ri} = 1$, temos

$$\sum_{r=1}^c h_r = \sum_{r=1}^c \sum_{i \in I} a_{ri} p_i = \sum_{i \in I} \sum_{r=1}^c a_{ri} p_i = \sum_{i \in I} p_i \sum_{r=1}^c a_{ri} = \sum_{i \in I} p_i = 1.$$

com estas definições de a_{ri} e de h_r temos que (3.10) é equivalente a (3.8) pois:

$$\begin{aligned} \sum_{i \in I} a_{ri} p_i = h_r &\Leftrightarrow \sum_{i \in I} t_r(u_r + b_{ri}) p_i = t_r(u_r + k_r) \Leftrightarrow \\ &\Leftrightarrow t_r u_r \sum_{i \in I} p_i + t_r \sum_{i \in I} b_{ri} p_i = t_r u_r + t_r k_r \Leftrightarrow \\ &\Leftrightarrow t_r u_r + t_r \sum_{i \in I} b_{ri} p_i = t_r u_r + t_r k_r \Leftrightarrow \\ &\Leftrightarrow t_r \sum_{i \in I} b_{ri} p_i = t_r k_r \Leftrightarrow \sum_{i \in I} b_{ri} p_i = k_r. \end{aligned}$$

como $\forall s, \exists i \in I : b_{si} \neq 0 \Rightarrow \forall s, \exists i \in I : a_{si} > 0 \Rightarrow \forall s, \sum_{i \in I} a_{si} p_i = h_s > 0$.

Definindo agora $\lambda_s = \nu \mu_s^{t_s^{-1}}$, $s = 1, 2, \dots, d$, $\lambda_c = \nu$ onde $\nu = \mu \prod_{s=1}^d \mu_s^{-u_s}$ escrevemos (3.7) como (3.9):

$$\begin{aligned}
p_i &= \pi_i \prod_{r=1}^c \lambda_r^{a_{ri}} \Leftrightarrow \frac{p_i}{\pi_i} = \prod_{r=1}^c \lambda_r^{a_{ri}} \Leftrightarrow \\
&\Leftrightarrow \log \frac{p_i}{\pi_i} = \sum_{r=1}^c a_{ri} \log \lambda_r = \sum_{r=1}^d a_{ri} \log \lambda_r + \left(1 - \sum_{s=1}^d a_{si}\right) \log \nu = \\
&= \sum_{r=1}^d a_{ri} \log \lambda_r + \left(1 - \sum_{s=1}^d a_{si}\right) \log \nu = \sum_{r=1}^d a_{ri} \log \nu \mu_r^{t_r^{-1}} + \left(1 - \sum_{s=1}^d a_{si}\right) \log \nu = \\
&= \sum_{r=1}^d a_{ri} \log \nu + \sum_{r=1}^d \frac{a_{ri}}{t_r} \log \mu_r + \log \nu - \log \nu \sum_{s=1}^d a_{si} = \\
&= \sum_{r=1}^d (u_r + b_{ri}) \log \mu_r + \log \mu - \sum_{s=1}^d u_s \log \mu_s = \\
&= \sum_{r=1}^d u_r \log \mu_r + \sum_{r=1}^d b_{ri} \log \mu_r + \log \mu - \sum_{s=1}^d u_s \log \mu_s = \sum_{r=1}^d b_{ri} \log \mu_r + \log \mu \Leftrightarrow \\
&\Leftrightarrow \log \frac{p_i}{\pi_i} = \sum_{r=1}^d \log \mu_r^{b_{ri}} + \log \mu \Leftrightarrow \frac{p_i}{\pi_i} = \mu \prod_{r=1}^d \mu_r^{b_{ri}} \Leftrightarrow p_i = \pi_i \mu \prod_{r=1}^d \mu_r^{b_{ri}}
\end{aligned}$$

Temos agora condições de apresentar o algoritmo GIS:

Teorema 1 *Considere a seqüência de distribuições de probabilidade $\{\mathbf{p}^{(n)} \mid n = 0, 1, 2, \dots\}$ onde $\mathbf{p}^{(n)} = \{p_i^{(n)} \mid i \in I\}$ definida por:*

$$\begin{aligned}
p_i^{(0)} &= \pi_i \\
p_i^{(n+1)} &= p_i^{(n)} \prod_{r=1}^c \left(\frac{h_r}{h_r^{(n)}} \right)^{a_{ri}}, \quad n = 1, 2, \dots
\end{aligned}$$

onde

$$h_r^{(n)} = \sum_{i \in I} a_{ri} p_i^{(n)}.$$

se as restrições (3.10) forem consistentes, então a seqüência $\{\mathbf{p}^{(n)}\}$ converge para uma solução de (3.9) a qual é única.

PROVA. Vamos mostrar primeiro que $\sum_{i \in I} p_i^{(n-1)} \leq 1$ para todo $n = 1, 2, \dots$. Para $n \geq 1$ temos:

$$\sum_{i \in I} p_i^{(n)} = \sum_{i \in I} p_i^{(n-1)} \prod_{r=1}^c \left(\frac{h_r}{h_r^{(n-1)}} \right)^{a_{ri}}$$

mas,

$$\log \prod_{r=1}^c \left(\frac{h_r}{h_r^{(n-1)}} \right)^{a_{ri}} = \sum_{r=1}^c a_{ri} \log \frac{h_r}{h_r^{(n-1)}}$$

de (3.2) temos:

$$\begin{aligned} \sum_{r=1}^c a_{ri} \log \frac{h_r}{h_r^{(n-1)}} &\leq \log \sum_{r=1}^c a_{ri} \frac{h_r}{h_r^{(n-1)}} \Rightarrow \\ \Rightarrow \prod_{r=1}^c \left(\frac{h_r}{h_r^{(n-1)}} \right)^{a_{ri}} &\leq \sum_{r=1}^c a_{ri} \frac{h_r}{h_r^{(n-1)}} \end{aligned}$$

assim,

$$\begin{aligned} \sum_{i \in I} p_i^{(n)} &\leq \sum_{i \in I} p_i^{(n-1)} \sum_{r=1}^c a_{ri} \frac{h_r}{h_r^{(n-1)}} = \sum_{r=1}^c \frac{h_r}{h_r^{(n-1)}} \sum_{i \in I} a_{ri} p_i^{(n-1)} = \\ &= \sum_{r=1}^c \frac{h_r}{h_r^{(n-1)}} h_r^{(n-1)} = \sum_{r=1}^c h_r = 1 \Rightarrow \sum_{i \in I} p_i^{(n)} \leq 1 \end{aligned}$$

temos ainda,

$$\sum_{r=1}^c h_r^{(n)} = \sum_{r=1}^c \sum_{i \in I} a_{ri} p_i^{(n)} = \sum_{i \in I} p_i^{(n)} \sum_{r=1}^c a_{ri} \leq 1.$$

Note que $\forall n, p_i^{(n)} > 0$ e $h_r^{(n)} > 0$. Assim, do Lema 1,

$$K(\mathbf{h}, \mathbf{h}^{(n)}) = \sum_{r=1}^c h_r \log \frac{h_r}{h_r^{(n)}} \geq 0.$$

Seja $\mathbf{q}$ uma distribuição de probabilidade que satisfaz (3.10), tal $\mathbf{q}$ existe da hipótese de consistência de (3.10). Assim,

$$\begin{aligned} K(\mathbf{q}, \mathbf{p}^{(n+1)}) &= K(\mathbf{q}, \mathbf{p}^{(n)}) - \sum_{i \in I} q_i \log \prod_{r=1}^c \left(\frac{h_r}{h_r^{(n)}} \right)^{a_{ri}} \\ &= K(\mathbf{q}, \mathbf{p}^{(n)}) - \sum_{r=1}^c \log \frac{h_r}{h_r^{(n)}} \sum_{i \in I} a_{ri} q_i \\ &= K(\mathbf{q}, \mathbf{p}^{(n)}) - K(\mathbf{h}, \mathbf{h}^{(n)}) \end{aligned}$$

Assim, a sequência $\{K(\mathbf{q}, \mathbf{p}^{(n)}); n = 1, 2, \dots\}$ é decrescente, e pelo Lema 1 é limitada inferiormente por 0. Portanto possui limite, isto implica que

$$\lim_{n \rightarrow \infty} K(\mathbf{h}, \mathbf{h}^{(n)}) = 0 \Rightarrow \mathbf{h}^{(n)} \rightarrow \mathbf{h}.$$

Assim, temos que a sequência $\{\mathbf{p}^{(n)}\}$ converge. Como o seu limite satisfaz (3.10), pois $\mathbf{h}^{(n)} \rightarrow \mathbf{h}$, e é da forma (3.9), o Lema 2 garante a unicidade do limite.

Este algoritmo é interessante, não apenas por nos permitir encontrar a distribuição de probabilidade que maximiza a entropia, mas também porque maximiza a verossimilhança, conforme mostra o lema a seguir.

Lema 5 *Seja I um conjunto finito e $\mathbf{p}$ uma distribuição de probabilidade da forma (3.7) onde $\{\pi_i\}$ e $\{b_{si}\}$ são conhecidos mas $\mu, \{\mu_s\}$ não são. Dada uma amostra de tamanho N com frequências positivas (contagem de cada evento), $\{f_i; i \in I, f_i > 0, \sum_{i \in I} f_i = N\}$, a estimativa $\hat{\mathbf{p}}$ que maximiza a verossimilhança de $\mathbf{p}$ é a única distribuição de probabilidade da forma (3.7) que satisfaz*

$$\sum_{i \in I} b_{si} p_i = \sum_{i \in I} b_{si} \frac{f_i}{N}, \quad s = 1, 2, \dots, d, \quad \sum_{i \in I} p_i = 1. \quad (3.11)$$

de forma que $\hat{\mathbf{p}}$ pode ser obtido pelo GIS.

PROVA. As restrições (3.11) são consistentes pois são satisfeitas quando $p_i = \frac{f_i}{N}$. Assim, pelo teorema 1, existe um único $\hat{\mathbf{p}}$ da forma

$$\hat{p}_i = \pi_i \hat{\mu} \prod_{s=1}^d \hat{\mu}_s^{b_{si}}$$

satisfaz (3.11) e pode ser obtido pelo GIS. Para mostrar que $\hat{\mathbf{p}}$ é único ao maximizar a verossimilhança logarítmica $\sum_{i \in I} f_i \log p_i$, temos:

$$\begin{aligned} \sum_{i \in I} f_i \log \hat{p}_i - \sum_{i \in I} f_i \log p_i &= \sum_{i \in I} f_i \log \frac{\hat{p}_i}{p_i} = \\ &= N \sum_{i \in I} \frac{f_i}{N} \left(\log \frac{\hat{\mu}}{\mu} + \sum_{s=1}^d b_{si} \log \frac{\hat{\mu}_s}{\mu_s} \right) \\ &= N \log \frac{\hat{\mu}}{\mu} \sum_{i \in I} \frac{f_i}{N} + N \sum_{i \in I} \frac{f_i}{N} \sum_{s=1}^d b_{si} \log \frac{\hat{\mu}_s}{\mu_s} = \\ &= N \log \frac{\hat{\mu}}{\mu} + N \sum_{s=1}^d \log \frac{\hat{\mu}_s}{\mu_s} \sum_{i \in I} b_{si} \frac{f_i}{N} = \\ &= N \log \frac{\hat{\mu}}{\mu} + N \sum_{s=1}^d \log \frac{\hat{\mu}_s}{\mu_s} \sum_{i \in I} b_{si} \hat{p}_i = \\ &= N \log \frac{\hat{\mu}}{\mu} \sum_{i \in I} \hat{p}_i + N \sum_{i \in I} \hat{p}_i \sum_{s=1}^d b_{si} \log \frac{\hat{\mu}_s}{\mu_s} = \\ &= N \sum_{i \in I} \hat{p}_i \left(\log \frac{\hat{\mu}}{\mu} + \sum_{s=1}^d b_{si} \log \frac{\hat{\mu}_s}{\mu_s} \right) = \\ &= N \sum_{i \in I} \hat{p}_i \log \frac{\hat{p}_i}{p_i} \end{aligned}$$

que, pelo Lema 1, é não negativo e igual a zero apenas se $p_i = \hat{p}_i$, para todo $i \in I$.

Assim, temos que a distribuição que maximiza a entropia é a mesma que maximiza a verossimilhança.

3.5 Dedução do Algoritmo de Maximização

Segundo o Lema 3, para que uma distribuição de probabilidade maximize a entropia ela precisa satisfazer (3.8) e ter a forma $p_i = \mu \prod_{s=1}^d \mu_s^{b_{si}}$. Definindo então $b_s(a, b)$ como uma função característica conforme feito anteriormente por outros pesquisadores [18, 7], temos que a distribuição de probabilidade e as restrições (3.8) se tornam:

$$p(a, b) = \mu \prod_{s=1}^d \mu_s^{b_s(a, b)} \quad (3.12)$$

$$\sum_{a, b} b_s(a, b) p(a, b) = \sum_{a, b} b_s(a, b) \tilde{p}(a, b) \quad s = 1, \dots, d \quad (3.13)$$

onde $d = ||CT \times A||$, com CT igual ao conjunto de funções contextuais definidas e $\tilde{p}(a, b)$ é a distribuição de probabilidade observada no corpus lingüístico. O GIS nos fornece então os parâmetros μ e μ_s que definem a distribuição de probabilidade $p_{a, b}$.

Observe ainda que em (3.12) os termos μ_s podem ser encarados como pesos para as funções características. E em (3.13) apenas dizemos que a esperança de uma função característica na distribuição de probabilidade estimada deve ser igual à encontrada na prática.

Devido a (3.12), temos ainda que a nossa distribuição de probabilidade condicional é da seguinte forma:

$$\begin{aligned} p(a|b) &= \frac{p(a, b)}{p(b)} = \frac{p(a, b)}{\sum_a p(a, b)} = \\ &= \frac{1}{Z(b)} \prod_{s=1}^d \mu_s^{b_s(a, b)}, \text{ com } Z(b) = \sum_a \prod_{s=1}^d \mu_s^{b_s(a, b)} \end{aligned} \quad (3.14)$$

Podemos então aplicar o GIS e obter $p(a, b)$. Nos trabalhos anteriores [7, 16, 15, 18], foi usada ainda mais uma restrição, a de que $p(b) = \tilde{p}(b)$. Na verdade, esta restrição é mais forte do que se precisa, pois será usada apenas em (3.13), de forma que precisamos dessa afirmação apenas quando $b_s(a, b) \neq 0$:

$$\sum_{a, b} b_s(a, b) \tilde{p}(b) p(a|b) = \sum_{a, b} b_s(a, b) \tilde{p}(a, b) \quad s = 1, \dots, d \quad (3.15)$$

Como o subconjunto de B induzido por $b_s(a, b)$ costuma ser pequeno quando comparado com B , esta restrição adotada não é tão forte quanto parece. Entretanto, agora a distribuição de probabilidade obtida pelo GIS não é mais $p(a, b)$

que maximiza a entropia, porém uma outra distribuição $q(a, b)$, mas tal que $q(a|b) = p(a|b)$, não influenciando a função de classificação que procuramos. Isso pode ser visto notando-se que $q(a, b) = \tilde{p}(b)p(a|b)$:

$$\begin{aligned} q(a, b) &= \tilde{p}(b)p(a|b) \Leftrightarrow \\ q(a|b) &= \frac{\tilde{p}(b)p(a|b)}{\sum_a \tilde{p}(b)p(a|b)} = \frac{p(a|b)}{\sum_a p(a|b)} = p(a|b). \end{aligned}$$

Para encontrar a distribuição de probabilidade $\mathbf{p}$ ou $\mathbf{q}$, além do GIS visto na Seção 3.4, podemos usar os algoritmos *Improved Iterative Scaling* (IIS) [7, 17], o qual é um caso geral do GIS, e a variação do GIS usado em [18] e mostrado a seguir.

Considere a sequência $\{\mathbf{p}^{(n)}; n = 0, 1, 2, \dots\}$ onde $\mathbf{p}^{(n)} = \{p^{(n)}(a|b); a \in A, b \in B\}$ definida por:

$$\begin{aligned} \mu_s^{(0)} &= 1 \\ \mu_s^{(n+1)} &= \mu_s^{(n)} \left[\frac{E_{\tilde{p}} b_s}{E_{p^{(n)}} b_s} \right]^{\frac{1}{c}} \end{aligned}$$

onde

$$\begin{aligned} E_{\tilde{p}} b_s &= \sum_{a,b} b_s(a, b) \tilde{p}(a, b) \\ E_{p^{(n)}} b_s &= \sum_{a,b} b_s(a, b) \tilde{p}(b) p^{(n)}(a|b) \\ p^{(n)}(a|b) &= \frac{1}{Z(b)} \prod_{s=1}^c \mu_s^{(n) b_s(a,b)} \\ C &= \max_{a,b} \sum_{s=1}^d b_s(a, b) \end{aligned}$$

se as restrições (3.15) forem consistentes, então a sequência $\{\mathbf{p}^{(n)}\}$ converge para uma solução de (3.14) a qual é única.

Se tomarmos $q(a, b)$ como definido anteriormente no cálculo de $E_{p^{(n)}} b_s$ no algoritmo acima, o detalhe da distribuição condicional some do algoritmo. Tendo isso em mente, vamos tentar mostrar que o algoritmo acima é um caso particular do GIS, tendo portanto a sua convergência garantida.

Vamos usar as mesmas convenções usadas na Seção 3.4, lembrando apenas que $I = A \times B$. Do Lema 4, temos:

$$a_{si} = t_s(u_s + b_{si}), \quad \forall i \in I, \quad h_s = t_s(u_s + k_s), \quad s = 1, \dots, d$$

$$\text{tomando } u_s = 0 \text{ e } t_s = \frac{1}{\max_{i \in I} \sum_{r=1}^d b_{ri}} \text{ temos que } \forall i \sum_{s=1}^d a_{si} \leq 1.$$

Se $\forall i, b_{si} \geq 0$, temos $a_{si} \geq 0$. Então

$$\forall i, a_{si} = t_s b_{si} \text{ e } h_s = t_s k_s, s = 1, \dots, d. \quad (3.16)$$

Como $\lambda_s = \nu \mu_s^{1/t_s}, s = 1, \dots, d$, $\lambda_c = \nu$ onde $\nu = \mu \prod_{s=1}^d \mu_s^{-u_s}$, definindo $\max_{i \in I} \sum_{r=1}^d b_{ri} = C$, temos que $t_s = \frac{1}{C}$ e portanto $\lambda_s = \nu \mu_s^C, s = 1, \dots, d$.

Como $q(a, b)$ possui $\mu = 1$, tomando $\mu = 1$ e lembrando que $u_s = 0$, resulta em

$$\nu = \mu, \text{ e } \lambda_s = \mu_s^C, s = 1, \dots, c \text{ com } \mu_c = 1. \quad (3.17)$$

Do GIS, temos que:

$$\begin{aligned} p_i^{(n+1)} &= p_i^{(n)} \prod_{r=1}^c \left(\frac{h_r}{h_r^{(n)}} \right)^{a_{ri}}, \text{ como } p_i = \pi_i \prod_{r=1}^c \lambda_r^{a_{ri}}, \text{ temos} \\ \pi_i \prod_{r=1}^c \lambda_r^{(n+1)a_{ri}} &= \pi_i \prod_{r=1}^c \lambda_r^{(n)a_{ri}} \prod_{r=1}^c \left(\frac{h_r}{h_r^{(n)}} \right)^{a_{ri}} \Leftrightarrow \\ \pi_i \prod_{r=1}^c \lambda_r^{(n+1)a_{ri}} &= \pi_i \prod_{r=1}^c \lambda_r^{(n)a_{ri}} \left(\frac{h_r}{h_r^{(n)}} \right)^{a_{ri}} \Leftrightarrow \\ \prod_{r=1}^c \lambda_r^{(n+1)a_{ri}} &= \prod_{r=1}^c \lambda_r^{(n)a_{ri}} \left(\frac{h_r}{h_r^{(n)}} \right)^{a_{ri}} \end{aligned}$$

se $\lambda_r^{(n+1)a_{ri}} = \lambda_r^{(n)a_{ri}} \left(\frac{h_r}{h_r^{(n)}} \right)^{a_{ri}} \Rightarrow \lambda_r^{(n+1)} = \lambda_r^{(n)} \frac{h_r}{h_r^{(n)}}$ a última equação é satis-

feita. Usando (3.16) e (3.17) a última equação se torna $\mu_s^{(n+1)} = \mu_s^{(n)} \left(\frac{k_s}{k_s^{(n)}} \right)^{\frac{1}{C}}$.

A inicialização do GIS, dada por $p_i^{(0)} = \pi_i$, pode ser escrita como $\mu_s^{(0)} = 1$, de forma que o seguinte algoritmo é um caso particular do GIS:

$$\begin{aligned} \mu_s^{(0)} &= 1 \\ \mu_s^{(n+1)} &= \mu_s^{(n)} \left(\frac{k_s}{k_s^{(n)}} \right)^{\frac{1}{C}} \end{aligned}$$

onde

$$\begin{aligned} k_s &= \sum_{i \in I} b_{si} p_i = \sum_{i \in I} b_{si} \tilde{p}_i, s = 1, \dots, d \quad \text{e} \\ k_s^{(n)} &= \sum_{i \in I} b_{si} p_i^{(n)}, s = 1, \dots, d \quad \text{e} \\ p_i^{(n)} &= \pi_i \prod_{r=1}^c \lambda_r^{(n)a_{ri}} = \pi_i \prod_{r=1}^c \mu_r^{(n)b_{ri}} \end{aligned}$$

Mas o algoritmo descrito é igual ao anterior, pois $\pi_i = \tilde{p}(b)/Z(b)$ para $q(a, b)$. Com relação a este algoritmo, resta apenas um problema: $Z(b)$. Segundo as referências utilizadas [18], $Z(b)$ é constante durante o algoritmo, mas isso implica em conhecermos $p(a|b)$ o que não é o caso, pois queremos justamente encontrar $p(a|b)$.²

Como o leitor facilmente notará, com vistas a melhorar um pouco o desempenho do algoritmo, no passo 8, ao invés de calcularmos $E_{\tilde{p}}b_s$ como sendo $\sum_{a,b} \tilde{p}(a, b)b_s(a, b)$ calculamos $E_{\tilde{p}}b_s$ como $\frac{1}{N} \sum_{i=1}^N b_s(a_i, b_i)$. Podemos fazer isso pois N corresponde a todos os pares do conjunto de teste, assim estamos implicitamente calculando $\tilde{p}(a, b)$ a partir da soma dos pares no conjunto de treinamento. Isso elimina a necessidade de calcularmos explicitamente $\tilde{p}(a, b)$ que seria mais custoso.

A Figura 3.2 representa novamente o algoritmo em pseudo-código.

Entrada: conjunto de treinamento T .

Saída: distribuição de probabilidade $p^{(n)}(a|b)$.

1. Define-se o conjunto de funções características: $b_1 \dots b_k$.
2. Calcula-se $C = \max_{a \in A, b \in B} \sum_{s=1}^k b_s(a, b)$.
3. Cria-se nova função característica $b_{k+1}(a, b) = C - \sum_{s=1}^k b_s(a, b), \forall (a, b)$ e adiciona-se ao conjunto de funções características.
4. Para todo contexto b no conjunto de treinamento
5. calcula $\tilde{p}(b)$;
6. Para $s=1$ até $k+1$
7. $\mu_s^{(0)} = 1$
8. $E_{\tilde{p}}b_s = \frac{1}{N} \sum_{i=1}^N b_s(a_i, b_i)$
9. Para $n=1$ até Parada
10. $Z^{(n)}(b) = \sum_{a \in A} \prod_{s=1}^{k+1} (\mu_s^{(n)})^{b_s(a, b)}$
11. Para cada par (a, b)
12. $p^{(n)}(a|b) = \frac{1}{Z^{(n)}(b)} \prod_{s=1}^{k+1} (\mu_s^{(n)})^{b_s(a, b)}$
13. Para cada $s=1$ até $k+1$
14. $E_{p^{(n)}}b_s = \sum_{a,b} \tilde{p}(b)p^{(n)}(a|b)b_s(a, b)$
15. $\mu_s^{(n+1)} = \mu_s^{(n)} \left[\frac{E_{\tilde{p}}b_s}{E_{p^{(n)}}b_s} \right]^{\frac{1}{C}}$
16. Atualiza-se critério de parada.

Figura 3.2: Algoritmo para maximização de entropia.

Observe que devido aos loops em 11 e 13, ao somatório em 14 e ao produto

²Outra opção é que $Z(b)$ não é constante entre as iterações e é na verdade $Z^{(n)}(b)$, porém neste caso não sei se o algoritmo ainda converge. Muito provavelmente sim, mas não pude provar teoricamente isso. De qualquer forma, implementamos $Z^{(n)}(b)$ e aparentemente funciona.

em 12, o tempo de execução do algoritmo é $O(N.C.F)$, onde N é o número de contextos, C é o número de classes e F o número de funções características.

Capítulo 4

Aplicação ao Problema da Análise Morfossintática

O objetivo deste capítulo é apresentar o problema da análise morfossintática e mostrar como o algoritmo de maximização de entropia pode ser adaptado para resolver este problema.

O problema da Análise Morfossintática pode ser definido como: dada uma sentença válida de uma determinada linguagem, fornecer com correção a classe gramatical, também chamada de categoria morfossintática, de cada uma de suas palavras componentes. A princípio, pode-se imaginar que apenas um dicionário de palavras e classificações resolveria o problema, mas há muitas palavras que possuem diversas classificações, que variam de acordo com o contexto. Por exemplo:

Eu **gosto** de banana.

Isso tem **gosto** de banana.

A palavra **gosto** é um exemplo da ambiguidade existente na maioria das línguas e em particular na portuguesa. Na primeira sentença, **gosto** é a conjugação no tempo presente do indicativo do verbo gostar na primeira pessoa do singular, enquanto que na segunda, **gosto** é um substantivo.

Mas por que resolver este problema? A motivação é o fato deste ser um problema relativamente básico no campo da lingüística computacional. Tendo a informação das classes gramaticais das palavras, fica muito mais fácil resolver problemas mais complexos como análise sintática, tradução ou até mesmo extração de informação de textos.

Por causa da ambiguidade inerente à linguagem, não é trivial a resolução deste, ou de qualquer outro problema lingüístico de forma automática por um programa de computador, embora para um especialista tal tarefa não seja tão complicada salvo raras exceções. Procuraremos resolver este problema usando uma abordagem de aprendizado computacional estatístico, apresentado no capítulo anterior, a maximização de entropia. Esta técnica pode ser adaptada

a outros problemas, e nos pontos onde essa adaptação deve ser feita chamaremos a atenção do leitor. A adaptação que nós usamos é a mesma usada por Ratnaparkhi em sua tese de doutorado [18].

4.1 Resumo da Solução

Para realizar a classificação estaremos obtendo uma distribuição de probabilidade conjunta de pares classe-contexto (definido mais adiante) com a qual poderemos, dado um contexto, dizer qual a sua classe mais provável. Esta distribuição será obtida usando o algoritmo apresentado no capítulo anterior.

Segundo o algoritmo da Figura 3.2, precisamos de um conjunto de treinamento, a partir do qual estaremos definindo um conjunto de funções características. Com estas funções características e o conjunto de treinamento calculamos a distribuição de probabilidade das classes dados os contextos $p^{(n)}(a|b)$. Na próxima seção discutimos a arquitetura desta solução.

4.2 Arquitetura Teórica da Solução

A solução pode ser dividida em duas partes, o treinamento e a classificação. O treinamento consiste apenas em obter a distribuição de probabilidade conjunta de pares classe-contexto a partir do conjunto de treinamento. Com base no algoritmo de maximização de entropia visto no capítulo anterior e no resumo da solução, podemos estruturar o treinamento conforme a Figura 4.1.

Figura 4.1: Arquitetura teórica do treinamento.

Em linhas gerais, a partir de um conjunto de treinamento, são criadas as funções características e dentre estas geradas, são selecionadas as mais relevantes para fins de maximização de entropia. Com o conjunto de funções características relevantes definido, é realizada a maximização da entropia, a qual terá como resultado um conjunto de pesos μ_s usados para gerar a distribuição de probabilidade conjunta $p^{(n)}(a|b)$.

A classificação é bem mais simples que o treinamento com relação à arquitetura, pois dado o conjunto de testes, o classificador apenas retorna como resultado o conjunto classificado. A Figura 4.2 ilustra este processo:

Figura 4.2: Arquitetura teórica da Classificação.

Nas próximas seções discutimos o conjunto de treinamento utilizado (4.3), como obter as funções características (4.4, 4.5, 4.6), como classificar um termo com base no resultado do algoritmo (4.7) e os procedimentos de treinamento (4.7.1), ajuste (4.7.2) e teste (4.7.3) usados para obter a distribuição de probabilidade $p^{(n)}(a|b)$. No próximo capítulo discutiremos como ficou a implementação desta arquitetura.

4.3 Conjunto de Treinamento

O conjunto de treinamento e testes utilizado é o Corpus Tycho Brahe do Português Histórico [3]. Este Corpus faz parte de um projeto que objetiva modelar a relação entre prosódia e sintaxe na mudança linguística que deu origem ao Português Europeu Moderno a partir do Português Clássico. O Corpus contém textos de autores portugueses que nasceram entre 1500 e 1850 e que ficaram pouco tempo fora de Portugal.

O Corpus Tycho Brahe já possui mais de 1.300.000 palavras, 710.000 das quais já classificadas, o que nos fornece uma boa base para testar o algoritmo de maximização de entropia aplicado à classificação morfossintática. O corpus compreende atualmente 26 autores e 30 textos.

No Corpus Tycho Brahe são usados 383 tags diferentes para cada uma das classificações morfossintáticas possíveis do Português. Esta quantidade excede largamente a quantidade de tags do inglês que no Penn Treebank Wall Street Journal [1] é por volta de 30. Para uma lista completa dos tags do Português, veja o manual do Corpus Tycho Brahe [2].

A seguir temos como exemplo um pequeno extrato do texto classificado de Matias Aires do Corpus Tycho Brahe:

```

... A/D-F fraqueza/N dos/P+D-P nossos/PRO$-P sentidos/N-P nos/CL
impede/VB-P o/D gozar/VB das/P+D-F-P cousas/N-P na/P+D-F
sua/PRO$-F simplicidade/N natural/ADJ-G ./...

```

4.4 Definição de Funções Características

Vimos no capítulo anterior que a distribuição de probabilidade das classes a dados os contextos b que maximiza a entropia é da forma de (3.14):

$$p(a|b) = \frac{1}{Z(b)} \prod_{s=1}^d \mu_s^{b_s(a,b)}, \text{ com } Z(b) = \sum_a \prod_{s=1}^d \mu_s^{b_s(a,b)} \quad (4.1)$$

Nesta equação, μ_s são pesos para as funções características $b_s(a, b)$. Estas funções $b_s : A \times B \rightarrow \mathbb{R}$ devem relacionar as classes com os contextos. Pois analisando a equação percebemos que para um dado contexto b , o único termo que varia é $b_s(a, b)$. Para fazer com que $b_s(a, b)$ relacione uma classe a um contexto, começaremos definindo b_s da seguinte forma:

$$b_s(a, b) = \begin{cases} 1, & \text{se } b = \bar{b} \text{ e } a = \bar{a}. \\ 0, & \text{caso contrário.} \end{cases}$$

onde $\bar{b}$ é um contexto específico e $\bar{a}$ é uma classe específica.

Embora esta definição relacione bem classes com contextos, pois quando $b_s(a, b) = 0$ temos $\mu_s^{b_s(a,b)} = 1$ e o termo correspondente sai do produto, ela é muito pouco prática, pois não nos permite generalizações, além de precisarmos de um número muito grande de funções características para obtermos a distribuição de probabilidade condicional de classes dados os contextos. Para melhorarmos nossa definição de b_s , precisamos analisar melhor como relacionar classes com contextos.

O contexto de um *token*¹ deve ser um conjunto de informações extraídas do texto suficiente para fornecer a classificação do token. Estaremos usando como contexto o conjunto que engloba o token, o token posterior, o token anterior e a classe do token anterior. Escolhemos este contexto, embora para muitos possa parecer insuficiente, pois o tempo de processamento necessário para usar contextos maiores com os recursos disponíveis ultrapassa o razoável como veremos no capítulo 5.

Por exemplo, seja o texto: “Paulo comprou um sapato novo.” Se a palavra sendo analisada for “sapato”, o contexto usado será: “um/D-UM sapato novo”. Onde D-UM é o código correspondente a artigo indefinido do Corpus Tycho Brahe. Sabendo que o termo anterior é um artigo, podemos classificar com segurança a palavra “sapato” como substantivo. Note que esta regra que usamos é bastante geral e pode ser aplicada em qualquer caso. Se usarmos o conjunto de contextos definido por esta regra para definir as funções características, teremos a generalização que buscamos.

Assim a definição das funções características b_s fica:

$$b_s(a, b) = \begin{cases} 1, & \text{se } b \in \bar{B} \text{ e } a = \bar{a}. \\ 0, & \text{caso contrário.} \end{cases}$$

¹um token no problema em questão é uma palavra ou um sinal de pontuação, ou seja, algo passível de receber uma classificação morfofssintática.

onde $\bar{B} \subset B$ é um conjunto de contextos definidos por alguma regra.

Para representar computacionalmente estes conjuntos, vamos usar o que foi chamado por Ratnaparkhi [18] de predicados contextuais. Os predicados contextuais são funções $pc(b) : B \rightarrow \{0, 1\}$ definidas como:

$$pc(b) = \begin{cases} 1, & \text{se } b \text{ satisfaz a regra } \bar{r}. \\ 0, & \text{caso contrário.} \end{cases}$$

No caso particular do exemplo acima, o predicado contextual seria:

$$pc(b) = \begin{cases} 1, & \text{se a classe de } b_{-1} = 'D'. \\ 0, & \text{caso contrário.} \end{cases}$$

onde b_{-1} corresponde ao termo anterior ao que está sendo analisado no contexto b e 'D' é a classe correspondente a artigo no Corpus Tycho Brahe.

Alterando um pouco a definição das funções características b_s para usar os predicados contextuais, ficamos finalmente com:

$$b_s(a, b) = \begin{cases} 1, & \text{se } \overline{pc}(b) = 1 \text{ e } a = \bar{a}. \\ 0, & \text{caso contrário.} \end{cases}$$

Vejamos a seguir como obter a partir do conjunto de treinamento os predicados contextuais e as funções características.

4.5 Geração Automática de Funções Características

A forma mais simples (e mais trabalhosa) de se obter os predicados contextuais e as funções características relevantes é definindo-as manualmente. Como o conjunto de tais funções é potencialmente grande, desestimula-se o seu uso.

Vamos então fazer com que o próprio computador gere os predicados contextuais e as funções características. Vejamos primeiro os predicados contextuais. Vamos usar um método semelhante ao usado por Ratnaparkhi no cap. 5 de [18].

Considere um contexto genérico b no qual designaremos o termo sendo analisado por $b_0(b)$, o termo anterior e o posterior a este por $b_{-1}(b)$ e $b_1(b)$ respectivamente e a classe do termo anterior por $t_{-1}(b)$.

Para cada contexto $\bar{b}$, qualquer que seja $b_0(\bar{b})$, criaremos os seguintes predicados contextuais:

$$\begin{aligned} pc_1(b) &= \begin{cases} 1, & \text{se } t_{-1}(b) = t_{-1}(\bar{b}). \\ 0, & \text{caso contrário.} \end{cases} \\ pc_2(b) &= \begin{cases} 1, & \text{se } b_{-1}(b) = b_{-1}(\bar{b}). \\ 0, & \text{caso contrário.} \end{cases} \\ pc_3(b) &= \begin{cases} 1, & \text{se } b_1(b) = b_1(\bar{b}). \\ 0, & \text{caso contrário.} \end{cases} \end{aligned}$$

Se $b_0(\bar{b})$ não for “raro”, ou seja, aparecer mais de n vezes (no nosso caso usamos $n = 5$) no conjunto de treinamento, criaremos o seguinte predicado contextual:

$$pc_4(b) = \begin{cases} 1, & \text{se } b_0(b) = b_0(\bar{b}). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.2)$$

E por fim, se $b_0(\bar{b})$ for “raro”, criaremos os seguintes predicados contextuais:

$$pc_5(b) = \begin{cases} 1, & \text{se } b_0(b) \text{ for um número .} \\ 0, & \text{caso contrário.} \end{cases}$$

$$pc_6(b) = \begin{cases} 1, & \text{se } b_0(b) \text{ possuir uma letra maiúscula .} \\ 0, & \text{caso contrário.} \end{cases}$$

$$pc_7(b) = \begin{cases} 1, & \text{se } b_0(b) \text{ possuir um hífen .} \\ 0, & \text{caso contrário.} \end{cases}$$

$$pc_8(b) = \begin{cases} 1, & \text{se } prefixo(b_0(b), 1) = prefixo(b_0(\bar{b}), 1). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.3)$$

$$pc_9(b) = \begin{cases} 1, & \text{se } prefixo(b_0(b), 2) = prefixo(b_0(\bar{b}), 2). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.4)$$

$$pc_{10}(b) = \begin{cases} 1, & \text{se } prefixo(b_0(b), 3) = prefixo(b_0(\bar{b}), 3). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.5)$$

$$pc_{11}(b) = \begin{cases} 1, & \text{se } prefixo(b_0(b), 4) = prefixo(b_0(\bar{b}), 4). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.6)$$

$$pc_{12}(b) = \begin{cases} 1, & \text{se } prefixo(b_0(b), 5) = prefixo(b_0(\bar{b}), 5). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.7)$$

$$pc_{13}(b) = \begin{cases} 1, & \text{se } sufixo(b_0(b), 1) = sufixo(b_0(\bar{b}), 1). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.8)$$

$$pc_{14}(b) = \begin{cases} 1, & \text{se } sufixo(b_0(b), 2) = sufixo(b_0(\bar{b}), 2). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.9)$$

$$pc_{15}(b) = \begin{cases} 1, & \text{se } sufixo(b_0(b), 3) = sufixo(b_0(\bar{b}), 3). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.10)$$

$$pc_{16}(b) = \begin{cases} 1, & \text{se } sufixo(b_0(b), 4) = sufixo(b_0(\bar{b}), 4). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.11)$$

$$pc_{17}(b) = \begin{cases} 1, & \text{se } sufixo(b_0(b), 5) = sufixo(b_0(\bar{b}), 5). \\ 0, & \text{caso contrário.} \end{cases} \quad (4.12)$$

onde $prefixo(b_0(b), i)$ é uma função que retorna o prefixo com i letras do string $b_0(b)$ e $sufixo(b_0(b), i)$ é a função análoga para sufixos.

Note que estes são apenas os predicados contextuais gerados automaticamente, isto não significa que outros não possam ser adicionados manualmente. Podem e devem, para isso existe o procedimento de ajuste que veremos na Subseção 4.7.2 onde não só adiciona-se como também retira-se predicados contextuais.

É interessante observar ainda os predicados contextuais (4.2) e de (4.3) a (4.12). No primeiro estamos explicitamente dizendo para o computador “memorizar” os resultados mais comuns, enquanto que nos seguintes estamos supondo que existe alguma relação entre os prefixos e sufixos de uma palavra e a sua classificação.

Cada vez que geramos um predicado contextual, geraremos também a função característica correspondente com a classe $t_0(b)$ para o contexto analisado. Deve-se tomar o cuidado para não criar predicados contextuais e funções características repetidos.

4.6 Seleção de Funções Características

O comportamento de funções características que ocorrem raramente no conjunto de treinamento é de difícil previsão pois não é estatisticamente confiável, segundo Ratnaparkhi [18]. Para contornar este problema, realiza-se uma seleção das funções características. Além desse problema, o algoritmo de maximização de entropia não se comporta bem quando $p(a|b) = 0 \vee p(a|b) = 1$, este segundo caso podendo ocorrer para funções características pouco frequentes no conjunto de treinamento. Veja o cap. 9 da tese de Ratnaparkhi para uma discussão a respeito [18].

Seguindo sugestão de Ratnaparkhi, usaremos uma heurística para selecionar as funções a serem usadas. Qualquer função característica que não use um predicado contextual do tipo de (4.2) e que ocorra menos de 10 vezes no conjunto de treinamento será descartada.

Este não é o único método para selecionar as funções características, outro método interessante é usado por Berger e Della Pietra em [7] e discutido por Ratnaparkhi no capítulo 8 de [18]. Consiste em um algoritmo no qual a cada iteração adiciona a função que maximiza o ganho na verossimilhança ao conjunto de funções características a serem usadas. A vantagem deste método está em evitar a redundância no conjunto de funções características usadas. Por outro lado, ele é computacionalmente mais caro que o proposto.

4.7 Classificação Morfossintática de um Termo

Após a aplicação do algoritmo de maximização de entropia, obtemos a distribuição de probabilidade condicional $p(a|b)$. Com $p(a|b)$ podemos realizar a classificação das palavras de um texto.

A forma mais simples de proceder a classificação é percorrer o texto e para cada contexto b_i , encontrar a classificação a_i que maximiza $p(a|b_i)$. Este no entanto não é o procedimento que usaremos. Procuraremos maximizar a probabilidade de uma sentença inteira posto que esse é o menor conjunto autocontido de classificação, ou seja, a classificação de todos os seus elementos não dependem de elementos externos ao conjunto. Para tanto, usaremos o mesmo método de Ratnaparkhi, uma busca em largura limitada a K termos. Nesta busca, mantêm-se apenas as K seqüências de classificações com maior probabilidade.

Dada um sentença $\{w_1 \dots w_n\}$ uma candidata à seqüência de classificações $\{a_1 \dots a_n\}$ possui probabilidade condicional:

$$p(a_1 \dots a_n | w_1 \dots w_n) = \prod_{i=1}^n p(a_i | b_i)$$

onde b_i é o contexto correspondente à i-ésima palavra da sentença.

Para acelerar o processo de busca, usaremos ainda um dicionário de classificações observadas. Para cada palavra conhecida, ficarão armazenados todas as classes vistas para essa palavra. Desta forma, não será necessário percorrer todo o conjunto de classes sempre, isso será feito apenas para palavras desconhecidas.

Na Figura 4.3 temos a descrição do algoritmo de classificação em mais detalhes para facilitar o leitor.

Para testar o algoritmo apresentado, vamos usar o recurso de que dispomos, o Corpus Tycho Brahe. Não adianta encontrar a distribuição de probabilidade e testar a distribuição encontrada no mesmo conjunto de dados. Assim sendo, estaremos dividindo o corpus em uma parte de treinamento e uma parte de teste. Esta divisão será feita autor por autor, para evitar que usemos um autor para treinar e outro para testar, o que diminuiria sensivelmente a qualidade da classificação. A parte de treinamento por sua vez será dividida em duas partes, um treinamento propriamente dito e uma parte de ajuste. Nas próximas subseções descrevemos cada uma das etapas com mais detalhes.

4.7.1 Procedimento de Treinamento

Para o treinamento estaremos separando 80% das sentenças do Corpus. uma vez com as sentenças separadas, aplicaremos nestas sentenças o algoritmo de maximização de entropia para obter a distribuição de probabilidade condicional.

4.7.2 Procedimento de Ajuste

Com o treinamento encerrado, passamos para a fase de ajuste. Para esta fase, separaremos 10% do Corpus. Usando a distribuição de probabilidade obtida no treinamento, classificaremos as sentenças separadas para ajuste e verificaremos o resultado obtido na classificação. Em particular, estaremos interessados no erros mais freqüentes cometidos pelo classificador. Com base nos erros deveremos conforme o caso, adicionar ou remover funções características específicas.

```

    avançar:  $s \rightarrow s_1 \dots s_m$  /* Dada uma seqüência de classes  $s$ , produz novas
                                seqüências  $s_1 \dots s_m$ , cada uma de comprimento  $|s| + 1$ .
                                Quando necessário, este método consulta o
                                dicionário de classificações. */
    inserir:  $s \times h \rightarrow \text{void}$  /* insere a seqüência  $s$  no heap  $h$ . */
    extrair:  $h \rightarrow s$  /* retorna a seqüência de classes em  $h$  com
                        a maior probabilidade e também a remove de  $h$ . */

 $n$  = comprimento da sentença de entrada
 $N = 10$ 
 $s$  = seqüência de classes vazia
 $h_0$  = heap vazio /*  $h_i$  contém seqüências de classes de comprimento  $i$ . */
inserir( $s, h_0$ ) /* inicializa  $h_0$  com a seqüência vazia. */

for  $i = 0$  to  $n - 1$ 
     $sz = \min(N, |h_i|)$ 
    for  $j = 1$  to  $sz$ 
         $s_1 \dots s_m = \text{avançar}(\text{extrair}(h_i))$ 
        for  $p = 1$  to  $m$ 
            inserir( $s_p, h_{i+1}$ )

return extrair( $h_n$ )

```

Figura 4.3: Método de classificação de sentença.

Uma vez realizada a análise dos erros, realiza-se novamente o procedimento acima descrito até o operador ficar satisfeito com o resultado. Agora podemos prosseguir para a fase de teste.

4.7.3 Procedimento de Teste

Nesta fase realiza-se a classificação dos 10% de sentenças restantes do Corpus, verificando a taxa de acerto do algoritmo. Além da taxa de acerto de classificações de palavras individuais, também é interessante calcular-se a taxa de acerto de classificação de uma sentença completa, bem como de palavras desconhecidas, ou seja, palavras que não foram vistas no treinamento.

Capítulo 5

Aplicação Prática da Técnica

O objetivo deste capítulo é apresentar a arquitetura da implementação, explicar detalhes e problemas verificados com a mesma, bem como apresentar os resultados obtidos, comparando-os com o método de Brill [5] aplicado ao mesmo problema e ao mesmo Corpus Tycho Brahe e com a implementação realizada por Ratnaparkhi [18] para o Inglês usando o Corpus Penn Treebank.

5.1 Arquitetura da Implementação

Implementamos uma solução muito parecida com a teórica descrita na Seção 4.2, apenas com o acréscimo de uma fase de pré-processamento dos dados de treinamento, conforme mostra a Figura 5.1.

Figura 5.1: Arquitetura implementada.

Inicialmente usamos esta arquitetura para as etiquetas completas, no entanto o tempo de execução mostrou-se impraticável como aliás era de se esperar, pois

conforme vimos ao final da Seção 3.5 o tempo de execução do algoritmo de maximização é $O(N.C.F)$, onde N é o número de contextos, C é o número de classes e F é o número de funções características. A implementação do algoritmo na sua forma original mostrou-se muito ineficiente, forçando-nos a diminuir o número de classes, diminuir o número de funções características, otimizar o código e por fim diminuir o tamanho do corpus utilizado.

Embora não tenhamos testado para etiquetas completas, o programa foi implementado tendo-se isso em vista. Como o algoritmo fica muito lento para as mais de 300 etiquetas do corpus, o programa divide o processo de treinamento e classificação em duas fases. A primeira usa apenas os núcleos das etiquetas (≈ 30) e a segunda faz um refinamento para cada etiqueta usando o mesmo algoritmo de maximização de entropia, porém em um subconjunto de contextos relevantes para este refinamento. Assim, além do classificador geral para os núcleos de etiquetas, obtém-se um classificador extra para cada núcleo de etiqueta.

Nas próximas seções estaremos explicando em detalhes a função de cada fase, os problemas enfrentados e as soluções encontradas.

5.2 Detalhes da Implementação e Problemas Enfrentados

Conforme dissemos anteriormente, o tempo de execução do algoritmo é $O(N.C.F)$, onde N é o número de contextos, C é o número de classes e F o número de funções características. Assim o algoritmo consome muito processamento, pois temos o produto $N.C.F$ da ordem de $10^6 \cdot 10^2 \cdot 10^4 = 10^{12}$. Isto fez com que a implementação do algoritmo fosse realizada tendo-se em mente desde o início o uso de algoritmos e estruturas de dados eficientes.

A função mais crítica do sistema é a função que avalia uma função característica. Esta função é chamada diversas vezes e o tempo de execução do programa depende diretamente desta função. Este método precisa comparar diversas vezes contextos e classes. Esta tarefa de comparação, como é uma comparação de strings, é relativamente custosa. Com o intuito de tornar este método mais eficiente, foram criados mapas de strings para inteiros e de classes para inteiros, que nos permitem comparar inteiros ao invés strings.

5.2.1 Processamento Inicial

O processamento inicial consiste em processar o corpus, criando um conjunto de classes e de contextos que serão usados nos passos seguintes. As classes e contextos são definidos por meio de strings. Para cada classe e contexto únicos é fornecido ainda um identificador inteiro que permite que comparemos classes e contextos, não por sua definição, mas por seus identificadores. Esta comparação nos permite um ganho muito grande de performance na execução do algoritmo, embora nos exija uma fase de processamento inicial mais demorada, para poder definir estes identificadores. Para realizar esta atribuição de valores, são usadas

tabelas de hashing para verificar se um dado contexto sendo analisado já foi adicionado ao conjunto de contextos ou não, e se já foi, qual o seu identificador.

Além dos contextos, foram atribuídos identificadores para os strings. No entanto, o objetivo neste caso era minimizar o uso da memória uma vez que na definição de cada contexto são usados 3 strings, um para o token anterior, um para o token atual e um para o token seguinte. Por causa disso, cada string do corpus participa de 3 contextos. Igualmente foi necessário o uso de uma tabela de hashing para definir os identificadores de strings.

Além da definição de identificadores para contextos, classes e strings, os contextos únicos são contados em suas ocorrências no corpus. Isto é feito para aproveitar o trabalho de identificar os contextos, trabalho este que já percorre todos os contextos do corpus, para fornecer uma informação necessária para o cálculo de $\tilde{p}(b)$. São criados ainda vetores para acelerar futuras buscas de contextos, classes ou strings a partir dos seus identificadores.

Com todas estas estruturas e identificadores definidos, passa-se à primeira parte do algoritmo que é a criação das funções características.

5.2.2 Criação e Seleção de Funções Características

Esta fase foi originalmente implementada conforme descrito na Seção 4.5. Posteriormente as regras (4.3), (4.4), (4.5), (4.6), (4.7), (4.8) e (4.12) foram eliminadas pois não acrescentam informações relevantes à classificação, além de aumentarem o tempo de execução do algoritmo. A quantidade de funções geradas é bem grande, pois para cada contexto são geradas as funções definidas na Seção 4.5 de acordo com a frequência do mesmo no corpus. Destas funções, muitas não são relevantes para a classificação pois dizem respeito a um caso muito particular e não permitem generalizações.

Embora não tenhamos realizado explicitamente a fase de ajuste, realizando a eliminação de funções características citada acima já estávamos implicitamente ajustando, percebemos a necessidade da mesma, pois apenas esta correção para fins de otimização já melhorou substancialmente a precisão do método de cerca de 75% para cerca de 85%. Além disso, observando-se os erros gerados, pode-se perceber uma repetição das condições de erro, sugerindo que a criação manual de novas funções características pode melhorar mais a precisão do algoritmo.

Para evitar o uso de funções redundantes, e tornar mais rápido o algoritmo de maximização, procede-se a seleção das funções características. Nesta fase, percorremos o conjunto de funções características geradas, contando o número de vezes que a mesma é satisfeita no corpus. As funções que aparecerem mais que um determinado número de vezes, no nosso caso 10, são mantidas e as pouco frequentes, descartadas.

5.2.3 Maximização de Entropia

Após o conjunto de funções características estar bem definido, procedemos o cálculo da maximização de entropia. Este cálculo é realizado conforme descreve o algoritmo da Figura 3.2. Com base nesse algoritmo, algumas estruturas foram

introduzidas com vistas a acelerar o cálculo. A principal delas é um vetor que associa a cada par contexto-classe um vetor com os identificadores das funções características que retornam 1 para esse par. Este vetor, embora demorado de construir compensa no tempo de execução de cada iteração do algoritmo. Pois anteriormente percorria-se o conjunto de contextos do corpus aplicando as funções características diversas vezes no decorrer do algoritmo. Com a criação deste vetor, calcula-se as funções características apenas uma vez e utiliza-se o resultados diversas.

Além da criação deste vetor, outras alterações foram feitas na implementação do algoritmo, valendo a pena destacar termos levado em consideração as funções características retornarem apenas 1 ou 0. Levando isso em consideração, foi possível simplificar alguns cálculos em loops críticos e criar o vetor descrito anteriormente, aumentando a velocidade de execução do algoritmo. Por outro lado, isso limitou a implementação feita a esses casos particulares de funções características.

5.3 Resultados Obtidos

O algoritmo foi aplicado a cada um dos arquivos do corpus sendo obtidos os resultados apresentados na tabela 5.1.

De cada arquivo analisado, 90% das sentenças foram separadas para a fase de treino, 5% para ajuste e 5% para teste. Os 5% para ajuste não foram usados devido a limitações de tempo, de forma que os resultados apresentados referem-se apenas aos 5% de teste. O algoritmo, conforme discutido anteriormente, foi aplicado apenas para os núcleos das etiquetas. O resultado médio é apresentado na última linha da tabela. Como pode-se observar, o percentual de acerto por contexto do algoritmo sem nenhum ajuste manual é de 85%.

Quando ao invés dos arquivos individuais foram analisados conjuntos de arquivos agrupados por século, com vistas a aumentar a base de contextos de análise sem aumentar muito a variabilidade contextual, foram obtidos os resultados mostrados na tabela 5.2.

A constituição de cada século é apresentada na tabela 5.3.

Observa-se dos resultados apresentados por século que o aumento na quantidade do conjunto de contextos provocada pelos distintos autores, propiciou uma melhoria sensível no resultado do algoritmo. Este aumento no entanto saturou por volta de 88% com 300 mil ou 500 mil contextos. Esta saturação indica que um aumento no número de contextos analisados não melhorará substancialmente essa precisão.

Estes resultados mostraram-se parecidos com os primeiros obtidos por Chacur e Finger (85.86%) em [5] baseados no método de Brill. Ficaram no entanto muito aquém dos obtidos por Ratnaparkhi em [18] que foi cerca de 96%. Este resultado é parecido com os últimos obtidos por Chacur e Finger (95.43%) em [4]. Esta discrepância pode ser explicada pelo fato de não termos realizado o ajuste do algoritmo, e devido à maior variedade gramatical e ortográfica do Corpus Tycho Brahe, uma vez que o mesmo é um Corpus histórico, enquanto

Tabela 5.1: Resultados por arquivo do Corpus

arquivo	total de contextos	contextos analisados	acertos	% de acerto
aires_00_tcc.txt	66722	4860	4236	87.1605%
alorna_99_tcc.txt	49881	2191	1879	85.7599%
barros_02_pos.txt	50207	2067	1804	87.2762%
bernardes_02_pos.txt	59116	3242	2607	80.4133%
brochado_00_pos.txt	34633	1691	1438	85.0384%
cavaleiro_01_pos.txt	53672	2906	2480	85.3407%
chagas_99_tc.txt	57429	2666	2332	87.4719%
costa_99_tcc.txt	58400	1506	1268	84.1965%
couto_02_pos.txt	53916	2382	2083	87.4475%
garcao_00_tcc.txt	26845	1527	1313	85.9856%
garrett_00_tcc.txt	51473	2208	1985	89.9004%
gusmao_02_pos.txt	31235	1041	864	82.9971%
lobo_99_tc.txt	59685	3587	3070	85.5868%
luis.sousa_99_tc.txt	59157	2748	2332	84.8617%
maria_ceu_99_pos.txt	29775	1883	1614	85.7143%
melo_00_tcc.txt	58118	6492	5051	77.8035%
ortigao_01_pos.txt	34137	1540	1373	89.1558%
verney_00_tcc.txt	54952	2881	2502	86.8448%
vieira_letters_99_tc.txt	57823	2309	2071	89.6925%
vieira_sermons_99_pos.txt	62387	3718	3127	84.1044%
média	50478	2672	2271	85.0014%

Tabela 5.2: Resultados por século

arquivo	total de contextos	contextos analisados	acertos	% de acerto
século 16	172758	8717	7665	87.9316%
século 17	499123	29422	25973	88.2775%
século 18	303545	14303	12642	88.3871%
século 19	34137	1540	1373	89.1558%
média	252391	13496	11913	88.2757%

Tabela 5.3: Arquivos por século

século	arquivos
século 16	couto_02_pos.txt luis_sousa_99_tc.txt lobo_99_tc.txt
século 17	costa_99_tcc.txt vieira_letters_99_tc.txt vieira_sermons_99_pos.txt melo_00_tcc.txt chagas_99_tc.txt bernardes_02_pos.txt brochado_00_pos.txt maria_ceu_99_pos.txt barros_02_pos.txt gusmao_02_pos.txt
século 18	cavaleiro_01_pos.txt aires_00_tcc.txt verney_00_tcc.txt garcao_00_tcc.txt alorna_99_tcc.txt garrett_00_tcc.txt
século 19	ortigao_01_pos.txt

que o Penn Treebank é um Corpus do inglês moderno, com uma consistência ortográfica e gramatical muito maior.

Como o algoritmo utilizado é essencialmente o mesmo usado por Ratnaparkhi imaginamos que a discrepância no resultado pode ser causada por 3 motivos: a não realização da fase de ajuste, a maior complexidade e riqueza nas construções em Português, o fato do corpus usado por Ratnaparkhi ser baseado em textos modernos de jornal, provavelmente com uma maior homogeneidade ortográfica do que o corpus histórico utilizado. Embora não seja possível definir até que ponto a precisão pode ser melhorada, a fase de ajuste provavelmente proporcionaria um incremento na precisão do algoritmo, conforme atestam os resultados preliminares obtidos pela eliminação dos prefixos.

O tempo de execução inicial do algoritmo não foi verificado, entretanto estimativas com base no andamento parcial do programa implementado indicavam um tempo de execução para apenas um dos arquivos do corpus de mais de 4 dias (≈ 100 hs). Este tempo, após a implementação das otimizações descritas na Seção 5.2, foi reduzido para cerca de 1 h por arquivo.

Capítulo 6

Conclusão

Nesta tese apresentamos o algoritmo de maximização de entropia, mostrando sua base matemática e aplicando o mesmo ao problema prático da análise morfo-sintática em lingüística computacional. Esperamos ter tornado o entendimento do método mais acessível e de implementação mais imediata.

O algoritmo não obteve a precisão esperada e conseguida por Ratnaparkhi, entretanto o estado inicial da solução, as diferenças entre os Corpus e entre as Línguas usadas foram provavelmente as causas da baixa precisão. Embora a comparação não seja precisa, o estado inicial da implementação do algoritmo de Brill para o Português obteve uma precisão semelhante o obtido pela Maximização de Entropia. Atualmente obtém-se quase 96% de precisão com o algoritmo de Brill, e para o inglês Ratnaparkhi conseguiu obter também cerca de 96% de precisão. Isto cria a expectativa que progresso semelhante possa ser obtido trabalhando-se com a Maximização de Entropia.

O algoritmo, na implementação proposta, consome muita memória, entretanto consideramos que no futuro este não será um grande problema devido ao grande avanço apresentado pela indústria de hardware.

Embora os problemas enfrentados para implementar o algoritmo, principalmente no que diz respeito ao tempo de aprendizado e aplicação do algoritmo, o mesmo é de fácil adaptação para resolução de outros problemas e a experiência aqui apresentada será muito útil para os próximos que venham a usar o mesmo algoritmo encontrem menos dificuldade.

Além dos problemas intrínsecos ao algoritmo, encontramos alguns problemas com a literatura a respeito. Diversos autores escreveram sobre o mesmo, entretanto nenhuma abordagem completa foi encontrada, exigindo a consolidação dos diversos artigos analisados, sendo necessário, além de tornar consistente a nomenclatura encontrada, realizar algumas deduções pois existe um buraco entre a dedução matemática e o algoritmo implementado por outros pesquisadores.

Com relação à abordagem do algoritmo esperamos ter sido suficientemente completos para que com pouco esforço o mesmo possa ser aplicado a outros problemas. Caso se deseje utilizar a implementação apresentada para o problema da análise morfo-sintática, deve-se ainda ajustar a solução apresentada,

conforme descrito em 4.7.2, e completar a classificação das etiquetas.

Este método já foi aplicado no contexto de lingüística computacional para os problemas de análise sintática, classificação de textos (*text categorization*) e colocação de sintagmas preposicionais (*prepositional phrase attachment*) além da classificação morfossintática ora apresentada. Para maiores informações sobre a adaptação do método para a resolução destes problemas, sugerimos a leitura de Ratnaparkhi [18].

Fora do contexto de lingüística computacional o mesmo algoritmo já foi usado em reconhecimento de objetos em imagens (*Image Object Recognition*) por Daniel Keysers, Franz Josef Och e Hermann Ney em [14] e para modelamento de seqüências biológicas (*biological sequence modeling*) por Eugen C. Buehler e Lyle H. Ungar em [9].

Esta tese com o código fonte da implementação descrita pode ser encontrado em <http://www.ime.usp.br/~archias>. No apêndice encontram-se instruções sucintas para uso do programa.

Bibliografia

- [1] Homepage do penn treebank wall street journal corpus, 1998. Available at <http://www.cis.upenn.edu/~treebank>.
- [2] General manual: The tycho brahe parsed corpus of historical portuguese, 1999. Available at <http://www.ime.usp.br/~tycho/corpus/manual>.
- [3] Homepage do projeto *Rhythmic Patterns, Parameter Setting & Language Change*, 1999. Available at <http://www.ime.usp.br/~tycho>.
- [4] Técnicas de otimização da precisão empregadas no etiquetador tycho brahe, 2000. Available at <http://www.ime.usp.br/~mfinger>.
- [5] Carlos D. C. Alves and Marcelo Finger. Etiquetagem do português clássico baseada em corpóra. In *IV Encontro para Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR99)*, Évora, Portugal, September 1999.
- [6] Martin Anthony and Norman Biggs. *Computational Learning Theory*. Cambridge University Press, Cambridge, 1992.
- [7] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–68, 1996.
- [8] E. Brill. Transformation-based error driven learning and natural language. a case study in part of speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.
- [9] Eugen C. Buehler and Lyle H. Ungar. Maximum entropy methods for biological sequence modeling. In *BIOKDD*, pages 60–64, 2001. Available at: <http://citeseer.nj.nec.com/449883.html>.
- [10] Eugene Charniak. *Statistical Language Learning*. The MIT Press, Cambridge, Massachusetts, 1993.
- [11] J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43:1470–1480, 1972.

- [12] Silviu Guiasu and Abe Shenitzer. The principle of maximum entropy. *The Mathematical Intelligencer*, 7(1):42–48, 1985.
- [13] E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106:620–630, 1957. Also available on the internet: <http://bayes.wustl.edu/etj/node1.html>.
- [14] Daniel Keysers, Franz Josef Och, and Hermann Ney. Maximum entropy and gaussian models for image object recognition. Available at: <http://citeseer.nj.nec.com/531771.html>.
- [15] Raymond Lau, Ronald Rosenfeld, and Salim Roukos. Adaptive language modeling using the maximum entropy principle. In *Proceedings of the Human Language Technology Workshop*, pages 108–113. ARPA, 1993. Also available on the internet: <http://www.raylau.com/publications.html>.
- [16] Raymond Lau, Ronald Rosenfeld, and Salim Roukos. Trigger-based language models: a maximum entropy approach. In *Proceedings of ICASSP-93*, April 1993. Also available on the internet: <http://www.raylau.com/publications.html>.
- [17] Stephen A. Della Pietra, Vincent J. Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.
- [18] Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania, 1998. Also available on the internet: <http://www.cis.upenn.edu/~adwait/statnlp.html>.
- [19] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, 1949.

Apêndice A

Programa de Exemplo

Para poder testar o algoritmo descrito na tese, foi criado um programa auxiliar que implementa o algoritmo. Este programa foi feito em C++ usando a ferramenta Visual C++. O programa é apenas um protótipo, não estando nem completo nem sendo garantido que o seu funcionamento é correto, devendo o usuário levar essas limitações em consideração quando estiver usando o programa. O programa pode ser alterado livremente por quem quiser e para quaisquer fins.

A.1 Onde Obter

O programa executável para Windows (GramClass.exe) e o código fonte com projeto para VC6 do mesmo (GramClass.zip) podem ser encontrados em <http://www.ime.usp.br/~archias>. O programa foi feito para usar os arquivos etiquetados do corpus Tycho Brahe para treinamento e teste. Estes arquivos podem ser encontrados em <http://www.ime.usp.br/~tycho>.

A.2 Como Instalar

O programa não necessita de instalação porém aconselhamos que seja criado um diretório para colocar o executável, pois o mesmo cria diversos arquivos no diretório em que se encontra.

Já o projeto, também não precisa ser instalado, entretanto provavelmente será necessário alterar o diretório de trabalho nas configurações do projeto.

A.3 Como Utilizar o Programa

Uma vez iniciado o programa, deve-se selecionar os arquivos para serem usados no treino e no teste:

1. clique no botão *Corpora...*;

2. na janela de diálogo que surge, clique em *Adiciona...*;
3. selecione os arquivos desejados e clique *Abrir*;

Os arquivos selecionados aparecem com a quantidade de palavras e a quantidade de sentenças contidas no mesmo. Caso tenha ocorrido algum erro na leitura do arquivo, o mesmo encontrar-se-á registrado no arquivo de texto *errors.err* no mesmo diretório do arquivo que foi lido pelo programa. Uma vez selecionados os arquivos, defina como os mesmos serão distribuídos entre as fases de treino, ajuste e teste. Para tanto:

1. selecione o arquivo na tabela de arquivos carregados;
2. clique em *Configurações...*;
3. Na tela que surge, defina os valores de sentenças ou tokens que serão usados em cada fase, ou em termos percentuais ou em termos absolutos. e clique em *OK*;

Agora vamos retornar à tela principal.

1. clique em *OK*;

Antes de começar o treinamento, pode-se visualizar as classes carregadas com o conjunto de arquivos clicando-se em *Classes*. Na tela que surge pode-se apenas navegar na árvore de classes criadas e ver alguns detalhes das classes.

Retornando-se à tela principal, pode-se ainda gerar previamente o conjunto de funções características e visualizá-lo clicando-se no botão *F. Características* que abre a tela de funções características. Nesta tela, clicando-se em *Criar e Selecionar* são criadas e selecionadas as funções características caso já não o tenham sido devido a um procedimento de treinamento realizado anteriormente. Na tabela mostrada nesta tela, o tipo do predicado contextual usado pela função característica está codificado da seguinte forma:

$X = y$, onde X pode assumir um dos seguintes valores: W , $W-1$, $W+1$, $C-1$, $s(k,W)$; y pode assumir um dos seguintes valores: *cte*, *Maiúscula*, *número* ou com hífen.

W significa a palavra central do contexto; $W-1$ a palavra anterior; $W+1$ a palavra posterior; $C-1$ a classe da palavra anterior; $s(k,W)$ o sufixo de tamanho k da palavra central do contexto; *cte* significa constante; e as outras são auto-explicativas. Assim, $W = cte$ significa que o predicado contextual correspondente à função característica define que a palavra analisada é uma constante.

O valor extra que aparece na tabela indica qual constante está sendo usada no predicado contextual.

Voltando à tela principal, os botões *Carregar...* e *Salvar...* não funcionam, mas não se preocupe, pois o programa salva automaticamente o seu estado quando é encerrado e recupera o mesmo estado quando é iniciado. Isto ocorre pois inicialmente o algoritmo era muito lento e para facilitar o debug, salvamos o estado do programa em arquivos *.sv. O Botão *Parâmetros...* é usado para

alterar os parâmetros de treinamento, porém também não indicamos o uso do mesmo, posto que o programa só foi usado para a fase 1. A fase 2 seria a classificação completa das palavras, e não apenas os núcleos, como faz a fase 1.

Para treinar, basta apertar o botão *Treinar*

Após realizado o treinamento, pode-se classificar os dados do corpus lidos. Esta classificação na realidade funciona como um teste, pois não implementamos o suporte para arquivos externos. Para classificar os arquivos previamente carregados, defina um arquivo de saída, selecione se é a parte separada para ajuste ou para teste, mantenha o checkbox *Classificar Corpus* selecionado e aperte o botão *Classificar!*. Ao final do procedimento de classificação o resultado pode ser encontrado no arquivo definido para saída. Pode-se visualizar os erros de classificação clicando-se no botão *Erros...* na janela principal. A janela de erros é mostrada. Nesta janela, pode-se observar todos os erros de classificação e clicando-se em um dos erros são mostrados os contextos nos quais os erros ocorreram.

A.4 Descrição Sucinta das Classes do Programa

Devido ao uso da MFC para a construção da interface gráfica, o método de entrada do programa é *CGramClassApp::InitInstance()*. A classe do diálogo principal é *CGramClassDlg* e as principais classes de organização dos dados são: *CClassManager*, *CStringManager*, *CContextManager*, *CCorpora*, *CPredicateMgr* e *CFCManager*. Destas, *CClassManager* organiza as classes, *CStringManager* faz um mapeamento de strings para ints, *CContextManager* organiza os contextos usados, *CCorpora* organiza os arquivos do Corpus, *CPredicateMgr* coordena a criação de predicados contextuais e suas definições e *CFCManager* cria e organiza as funções características.

As principais classes de cálculo são: *CEntropyMaximizer*, *MECoordinator* e *CClassifier*. Destas, *MECoordinator* coordena o processo de treinamento, *CEntropyMaximizer* realiza o cálculo da maximização de entropia propriamente dito e *CClassifier* é responsável pela classificação pós treinamento.

As classes da interface são todas terminadas em *Dlg*, e as demais classes são todas estruturas de dados auxiliares.