UNIVERSIDADE DE SÃO PAULO

Escola de Artes, Ciências e Humanidades

Cláudio Frizzarini

Algoritmo para Indução de Árvores de
Classificação para Dados Desbalanceados

Algorithm for Induction of Classification
Trees for Unbalanced Data

São Paulo
Novembro de 2013

UNIVERSIDADE DE SÃO PAULO

Escola de Artes, Ciências e Humanidades

Cláudio Frizzarini

Algoritmo para Indução de Árvores de Classificação para Dados Desbalanceados

Algorithm for Induction of Classification Trees for Unbalanced Data

São Paulo

Novembro de 2013

Cláudio Frizzarini

Algoritmo para Indução de Árvores de Classificação para Dados Desbalanceados

Algorithm for Induction of Classification Trees for Unbalanced Data

Dissertação apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo para obtenção do título de Mestre em Ciências. Programa: Sistemas de Informação.

Versão corrigida contendo as alterações solicitadas pela comissão julgadora. A versão original encontra-se disponível na Escola de Artes, Ciências e Humanidades.

Orientador: Prof. Dr. Marcelo de Souza Lauretto

São Paulo
Novembro de 2013

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

CATALOGAÇÃO-NA-PUBLICAÇÃO

Biblioteca

Escola de Artes, Ciências e Humanidades da Universidade de São Paulo

Frizzarini, Cláudio

Algoritmo para indução de árvores de classificação para dados desbalanceados / Cláudio Frizzarini ; orientador, Marcelo de Souza Laretto. – São Paulo, 2013.

[95] f. : il.

Dissertação (Mestrado em Ciências) - Programa de Pós-Graduação em Sistemas de Informação, Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, em 2013.

Versão corrigida.

1. Algoritmos e estrutura de dados. 2. Mineração de dados. 3. Aprendizado computacional. 4. Algoritmos para processamento. I. Laretto, Marcelo de Souza, orient. II. Título.

CDD 22.ed. – 005.1

Folha de Aprovação

Dissertação sob o título “*Algoritmo para Indução de Árvores de Classificação para Dados Desbalanceados*”, defendida por Cláudio Frizzarini e aprovada em 21 de novembro de 2013, em São Paulo, Estado de São Paulo, pela banca examinadora constituída pelos doutores:

Prof. Dr. Marcelo de Souza Lauretto
Orientador

Prof. Dr. Roberto Hirata Junior
Universidade de São Paulo

Prof. Dr. Clodoaldo Aparecido de Moraes
Lima
Universidade de São Paulo

Dedicatória

Dedico esta dissertação a minha família e todos os amigos que apoiaram e incentivaram o desenvolvimento deste trabalho e com muita paciência compreenderam minha ausência em vários momentos.

Agradecimentos

Agradeço ao Prof. Dr. Marcelo de Souza Lauretto por todo ensinamento, amizade, incentivo e pelo exemplo de colaboração e determinação. Muito obrigado por tudo.

A todos os meus familiares por todo incentivo, apoio e compreensão, vocês são minha vida.

Aos meus amigos do PPgSI por toda contribuição e amizade, boa sorte a todos.

Aos meus amigos da Copersucar pelo interesse e apoio ao meu mestrado.

A todos os professores do PPgSI por serem parte importante desta minha formação e por todo incentivo durante o mestrado.

A Copersucar por permitir e apoiar a minha participação neste mestrado.

Lute com determinação, abrace a vida com paixão,
perca com classe e vença com ousadia,
porque o mundo pertence a quem se atreve e a vida é
muito para ser insignificante..

(Charles Chaplin)

Resumo

FRIZZARINI, Cláudio. **Algoritmo para Indução de Árvores de Classificação para Dados Desbalanceados**. 2013. 95 f. Dissertação (Mestrado em Sistemas de Informação) – Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo, 2013

As técnicas de mineração de dados, e mais especificamente de aprendizado de máquina, têm se popularizado enormemente nos últimos anos, passando a incorporar os Sistemas de Informação para Apoio à Decisão, Previsão de Eventos e Análise de Dados. Por exemplo, sistemas de apoio à decisão na área médica e ambientes de *Business Intelligence* fazem uso intensivo dessas técnicas.

Algoritmos indutores de árvores de classificação, particularmente os algoritmos TDIDT (Top-Down Induction of Decision Trees), figuram entre as técnicas mais comuns de aprendizado supervisionado. Uma das vantagens desses algoritmos em relação a outros é que, uma vez construída e validada, a árvore tende a ser interpretada com relativa facilidade, sem a necessidade de conhecimento prévio sobre o algoritmo de construção.

Todavia, são comuns problemas de classificação em que as frequências relativas das classes variam significativamente. Algoritmos baseados em minimização do erro global de classificação tendem a construir classificadores com baixas taxas de erro de classificação nas classes majoritárias e altas taxas de erro nas classes minoritárias. Esse fenômeno pode ser crítico quando as classes minoritárias representam eventos como a presença de uma doença grave (em um problema de diagnóstico médico) ou a inadimplência em um crédito concedido (em um problema de análise de crédito). Para tratar esse problema, diversos algoritmos TDIDT demandam a calibração de parâmetros *ad-hoc* ou, na ausência de tais parâmetros, a adoção de métodos de balanceamento dos dados. As duas abordagens não apenas introduzem uma maior complexidade no uso das ferramentas de mineração de dados para usuários menos experientes, como também nem sempre estão disponíveis.

Neste trabalho, propomos um novo algoritmo indutor de árvores de classificação para problemas com dados desbalanceados. Esse algoritmo, denominado atualmente DDBT (Dynamic Discriminant Bounds Tree), utiliza um critério de partição de nós que, ao invés de se basear em frequências absolutas de classes, compara as proporções das classes nos nós com as proporções do conjunto de treinamento original, buscando formar subconjuntos com maior discriminação de classes em relação ao conjunto de dados original. Para a rotulação de nós terminais, o algoritmo atribui a classe com maior prevalência relativa no nó em relação à prevalência no conjunto original. Essas características fornecem ao algoritmo a flexibilidade para o tratamento de conjuntos de dados com desbalanceamento de classes, resultando em um maior equilíbrio entre as taxas de erro em classificação de objetos entre as classes.

Palavras-chave: Árvore de Classificação, Árvore de Decisão, Dados Desbalanceados, Aprendizado Supervisionado, Mineração de Dados

Abstract

FRIZZARINI, Cláudio. **Algorithm for Induction of Classification Trees for Unbalanced Data**. 2013. 95 p. Dissertation (Master in Information Systems) – School of Arts, Sciences and Humanities, University of São Paulo, São Paulo, 2013.

Data mining techniques and, particularly, machine learning methods, have become very popular in recent years. Many decision support information systems and business intelligence tools have incorporated and made intensive use of such techniques.

Top-Down Induction of Decision Trees Algorithms (TDIDT) appear among the most popular tools for supervised learning. One of their advantages with respect to other methods is that a decision tree is frequently easy to be interpreted by the domain specialist, precluding the necessity of previous knowledge about the induction algorithms.

On the other hand, several typical classification problems involve unbalanced data (heterogeneous class prevalence). In such cases, algorithms based on global error minimization tend to induce classifiers with low error rates over the high prevalence classes, but with high error rates on the low prevalence classes. This phenomenon may be critical when low prevalence classes represent rare or important events, like the presence of a severe disease or the default in a loan. In order to address this problem, several TDIDT algorithms require the calibration of *ad-hoc* parameters, or even data balancing techniques. These approaches usually make data mining tools more complex for less expert users, if they are ever available.

In this work, we propose a new TDIDT algorithm for problems involving unbalanced data. This algorithm, currently named DDBT (Dynamic Discriminant Bounds Tree), uses a node partition criterion which is not based on absolute class frequencies, but compares the prevalence of each class in the current node with those in the original training sample. For terminal nodes labeling, the algorithm assigns the class with maximum ration between the relative prevalence in the node and the original prevalence in the training sample. Such characteristics provide more flexibility for the treatment of unbalanced data-sets, yielding a higher equilibrium among the error rates in the classes.

Keywords: Classification tree, unbalanced data, Supervised Learning, Decision Tree, Data Mining

Lista de Figuras

1	Representação esquemática do processo de aprendizado supervisionado . . .	8
2	Estruturas de uma árvore de classificação [1]	9
3	Exemplo das medidas de impureza de um nó para classificação binária [2] .	16
4	Exemplo de Poda “ <i>Subtree Raising</i> ” [3]	24
5	Validação Cruzada - V-fold	27
6	Espaço ROC para análise do desempenho entre as classes	33
7	Análise da linha de isodesempenho [4]	35
8	Gráficos da função de perda do REAL	41
9	Gráficos da função de convicção do REAL	42
10	Gráficos da função de convicção do DDBT	44
11	Amostras de Partições do nó, com e sem sucesso	45
12	Gráficos ROC para os conjuntos de dados - 1ª parte	71
13	Gráficos ROC para os conjuntos de dados - 2ª parte	72

Lista de Tabelas

1	Principais características dos níveis de escalas de medição	6
2	Exemplo de conjunto de treinamento [1]	8
3	Custo de Erro de Classificação - exemplo em diagnóstico médico	13
4	Exemplo hipotético das taxas de acerto de dois classificadores ψ_1 e ψ_2 sobre um conjunto de dados com desbalanceamento entre as classes	24
5	Matriz de Confusão	28
6	Matriz de Confusão do exemplo hipotético do classificador ψ_1 apresentado na seção 2.5	30
7	Sumários dos conjuntos de dados utilizados	60
8	Matriz de Confusão comparativa do DDBT e REAL por conjunto de dados.	63
9	Comparativo DDBT e REAL - Taxas de Precisão; Falso Negativo e Falso Positivo	63
10	Comparativo DDBT e REAL - F-Score com $\beta = 1$ e do AUC para cada conjunto de dados	64
11	Matriz de Confusão por algoritmo e por conjunto de dados.	65
12	Média dos valores obtidos para as Taxas de Precisão; Falso Negativo e Falso Positivo	66
13	Resultado do F-Score com $\beta = 1$ e do AUC para cada conjunto de dados .	68

Lista de Símbolos

$\mathcal{U}$	Conjunto Universo.....	7
$\mathcal{L}$	Conjunto de Treinameto	7
N	Tamanho do conjunto de treinamento	7
M	Número de atributos	7
K	Número de classes	7
χ_j	Conjunto dos possíveis valores do atributo j	7
χ	Espaço dos atributos ($\chi = \chi_1 \times \chi_2 \times \dots \times \chi_M$)	7
a	conjunto de M atributos ou características de cada elemento de $\mathcal{U}$..	7
$\mathbf{x}$	Vetor de atributos	7
x_j	Valor do atributo j no vetor de atributos	7
$n_{\bullet t}$	Número de exemplos incidentes nó t	12
n_{kt}	Número de exemplos incidentes nó t pertencentes à classe k	12
$\psi(\bullet, \mathcal{L})$	Classificador induzido a partir do conjunto de treinamento $\mathcal{L}$	7
t	Nó atual na árvore	11
$\pi_{k,t}$	Probabilidade desconhecida de $\mathbf{x} \in \chi$ do nó t pertencer à classe k .	12
$\hat{\pi}_{k,t}$	Estimador pontual para $\pi_{k,t}$	12
$r(t)$	Erro estimado no nó t	12
$C(l, k)$	Custo do erro em classificar um exemplo de classe l como k	13
$\mathcal{G}(t)$	Índice Gini no nó t	15
$\mathcal{E}(t)$	Entropia no nó t	15
$GR(t)$	Razão de Ganho de informação no nó t	17

Sumário

1	Introdução	1
1.1	Considerações Iniciais	1
1.2	Objetivos	3
1.3	Organização	3
2	Aprendizado Supervisionado e Árvores de Classificação	4
2.1	Aprendizado Supervisionado	4
2.2	Árvores de Classificação	8
2.3	Construção da Árvore de Classificação	10
2.3.1	Rotulação de Nós Terminais	12
2.3.1.1	Critério de Minimização do Erro Estimado	12
2.3.1.2	Critério de Minimização do Custo Estimado do Erro . . .	13
2.3.2	Seleção de Atributo	14
2.3.2.1	Índice Gini	15
2.3.2.2	Entropia	15
2.3.2.3	Razão de Ganho de Informação	16
2.4	Poda da Árvore	17
2.4.1	Pré-poda	18
2.4.2	Pós-poda	18
2.4.2.1	Redução de Erros - <i>Reduced-error pruning</i>	19
2.4.2.2	Custo-Complexidade - <i>Cost-complexity pruning</i>	19

2.4.2.3	Erro Pessimista - <i>Pessimistic error pruning</i>	21
2.4.2.4	Valor Crítico - <i>Critical value pruning</i>	22
2.4.2.5	Erro Mínimo - <i>Minimum-error pruning</i>	22
2.4.2.6	Poda por Estimativa de Erro - <i>Error-based pruning</i>	23
2.5	Problemas com Dados Desbalanceados	23
3	Desempenho de Classificadores	26
3.1	Validação Cruzada	26
3.2	Matriz de Confusão	27
3.3	Medidas de Desempenho	28
3.3.1	Taxas de Erros e Acertos	29
3.3.2	Indicadores	31
3.4	Análise Gráfica - Gráfico ROC	32
3.5	Equivalência Estatística - Teste t Pareado	34
4	O método DDBT	38
4.1	Regra de Rotulação	39
4.2	Função de Convicção	39
4.2.1	Função de Convicção do REAL	40
4.2.2	Função de Convicção do DDBT	42
4.3	Divisão dos Nós	44
5	Algoritmos TDIDT Comparados	46
5.1	Considerações Iniciais	46
5.2	CTree (Conditional Inference Trees)	46
5.3	J48	48
5.3.1	Tipo de árvore	48
5.3.2	Expansão da árvore	49

5.3.3	Poda da árvore	49
5.4	LMT – Logistic Model Trees	49
5.4.1	Regressão Logística	49
5.4.2	Estimação dos Coeficientes	51
5.4.3	Seleção de Atributos	52
5.4.4	Tratamento de Atributos Categóricos	53
5.4.5	Indução da Árvore	53
5.5	Random Forest	54
5.6	RPART	56
6	Avaliações Numéricas do Algoritmo	59
6.1	Datasets	59
6.2	Considerações Iniciais	61
6.3	Análise do Desempenho comparativa entre DDBT e REAL	62
6.3.1	Matriz de Confusão	62
6.3.2	Taxas de Desempenho	62
6.3.3	Análise do F-Score e do AUC	64
6.4	Análise do Desempenho comparativa entre DDBT e demais algoritmos	65
6.4.1	Matriz de Confusão	65
6.4.2	Taxas de Desempenho	66
6.4.3	Análise do F-Score e do AUC	67
6.4.4	Análise do Gráfico ROC e Teste t-Pareado por Conjunto de Dados	69
7	Conclusões	73
7.1	Trabalhos Futuros	74
	Referências	75

Capítulo 1

Introdução

1.1 Considerações Iniciais

A tomada de decisão baseada em análise de dados é uma atividade bastante corriqueira em diversas áreas. Por exemplo: o médico, com base em exames realizados, estabelece o diagnóstico do paciente; o bancário, com base nos dados financeiros de uma pessoa, disponibiliza a concessão de crédito ou não; o militar, com base nas características da ligação, identifica se a chamada de socorro é um trote ou não. Quando a tomada de decisão precisa ser feita em série ou o volume de informação a ser analisado é muito grande, a necessidade de automação desse processo passa a ser fundamental, e uma das ferramentas que podem ser utilizadas para esse propósito são os classificadores.

A *árvore de decisão* (ou *árvore de classificação*) é um tipo de classificador muito utilizado devido ao fato de suas regras serem estruturadas de forma lógica, facilitando o entendimento de seu funcionamento, mesmo por pessoas sem o conhecimento prévio sobre o algoritmo de construção.

Árvore de Decisão é um classificador preditivo de aprendizado supervisionado ou de aprendizado semi-supervisionado, sendo que o interesse neste trabalho é pela Árvore de Decisão de aprendizado supervisionado. Aprendizado Supervisionado porque a construção desse tipo de classificador é feita com base em um conjunto de eventos (exemplos) já rotulados com a classe correta (classificado), sendo que o classificador “aprende” a rotulação já realizada para poder determinar a classe de novos exemplos. Preditivo porque sua função é, com base nas características observadas de um novo elemento, predizer sua classe.

TDIDT (*Top-Down Induction of Decision Trees*) é uma técnica de construção (indução) de árvores de decisão. A construção da árvore começa com um conjunto de exemplos, chamado de conjunto de treinamento, que é dividido em subconjuntos de acordo com critérios de particionamento que buscam a melhor separação entre as classes. A cada subconjunto gerado é associado um novo nó da árvore. O processo de particionamento

é aplicado recursivamente em cada nó até que um critério de parada seja atingido, ou que todos os exemplos em cada subconjunto tenham a mesma classe. Cada subconjunto não particionado dá origem a um nó folha rotulado por uma classe. O resultado deste processo é representado por uma árvore em que: cada nó especifica um atributo testado; cada ramo que emana de um nó especifica os valores ou intervalos possíveis do atributo; e as extremidades da árvore são nós folha. Quando um novo exemplo precisa ser classificado, seus atributos são testados a partir do nó raiz (primeiro nó da árvore), seguindo sucessivamente pelos ramos e testes dos nós internos, até chegar a um nó folha; a classe atribuída ao exemplo será aquela que rotula a folha.

O critério para particionamento de um nó é determinado pelo algoritmo de indução da árvore. Por exemplo, o algoritmo C4.5 [5] utiliza o critério de maior Razão de Ganho de informação; já o algoritmo CART (*Classification and Regression Trees*) [6] utiliza o índice de Gini.

É normal encontrarmos problemas envolvendo conjunto de dados desbalanceados, ou seja, com alta prevalência de uma ou mais classes. Nesses casos, normalmente os classificadores baseados na minimização do erro global não têm um bom desempenho na classificação dos exemplos de classe minoritária. Tomemos como exemplo o diagnóstico de uma doença grave e rara, sendo que na maioria dos casos o diagnóstico é negativo (não apresenta a doença), e em alguns poucos casos observados o diagnóstico é positivo. Essa é uma situação característica de conjunto de dados desbalanceados, na qual se espera que um classificador apresente baixas taxas de erro na predição da classe minoritária (diagnóstico positivo), mesmo que apresente alguns erros na classificação da classe majoritária. Isto se justifica porque errar a predição da classe minoritária significa potencialmente negligenciar o tratamento a pacientes com a doença. Por outro lado, errar a predição da classe majoritária significa dizer que paciente tem a doença quando ele não tem; tal situação é menos grave, pois normalmente se pode recorrer a exames confirmatórios. Esse exemplo ilustra por que um bom desempenho na classificação dos exemplos da classe minoritária pode ser um fator determinante na escolha do classificador.

Na busca por melhor desempenho na classificação de exemplos da classe minoritária, alguns algoritmos utilizam parâmetros para calibração, como por exemplo pesos ou custos de erros de classificação. Existem também métodos que buscam minimizar o desbalanceamento das classes do conjunto de treinamento. Normalmente, tais métodos buscam: eliminar exemplos menos significativos da classe majoritária; ou replicar exemplos da classe minoritária; ou ainda, combinar essas duas abordagens [7]. Todavia, normalmente

a calibração desses parâmetros não é automática, tornando a utilização dos algoritmos de classificação mais complexa para usuários menos experientes.

1.2 Objetivos

O principal objetivo deste trabalho é apresentar um algoritmo de indução de árvore de classificação alternativo para problemas envolvendo conjuntos de dados com desbalanceamento entre as classes, bem como comparar seu desempenho com outros cinco algoritmos.

Esse algoritmo, denominado atualmente DDBT (*Dynamic Discriminant Bounds Tree*), é uma extensão do algoritmo REAL (*Real-Valued Attribute Learning*)[8, 9]. Tanto para o REAL como para o DDBT, o critério de particionamento de um nó é dado por uma função de ganho de convicção, detalhada no Capítulo 4. Entretanto, para o DDBT, essa função foi modificada para incorporar o tratamento automático de conjuntos de dados com desbalanceamento entre as classes. A ideia principal é que, para o DDBT, o particionamento de um nó seja orientado por um critério de significância estatística, que busca dinamicamente os limitantes da discriminação das classes do nó em relação à distribuição das classes no conjunto de treinamento original.

Por restrições na concepção da função de ganho de convicção, o DDBT é aplicável em problemas de classificação binária, onde as classes estão restritas a dois valores.

1.3 Organização

O Capítulo 2 descreve os conceitos básicos relacionados aos algoritmos de indução de árvores de classificação. O Capítulo 3 apresenta os métodos de avaliação e seleção de classificadores utilizados neste trabalho. No Capítulo 4 é apresentado o DDBT como uma alternativa de algoritmo de indução de árvore de classificação, voltado para problemas com dados desbalanceados. O Capítulo 5 apresenta os cinco algoritmos adotados nas análises comparativas. O Capítulo 6 descreve a avaliação numérica do DDBT frente aos demais algoritmos, e discute seus resultados. Finalmente, o Capítulo 7 apresenta a conclusão deste trabalho e os possíveis trabalhos futuros.

Capítulo 2

Aprendizado Supervisionado e Árvores de Classificação

2.1 Aprendizado Supervisionado

Um *classificador* é um conjunto de regras, comandos ou funções construídos com o objetivo de prever a classe de um objeto, com base em seus *atributos* ou *características* observados. A construção – também denominada *indução* – do classificador pode ser realizada por *aprendizado supervisionado*, *aprendizado semi-supervisionado* ou por *aprendizado não supervisionado*.

O classificador de *aprendizado não supervisionado* realiza a identificação da classe sem que os dados sejam previamente rotulados. Neste caso, o objetivo do algoritmo é particionar os dados de treinamento em um certo número de *clusters* (ou classes) com alta homogeneidade interna e alta heterogeneidade externa. Ou seja, espera-se que objetos pertencentes aos mesmos *clusters* tenham alta similaridade, enquanto objetos pertencentes a *clusters* distintos apresentem baixa similaridade entre si. Essa abordagem é usualmente denominada *Análise de Agrupamentos* [10].

Os classificadores de *aprendizado supervisionado*, que são o foco deste estudo, são aqueles construídos com base em um conjunto de exemplos com a classe já rotulada. Ou seja, o algoritmo busca induzir classificadores que discriminem adequadamente os exemplos fornecidos segundo suas classes, de forma que, uma vez construídos, tais classificadores possam prever adequadamente as classes de novos exemplos.

Os classificadores de *aprendizado semi-supervisionado*, utilizam os conceitos dos classificadores de *aprendizado supervisionado* e de *aprendizado não supervisionado*, ou seja, neste caso, o classificador é construído utilizando conjuntos de exemplos com a classe já rotulada e com conjunto de exemplos onde a classe não foi previamente rotulada. [11]

Um domínio clássico de interesse de aprendizado supervisionado é o de diagnósticos médicos. Dado um conjunto de dados de pacientes já diagnosticados como positivos

ou negativos em relação a uma determinada doença, usualmente busca-se realizar um conjunto de análises para identificar o relacionamento entre a presença e ausência da doença com as informações relativas ao paciente (resultados de exames, hábitos, etc). A partir dessas análises, obtém-se um procedimento (ou classificador) que possa prever o diagnóstico de novos pacientes com base em suas informações fornecidas (atributos).

Um atributo é uma variável observável e independente que representa a propriedade ou característica de um elemento [12]. Conforme descrito por Maimom [13], o atributo pode ser classificado em dois tipos: *qualitativo* (também conhecido como *categorico*) ou *quantitativo*.

Os valores dos atributos *qualitativos* podem ser agrupados em diferentes categorias que se distinguem por alguma característica, por exemplo: o tipo de sangue de uma pessoa (A, B, AB, O); e resultado de uma avaliação (ruim, regular, bom, excelente). Os atributos qualitativos podem apresentar ordenação dos valores, como é o caso do segundo exemplo acima. Todavia, as operações aritméticas (soma, subtração, média, etc.) não podem ser aplicadas a eles, mesmo que sua representação seja numérica.

Os valores dos atributos *quantitativos* são de natureza numérica e representam contagens e medidas. Eles podem ser classificados e também admitem as operações aritméticas. Os atributos quantitativos podem ser classificados em dois grupos: *discretos* ou *contínuos*.

- Os valores dos atributos *discretos* estão contidos em um conjunto finito e que podem ser contados, por exemplo: número de crianças na família.
- Os valores dos atributos *contínuos* estão contidos nos números reais, ou seja, podem assumir qualquer valor dentro de um intervalo. Os valores são obtidos através de medição ou observação, por exemplo: temperatura.

Além da classificação por tipo qualitativo ou quantitativo, o atributo também pode ser classificado pela forma como seus valores são agrupado, contados ou medidos. Esse tipo de classificação utiliza as escalas de medição, onde os quatro níveis mais comuns dessa escala são: *nominal*, *ordinal*, *intervalar* e *razão*. A Tabela 1 apresenta as principais características das escalas de medição.

- O nível *nominal* é composto por atributos cujos valores identificam nomes, rótulos ou categorias. Os valores de um atributo do tipo nominal não têm uma ordem natural e não devem ser usados em operações aritméticas, mesmo que sua representação

seja numérica. Exemplos: sexo (masculino, feminino), código de peça (3322, 1704, 6167).

- O nível *ordinal* é composto por atributos cujos valores podem ser organizados em alguma ordem. No entanto, a magnitude das diferenças entre as classificações não pode ser calculada. Por exemplo, em um resultado degustativo (ruim, aceitável, gostoso) é possível ordenar os valores, mas não sabemos se a diferença entre aceitável e ruim é a mesma que a existente entre gostoso e aceitável.
- O nível *intervalar* é semelhante ao nível ordinal (os valores podem ser organizados em alguma ordem), mas as diferenças entre as classificações podem ser calculadas. No entanto, esses valores não possuem um ponto inicial “zero” no intervalo de dados. Um exemplo é a escala de temperatura Fahrenheit, onde cada grau é uma diferença significativa nessa unidade de medida. Ou seja, 74 graus são dois graus mais quente que 72 graus, mas 0 graus Fahrenheit não significa ausência de calor, e também não significa dizer que 100 graus é duas vezes mais quente que 50 graus.
- O nível de *razão* é semelhante ao nível intervalar, com a diferença de possuir o ponto inicial “zero” no espaço de valores. Assim, existem verdadeiras relações entre os diferentes valores medidos. Por exemplo, duas famílias, onde a primeira tem 4 crianças e a segunda tem 2 crianças, é significativo a dizer que a primeira família tem o dobro das crianças da segunda família.

Tabela 1 – Principais características dos níveis de escalas de medição

Nível	Ordenação	Operação Aritimética	Ponto inicial zero
Nominal	não	não	não
Ordinal	sim	não	não
Intervalar	sim	sim	não
Razão	sim	sim	sim

Em resumo, temos a seguinte classificação de tipos de atributo e níveis de escalas: **Atributos Qualitativos** - nominal ou ordinal; e **Atributos Quantitativos** - Intervalar (discreto ou contínuo) ou Razão (discreto ou contínuo).

Introduzimos a notação apresentada neste trabalho. Denotemos por $\mathcal{U}$ o conjunto universo, ou seja, o conjunto de objetos observáveis do domínio de interesse. Suponha

que cada elemento de $\mathcal{U}$ seja descrito por um conjunto de M atributos ou características $a_1, \dots, a_M$. O vetor $\mathbf{x} = (x_1, x_2, \dots, x_M)$ denota os valores dos atributos $a_1, \dots, a_M$, de um dado elemento de $\mathcal{U}$, onde x_j corresponde ao valor do atributo a_j , com $j = 1 \dots M$. Esse vetor é usualmente denominado de vetor de atributos (ou de características) do elemento. Denotando por χ_j o conjunto dos possíveis valores de a_j , o produto cartesiano $\chi = \chi_1 \times \chi_2 \times \dots \times \chi_M$ é denominado *espaço de atributos* ou *espaço amostral* e corresponde ao conjunto de todos os vetores de atributos possíveis.

No contexto de classificação supervisionada, assume-se a existência de uma partição do conjunto universo $\mathcal{U}$ em K subconjuntos disjuntos e não vazios $\mathcal{U}_1, \mathcal{U}_2, \dots, \mathcal{U}_K$. Neste trabalho as classes são denotadas por seus respectivos índices $k = 1, 2, \dots, K$.

Um *conjunto de treinamento*, denotado por $\mathcal{L}$, é um conjunto de N *exemplos* (observações),

$$\mathcal{L} = \{(\mathbf{x}_{i,\bullet}, y_i), \quad i = 1, 2, \dots, N\}, \quad (2.1)$$

onde $\mathbf{x}_{i,\bullet} = (x_{i,1}, x_{i,2}, \dots, x_{i,M}) \in \chi$ e $y_i \in \{1, 2, \dots, K\}$ denotam, respectivamente, o vetor de atributos e a classe do i -ésimo exemplo. O valor do atributo a_j no i -ésimo exemplo é denotado por $x_{i,j}$. Assume-se, tradicionalmente, que as N observações são independentes e identicamente distribuídas.

Um classificador induzido a partir do conjunto de treinamento $\mathcal{L}$, denotado por $\psi(\bullet, \mathcal{L})$, é uma função que atribui, para todo vetor de atributos $\mathbf{x} \in \chi$, uma classe de $\{1 \dots K\}$:

$$\psi(\bullet, \mathcal{L}) : \chi \rightarrow \{1 \dots K\}. \quad (2.2)$$

A Figura 1 contém a representação da construção de um classificador de aprendizado supervisionado. A partir dos objetos coletados, constrói-se o conjunto de treinamento $\mathcal{L}$, o qual é utilizado pelo algoritmo indutor para construir o classificador $\psi(\bullet, \mathcal{L})$. Esse classificador é então utilizado para predizer a classe de cada novo objeto (representado por seu vetor de atributos $\mathbf{x}$).

$$\hat{y} = \psi(\mathbf{x}, \mathcal{L}) \quad (2.3)$$

A Tabela 2 exemplifica um conjunto de treinamento referente ao comportamento de leitura de e-mails por um dado usuário. Cada linha contém um exemplo do comportamento do usuário quanto à leitura de e-mail. A primeira coluna (Autor) é um atributo que indica se o autor do e-mail é conhecido ou não pelo usuário. A segunda coluna (Assunto) indica se o e-mail é relacionado a um novo assunto ou se trata de uma resposta a mensagens anteriores. A terceira coluna (Tamanho) está relacionada ao tamanho do

Figura 1 – Representação esquemática do processo de aprendizado supervisionado

e-mail (curto ou longo). A última coluna corresponde à classe, indicando se o e-mail foi lido em casa (sim) ou lido no próprio local de trabalho (não) [1].

Tabela 2 – Exemplo de conjunto de treinamento [1]

Autor	Assunto	Tamanho	Ler em casa?
conhecido	novo	curto	sim
desconhecido	novo	longo	sim
desconhecido	antigo	curto	não
conhecido	antigo	curto	sim
conhecido	novo	longo	sim
conhecido	antigo	longo	sim
desconhecido	antigo	longo	não
desconhecido	novo	longo	sim
conhecido	antigo	curto	sim
conhecido	novo	curto	sim
desconhecido	antigo	longo	não
conhecido	novo	curto	sim
conhecido	antigo	longo	sim
conhecido	novo	longo	sim

→ Exemplo

atributos
classe

2.2 Árvores de Classificação

Uma Árvore de Classificação (ou Árvore de Decisão) é uma estrutura hierárquica que inicia no *nó raiz*, passa pelos *nós internos* e termina em *nós respostas (folhas)*. A ligação

entre os nós é feita pelos *ramos*. O antecessor de um determinado nó é chamado de *nó pai*, e os nós sucessores a um determinado nó são chamados de *nós filhos*.

Os nós internos são os nós de decisão, e são rotulados por um dos atributos do conjunto de dados. Assim, cada nó interno representa um teste sobre um atributo. Os possíveis resultados do teste dão origem aos ramos que emanam desse nó. Cada folha da árvore (nó resposta) é rotulada por uma classe.

Na Figura 2 [1], temos a representação de uma árvore de classificação, construída a partir do conjunto de dados da Tabela 2, (comportamento relacionado à leitura de e-mails de um dado usuário). O primeiro nó dessa árvore (nó raiz) testa o atributo **autor**, ou seja, se o autor do e-mail é ou não conhecido pelo usuário. Desse nó partem dois ramos correspondentes às duas resposta possíveis a esse teste. O segundo nó (nó interno) testa o atributo **assunto**, indicando se a mensagem é referente a um assunto novo ou antigo. Os nós terminais são rotulados com as duas classes possíveis, correspondentes ao local em que o usuário lê o email (**casa** ou **trabalho**).

Figura 2 – Estruturas de uma árvore de classificação [1]

As árvores de classificação têm a propriedade de serem facilmente interpretadas. Tal característica é conferida pelo fato de suas estruturas serem simples e intuitivas, além de suas regras serem organizadas de forma lógica. Utilizando como ilustração a Figura 2, caso o autor seja desconhecido e o assunto seja antigo, a recomendação dada pela árvore é de ler o email no trabalho. Onde o conjunto de regras dessa árvore pode ser descrita

como:

- Se o autor = conhecido, então ler o e-mail em casa;
- Se o autor = desconhecido e o assunto = novo, então ler o e-mail em casa;
- Se o autor = desconhecido e o assunto = antigo, então ler o e-mail no trabalho;

2.3 Construção da Árvore de Classificação

A construção da árvore de classificação passa por duas fases distintas e complementares. A primeira é a fase de expansão da árvore, sendo a técnica TDIDT (*Top-Down Induction of Decision Tree*) a mais utilizada para a indução de árvore de classificação. A segunda fase é a poda da árvore, que será apresentada na seção 2.4

O princípio básico dos algoritmos que utilizam a técnica TDIDT é particionar recursivamente o conjunto de treinamento, a partir do Nó Raiz até a Folha[8]. Normalmente os algoritmos que utilizam a técnica TDIDT para a expansão da árvore de classificação passam pelos seguintes passos:

1. Encontrar o atributo com melhor poder de discriminação do conjunto de exemplos, ou seja, selecionar o atributo que melhor divide os exemplos em suas classes;
2. Fazer a partição dos dados em dois ou mais subconjuntos disjuntos, de acordo com os valores desse atributo;
3. Para cada subconjunto de dados é feita a seguinte análise:
 - Se todos os exemplos desse subconjunto são da mesma classe, ou se um critério de parada foi atingido, ou ainda, se no **passo 1** nenhum atributo foi encontrado, então, associa-se a esse subconjunto uma Folha rotulada com a classe predominante entre os exemplos;
 - Se a condição de parada não é satisfeita, então, associa-se ao subconjunto um nó interno, e retorna-se ao passo 1.

um conjunto de treinamento $\mathcal{L}$
 um critério de parada
Entrada: uma regra de rotulação de nós folhas
 uma função de avaliação de partições de atributos, $\text{score}(\mathcal{L}, j, s)$

- 1 Denote por t o nó correspondente ao conjunto $\mathcal{L}$;
- 2 **Se** o conjunto $\mathcal{L}$ satisfaz à condição de parada,
- 3 **então** rotule t conforme a regra $\text{classe}(t)$
- 4 **caso contrário**
 - a) Para cada atributo a_j , $j = 1 \dots M$, teste todas as possíveis partições $s_1^j, s_2^j, \dots, s_q^j$ avaliando o valor da função $\text{score}(\mathcal{L}, j, s_q^j)$;
 - b) Escolha a partição e o respectivo atributo que maximizam o valor da função de score ,

$$(a_j^*, s^*) = \arg \max_{j,q} \text{score}(\mathcal{L}, j, s_q^j);$$
 - c) Rotule t com o atributo j^*
 - c) Sejam $\chi_{j^*1}, \chi_{j^*2}, \dots, \chi_{j^*Z}$ os sub-conjuntos de χ_{j^*} formados pela partição s^* do atributo a_{j^*} ;
 Crie um ramo correspondente a cada subconjunto, rotulando-o com uma condição do tipo

$$x_{j^*} \in \chi_{j^*z}, z = 1, \dots, Z;$$
 - d) particione o conjunto de treinamento $\mathcal{L}$ em Z sub-conjuntos $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_Z$ de acordo com os sub-espacos de χ_{j^*} ;
 - e) Aplique o algoritmo recursivamente para cada subconjunto $\mathcal{L}_z, z = 1, \dots, Z$.

Algoritmo 1: Construção de uma árvore T a partir de um conjunto de treinamento $\mathcal{L}$. Adaptado de Lauretto [8]

O Algoritmo 1 apresenta a construção das árvores. Inicialmente, a árvore é constituída por um único nó t sobre o qual armazenamos o conjunto de treinamento $\mathcal{L}$. Se $\mathcal{L}$ for “puro” o suficiente (isto é, se houver uma classe que domina em $\mathcal{L}$ com grau suficiente de convicção), podemos considerar o nó t como um nó terminal (rotulado com a classe dominante em $\mathcal{L}$) e interromper a expansão da árvore. Caso contrário, selecionamos o melhor atributo (em geral, o atributo que melhor discrimina as classes em $\mathcal{L}$) e particionamos $\mathcal{L}$ de acordo com os valores desse atributo. Para cada sub-conjunto de χ_{j^*} formado pela partição do atributo j^* , criamos um novo nó filho de t . Repetimos a mesma operação recursivamente para cada nó terminal obtido.

As componentes do algoritmo serão descritas brevemente a seguir. Adotamos a seguinte notação: denotamos por $n_{\bullet,t}$ a quantidade de elementos de $\mathcal{L}$ incidentes sobre o nó t e por $n_{k,t}$ a quantidade de elementos de $\mathcal{L}$ incidentes sobre o nó t e pertencentes à classe k . Denotamos também por $\pi_{k,t}$ a probabilidade (desconhecida) de um elemento $\mathbf{x} \in \chi$ incidente sobre o nó t pertencer à classe k . Um dos estimadores pontuais mais comuns para $\pi_{k,t}$ é a frequência relativa da classe k no nó t :

$$\hat{\pi}_{k,t} = \frac{n_{k,t}}{n_{\bullet,t}} \quad (2.4)$$

2.3.1 Rotulação de Nós Terminais

Nos algoritmos de indução de árvores de classificação, quando todos os exemplos incidentes em um nó terminal são da mesma classe, então o nó será rotulado com essa classe. Entretanto, quando em um nó terminal existem duas ou mais classes, normalmente a regra de rotulação da classe desse nó é baseada no critério de minimização do erro estimado de classificação, ou no critério de minimização do custo estimado do erro de classificação.

2.3.1.1 Critério de Minimização do Erro Estimado

O critério de rotulação pelo menor erro estimado de classificação consiste simplesmente em atribuir a classe predominante ao nó terminal. De fato, supondo que t seja rotulado com a classe k , o erro de classificação estimado em t , denotado por $\widehat{err}_t(k)$, corresponde à probabilidade estimada de um exemplo incidente sobre t ser de classe diferente de k :

$$\widehat{err}_t(k) = \sum_{l \neq k} \hat{\pi}_{l,t} = 1 - \hat{\pi}_{k,t}. \quad (2.5)$$

Logo, o erro mínimo de classificação no nó, aqui denotado por $r(t)$, será dado por:

$$r(t) = \min_{k=1 \dots K} \widehat{err}_t(k) = 1 - \max_{k=1 \dots K} \hat{\pi}_{k,t}, \quad (2.6)$$

e a classe que rotulará o nó será a classe majoritária:

$$k^* = \arg \min_{k=1 \dots K} \widehat{err}_t(k) = \arg \max_{k=1 \dots K} \hat{\pi}_{k,t}. \quad (2.7)$$

2.3.1.2 Critério de Minimização do Custo Estimado do Erro

O critério de minimização do custo estimado do erro de classificação, adotado por alguns algoritmos [6], tem como ideia principal que classes com importâncias distintas resultam em erros de classificação com importâncias (custos) distintas também. Tomemos o exemplo hipotético de diagnóstico médico para pacientes de acordo com a severidade de uma doença (*leve*, *média* e *grave*). Observe que os erros de classificação (diagnóstico de intensidade) têm importâncias e consequências diferentes conforme seu tipo. Por exemplo, se um paciente possui a doença com grau *leve* mas é erroneamente diagnosticado com severidade *grave*, o custo associado será aquele envolvido com exames mais complexos e específicos; entretanto, se um paciente com severidade *grave* da doença é erroneamente diagnosticado como possuindo severidade *leve*, o “custo” associado pode ser algumas ordens de grandeza superior, já que a subestimação da gravidade pode acarretar consequências mais severas, incluindo, em casos extremos, o óbito do paciente. Os custos ou penalidades dos erros de classificação podem ser representados através de uma tabela de custos, como a ilustrada na Tabela 3.

Tabela 3 – Custo de Erro de Classificação - exemplo em diagnóstico médico

		Classe Predita		
		Leve	Média	Grave
Classe Real	Leve	0	1	2
	Média	5	0	1
	Grave	10	6	0

Denotamos por $C(l, k)$ o custo de atribuir a classe k a um elemento cuja classe verdadeira seja l , sendo:

$$C(l, k) = \begin{cases} 0 & \text{se } k = l \\ \geq 0 & \text{se } k \neq l. \end{cases}$$

Supondo que t seja rotulado pela classe k , o custo esperado do erro de classificação é dado por:

$$\widehat{Cerr}_t(k) = \sum_{l \neq k} \hat{\pi}_{l,t} C(l, k). \quad (2.8)$$

Sob esse critério, $r(t)$ denota a estimativa do menor custo de erro de classificação para um exemplo incidente sobre o nó t :

$$r(t) = \min_k \widehat{Cerr}_t(k), \quad (2.9)$$

E a classe k^* escolhida é dada por:

$$k^* = \arg \min_{k=1\dots K} \widehat{Cerr}_t(k). \quad (2.10)$$

Note que o critério de erro mínimo é um caso particular do critério de custo mínimo, definindo-se

$$C(l, k) = \begin{cases} 0 & \text{se } k = l \\ 1 & \text{se } k \neq l. \end{cases}$$

Nesse caso, as equações 2.8, 2.9 e 2.10 tornam-se equivalentes às equações 2.5, 2.6 e 2.7, respectivamente.

A matriz de custo do erro de classificação é uma informação que necessita ser fornecida pelo usuário especialista. A incorporação de matrizes de erros não unitárias pode também ser utilizada como uma alternativa para o tratamento de dados desbalanceados..

O custo total estimado de erros de classificação é calculado pela média dos custos de erros nos nós terminais, ponderada pelas respectivas proporções de exemplos incidentes naqueles nós. Formalmente, denotando por $\bar{T}$ o conjunto de nós terminais de T , a estimativa do custo total de erros de classificação da árvore, denotado por $R(T)$, é dado por:

$$R(T) = \sum_{t \in \bar{T}} \frac{n_{\bullet t}}{N} r(t), \quad (2.11)$$

onde $n_{\bullet t}$ denota a quantidade de exemplos de $\mathcal{L}$ em t e N denota o tamanho de $\mathcal{L}$.

2.3.2 Seleção de Atributo

Os critérios de seleção de atributo para a melhor divisão de um nó são baseados na diminuição de impureza, sendo que a maior parte dos algoritmos de indução de árvores busca dividir o conjunto de dados de um nó-pai de forma a minimizar o grau de impureza dos nós-filhos [12]. Dado um nó t , quanto menor o grau de impureza, mais desbalanceada é a distribuição de classes nesse nó.

Em um determinado nó, a impureza é nula se todos os exemplos nele pertencerem à mesma classe: $\hat{\pi}_{kt} = 1$; e a impureza é máxima se todas as classes estão igualmente presentes no nó: $\hat{\pi}_{kt} = 1/K$, $k = 1, \dots, K$ [2].

As principais medidas de impureza são: Índice Gini, utilizado no algoritmo CART [6]; Ganho de Informação, utilizado no algoritmo ID3 [14]; e Razão de Ganho, utilizado no algoritmo C4.5 [5] e LMT [15].

2.3.2.1 Índice Gini

Esta medida foi desenvolvida e proposta por Corrado Gini em 1912. É muito empregada em análises econômicas e sociais, como por exemplo, o cálculo da desigualdade de distribuição de renda de uma população. A utilização dessa medida como critério de seleção de atributo em algoritmos de indução de árvores de classificação é descrita por Breiman [6]. Sua expressão é dada por:

$$\mathcal{G}(t) = \sum_{k=1}^K \hat{\pi}_{k,t} \cdot \sum_{k' \neq k} \hat{\pi}_{k',t} \quad (2.12)$$

$$\begin{aligned} &= \sum_{k=1}^K \hat{\pi}_{k,t} \cdot (1 - \hat{\pi}_{k,t}) \\ &= \sum_{k=1}^K \hat{\pi}_{k,t} - \sum_{k=1}^K \hat{\pi}_{k,t}^2 \\ &= 1 - \sum_{k=1}^K \hat{\pi}_{k,t}^2, \end{aligned} \quad (2.13)$$

sendo as formas das equações 2.12 e 2.13 as mais usuais.

Sob essa medida, o atributo selecionado é o que apresenta o maior decréscimo no índice Gini na divisão do nó, obtido por:

$$\Delta\mathcal{G}(t) = \mathcal{G}(t) - \sum_{z=1}^Z \frac{N_{t_z}}{N_t} \mathcal{G}(t_z) \quad (2.14)$$

onde Z é o número de nós filhos, N_t é o número total de objetos do nó pai e N_{t_z} é o número de exemplos do nó filho t_z [12].

2.3.2.2 Entropia

O conceito de entropia é fortemente aplicado na física como uma grandeza termodinâmica e é estendido para vários outros fenômenos. Em algoritmos de aprendizado supervisionado, a entropia é uma medida de impureza ou incerteza da informação. O cálculo da entropia em um nó da árvore de classificação é dado por:

$$\mathcal{E}(t) = - \sum_{k=1}^K \hat{\pi}_{k,t} \cdot \log_2[\hat{\pi}_{k,t}] \quad (2.15)$$

O atributo que será selecionado é o que apresentar a maior diferença entre a entropia do nó pai e a média ponderada das entropias dos nós filhos (diferença de entropia de

Shannon). Essa diferença é conhecida por *Ganho de Informação*, sua expressão é dada por:

$$\Delta\mathcal{E}(t) = \mathcal{E}(t) - \sum_{z=1}^Z \frac{N_{t_z}}{N_t} \mathcal{E}(t_z) \quad (2.16)$$

A figura 3 apresenta uma comparação das medidas de impureza para um problema de classificação binária, mostrando a diferença entre a medida Gini, a Entropia e o erro esperado $r(t)$ [2]. O eixo horizontal corresponde à proporção de exemplos de uma das classes (denotada por p). Nota-se que as medidas Gini e Entropia possuem formas bastante similares entre si, com diferença apenas de escala. As três medidas, independentemente da quantidade de exemplos resultantes nos nós, priorizam nós filhos cada vez mais puros, esse fato é observado nas extremidades de p (p próximo de 0 ou 1) onde as três medidas apresentam o melhor resultado. Nota-se também que, pela concavidade das curvas das medidas Gini e entropia, essas duas medidas são mais sensíveis às variações de p nas extremidades do que no centro (p próximo de 0,5), enquanto que, com a medida de erro a taxa de variação é constante.

Figura 3 – Exemplo das medidas de impureza de um nó para classificação binária [2]

2.3.2.3 Razão de Ganho de Informação

Conforme ressaltado por Quinlan [5], o critério de partição baseado no Ganho de Informação tende a dar maior preferência para atributos com muitas divisões possíveis. A medida de razão de ganho de informação foi proposta para contornar esse problema. Para o algoritmo C4.5, a Razão de Ganho é utilizada como o critério padrão de particionamento de um nó. A Razão de Ganho, representada por $GR(t)$ é conceitualmente o Ganho de

Informação ponderado:

$$\begin{aligned} SP(t) &= - \sum_{z=1}^Z \frac{N(t_z)}{N_t} \times \log_2 \left(\frac{N(t_z)}{N_t} \right) \\ GR(t) &= \frac{\Delta \mathcal{E}}{SP(t)} \end{aligned} \quad (2.17)$$

A Razão de Ganho favorece atributos com valor baixo de entropia, então, é sugerido que o cálculo da razão de ganho seja feita em duas etapas:

1. Calcular o ganho de informação para todos os atributos; e
2. Considerar apenas os atributos com ganho de informação acima da média, e dentre esses escolher o que apresentar a melhor razão de ganho.

$$\begin{aligned} \bar{g} &= \frac{\sum_{j=1}^M \Delta \mathcal{E}_j(t)}{M} \\ a_j^* &= \arg \max_{j: \Delta \mathcal{E}_j(t) \geq \bar{g}} GR(t) \end{aligned} \quad (2.18)$$

onde M denota o número de atributos no nó t , $\Delta \mathcal{E}_j(t)$ denota o maior ganho de informação obtido sobre todas as partições possíveis do atributo a_j .

2.4 Poda da Árvore

Normalmente o crescimento das árvores de classificação objetiva minimizar a taxa estimada de erro global da árvore, que tende a diminuir progressivamente à medida que mais e mais divisões são realizadas e a árvore torna-se cada vez maior. Quanto maior o porte da árvore, maior é a complexidade do modelo. O crescimento excessivo da árvore torna o modelo gerado excessivamente ajustado sobre o conjunto de treinamento, porém com baixo poder de generalização, isto é, com altas taxas de erro sobre novos casos da população. Esse fenômeno é usualmente denominado *over-fitting* [16].

A complexidade das árvores pode ser medida pela quantidade de nós gerados. A cada nova partição de um nó é feita a subdivisão do espaço de atributos, e por isso, a função de classificação representada pela árvore fica mais complexa. Além disso, normalmente a partição em nós filhos introduz novos atributos ao modelo (a menos que o mesmo atributo já tenha sido utilizado em outro nó).

Nesse contexto, métodos de poda de árvores analisam o balanço entre a diminuição do tamanho da árvore (que por sua vez é proporcional ao número de partições) e o aumento

na taxa de erro [15]. Assim, o objetivo do processo de poda é que, ao final do processo de construção, a árvore tenha um tamanho adequado, não sendo excessivamente geral (muito pequena) nem excessivamente especializada (muito grande). Esse problema tem sido abordado sob duas maneiras [8]: a primeira abordagem é a *pré-poda*, que consiste em estabelecer critérios de parada antecipada da expansão dos nós; a segunda abordagem é a *pós-poda*, que consiste em, após a indução da árvore, selecionar e eliminar alguns ramos ou sub-árvores. Alguns algoritmos, por exemplo RPART [17], combinam variações das duas abordagens na construção da árvore.

2.4.1 Pré-poda

A *pré-poda* consiste em estabelecer critérios de parada mais rígidos e assim interromper antecipadamente a expansão dos nós. Alguns critérios de parada são:

1. Se a árvore atingir a profundidade de d níveis, onde d é um parâmetro do algoritmo.
2. Se, a melhor divisão candidata do nó t apresentar Ganho de Informação menor que β (parâmetro definido pelo usuário) $\Delta\mathcal{E} < \beta$.
3. Se o número de exemplos que incidir sobre t for inferior a um parâmetro n .
4. Se a proporção dos exemplos incidentes no nó t em relação ao número total de exemplos em $\mathcal{L}$ for inferior a um parâmetro p .
5. Se a estimativa de erro (ou o custo de erro) $r(t)$ naquele nó for menor do que um parâmetro r .

A *pré-poda* é utilizada pelos sistemas C4.5 [18], LMT [15] e RPART [17].

2.4.2 Pós-poda

A *pós-poda* busca encontrar o tamanho adequado de uma árvore, sendo que após a árvore ser induzida completamente, é avaliada a confiabilidade de cada uma de suas sub-árvores, podando os ramos considerados não confiáveis. Dada uma árvore T , um nó interno $t \in T$ e as regras de poda da árvore, a *pós-poda* (ou simplesmente *poda*) do ramo T_t de T consiste em remover todos os descendentes próprios de t se o ramo T_t atender as condições da regra de poda, com isso, o nó t é declarado como nó terminal e é atribuída a classe k , conforme a regra de rotulação de nós terminais.

Normalmente os métodos de poda seguem a orientação *bottom-up* (de baixo para cima) ou a orientação *top-down* (de cima para baixo). Na orientação *bottom-up* o processo de poda é iniciado nas folhas árvore e sobe pelos ramos internos até chegar na raiz da árvore. Enquanto na orientação *top-down* o processo de poda é iniciado no nó raiz e evolui descendo na árvore até atingir os nós terminais. Para calcular a estimativa de erros e realizar as análises do processo de poda da árvore, alguns métodos utilizam um conjunto de testes $\mathcal{L}^A$, que consiste de um conjunto de instâncias cujas classes sejam conhecidas e que não tenham sido empregadas durante a construção da árvore T_0 .

Os principais métodos de pós-poda citados na literatura e descritos por Frank [19] e Esposito [20] são: Redução de Erros - *Reduced-error pruning*; Custo-Complexidade - *Cost-complexity pruning*; Erro Pessimista - *Pessimistic error pruning*; Valor Crítico - *Critical value pruning*; Erro Mínimo - *Minimum-error pruning*; e Poda por Estimativa de Erro - *Error-based pruning*.

2.4.2.1 Redução de Erros - *Reduced-error pruning*

Segundo Quinlan [21], a poda por Redução de Erros é um método simples e rápido. Esse método segue a orientação *bottom-up* e necessita de um conjunto de teste $\mathcal{L}^A$ específico para processo de poda.

Como regra para a poda esse método substituirá o ramo T_t por um nó terminal, caso essa substituição não incremente a taxa de erro estimada sobre o conjunto de teste.

2.4.2.2 Custo-Complexidade - *Cost-complexity pruning*

Este método, proposto por Breiman [6] e implementado no algoritmo CART, é constituído de dois estágios. No primeiro estágio, uma sequência de árvores $T_0, T_1, \dots, T_Z$ é gerada, onde: T_0 é a árvore original; T_Z é uma árvore constituída apenas por uma folha (a raiz da árvore original); e T_{z+1} é obtida pela substituição de uma ou mais sub-árvores de T_z por folhas. No segundo estágio, é selecionada a melhor árvore dessa sequência, levando-se em consideração o custo estimado dos erros de classificação e a complexidade (medida em número de folhas) de cada uma dessas árvores.

Para apresentar o primeiro estágio do método de poda por Custo-Complexidade, vamos considerar as seguintes definições:

1. Dada a sub-árvore T_1 defina sua complexidade como $|\overline{T}_1|$, o número de folhas em T_1 .

2. A relação de comparação de complexidade entre duas sub-árvores é representada $\prec$, $\preceq$, $\succ$ e $\succeq$, sendo que, por exemplo: $T_1 \succ T_2$ denota que T_1 é uma sub-árvore mais complexa que T_2 .
3. Para simplificar a formalização vamos considerar T_0 como a árvore de tamanho máximo.

A ideia principal desse primeiro estágio é a que segue - Seja $\alpha > 0$ um número real denominado o *parâmetro de complexidade* e defina a *medida de custo-complexidade* $R_\alpha(T)$ como:

$$R_\alpha(T) = R(T) + \alpha|\overline{T}|.$$

$R_\alpha(T)$ é uma combinação linear entre o custo de erro da árvore e sua complexidade. O problema central do método é encontrar, para cada valor de α , a sub-árvore $T(\alpha) \preceq T_0$ que minimiza $R_\alpha(T)$, isto é,

$$T(\alpha) = \arg \min_{T \preceq T_0} R_\alpha(T).$$

O parâmetro α pode ser visto como um custo por folha; assim, se α for pequeno, a penalização por haver muitas folhas será pequena e $T(\alpha)$ será grande. À medida que a penalidade α por folha aumenta, a sub-árvore $T(\alpha)$ passa a ter um número menor de nós terminais até que, para um valor suficientemente grande de α , $T(\alpha)$ consistirá apenas do nó raiz e a árvore T_0 terá sido completamente podada.

O segundo estágio do método de poda por Custo-Complexidade é a escolha da melhor sub-árvore. Uma vez obtida a sequência decrescente de complexidade de sub-árvores $T_1 \succ T_2 \succ \dots \succ T_Z \equiv \{t_1\}$, onde t_1 denota o nó raiz, o critério para essa decisão (escolha da melhor sub-árvore) é baseado na precisão de classificação e na complexidade de cada sub-árvore.

Inicialmente, deve-se encontrar uma boa estimativa de erro para cada uma das árvores. Para encontrar essa estimativa, não podemos simplesmente utilizar os mesmos exemplos que haviam sido empregados para a construção da árvore, sob pena de tal estimativa de erro ser demasiadamente otimista. Portanto, assim como o método de Redução de Erros, este método também necessita de um conjunto de teste $\mathcal{L}^A$ específico para processo de poda.

Seja T_z uma sub-árvore da sequência, e $Q(l|k)$ a probabilidade estimada de um objeto de classe k ser classificado por T_z como classe l . Denote por $R(l)$ o custo esperado no erro

de classificação dos objetos de classe k . $R(k)$ será dado por:

$$R(k) = \sum_{l=1}^K C(l, k)Q(k|l)$$

onde $C(l, k)$ é o custo de erro.

Finalmente, seja $\hat{\pi}_k$ a probabilidade *a priori* de um objeto qualquer de $\mathcal{L}^A$ ser de classe k . A estimativa do custo da árvore T_z é dada por:

$$R^C(T_z) = \sum_{k=1}^K R(k)\hat{\pi}_k.$$

Depois de calculada a estimativa de custo $R^C(T_z)$ para cada sub-árvore T_z da sequência, pode-se simplesmente escolher a sub-árvore

$$T_{z1} = \arg \min_{1 \leq z \leq Z} R^C(T_z).$$

2.4.2.3 Erro Pessimista - *Pessimistic error pruning*

Este método de poda analisa a árvore de cima para baixo (top-bottom), sendo que uma sub-árvore pode ser podada sem que seus nós descendentes sejam analisados. As estimativas de erro são obtidas a partir do conjunto de treinamento (assumindo-se uma distribuição binomial para os erros), portanto, não é necessário um conjunto de dados específico para o processo de poda [19].

Sejam: T_i uma sub-árvore de T que contem $|\bar{T}_i|$ folhas; $n_{\bullet T_i}$ a quantidade de exemplos do conjunto de treinamento incidentes na sub-árvore T_i ; e n_{eT_i} o número de exemplos classificados erroneamente por todas as folhas da sub-árvore T_i . A estimativa pessimista de erro de classificação para a sub-árvore T_i é dada por: [21]

$$E' = n_{eT_i} + \frac{F_{T_i}}{2}.$$

Supondo que a sub-árvore T_i fosse transformada em uma folha, a quantidade de exemplos do conjunto de treinamento classificados erroneamente por essa folha seria dado por D . O método de poda por Erro Pessimista irá substituir T_i por uma folha se:

$$D + 1/2 < E' + S_{E'}$$

sendo $S_{E'}$ o erro padrão, estimado por:

$$S_{E'} = \sqrt{\frac{E' \times (n_{\bullet T_i} - E')}{n_{\bullet T_i}}}$$

2.4.2.4 Valor Crítico - *Critical value pruning*

Valor Crítico é uma técnica que segue a orientação “bottom-up” para o processo de poda de forma semelhante à técnica de Redução de Erros. Entretanto, existe uma diferença fundamental na forma da poda da árvore: enquanto a técnica de Redução de Erros usa a estimativa de erro para avaliar a qualidade da sub-árvore para a sua poda, a técnica de Valor Crítico utiliza as informações coletadas durante a construção da árvore. Os algoritmos da família TDIDT empregam algum critério para fazer divisão do conjunto de treinamento, com o objetivo de incrementar a pureza nos conjuntos de dados menores. Portanto, a cada nó, o conjunto de treinamento é dividido de acordo com esse critério para maximizar esse valor, por exemplo, o Ganho de Informação.

Quando uma sub-árvore é analisada para a poda, o valor do critério de divisão é comparado com um “threshold” (valor fixo durante todo o processo de poda). Se para o nó correspondente a sub-árvore o valor do critério de divisão for menor que o valor do “threshold”, essa sub-árvore é transformada em folha. Entretanto, uma validação adicional é realizada: se a sub-árvore contém ao menos um nó cujo valor do critério de divisão é maior que o valor do “threshold”, essa sub-árvore não será podada. Isto significa que uma sub-árvore somente é podada se o valor do critério de divisão de todos os seus nós sucessores forem menores que o valor do “threshold”.

A técnica do Valor Crítico depende do valor do “threshold”: quanto maior esse valor, mais agressivo será o processo de poda. O melhor valor para o “threshold” pode ser obtido executando processo de validação cruzada do tipo *hold-out* ou *v-fold*.

2.4.2.5 Erro Mínimo - *Minimum-error pruning*

O Erro Mínimo é uma técnica que segue a orientação “bottom-up” para o processo de poda, onde uma sub-árvore é substituída por um nó terminal (folha), se a estimativa de erro apurada da sub-árvore candidata for menor que a soma das estimativas de erro das folhas contidas nessa sub-árvore. Analogamente ao Erro Pessimista, a estimativa de erro apurada da sub-árvore é obtida a partir do conjunto de treinamento, portanto, para esta técnica, não é necessário um conjunto dados específico para o processo de poda.

A estimativa de erro apurada da sub-árvore combina a probabilidade a priori das instâncias do conjunto de treinamento incidentes no nó com um coeficiente de severidade da poda. Para determinar o valor desse coeficiente, um processo de validação cruzada do tipo *v-fold* ou *leave-one-out* pode ser utilizado.

2.4.2.6 Poda por Estimativa de Erro - *Error-based pruning*

A Poda por Estimativa de Erro é uma técnica implementada pelo algoritmo C4.5 do Quinlan. Esta técnica segue a orientação “bottom-up”, onde uma sub-árvore é substituída por um nó terminal (folha) se a estimativa de erro da sub-árvore candidata for menor que a soma das estimativas de erro das folhas contidas nessa sub-árvore.

Assim como a poda por Erro Pessimista, esta também deriva da estimativa de erro obtida a partir do conjunto de treinamento, assumindo que as estimativas de erros tendem a uma distribuição binomial. Entretanto, esta técnica calcula um intervalo de confiança das contagens de erro baseado na aproximação da distribuição binomial pela distribuição normal, para conjuntos com muitos exemplos. Assim, o limite superior do intervalo de confiança é usualmente estimado com a taxa de erro das folhas. No algoritmo C4.5 o intervalo de confiança padrão é de 25%.

Além da Poda por Estimativa de Erro, o algoritmo C4.5 utiliza também outra técnica de poda chamada “*Subtree Raising*”, onde um nó interno é substituído pelo nó interno descendente (filho) mais populoso, desde que a substituição não provoque incremento na estimativa da taxa de erro. Tomemos como exemplo a árvore da Figura 4(a) - árvore antes do processo de poda, e a Figura 4(b) - árvore após o processo de poda: a sub-árvore C descendente de B substituiu a sub-árvore B. Note que os filhos de B e C são folhas, mas poderiam ser sub-árvores. Observar que realizando essa poda é necessário reclassificar os exemplos dos nós 4 e 5 para a nova sub-árvore C, este é o motivo pelo qual as folhas são marcadas como 1', 2' e 3' [3].

2.5 Problemas com Dados Desbalanceados

Alguns conjuntos de dados apresentam naturalmente alta prevalência de uma ou mais classes, e muitas vezes, a classe minoritária representa o objeto de interesse no tratamento do conjunto de dados, como por exemplo: detecção de operações fraudulentas em cartões de crédito; diagnóstico de doenças raras; etc.

Figura 4 – Exemplo de Poda “Subtree Raising” [3]

Tomemos como exemplo um conjunto de teste hipotético com duas classes, contendo 90 exemplos de classe 1 e 10 elementos de classe 2. Suponha que esse conjunto de teste seja classificado por dois classificadores ψ_1 e ψ_2 , cujas taxas de acerto nas classes sejam aquelas apresentadas na Tabela 4: ψ_1 classifica corretamente 89 exemplos de classe 1 e 6 exemplos de classe 2, enquanto ψ_2 classifica corretamente 85 exemplos de classe 1 e 9 exemplos de classe 2. Note que o classificador ψ_1 apresenta uma acurácia global um pouco melhor que a do classificador ψ_2 , 95% e 94% respectivamente, porém o classificador ψ_1 apresenta uma taxa de acerto na classe 2 (60%) consideravelmente menor do que a taxa de acerto do classificador ψ_2 naquela classe (células em destaque). Se a classe 2 representasse um evento severo (por exemplo, uma doença grave), o classificador ψ_2 seria preferível em relação a ψ_1 , mesmo obtendo resultados inferiores tanto na taxa de acerto da classe 1 como na acurácia global.

Tabela 4 – Exemplo hipotético das taxas de acerto de dois classificadores ψ_1 e ψ_2 sobre um conjunto de dados com desbalanceamento entre as classes

Classe	N	ψ_1		ψ_2	
		Classif Correta	(%) Acerto	Classif Correta	(%) Acerto
1	90	89	98.9	85	94.4
2	10	6	60.0	9	90.0
Total	100	95	95.0	94	94.0

Para minimizar a taxa de erro de classificação de classes minoritárias, uma das alternativas encontradas em alguns algoritmos é a atribuição de custos diferente para os erros por classe, onde para as classes minoritárias são atribuídos os maiores custos e para as classes majoritárias os menores custo[22]. O problema dessa alternativa é que essa atribuição normalmente não é automática, o que introduz maior complexidade na utilização

do algoritmo para usuários menos experientes.

Ainda na busca por minimizar a taxa de erro de classificação de classes minoritárias, diversos trabalhos têm apresentado técnicas para o balanceamento das classes do conjunto de dados, sendo que o objetivo principal delas é equilibrar artificialmente as classes do conjunto de dados de treinamento. Frequentemente, duas abordagens têm sido utilizadas: a primeira abordagem é a eliminação de exemplos da classe majoritária, sendo que o procedimento é identificar e eliminar os exemplos da classe majoritária que são menos relevantes para a indução do classificador; e a segunda abordagem é replicação de exemplos da classe minoritária [23]. Na procura por melhores resultados, alguns trabalhos combinam as duas abordagens [7].

Capítulo 3

Desempenho de Classificadores

Neste capítulo apresentaremos alguns métodos e métricas para avaliação de desempenho de classificadores. Em particular, apresentaremos a validação cruzada, as matrizes de confusão e as medidas mais usuais de desempenho.

3.1 Validação Cruzada

A validação cruzada é um método muito utilizado para estimar a precisão de um classificador induzido por um algoritmo de aprendizagem supervisionada.

As várias técnicas de validação cruzada têm em comum o conceito de particionar o conjunto de dados em subconjuntos mutuamente exclusivos, e posteriormente, utilizar alguns destes subconjuntos para a indução do classificador (conjunto de treinamento) e o restante dos subconjuntos (conjunto de teste) para a validação do classificador. As técnicas de validação cruzada mais utilizadas são: *v-fold*, *holdout*, e *leave-one-out* [24].

V-Fold - Esta técnica divide o conjunto de treinamento em V subconjuntos, sendo que o processo de validação do algoritmo é repetido V vezes, como é exemplificado na Figura 5. Em cada ciclo de validação, $V - 1$ subconjuntos são utilizados para a indução do classificador, que será testado com o subconjunto não utilizado na indução do classificador. Ao final de todos os ciclos o erro médio é calculado. Dessa forma, cada exemplo do conjunto de treinamento é utilizado apenas uma vez para teste e chega a ser utilizado $V - 1$ vezes para a indução de classificador, sendo que a variância da estimativa de erro é diminuída à medida que o V é aumentado. Quanto maior o valor para V , maior é o tempo computacional para processamento, muitos testes utilizam $V = 10$. Entretanto, para realização do teste t-pareado (que será detalhado mais adiante), Mitchell [25] recomenda calibrar V de forma que cada subconjunto contenha 30 ou mais exemplos, a fim de se obter estimativas mais confiáveis de erros. A técnica V-Fold com amostragem estratificada objetiva manter em todos os V subconjuntos a mesma proporção entre as classes do conjunto original.

Figura 5 – Validação Cruzada - V-fold

Leave-One-Out - Esta técnica é um caso particular do v-fold, onde V é igual ao número de exemplos do conjunto de dados, ou seja, $V = N$. Dessa forma, cada exemplo é utilizado uma vez para teste e $N - 1$ vezes para a indução de classificadores. Como o processamento será realizado em N ciclos o custo computacional é alto, assim esta técnica é mais indicada em situações no qual a quantidade de exemplos do conjunto de dados é pequena.

Holdout - Esta é a técnica mais simples de Validação Cruzada. Nela o conjunto de dados é dividido em dois subconjuntos mutuamente exclusivos, sendo que um é utilizado para a indução do classificador e o outro para a validação. Normalmente a divisão é feita na seguinte proporção: 2/3 dos dados para o subconjunto de treinamento (indução) e 1/3 para o subconjunto de teste (validação). A técnica Holdout com amostragem estratificada objetiva manter nos dois subconjuntos a mesma proporção entre as classes do conjunto original. Esta técnica é recomendada para conjunto de dados com grande quantidade de exemplos.

3.2 Matriz de Confusão

Depois de construído o classificador, sua aplicação sobre o conjunto de testes resulta em uma matriz de confusão, construída a partir da contagem (frequência absoluta) dos exemplos utilizados no teste do classificador. Em problemas de classificação binária, são

comuns as seguintes medidas de interesse:

- VP (Verdadeiros Positivos): quantidade de exemplos positivos classificados corretamente;
- FN (Falsos Negativos): quantidade de exemplos positivos classificados erroneamente como negativos;
- FP (Falsos Positivos): quantidade de exemplos negativos classificados erroneamente como positivos; e
- VN (Verdadeiros Negativos): quantidade de exemplos negativos classificados corretamente.

Na Tabela 5 é apresentada uma das formas de apresentação da matriz de confusão.

Tabela 5 – Matriz de Confusão

Classe Real	Classe Predita		
	VP Verdadeiro Positivo	FN Falso Negativo	POS Positivo Total (Real)
	FP Falso Positivo	VN Verdadeiro Negativo	NEG Negativo Total (Real)
	PP Positivo Total (Predito)	PN Negativo Total (Predito)	

A matriz de confusão ainda apresenta outras informações a partir dessas contagens:

POS - É a quantidade real de exemplos positivos, ou seja, a quantidade de verdadeiros positivos somados aos falsos negativos: $POS = VP + FN$;

NEG - É a quantidade real de exemplos negativos, ou seja, a quantidade de falsos positivos somados aos verdadeiros negativos: $NEG = FP + VN$;

PP - É a quantidade predita de exemplos positivos, ou seja, a quantidade de verdadeiros positivos somados aos falsos positivos: $PP = VP + FP$; e

PN - É a quantidade predita de exemplos negativos, ou seja, a quantidade de verdadeiros negativos somados aos falsos negativos: $PN = VN + FN$.

3.3 Medidas de Desempenho

As medidas de desempenho têm como objetivo: estimar a precisão do classificador e orientar a escolha de um classificador para o tipo de problema e condição testada [26].

3.3.1 Taxas de Erros e Acertos

A partir das contagens da Matriz de Confusão, são obtidas as taxas de erros e acertos do classificador. Naturalmente busca-se no classificador que ele tenha as menores taxas de erro e consequentemente as maiores taxas de acerto.

Taxa de verdadeiros positivos: $VP_r = VP/POS$ é a proporção de exemplos classificados corretamente como positivos em relação ao total real de casos positivos. Esta taxa também é conhecida como **Sensibilidade** ou **Recall**, que mede a capacidade do classificador atribuir corretamente a classe positiva. Quando $VP_r = 1$ indica que todos os exemplos positivos foram classificados corretamente (observe que podem ocorrer exemplos negativos classificados como positivos);

Taxa de falsos negativos: $FN_r = FN/POS$ é a proporção de exemplos classificados erroneamente como negativos em relação ao total real de casos positivos. Esta taxa é complementar a taxa de verdadeiros positivos, ou seja $FN_r = 1 - VP_r$;

Taxa de verdadeiros negativos: $VN_r = VN/NEG$ é a proporção de exemplos classificados corretamente como negativos em relação ao total real de casos negativos. Esta taxa também é conhecida como **Especificidade**, que mede a capacidade do classificador atribuir corretamente a classe negativa. Quando $VN_r = 1$ indica que todos os exemplos negativos foram classificados corretamente (observe que podem ocorrer exemplos positivos classificados como negativos);

Taxa de falsos positivos: $FP_r = FP/NEG$ é a proporção de exemplos classificados erroneamente como positivos em relação ao total real de casos negativos. Essa taxa é complementar a taxa de verdadeiros negativos, ou seja $FP_r = 1 - VN_r$;

Taxa de erro total: $ET_r = (FN + FP)/(NEG + POS)$ é a proporção de classificações errôneas em relação ao total de exemplos. A **Acurácia Global** de um classificador é medida pelo complemento desta taxa ($1 - ET_r$);

Taxa de precisão: $PP_r = VP/PP$ é a proporção de exemplos positivos classificações corretamente em relação ao total de classificações positivas ($VP + FP$), ou seja, essa taxa corresponde à probabilidade estimada de um exemplo ser de classe positiva, dado que foi classificado como positivo. Observe que mesmo com valores altos para essa taxa (próximo de 1), não necessariamente o algoritmo está classificando bem os exemplos da classe minoritária, pois ele pode ter classificado vários exemplos positivos como negativos, e nesse caso, esse exemplos não estão retratados neste

índice. Uma baixa taxa de precisão pode estar relacionada com baixos valores nas contagens de casos de VP (Verdadeiro Positivo) e/ou altos valores nas contagens de casos de FP (Falso Positivo).

A taxa de erro total (ET_r) é uma medida bastante utilizada na avaliação e comparação de classificadores porém, quando o problema envolve conjunto de dados com grande desbalanceamento entre as classes, somente essa média pode não ser suficiente. Para exemplificar essa afirmação vamos retomar o exemplo apresentado na Tabela 4 da Seção 2.5, neste exemplo estamos considerando a classe =1 como exemplos “negativos” e a classe = 2 como exemplos “positivos”. Esse conjunto de teste apresenta o desbalanceamento entre as classes de: 90% dos casos de classe = 1 versus 10% de classe = 2. Tomemos agora os resultados obtidos pelo classificador ψ_1 , onde foram classificados corretamente 89 dos 90 casos de classe = 1, e para a classe = 2 foram classificados corretamente 6 dos 10 casos existentes, com esses resultados podemos construir a matriz de confusão desse exemplo, representada pela Tabela 6, onde os casos da classe = 1 foram considerados com negativos e os casos da classe = 2 foram considerados como positivos.

Tabela 6 – Matriz de Confusão do exemplo hipotético do classificador ψ_1 apresentado na seção 2.5

Classe Real	Classe Predita		
	VP = 6	FN = 4	
	FP = 1	VN = 89	
	PP = 7	PN = 93	

As taxas de acertos e erros referente ao exemplo da Tabela 6 são:

- Taxa de verdadeiros positivos: $VP_r = 0,6$ ($VP_r = 06/10$)
- Taxa de falsos negativos: $FN_r = 0,4$ ($FN_r = 04/10$)
- Taxa de verdadeiros negativos: $VN_r = 0,989$ ($VN_r = 89/90$)
- Taxa de falsos positivos: $FP_r = 0,011$ ($FP_r = 01/90$)
- Taxa de erro total: $ET_r = 0,95$ ($ET_r = (04 + 01)/(10 + 90)$)
- Taxa de precisão: $PP_r = 0,857$ ($PP_r = 06/07$)

A taxa de falso positivo apresentada é de 1,1%. Tomando como referência somente esse resultado, podemos dizer que esse é um bom classificador. Porém temos que observar

que existe uma diferença significativa entre as taxas de falso positivo e de falso negativo (40%). Para esclarecer a importância da análise dessa diferença, vamos tomar como exemplo a classificação de diagnóstico médico com o objetivo de identificar a existência ou não de uma doença grave. A existência da doença ocorre em alguns poucos casos, ou seja, ela é a classe minoritária. Conforme discutido na seção 2.3.1, usualmente o indício (ou diagnóstico) da existência da doença implica na necessidade de exames adicionais para o completo diagnóstico e determinação do tratamento. Sob esse aspecto, o diagnóstico falso negativo (afirmar que um paciente não tem a doença quando na realidade ele é portador da mesma) é mais grave do que um diagnóstico de falso positivo (dizer que o paciente tem a doença quando na realidade ele não tem), pois o falso negativo pode interromper a investigação prematuramente. Fazendo analogia do diagnóstico médico com a matriz de confusão da Tabela 5, os casos positivos seriam aqueles com presença da doença; nesse caso, o classificador apresentaria uma taxa de falso negativo muito elevada, indicando que 40% dos pacientes com a doença seriam diagnosticados como não portadores. Portanto, para esse tipo de situação, classificadores com baixo desempenho na classificação da classe positiva têm baixa efetividade.

3.3.2 Indicadores

O exemplo apresentado na subseção anterior ilustra o fato de que, frequentemente, a análise isolada das taxas apresentadas acima não é suficiente, especialmente na presença de dados desbalanceados. Algumas medidas, como as apresentadas abaixo, combinam mais de uma das taxas previamente apresentadas em um mesmo indicador.

F-score : (ou F-measure) é um indicador que combina a taxa de precisão e a taxa de sensibilidade [27, 28]. O resultado desse indicador está no intervalo $[0,1]$, sendo o melhor resultado é o F-score = 1 e o pior resultado é o F-score = 0. O F-score para β real positivo é:

$$F_{\beta} = (1 + \beta^2) * \frac{(\text{precisão} * \text{sensibilidade})}{(\beta^2 * \text{precisão}) + \text{sensibilidade}} \quad (3.1)$$

O F-score é equilibrado quando $\beta = 1$. Quando $\beta < 1$ a ênfase é dada para a taxa de precisão e quando $\beta > 1$ a ênfase é dada para a sensibilidade. Os dois valores para β bastante utilizados são 0.5 e 2.

AUC (*Area Under Curve*) ou "Área abaixo da Curva ROC". Dadas diversas e diferentes interações em um algoritmos, plotar as taxas VP_r e FP_r resultantes irá fornecer a

curva ROC que, segundo Prati [4] é uma ferramenta de avaliação de desempenho de algoritmos. A área situada abaixo dessa curva (**AUC**) é um indicador usualmente adotado como medida de qualidade do classificador [7]. Powers [29] e Sokolova [28] apresentam a seguinte simplificação no cálculo desse indicador:

$$AUC = \frac{\text{sensibilidade} + \text{especificidade}}{2} \quad (3.2)$$

Outro indicador encontrado na literatura atual é o [EIG - Erro médio Intra-Grupo] (Mean Within Group Error Rate): é apresentado por Qiao [30] e Batista [7], em uma formulação geral incluindo multi-classes. Para problemas com classes raras, esse indicador tende a ser mais apropriado do que as medidas usuais de erro global de classificação. Para problemas de classificação binária, o EIG é dado por $EIG = (FN_r + FP_r)/2$, sendo portanto o complementar do AUC: $EIG = 1 - AUC$.

Além dos indicadores acima, algumas ferramentas são úteis na avaliação e seleção de classificadores, por exemplo: o gráfico ROC, e o teste t-pareado. O gráfico ROC possibilita a visualização do resultado de todos os classificadores avaliados no mesmo gráfico. O teste t-pareado permite a comparação entre pares de classificadores, fornecendo medidas de significância para as diferenças entre os resultados observados na validação cruzada.

3.4 Análise Gráfica - Gráfico ROC

ROC (*Receiver Operating Characteristic*) é uma ferramenta gráfica bastante útil na avaliação e seleção de classificadores, por permitir estudar a relação entre as medidas de sensibilidade e especificidade.

Conceitualmente o gráfico ROC é simples, mas muito poderoso na análise de classificadores construídos para problemas onde existe desbalanceamento de classes no conjunto de dados ou para problemas em que devem-se tratar diferentes custos para os erros de classificação [4].

O gráfico é construído plotando-se o percentual correspondente ao FP_r (ou $1 - \text{Especificidade}$) no eixo das ordenadas e o VP_r (ou Sensibilidade) no eixo das abscissas. A Figura 6 apresenta a estrutura geral do gráfico ROC. As quatro coordenadas extremas do gráfico (vértices do quadrado) têm as seguintes interpretações [4]:

coordenada (0,100) - neste ponto temos 0% para a taxa de Falso Positivo e 100% para a taxa de Verdadeiro Positivo. Este é o ponto em que todos os exemplos de

classe positiva e de classe negativa são classificados corretamente. Classificadores que resultam em pontos próximos à esta coordenada são os ideais;

coordenada (100,0) - neste ponto temos 100% para a taxa de Falso Positivo e 0% para a taxa de Verdadeiro Positivo. Ao contrário do anterior, este ponto representa os casos em que todos os exemplos de classe positiva e de classe negativa são classificados erroneamente;

coordenada (0,0) - neste ponto temos 0% tanto para a taxa de Falso Positivo como para a taxa de Verdadeiro Positivo. Classificadores expressos no gráfico em região próxima a esse ponto tendem fortemente a classificar os exemplos como negativos;

coordenada (100,100) - neste ponto temos 100% tanto para a taxa de Falso Positivo como para a taxa de Verdadeiro Positivo. Ao contrário do anterior, classificadores expressos no gráfico em região próxima à esse ponto tendem fortemente a classificar os exemplos como positivos.

O espaço ROC possibilita a análise visual do desempenho de cada algoritmo em relação às classes, permitindo identificar quais algoritmos classificam melhor a classe positiva e quais classificam melhor a classe negativa. A diagonal descendente (0,100 - 100,0) da figura 6 representa classificadores que têm desempenho igual para ambas as classes. Essa diagonal é dada por: $VP_r = 1 - FP_r = VN_r$. Os classificadores expressos por pontos do gráfico localizados acima dessa diagonal, tendem a favorecer a classe positiva em detrimento da negativa. [4]

Figura 6 – Espaço ROC para análise do desempenho entre as classes

A Figura 7 ilustra os resultados de três classificadores hipotéticos A, B e C. Nela são comparadas as diferenças de desempenho dos classificadores de acordo com o custo de erro ou diferenças na distribuição das classes do conjunto de dados.

A linha vermelha é chamada de linha de isodesempenho; nela todos os pontos têm uma característica em comum: a taxa de erro (ou custo médio esperado) é a mesma. O coeficiente angular dessa linha está relacionado a quanto um determinado erro é relativamente mais importante que o outro. A linha de isodesempenho pode representar, por exemplo, a proporção de exemplos entre as classes ou o custo de classificar erroneamente um exemplo positivo ou negativo [4].

Para comparação entre classificadores, as linhas de isodesempenho podem ser utilizadas da seguinte forma: fixado o coeficiente angular β , plotam-se todas as linhas de isodesempenho com coeficiente β passando sobre os pontos correspondentes aos classificadores. Escolhe-se então o classificador cuja linha de isodesempenho é a mais próxima da coordenada ideal (0, 100).

Na Figura 7(a) o coeficiente angular estabelecido é igual a 1, indicando que as proporções de classe positiva e negativa são iguais ou, analisando pelo custo de erros, que os custos de erro de classificação para a classe positiva e negativa são os mesmos. Nesta condição, o classificador B é o que apresenta o melhor desempenho, já que a linha de isodesempenho passando pelo seu ponto é a mais próxima da coordenada (0,100).

Na Figura 7(b) o coeficiente angular é igual a 0,5, o que pode indicar que a classe negativa é duas vezes mais populosa que a classe positiva, ou que o custo por classificar errado um exemplo de classe positiva é o dobro em relação ao custo do erro para exemplo de classe negativa. Nesta condição o classificador C passou a ser o que apresenta o melhor desempenho.

Portanto, a linha de isodesempenho é uma técnica que auxilia a avaliação e comparação do desempenho dos algoritmos analisados, permitindo estabelecer o critério de “peso” após a obtenção dos resultados dos testes.

3.5 Equivalência Estatística - Teste t Pareado

As medidas de desempenho até aqui apresentadas são estimativas muito úteis na avaliação dos algoritmos; seus resultados auxiliam as análises dos pontos fortes e fracos de cada algoritmo. Porém, os resultados obtidos são através de conjuntos de testes ex-

Figura 7 – Análise da linha de isodesempenho [4]

traídos do conjunto de dados original, e algumas vezes, a quantidade de amostras não é muito grande. Então, é possível que, quando o classificador induzido pelo algoritmo for submetido a casos reais, o desempenho do algoritmo apresente alguma variação.

O intervalo de confiança é uma estatística utilizada para indicar a confiabilidade de uma estimativa. Dessa forma, podemos deixar de ter um resultado estimado com um valor absoluto e passar a tê-lo como um intervalo de valores, de acordo com o grau de confiança estipulado.

Portanto, ao comparar desempenhos entre algoritmos distintos, levando em consideração o intervalo de confiança, podemos encontrar casos em que dois algoritmos apresentem resultados estatisticamente equivalentes, a despeito de eventuais diferenças entre as estimativas pontuais de desempenho obtidas nos testes.

Denotemos por ψ um algoritmo de classificação, por $\psi(\bullet, \mathcal{L}_{tr})$ um classificador induzido por ψ sobre um conjunto de treinamento $\mathcal{L}_{tr}$, e por $h(\psi(\mathcal{L}_{ts}, \mathcal{L}_{tr}))$ uma medida de interesse (acurácia, erro, etc) do classificador $\psi(\bullet, \mathcal{L}_{tr})$ quando aplicado sobre o conjunto de teste $\mathcal{L}_{ts}$ ($\mathcal{L}_{tr}, \mathcal{L}_{ts} \in \chi \times \{1 \dots K\}$).

Dados dois algoritmos distintos ψ_A e ψ_B , a variável aleatória de interesse é a diferença entre os desempenhos medidos para os dois classificadores:

$$\delta = h(\psi_A(\mathcal{L}_{ts}, \mathcal{L}_{tr}) - h(\psi_B(\mathcal{L}_{ts}, \mathcal{L}_{tr})), (\mathcal{L}_{ts}, \mathcal{L}_{tr}) \in (\chi \times \{1 \dots K\})^2.$$

Denotamos por $\mu_\delta = E_{(\chi \times \{1 \dots K\})^2}(\delta)$, ou seja, o valor esperado de δ sobre todos os pares

de conjuntos de treinamento e de teste $(\mathcal{L}_{ts}, \mathcal{L}_{tr}) \in (\chi \times \{1 \dots K\})^2$. Se os algoritmos ψ_A e ψ_B tiverem o mesmo desempenho sob a medida h , então teremos $\mu_\delta = 0$. Como o valor do parâmetro μ_δ é desconhecido, um procedimento possível é construir um intervalo dentro do qual espera-se que esse valor esteja presente, com um certo nível de confiança.

Um *intervalo de confiança* de $1 - \alpha$ para o parâmetro μ_δ é um intervalo $[a, b]$ obtido a partir de uma amostra tal que, com probabilidade $1 - \alpha$, deve incluir o valor de μ_δ [25]. Nesse caso, dizemos que θ pertence ao intervalo $[a, b]$ com *confiança* $1 - \alpha$. Valores comuns para o nível de confiança são 0.90, 0.95 e 0.99 ($\alpha = 0.1, 0.05$ e 0.01 , respectivamente).

A construção do intervalo de confiança para μ_δ envolve dois passos. O primeiro passo é obter a média da medida de desempenho dos dois algoritmos, sendo que a medida de desempenho é uma taxa de erro ou acerto ou um indicador de performance, conforme descrito nas seções 3.3.1 e 3.3.2. O segundo passo é calcular a aproximação estatística dos dois resultados pelo intervalo de confiança estipulado.

A média da medida de desempenho é obtida através do processo de validação cruzada, sendo que a cada interação os dois algoritmos são treinados com o mesmo conjunto de dados e os classificadores, gerados nessa interação, são avaliados com o mesmo conjunto de teste, conforme detalhado no Algoritmo 2.

Entrada: $\mathcal{L}$: Conjunto de Dados Original

V : número de ciclos da Validação Cruzada

- 1 Particionar $\mathcal{L}$ em V subconjuntos $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_V$ independentes e de tamanho igual;
- 2 **for** v from 1 to V **do**
- 3 Usar $\mathcal{L}_v$ como conjunto de teste;
- 4 Usar demais dados como conjunto de treinamento: $\mathcal{L}_v^c \leftarrow \mathcal{L} - \mathcal{L}_v$;
- 5 Construir os classificadores $\psi_A(\bullet, \mathcal{L}_v^c)$ e $\psi_B(\bullet, \mathcal{L}_v^c)$ a partir de $\mathcal{L}_v^c$;
- 6 Calcular as medidas de desempenho sobre $\mathcal{L}_v$:
 $h_{A,v} = h(\psi_A(\mathcal{L}_v, \mathcal{L}_v^c))$; $h_{B,v} = h(\psi_B(\mathcal{L}_v, \mathcal{L}_v^c))$;
- 7 Calcular a diferença entre as medidas: $\delta_v \leftarrow h_{A,v} - h_{B,v}$;
- 8 **end**
- 9 Retornar a média da diferença das medidas desempenho $\bar{\delta}$, dado por:

$$\bar{\delta} \equiv \frac{1}{V} \sum_{v=1}^V \delta_v$$

Algoritmo 2: Diferença média das medidas desempenho dos algoritmos

A aproximação do intervalo de confiança pela estimativa de quantidade é dada por:

$$\mu = \bar{\delta} \pm \tau_{1-\alpha/2, V-1} * S_{\bar{\delta}} \quad (3.3)$$

onde

$$S_{\bar{\delta}} \equiv \sqrt{\frac{1}{V(V-1)} \sum_{i=1}^V (\delta_i - \bar{\delta})^2} \quad (3.4)$$

e $\tau_{1-\alpha/2, V-1}$ denota o quantil $1 - \alpha/2$ da distribuição *t* de Student com $V - 1$ graus de liberdade.

Para testar a hipótese $H_0 : \mu_{\delta} = 0$ (hipótese de que os algoritmos ψ_A e ψ_B são equivalentes sob a medida h), após construído o intervalo de confiança $[a, b]$ para μ_{δ} , verificamos se $0 \in [a, b]$. Em caso afirmativo, rejeitamos essa hipótese, concluindo que ψ_A e ψ_B possuem desempenhos distintos; em caso afirmativo, considera-se um empate entre ψ_A e ψ_B , com nível de confiança $1 - \alpha$.

Capítulo 4

O método DDBT

O DDBT é uma extensão do algoritmo REAL – Real-Valued Attribute Learning [8, 9]. Enquanto os algoritmos tradicionais utilizam medidas de entropia ou medidas de impureza (índice GINI) para fazer a divisão dos nós, o REAL e o DDBT utilizam uma função de convicção. A função de convicção do DDBT é inspirada na função do REAL, e foi concebida para incorporar o tratamento de conjunto de dados com desbalanceamento entre as classes.

Para a geração das árvores, o DDBT inicia o processo de indução no nó raiz contendo o conjunto de treinamento completo. Em cada nó, realizam-se os seguintes passos:

1. Binarização de cada atributo, que consiste em encontrar a melhor divisão do conjunto de dados em duas partes, segundo o critério de ganho de convicção;
2. Seleção do atributo correspondente à binarização ótima;
3. Divisão apropriada do nó em dois novos nós filhos, de acordo com a binarização do atributo;
4. Caso não haja mais atributos que resultem em ganho positivo de convicção, o procedimento para a expansão do nó, rotulando-o com uma das classes, de acordo com a regra de rotulação.

A versão atual do DDBT trata exclusivamente problemas de classificação binária e portanto, consideramos $K = 2$ e as classes são indexadas por $k \in \{1, 2\}$.

Neste capítulo usamos a seguinte notação, sendo alguns símbolos definidos no Capítulo 2. Denotamos por N a quantidade de exemplos no conjunto de treinamento $\mathcal{L}$ e por N_k a quantidade de exemplos de $\mathcal{L}$ pertencentes a classe k ; denotamos ainda por π_k a probabilidade (desconhecida) de um elemento do conjunto universo $\mathcal{U}$ ser de classe k e por $\hat{\pi}_k$ o estimador pontual de π_k dado pela frequência relativa de exemplos de classe k em $\mathcal{L}$,

$$\hat{\pi}_k = \frac{N_k}{N}. \quad (4.1)$$

Dado um nó t da árvore, $n_{\bullet,t}$ denota a quantidade de exemplos de $\mathcal{L}$ incidentes sobre t e $n_{k,t}$ a quantidade de exemplos de t pertencentes à classe k . Denotamos também $\pi_{k,t}$ a probabilidade (desconhecida) de um elemento incidente sobre t pertencer à classe k . De forma análoga à Equação 2.4, $\hat{\pi}_{k,t}$ denota a frequência relativa da classe k no nó t ,

$$\hat{\pi}_{k,t} = \frac{n_{k,t}}{n_{\bullet,t}}. \quad (4.2)$$

4.1 Regra de Rotulação

Para a rotulação de um nó terminal t o DDBT atribui a classe com maior prevalência relativa em t em relação a prevalência observada no conjunto $\mathcal{L}$. Formalmente,

$$k^* = \max_k \frac{\hat{\pi}_{k,t}}{\hat{\pi}_k}. \quad (4.3)$$

Considere o seguinte exemplo: Suponha que no conjunto $\mathcal{L}$ as frequências relativas das classes 1 e 2 sejam 0.2 e 0.8 respectivamente. Se um nó terminal t contém 4 exemplos de classe 1 e 6 exemplos de classe 2, então teremos:

$$\frac{\hat{\pi}_{1,t}}{\hat{\pi}_1} = \frac{0,4}{0,2} = 2 \quad ; \quad \frac{\hat{\pi}_{2,t}}{\hat{\pi}_2} = \frac{0,6}{0,8} = 0,75,$$

e portanto a classe 1 será atribuída ao nó t .

No exemplo acima, a decisão de rotular o nó com a classe 1 (a despeito desta ser minoritária) tem a seguinte interpretação: em relação à frequência observada no conjunto $\mathcal{L}$, nosso algoritmo entende que 4 exemplos pertencentes à classe minoritária, cuja frequência original é de 0,2, possui uma importância maior que 6 exemplos da classe majoritária, cuja frequência em $\mathcal{L}$ é de 0,8. Algoritmos que rotulam o nó terminal pela classe predominante no nó, por exemplo REAL e J48, determinariam a classe 2 para esse nó.

Os algoritmos que utilizam métodos de custo por erro de classificação para rotular um nó, dependendo dos custos atribuídos à cada classe, poderiam produzir resultado equivalente ao resultados do nosso algoritmo, bastando estabelecer custos de erro de classificação inversamente proporcional à distribuição observada.

4.2 Função de Convicção

Assumindo que os exemplos incidentes sobre o nó sejam independentes e identicamente distribuídos, podemos considerar que $n_{k,t}$ segue uma distribuição Binomial com parâmetro

desconhecido $\pi_{k,t} \in [0, 1]$:

$$P(n_{k,t} | \pi_{k,t}, n_{\bullet,t}) = \binom{n_{\bullet,t}}{n_{k,t}} \pi_{k,t}^{n_{k,t}} (1 - \pi_{k,t})^{n_{\bullet,t} - n_{k,t}}. \quad (4.4)$$

O parâmetro $\pi_{k,t}$ representa a probabilidade de um exemplo incidente sobre o nó t ser da classe k , e seu estimador usual é $\hat{\pi}_{k,t} = n_{k,t}/n_{\bullet,t}$. Aqui, consideramos uma abordagem Bayesiana, através da qual, ao invés de considerar o estimador pontual $\hat{\pi}_{k,t}$, nosso interesse é considerar uma distribuição de probabilidade para $\pi_{k,t}$. Demonstra-se que, assumindo uma distribuição a priori uniforme para $\pi_{k,t}$, sua distribuição a posteriori após a observação de $n_{\bullet,t}, n_{k,t}$ segue uma distribuição Beta, com parâmetros $\alpha_1 = n_{k,t} + 1$ e $\alpha_2 = n_{\bullet,t} - n_{k,t} + 1$ [31], cap. 6:

$$f(\pi_{k,t} | \alpha_1, \alpha_2) = \frac{\Gamma(n_{\bullet,t} + 2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)} \pi_{k,t}^{n_{k,t}} (1 - \pi_{k,t})^{n_{\bullet,t} - n_{k,t}}. \quad (4.5)$$

4.2.1 Função de Convicção do REAL

O REAL rotula cada nó com a classe mais frequente naquele nó, e define uma função de convicção baseada em um limitante superior do erro de classificação.

Supondo, sem perda de generalidade, que a classe majoritária seja a classe 1 (e portanto a classe 2 é a minoritária), $n_{2,t}$ é o número de exemplos classificados erroneamente e $\pi_{2,t}$ representa a probabilidade de erro de classificação no nó. A medida de convicção no REAL é definida como $\text{conv}(t) = 1 - cm$, onde

$$cm = \min c \mid Pr(\pi_{2,t} \leq c) \geq g(c) \quad (4.6)$$

e $g(\cdot)$ é uma bijeção monotonicamente decrescente do intervalo $[0, 1]$ sobre si mesmo, preferencialmente côncava. No REAL, $g(c) = 1 - c^r$, onde $r > 1$ é um parâmetro de concavidade. A medida cm pode ser interpretada como uma função de perda e busca representar simultaneamente o limitante superior do erro, $Pr(\pi_{2,t} \leq c)$, e o nível de confiança dinâmico, $g(c)$.

Observando que $Pr(\pi_{2,t} \leq c) = F(c | \alpha_1, \alpha_2)$, onde F denota a função acumulada da densidade f , a medida cm pode ser definida pelo ponto de intersecção entre $F(c | \alpha_1, \alpha_2)$ e

$g(c)$, isto é, a raiz da função abaixo:

$$\begin{aligned} cm(n, n_{2,t}, r) &= c \mid h(c) = 0 \\ h(c) &= 1 - c^r - F(c|\alpha_1, \alpha_2). \end{aligned}$$

A Figura 8 ilustra graficamente a obtenção da função de convicção do REAL. No exemplo apresentado foi considerado um nó com oito exemplos de uma classe e dois exemplos da outra classe, ou seja, $n_{\bullet,t} = 10$ e $n_{2,t} = 2$. Na Figura 8(a) temos o Gráfico da distribuição Beta para o nó. Na Figura 8(b) temos: a linha preta representa a distribuição Beta acumulada (F); a linha vermelha representa a bijeção monotonicamente decendente $g(\cdot)$ com expoente 2; e a linha preta pontilhada, que é formada a partir do encontro das duas linha anteriores, é a perda cm , que neste exemplo é $cm = 0,381$ – e portando a convicção é $1 - cm = 0,619$.

Figura 8 – Gráficos da função de perda do REAL

Na Figura 9(a) são ilustradas as funções de densidade das distribuições dos erros em dois nós hipotéticos, sendo o primeiro com $n_{\bullet,t} = 5, n_{2,t} = 1$ (linha verde), e o segundo com $n_{\bullet,t} = 100, n_{2,t} = 20$ (linha preta). Na Figura 9(b), são apresentadas as funções de distribuição de probabilidade acumulada dos erros nos dois nós (linhas verde e preta), juntamente com a função $g(\cdot)$ com expoente 2 (linha vermelha). Note que, embora nos dois nós o erro estimado seja igual $\hat{\pi}_{2,t} = 0.2$, o nó com maior quantidade de exemplos teria uma medida de perda cm (linha tracejada preta) consideravelmente menor do que a perda do nó com menos exemplos (linha tracejada azul). Esse exemplo ilustra uma característica importante no algoritmo REAL, que é o fato desse método fornecer naturalmente um

balanço entre a taxa pontual de erro e a quantidade de exemplos incidentes sobre os nós da árvore, prevenindo assim o *overfitting*.

Figura 9 – Gráficos da função de convicção do REAL

No algoritmo REAL, a função de convicção $1 - cm$ busca minimizar o erro global de classificação, e um nó folha é rotulado com classe predominante naquele nó. Porém, tanto a função de convicção como a regra de rotulação de nós terminais, não são adequados para a indução de classificadores para conjunto de dados desbalanceados. Assim, propõem-se no algoritmo DDBT versões modificadas da função de convicção cm e da regra de rotulação.

4.2.2 Função de Convicção do DDBT

A função de convicção modificada busca medir a discriminação das classes do nó t em relação ao conjunto de treinamento original. Isso é feito em três etapas:

- definir um nó teórico de referência t_0 com a mesma quantidade de exemplos de $n_{\bullet,t}$, mas com proporções nas classes iguais ao do conjunto de treinamento original; o nó de referência t_0 representa um nó “neutro”, sem nenhum ganho de informação em relação aos dados originais;
- adotar uma distribuição de referência para as proporções das classes no nó t_0 ;
- comparar a distribuição das classes em t com a distribuição de referência em t_0 .

Denotemos por ρ_k a probabilidade de um exemplo incidente em t_0 pertencer à classe k . Na ausência de uma distribuição a posteriori para ρ_0 , adotamos uma distribuição de

referência aproximada: a distribuição beta com parâmetros $(p_{k,t}n_{\bullet,t} + 1, (1 - p_{k,t})n_{\bullet,t} + 1)$:

$$f(\rho_k | p_{k,t}n_{\bullet,t} + 1, (1 - p_{k,t})n_{\bullet,t} + 1) = \frac{\Gamma(n_{\bullet,t} + 2)}{\Gamma(p_{k,t}n_{\bullet,t} + 1)\Gamma((1 - p_{k,t})n_{\bullet,t} + 1)} \rho_k^{p_{k,t}n_{\bullet,t}} (1 - \rho_k)^{(1 - p_{k,t})n_{\bullet,t}}. \quad (4.7)$$

Suponha, sem perda de generalidade, que o nó t seja rotulado com a classe 1. Note que, nesse caso, quanto menor for $\hat{\pi}_2$ em relação a p_2 , maior será a evidência de discriminação entre as classes. Uma ideia é então testar a significância estatística da diferença entre $\hat{\pi}_2$ e p_2 , usando as distribuições de π_2 e de ρ_2 , respectivamente. A hipótese de interesse é $H_0 : \pi_2 = \rho_2$, contra a hipótese alternativa $H_1 : \pi_2 \neq \rho_2$.

Adotamos a abordagem proposta por Vêncio [32], definindo o parâmetro $\tau = \pi_2 / (\pi_2 + \rho_2)$. Observe que esse parâmetro está definido sobre o intervalo $[0, 1]$ e que, além disso, a hipótese $H_0 : \pi_2 = \rho_2$ equivale a $H_0 : \tau = 0,5$.

Assim como na formulação do REAL são privilegiados nós com baixas probabilidades de erros de classificação, na formulação do DDBT busca-se privilegiar nós com baixos valores de τ . Assim, definimos a função de convicção modificada

$$\text{convm}(t) = 1 - cm, \quad (4.8)$$

onde

$$cm = \min c \mid Pr(\tau \leq c) \geq 1 - c^r. \quad (4.9)$$

Embora Pham-Gia [33] apresente uma aproximação analítica da função de densidade de probabilidade de τ , não conhecemos uma expressão para a probabilidade acumulada $Pr(\tau \leq c)$. A abordagem utilizada neste trabalho é aproximar τ por uma distribuição Beta, gerando amostras independentes de π_2 e ρ_2 , computando os valores de τ correspondentes e realizando um ajuste pelo método de máxima verossimilhança.

A implementação atual do DDBT, a Função de Convicção é obtida em três passos:

- São geradas 100 amostras independentes de π_2 e ρ_2 , denotadas por $\pi_2^{(1)}, \pi_2^{(2)}, \dots, \pi_2^{(100)}, \rho_2^{(1)}, \rho_2^{(2)}, \dots, \rho_2^{(100)}$
- para cada par $\pi_2^{(k)}, \rho_2^{(k)}$, calcula-se a razão $\tau^{(k)} = \pi_2^{(k)} / (\pi_2^{(k)} + \rho_2^{(k)})$.
- sobre os pontos $\tau^{(1)}, \tau^{(2)}, \dots, \tau^{(100)}$ é ajustada uma distribuição Beta pelo método de máxima verossimilhança.

A Figura 10 ilustra graficamente a obtenção da função de convicção do DDBT, para o seguinte exemplo hipotético: o nó t tem dois exemplos da classe 1 e oito exemplos

da classe 2; e a proporção de classe 1, no conjunto de treinamento original, é de 40%, portanto o nó de referência t_0 terá 4 exemplos de classe 1 e seis exemplos da classe 2.

Na Figura 10(a) são apresentadas as funções de densidade a posteriori para π_2 (proporção populacional da classe 2 no nó t) e de ρ_2 (proporção populacional da classe 2 no nó t_0), denotadas por $f(\pi_2)$ e $f(\rho_2)$, respectivamente. A Figura 10(b) apresenta a função de densidade do parâmetro $\tau = \pi/(\pi + \rho)$. Na Figura 10(c) são apresentadas: a função de probabilidade acumulada para τ (linha marrom), a bijeção $g()$ com expoente 2 (linha verde), e o valor da função de perda $cm = 0,52$ (linha preta pontilhada).

Figura 10 – Gráficos da função de convicção do DDBT

O nome *Dynamic Discriminant Bounds Tree* (DDBT) é inspirado na função de convicção, pois esta busca estimar, dinamicamente, limitantes inferiores da discriminação das classes em t em relação à distribuição original no conjunto de treinamento.

Como mencionado acima, o parâmetro p_k é usualmente assumido como a proporção original da classe k ($k = 1, 2$) no conjunto de treinamento. Todavia, esse parâmetro pode também ser calibrado pelo usuário, a fim de tornar o algoritmo mais sensível à classe 1 ou à classe 2.

4.3 Divisão dos Nós

Uma regra de binarização de um atributo a_j consiste em uma regra da forma $x_j \in R$, onde x_j denota o valor do atributo j , e R denota uma região dos valores j -ésimo no vetor de atributos $\mathbf{x}$. Para atributos numéricos, $R = (-\infty, c]$, onde c é uma constante. Para atributos categóricos, R é um subconjunto não vazio das categorias do atributo a_j .

A partição binária do nó t pela regra “ $x_j \in R$ ” consiste em dividir o conjunto de

exemplos incidentes sobre o nó t em dois subconjuntos, de acordo com a validade da regra. Considere uma divisão candidata do nó t , com $n_{\bullet,t}$ exemplos, e denote por t_e e t_d os dois nós-filhos resultantes dessa divisão, cada qual com n_{t_e} e n_{t_d} exemplos. O ganho de convicção é obtido pela equação abaixo:

$$\Delta conv = n_{t_e} \times convm(t_e) + n_{t_d} \times convm(t_d) - n \times convm(t). \quad (4.10)$$

A Figura 11 apresenta duas possibilidades de partição ótima para um nó pai hipotético contendo cinco exemplos de classe 1 e trinta exemplos de classe 2, hipoteticamente ainda, vamos supor que o conjunto de treinamento original apresentasse a proporção de 13% dos exemplos para a classe 1, nessas condições, a convicção desse nó pai hipotético seria de $conv = 20,2$ ($c = 0,5763$). A primeira possibilidade de partição é apresentada pela Figura 11(a), onde a soma da convicção dos nós filhos é maior que a convicção do nó pai ($5,8 + 15,6 > 20,2$), portanto o DDBT promoveria a divisão do nó, pois com essa divisão haveria ganho de convicção. A segunda possibilidade, representada pela Figura 11(b), não apresenta ganho de convicção na divisão do nó pai ($14,1 + 5,4 < 20,2$), neste caso a divisão não seria realizada, esse nó seria considerado como nó terminal (folha) rotulado como a classe 1, pois a frequência relativa da classe 1 nesse nó (14,3%) é maior do que a do conjunto de treinamento (13%).

Figura 11 – Amostras de Partições do nó, com e sem sucesso

Capítulo 5

Algoritmos *TDIDT* Comparados

5.1 Considerações Iniciais

Para análise comparativa de desempenho do DDBT, foram adotados cinco algoritmos indutores de árvores de classificação: *CTree*, *J48*, *LMT*, *Random Forests* e *RPART*. Os cinco algoritmos possuem implementações disponíveis no ambiente R [34]. Neste capítulo apresentamos uma breve descrição desses métodos.

5.2 CTree (Conditional Inference Trees)

O CTree é um algoritmo de árvores de classificação e regressão que utiliza a inferência condicional como método de partição recursiva dos nós [35].

Por simplicidade, apresentaremos brevemente a descrição do ctree no contexto de classificação. Considere um elemento do conjunto universo $\mathcal{U}$ com classe (desconhecida) denotada por y . No ctree, o interesse em cada passo é testar a independência entre a ocorrência da classe e os valores do vetor de atributo. Mais especificamente, para $k = 1, \dots, K$, denotemos por $Pr(y = k)$ a probabilidade da classe do elemento ser k e por $Pr(y = k|\mathbf{x})$ a probabilidade do elemento ser de classe k , dado seu vetor de atributos $\mathbf{x} = (x_1, x_2, \dots, x_M)^T$. A hipótese de independência global é dada por

$$Pr(y = k|\mathbf{x}) = Pr(y = k).$$

Todavia, a hipótese global de independência não é testada diretamente sobre os vetor de atributos completo, mas é formulada como a conjunção das hipóteses de independência entre a ocorrência da classe e os valores de cada parâmetro:

$$H_0 = \bigcap_{j=1}^M H_0^j, \quad H_0^j : Pr(y = k|x_j) = Pr(y = k), k = 1, j = 1 \dots M.$$

Os passos gerais do ctree estão descritos no Algoritmo 3.

1 Regra de parada:

- Teste a hipótese nula global H_0 de independência entre as classes e todos os atributos a_j com $H_0 = \bigcap_{j=1}^M H_0^j$ e $H_0^j : \Pr(y = k|x_j) = \Pr(y = k), j = 1 \dots M$.
- Se H_0 não é rejeitada, pare.

Seleção de atributo:

- Selecione o atributo a_j^* que apresenta maior associação com as classes (menor p -valor).

Busca pelo melhor ponto de partição:

- Procure pela melhor partição do atributo a_j^* e particione os dados em dois subconjuntos não vazios e com intersecção nula.

Repita:

- Repita os passos 1, 2 e 3 para os novos subconjuntos.

Algoritmo 3: Algoritmo geral do Ctree [36]

Os passos 1 e 2 do algoritmo são baseados em testes de hipóteses, as quais são realizadas por métodos de permutação. A associação entre a classe y e cada atributo a_j é medida pela estatística linear da forma [35]

$$T_j(\mathcal{L}, \mathbf{w}) = \text{vec} \left(\sum_{i=1}^N w_i g_j(x_{i,j}) h(y_i, (y_1 \dots y_n))^\top \right) \in \mathbb{R}^{p_j q}, \quad (5.1)$$

onde:

- $\mathbf{w} = (w_1, w_2, \dots, w_N)^\top$ é um vetor de pesos não negativos ($w_i > 0$ se o i -ésimo pertence ao nó, e $w_i = 0$ caso contrário);
- $g_j : \chi_j \rightarrow \mathbb{R}^{p_j}$ é uma transformação não aleatória dos valores de a_j ;
- $h : \{1 \dots K\}^{N+1} \rightarrow \mathbb{R}^q$ é a função de influência que depende das classes.

Tanto para o teste de independência realizado no passo 1 como na seleção da melhor partição de um atributo, o vetor de estatística T_j precisa ser estandardizada e mapeada em um valor escalar. Para isso, uma escolha possível é adotar o máximo das estatísticas

estandardizadas,

$$c(t, \mu, \Sigma) = \max z = 1 \dots pq \left| \frac{(t - \mu)_z}{\sqrt{(\Sigma)_{zz}}} \right|,$$

onde t é um valor real da estatística T , μ é o valor esperado de T sob independência e $(\Sigma)_{zz}$ é o z -ésimo elemento da diagonal da matriz de covariância.

Molnar [36] apresenta o detalhamento para a obtenção de w, g, h, μ e Σ no contexto de classificação.

O algoritmo CTree está implementado e disponível no ambiente R [34] através do pacote *party* [35].

5.3 J48

O J48 é uma implementação em código aberto Java do algoritmo C4.5 na plataforma de mineração de dados Weka [37] e encontra-se disponível no ambiente R [34] através do pacote RWeka [38]. O C4.5 é um algoritmo desenvolvido por Quinlan [5], utilizado para gerar uma árvore de decisão já podadas ou não. O C4.5 é uma extensão do algoritmo ID3, também desenvolvido por Quinlan. As notações estão baseadas em Quinlan [18].

O J48 é um algoritmo que utiliza a técnica TDIDT, portanto ele faz a partições sucessivas do conjunto de treinamento, desde o nó raiz até que um critério de parada da expansão seja atendido, transformando os nós terminais em nós folhas e rotulando-os com uma classe.

5.3.1 Tipo de árvore

Se o atributo a_j^* selecionado para o particionamento de um nó t for do tipo contínuo, o subconjunto de treinamento $\mathcal{L}_t$ pertencente à esse nó será dividido em dois, de acordo com valor limite b para o atributo, gerando assim dois ramos: um para o resultado “verdadeiro”; e outro para o resultado “falso” do teste $x_j \leq b$.

Os valores do atributo a_j^* entre os exemplos do nó t são ordenados em $v_1, v_2, \dots, v_N$, e cada par de valores adjacentes são candidatos a estabelecer o limite b , sendo $t = (v_i + v_{i+1})/2$. O valor de b que apresentar o melhor critério de divisão de D será selecionado.

Se o atributo a_j^* for do tipo discreto, para cada valor do atributo a_j^* será gerado um ramo. Portanto, a árvore gerada pelo J48 não necessariamente é binária.

5.3.2 Expansão da árvore

A Razão de Ganho (descrita na seção 2.3.2.3 é o critério utilizado para o particionamento de um nó t é determinado da seguinte forma:

1. Determinar o ganho de informação para todos os testes possíveis no nó;
2. Calcular o ganho de informação médio;
3. Calcular a Razão de Ganho para os testes com ganho de informação acima da média;
4. Selecionar a melhor Razão de Ganho para o particionamento do nó.

Quando todos os exemplos de um nó são da mesma classe, esse nó é transformado em nó folha (terminal), rotulada com essa mesma classe. Se nenhum particionamento do nó apresentar ganho de informação positivo ou se o particionamento apresentar muitos erros de classificação, esse nó é transformado em nó folha, rotulada com a classe mais frequente.

5.3.3 Poda da árvore

Depois que a árvore é construída, todas as sub-árvores recursivamente serão analisadas para identificar se a transformação daquela sub-árvore em uma folha irá reduzir o índice de erro, se isso ocorrer a poda da sub-árvore é realizada.

5.4 LMT – Logistic Model Trees

Landwehr [15] propôs as *árvores de modelo logístico* (do inglês *Logistic Model Trees*, abreviadas por LMT), que basicamente consistem de estruturas de árvores de decisão padrão, nas quais as folhas são rotuladas por funções de regressão logística. Essas funções têm por objetivo fornecer estimativas de probabilidades de um elemento pertencer a cada uma das possíveis classes, a partir de seu vetor de atributos.

5.4.1 Regressão Logística

Considere um elemento de $\mathcal{U}$ com classe (desconhecida) denotada por y . Na regressão logística, o interesse é utilizar uma função de regressão que estima a probabilidade do elemento ser de classe k , $k = 1 \dots K$, a partir de seu vetor de atributos $\mathbf{x} = (x_1, x_2, \dots, x_M)^T$.

Aqui denotamos essa probabilidade por $\Pr(y = k|\mathbf{x})$, $k = 1, \dots, K$. Sob essa abordagem, o critério de classificação é atribuir a classe de maior probabilidade,

$$\hat{y} = \arg \max_k \Pr(y = k|\mathbf{x}). \quad (5.2)$$

Note que o vetor de probabilidades $(\Pr(y = 1|\mathbf{x}), \dots, \Pr(y = K|\mathbf{x}))$ pertence a um espaço Simplex de dimensão $K - 1$ (denotado por $\mathcal{S}^{K-1}$), pois cada probabilidade está no intervalo $[0, 1]$ e a soma das probabilidades é igual a 1. A regressão logística modela essas probabilidades utilizando funções lineares em $\mathbf{x}$, e ao mesmo tempo assegura que as mesmas satisfaçam às propriedades do Simplex. O modelo é especificado em termos de $K - 1$ funções de regressão na forma

$$\log \frac{\Pr(y = k|\mathbf{x})}{\Pr(y = K|\mathbf{x})} = \beta_{k,0} + \beta_{k,1}x_1 + \dots + \beta_{k,M}x_M, \quad k = 1 \dots K - 1. \quad (5.3)$$

Por simplicidade de notação, é usual assumir que $\mathbf{x} = (1, x_1, x_2, \dots, x_M)^T$, ou seja, que $\mathbf{x}$ possui, além dos valores dos atributos originais, uma constante unitária $x_0 = 1$. Assim, denotando por $\boldsymbol{\beta}_k = (\beta_{k,0}, \beta_{k,1}, \dots, \beta_{k,M})^T$, a equação 5.3 pode ser apresentada em forma vetorial,

$$\log \frac{\Pr(y = k|\mathbf{x})}{\Pr(y = K|\mathbf{x})} = \boldsymbol{\beta}_k^T \mathbf{x}, \quad k = 1 \dots K - 1. \quad (5.4)$$

A transformação $\log(\Pr(y = k|\mathbf{x})/\Pr(y = K|\mathbf{x}))$, usualmente denominada *transformação logito*, corresponde ao logaritmo da razão de chances – *log-odds* – entre a classe k e a classe K . Sob essa transformação, cada parâmetro $\beta_{k,j}$ representa a variação esperada no *log-odds* da classe k com respeito à classe K por unidade de variação no valor do atributo a_j . Valores positivos de $\beta_{k,j}$ indicam uma associação positiva entre o valor do atributo a_j e as chances (e, conseqüentemente, na probabilidade) do exemplo ser de classe k em relação à classe K ; valores negativos de $\beta_{k,j}$ indicam associação negativa; $\beta_{k,j} = 0$ indica que as chances do exemplo ser de classe k não são alteradas pelo valor do atributo a_j .

Note que essa transformação é uma bijeção entre cada probabilidade $\Pr(y = k|\mathbf{x})$ e a reta real. De fato, dado o vetor de atributos $\mathbf{x}$ e os coeficientes $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \dots, \boldsymbol{\beta}_{K-1}$, as probabilidades $\Pr(y = k|\mathbf{x})$ são obtidas por inversão da equação 5.4: para $k = 1 \dots K - 1$,

denotando por $P_k \equiv \Pr(y = k|\mathbf{x})$ e $P_K \equiv \Pr(y = K|\mathbf{x})$, tem-se:

$$\log(P_k/P_K) = \boldsymbol{\beta}_k^T \mathbf{x} \Rightarrow \quad (5.5)$$

$$P_k = \exp(\boldsymbol{\beta}_k^T \mathbf{x}) P_K \Rightarrow \quad (5.6)$$

$$P_k = \exp(\boldsymbol{\beta}_k^T \mathbf{x}) \left(1 - \sum_{l=1}^{K-1} P_l\right) \quad (5.7)$$

$$= \exp(\boldsymbol{\beta}_k^T \mathbf{x}) - \exp(\boldsymbol{\beta}_k^T \mathbf{x}) \sum_{l=1}^{K-1} P_l \quad (5.8)$$

$$= \exp(\boldsymbol{\beta}_k^T \mathbf{x}) - \exp(\boldsymbol{\beta}_k^T \mathbf{x}) P_K \sum_{l=1}^{K-1} \exp(\boldsymbol{\beta}_l^T \mathbf{x}) \quad (5.9)$$

$$= \exp(\boldsymbol{\beta}_k^T \mathbf{x}) - P_k \sum_{l=1}^{K-1} \exp(\boldsymbol{\beta}_l^T \mathbf{x}) \Rightarrow \quad (5.10)$$

$$P_k = \frac{\exp(\boldsymbol{\beta}_k^T \mathbf{x})}{1 + \sum_{l=1}^{K-1} \exp(\boldsymbol{\beta}_l^T \mathbf{x})}, k = 1 \dots K-1. \quad (5.11)$$

Da equação 5.11 tem-se

$$P_K = 1 - \sum_{k=1}^{K-1} P_k = 1 - \frac{\sum_{k=1}^{K-1} \exp(\boldsymbol{\beta}_k^T \mathbf{x})}{1 + \sum_{l=1}^{K-1} \exp(\boldsymbol{\beta}_l^T \mathbf{x})} \quad (5.12)$$

$$= \frac{1}{1 + \sum_{l=1}^{K-1} \exp(\boldsymbol{\beta}_l^T \mathbf{x})} \quad (5.13)$$

A formulação aqui apresentada utiliza a última classe como a base para os logs das razões de chances; entretanto, a escolha da classe base é arbitrária, uma vez que as estimativas das probabilidades são as mesmas, independentemente da escolha da classe base [39] (Cap. 5).

5.4.2 Estimação dos Coeficientes

O ajuste do modelo de regressão logística compreende a estimação dos vetores de coeficientes $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \dots, \boldsymbol{\beta}_{K-1}$. A abordagem padrão para este problema é a estimação por máxima verossimilhança, que consiste em escolher os parâmetros que maximizam a probabilidade dos exemplos observados. Na abordagem tradicional, a otimização dos parâmetros é feita através de métodos numéricos e a seleção dos atributos é feita através de testes de hipóteses [40]. O LMT, por outro lado, utiliza o método *LogitBoost* para ajuste de modelos de regressão logística aditiva, por máxima verossimilhança. Esses modelos são uma generalização dos modelos de regressão logística linear apresentados acima possuindo a forma geral:

$$\Pr(y = k|\mathbf{x}) = \frac{\exp(F_k(\mathbf{x}))}{\sum_{l=1}^K \exp(F_l(\mathbf{x}))}, \quad \sum_{l=1}^K F_l(\mathbf{x}) = 0, \quad (5.14)$$

onde $F_k(\mathbf{x}) = \sum_{q=1}^Q f_{kq}(\mathbf{x})$, $f_{kq}(\mathbf{x})$ são funções do vetor de atributos (não necessariamente lineares) e Q é a quantidade de funções adotadas (ou mais especificamente, o número de iterações do algoritmo).

O LogitBoost é um método iterativo que, a cada iteração, computa o erro do modelo ajustado corrente, e tenta melhorar esse modelo adicionando uma função f_{kq} ao comitê F_k , usando ajuste por mínimos quadrados.

A regressão linear logística (equações 5.11–5.13) é um caso particular da formulação acima, na qual $F_k(\mathbf{x})$ assume a forma

$$F_k(\mathbf{x}) = \beta_k^T \mathbf{x} \quad (5.15)$$

estabelecendo-se $\beta_k = \beta_{k-K}$ para $k = 1 \dots K-1$ e $\beta_K = \sum_{l=1}^{K-1} \beta_l / (K-2)$.

5.4.3 Seleção de Atributos

Para selecionar os atributos mais relevantes, a estratégia adotada no LMT é a seguinte: a cada iteração q , ajustar uma função de regressão a cada atributo utilizando mínimos quadrados como critério de erro, e então adotando f_{kq} como a regressão sobre o atributo com o menor erro. Esse processo é equivalente a um método quasi-Newton de otimização [41].

Landwehr [15] lembra que, um vez que toda regressão linear múltipla pode ser expressa como uma soma de funções de regressão linear simples, o modelo geral não muda ao se utilizar regressão simples ao invés de múltipla para f_{kq} . Além disso, o modelo final encontrado pelo LogitBoost será o mesmo, devido à convergência dos métodos quasi-Newton ao ponto de máxima verossimilhança se o número de iterações for grande o suficiente.

Por outro lado, para evitar o superajustamento aos dados, o LMT estabelece uma política para interromper o número de iterações antes do algoritmo atingir o ponto de máxima verossimilhança. O número adequado de iterações do LogiBoost é determinado por validação cruzada: divide-se os dados originais em cinco subconjuntos de treinamento e testes, roda-se o algoritmo sobre cada conjunto de treinamento com até 500 iterações, e mede-se os erros de classificação no respectivo conjunto de testes (em cada uma das 500 iterações). Em seguida, o LogitBoost é executado novamente sobre o conjunto de treinamento original, usando o número de iterações que resultou no menor erro médio sobre os cinco conjuntos de testes.

A abordagem acima resulta em um ajuste adequado do modelo de regressão, utilizando-

se apenas os atributos mais relevantes.

5.4.4 Tratamento de Atributos Categóricos

O LMT utiliza a abordagem tradicional em modelos de regressão para converter atributos categóricos em numéricos: suponha que o atributo a_j seja categórico, com o conjunto $\{a_j^1, a_j^2 \dots a_j^{C_j}\}$ de valores possíveis; então, o vetor de atributos $\mathbf{x}$ é estendido da seguinte maneira: x_j é convertido em um vetor binário de C_j componentes $(x_j^1, x_j^2 \dots x_j^{C_j})^T$, de forma que $x_j^c = 1$ se $x_j = a_j^c$ e $x_j^c = 0$ caso contrário, para $c = \dots C_j$.

5.4.5 Indução da Árvore

O LMT utiliza a seguinte estratégia básica:

1. A expansão da árvore inicia construindo-se um modelo logístico na raiz, utilizando-se o algoritmo LogitBoost, ajustando iterativamente funções de regressão linear simples, usando a validação cruzada (descrita na subseção 5.4.3) para determinar o número adequado de iterações.

Realiza-se a partição da raiz (e, conseqüentemente, dos dados) através do método de razão de ganho de informação (implementado no algoritmo C4.5 e descrito na Subseção 2.3.2.3). A partição pode ser binária (para atributos numéricos) ou múltipla (para atributos categóricos).

2. Em cada novo nó obtido, o modelo logístico é estendido a partir do modelo anteriormente obtido no nó pai. Ou seja, o algoritmo LogitBoost é retomado a partir de seu ponto de parada no nó pai, usando como ponto de reinício os atributos e coeficientes previamente selecionados. O número ótimo de iterações é novamente determinado por validação cruzada (utilizando o conjunto de treinamento incidente sobre esse nó).
3. A expansão de um nó somente ocorre se todas as condições abaixo forem satisfeitas:
 - Se o nó contiver pelo menos 15 exemplos. Em comparação com os principais algoritmos, o LMT necessita de folhas com mais exemplos para o modelo logístico.
 - Para evitar fragmentação demasiada de um nó, a divisão do nó só é considerada se houver pelo menos dois subconjuntos com dois exemplos em cada (atributo categórico).

- O modelo logístico somente é criado para nós com mais de 5 exemplos; nós com menos de 5 exemplos herdam o modelo do nó pai.
4. Após a expansão da árvore, utiliza-se o critério de poda por custo-complexidade (implementado no algoritmo CART e descrito na Subseção 2.4)

Duas motivações são apontadas por Landwehr [15] para adotar, nos nós-filhos, modelos logísticos estendidos a partir dos nós pais (passo 2): a primeira é que, uma vez que o nó pai já contém uma função de regressão ajustada, é razoável utilizá-la como uma base para o ajuste da regressão nos nós filhos. Espera-se que os parâmetros do modelo no nó pai incorporem influências “globais” de alguns atributos sobre a classe; no nó filho, o modelo pode ser adicionalmente refinado levando-se em conta atributos que possuem influência “local”. A segunda motivação é que é computacionalmente mais eficiente construir os modelos logísticos nos níveis inferiores da árvore refinando modelos já construídos nos níveis superiores do que construir novos modelos do início.

5.5 Random Forest

O algoritmo Random Forest está implementado e disponível no ambiente R [34] através do pacote randomForest [42].

O Random Forest é um conjunto de árvores de decisão, sendo que, para problemas de classificação a classe de um exemplo é dada pelo voto majoritário entre as árvores da floresta [43]. As principais características da indução das árvores da floresta são: as árvores crescem utilizando a metodologia CART até o tamanho máximo possível; não é utilizado processo de poda (nem pré-poda nem pós-poda); e que para a divisão de cada nó, poucos atributos são selecionados aleatoriamente [44]. A indução das árvores seguem a técnica TDIDT, sendo que o conjunto de treinamento é particionado sucessivamente, dando origem aos ramos, nós internos até chegar nas folhas da árvore.

A indução de cada árvore da floresta depende dos valores de um vetor aleatório amostrado de forma independente e com a mesma distribuição [44]. Esse conjunto de árvores é construído baseado no método de *Bagging* (**bootstrap aggregating**) *Predictors*. Ao final do processo de indução de todas as árvores, é possível obter a taxa de erro para cada um dos exemplos do conjunto de dados originais, e adicionalmente, é possível calcular a estimativa de erro de todos os classificadores.

Bootstrap - É o método proposto por Bradley Efron em 1979 para fazer deter-

minados tipos de inferências estatísticas [45]. Para as inferências estatísticas o processo produz amostras do conjunto de dados original, sorteadas com reposição e cada amostra tem o mesmo tamanho do conjunto de dados original. Essas amostras são usualmente denominadas amostras *bootstrap* [46].

Bagging Predictors - É um método para a geração de múltiplas versões de um preditor, as várias versões do preditor (classificador) são construídas utilizando amostras bootstrap do conjunto de dados original, sendo que, a predição da classe de um exemplo é determinada pelo voto majoritário de todas as versões do classificador. [47].

Adaptado das notações realizadas por Breiman [47] e [44] temos que $\mathcal{L}$ é um conjunto de treinamento composto pelos dados $\{(\mathbf{x}_i, y_i), i = 1, 2, \dots, N\}$, sendo que: N é a quantidade de exemplos do conjunto de treinamento; $\mathbf{x}_i$ é o vetor dos atributos dos exemplos; e $y_i \in \{1, 2, \dots, K\}$ é a classe do i -ésimo exemplo.

O preditor da classe de $\mathbf{x}$ que foi induzido a partir do conjunto de treinamento $\mathcal{L}$ é denotado por $\psi(\bullet, \mathcal{L})$. Tomemos uma sequência finita amostras bootstrap do conjunto de treinamento $\mathcal{L}$, sendo que cada amostra contém N exemplos independentes provenientes da mesma distribuição subjacente ao conjunto $\mathcal{L}$, denotemos essa sequência de amostras por $\{\mathcal{L}^{(v)}\}, v = 1, 2, \dots, V$. O objetivo é com essas V amostras induzir V preditores, e com um método de agregação obter um preditor melhor, denotemos portanto, $\psi(\bullet, \mathcal{L}^{(v)})$ como a sequência de preditores induzidos pela sequência de amostras $\{\mathcal{L}^{(v)}\}$.

Indexando-se as classes por $k = 1, 2, \dots, K$, um método de agregar os preditores $\psi(\bullet, \mathcal{L}^{(v)})$ é através de votação, escolhendo para $\mathbf{x}$ a classe mais votada entre os preditores. Formalmente, denotando por

$$N_k = |\{v \in \{1 \dots V\} : \psi(\mathbf{x}, \mathcal{L}^{(v)}) = k\}| \quad (5.16)$$

como o número de “votos” na classe k , o classificador agregado pode ser definido por

$$\psi_A = \arg \max_k N_k. \quad (5.17)$$

sendo o subscrito A em ψ_A denota agregação [48].

A obtenção de $\{\mathcal{L}^{(v)}\}, v = 1, 2, \dots, V$ é feita tomando-se re-amostras *bootstrap* de $\mathcal{L}$, via sorteio com repetição, cada qual de tamanho N .

Out-Of-Bag (OOB) - Na geração de cada amostra bootstrap, alguns exemplos são deixados de fora e não são usados na indução do classificador. Esse conjunto de exemplos é chamado de Out-Of-Bag (OOB) e é usado no teste do classificador construído. A

estimativa de erro obtida por este processo evita a necessidade de cross-validation ou de um conjunto de dados para teste em separado [47].

O erro *out-of-bag* de cada classificador $\psi(\bullet, \mathcal{L}^{(v)})$ é definido como o percentual do conjunto de teste (constituído por $\mathcal{L} \setminus \mathcal{L}^{(v)}$) classificado erroneamente.

5.6 RPART

O algoritmo RPART está implementado e disponível no ambiente R [34] através do pacote rpart [17].

O RPART implementa várias ideias do método CART (Classification and Regression Trees) [6]. O critério de seleção de atributos para partição dos nós é o índice de Gini. Este algoritmo também segue a técnica TDIDT para a indução das árvores.

As árvores de construídas são binárias e o crescimento delas é limitado a 31 (trinta e um) níveis de profundidade. O algoritmo implementa processo de poda para minimizar a estimativa de risco.

Critério de partição do nó - O RPART usa uma das medidas de impureza para orientar o critério de partição do nó. Sendo f a função para definir a impureza do nó t como: $I(t) = \sum_{k=1}^K f(\pi_{k,t})$, onde $\pi_{k,t}$ é a probabilidade estimada de um exemplo do nó t pertencer à classe k . As duas medidas de impurezas candidatas são: índice de informação (vide Equação 2.15); e o índice Gini (vide Equação 2.12). Para a separação do nó, é escolhido o índice I que maximiza a redução de impureza:

$$\Delta I_t = \pi_{k,t} I(t) - \pi_{k,t_d} I(t_l) - \pi_{k,t_e} I(t_e)$$

onde t_d e t_e são os nós da direita e da esquerda candidatos à partição do nó t

Incorporação da Função de Perda no índice de impureza A incorporação da Função de Perda no índice de impureza tem o objetivo de redução de risco. No CART as funções de perda são extensões dos critérios de impureza, o índice de Gini generalizada e priori alterada. O RPART implementa apenas o método de priori alterada.

A Priori Alterada: A técnica da priori alterada muda o $p_{k,t}$ para que sua máxima seja representada pelo 'peso' do erro de classificação das classes, ou seja, o índice de impureza $\Delta I_t = \sum_{k=1}^K f(\pi_{k,t})$ tem a sua máxima em $\pi_{1,t} = p_{i2,t} = \dots = p_{iK,t} = 1/K$, sendo que o erro de classificação tem o mesmo peso para todas as classes, porém, se para um determinado problema, a classe 1 tem uma perda duas vezes a perda de uma classe 2 ou 3, então o ΔI_t

ter o seu valor máximo alterado para $\pi_{1,t} = 1/5$, $pi_{2,t} = pi_{2,t} = 2/5$, para poder decidir a classe do nó.

As priores alteradas são utilizadas para auxiliar o processo de escolha da melhor partição do nó em termos de risco. As perdas normais e priores são utilizados para calcular o risco do nó.

Importância de uma variável: Uma variável pode aparecer muitas vezes na árvore, tanto como um primária, como substituta. Uma medida geral de importância da variável é a soma da medida de ganho (*goodness*) de cada subconjunto do particionamento da variável principal, mais ganho * (fator de ajuste) para todos os particionamentos em que tinha uma variável substituta. As variáveis substitutas que apresentam uma proporção menor que 1% são omitidas.

Processo de Poda O processo de poda no RPART é orientado pelo método de custo-complexidade descrito na seção 2.4, esse método calcula do custo-complexidade em manter o particionamento de cada sub-árvore no modelo. O objetivo é minimizar o custo-complexidade geral do modelo, eliminando partes da árvore que apresentem elevação no custo-complexidade naquele seguimento do modelo. A ideia geral é: determinar o fator de complexidade, analisando todas as sub-árvores possíveis; calcular o custo-complexidade para cada sub-árvore; e eliminar as sub-árvores que aumentem o custo-complexidade, transformando a sub-árvore em um nó terminal.

Dados Faltantes (*Missing Values*) A maioria dos procedimentos adotados para missing-values é rejeitar o exemplo que não esteja com todos dados preenchidos, porém para o RPART qualquer exemplo com valor para a variável dependente e pelo menos uma variável independente preenchida participará da modelagem.

Decidindo a partição de um nó - Para a partição proposta de um nó é analisada a diminuição do índice de impureza (diferença entre a impureza do nó pai e a soma dos nós filhos), para o cálculo da probabilidade e da impureza do nó pai, são considerados todos os exemplos, independente dos missing-values, porém, para o cálculo da probabilidade e da impureza dos nós filhos, os exemplos com missing-values não são considerados. OBS. Este método tem o viés de quando somente dois exemplos não são missing-values, um para cada filho proposto.

Substituindo valores - Um atributo que contém missing-values foi escolhido para particionar um nó, então os missing-values serão preenchidos com valores por duas regras possíveis, a regra escolhida e a que minimizar o erro de classificação: A primeira regra e

prever os missing-values de acordo com o critério de particionamento, por exemplo, o atributo idade foi selecionado para particionar o nó, sendo a primeira parte os exemplos com idade > 40 , e a segunda parte os exemplos com idade ≤ 40 , com base nos outros atributos, o RPART irá prever os missing-values do atributo idade para essas duas condições (sem considerar priores e perdas). A segunda regra é atribuir pela maioria.

A implementação utilizada encontra-se no Pacote rpart [17], a qual permite a incorporação de custos distintos de erros de classificação nas classes, possibilitando assim aumentar a sensibilidade do classificador às classes minoritárias.

Capítulo 6

Avaliações Numéricas do Algoritmo

Na avaliação do DDBT foram conduzidos testes numéricos para analisar seu desempenho frente a cinco outros algoritmos de indução de árvores de classificação, além da comparação específica com o algoritmo REAL. Foram utilizados vinte conjuntos de dados públicos, obtidos na página de internet da UCI Machine Learning Repository [49]. A técnica de validação cruzada foi utilizada para conduzir os testes numéricos e os resultados foram submetidos à matriz de confusão para análise de algumas taxas e indicadores.

6.1 Datasets

Todos os conjuntos de dados selecionados para a avaliação de desempenho do DDBT são caracterizados por classes binárias e ausência de *missing values*, uma vez que a versão atual do algoritmo não incorpora tratamento para conjunto de dados que contenha multi-classes ou *missing values*.

A Tabela 7 apresenta um sumário dos conjuntos de dados. As colunas da tabela apresentam as seguintes informações:

- Identificação do conjunto de dados para este trabalho;
- Identificação do conjunto de dados na origem (UCI);
- Tipos de dados (Real, numérico, categórico, etc.) que estão presentes no conjunto de dados;
- Quantidade de atributos (sem considerar a classe) que o conjunto de dados possui;
- Quantidade total de exemplos (soma dos exemplos da classe majoritária e da minoritária);
- Valores possíveis para as duas classes de cada conjunto de dados;
- Percentuais das classes minoritárias e majoritárias em cada conjunto de dados.

As tabelas a seguir que apresentam os resultados de análises por conjunto de dados, estarão dispostas com as linhas (conjunto de dados) em ordem crescente da participação da classe minoritária.

Tabela 7 – Sumários dos conjuntos de dados utilizados

Abreviação	Nome do Dataset	Tipo de Atributo	Qtde Atributos	Qtde exemplos	Classes (minoritária, majoritária)	% nas Classes
PageBlock	Page Blocks Classification	Real, Inteiro	10	5242	Blocos de uma página: linha horizontal, texto	[6,3 - 93,7]
Bank	Bank Marketing	Real, Categórico	16	4521	Contratação de produto bancário: sim, não	[11,5 - 88,5]
Spect	SPECT Heart	Categórico	22	267	Avaliação cardíaca: Anormal, Normal	[20,6 - 79,4]
SpectF	SPECTF Heart	Inteiro	44	267	Avaliação cardíaca: Anormal, Normal	[20,6 - 79,4]
Blood	Blood Transfusion Service Center	Real	4	748	Doação de sangue feita em mar/07: sim, não	[23,8 - 76,2]
Haberman	Haberman's Survival	Inteiro	3	306	Sobrevivência após cirurgia: óbito, sobrevivência	[26,5 - 73,5]
Planning	Planning Relax	Real	12	182	Sinais de dois estágios mentais: Relaxado, Planejamento	[28,6 - 71,4]
St_credit	Statlog (German Credit Data)	Inteiro, Categórico	20	1000	Risco de crédito: ruim, bom	[30 - 70]
St_credit_n	Statlog (German Credit Data)	Inteiro	24	1000	Risco de crédito: ruim, bom	[30 - 70]
Column_2C	Vertebral Column	Real	6	310	Diagnóstico de coluna vertebral: normal, anormal	[32,3 - 67,7]
Monks2	MONK's Problems	Categórico	6	432	Classificação do exemplo: 0, 1	[32,9 - 67,1]
TicTacToe	Tic-Tac-Toe Endgame	Categórico	9	958	Vitória para o primeiro jogador: negativo, positivo	[34,7 - 65,3]
Magic	MAGIC Gamma Telescope	Real	10	19020	Tipo de partícula: hadrão, gama	[35,2 - 64,8]
Ionosphere	Ionosphere	Real, Inteiro	34	351	Qualidade do sinal recebido: ruim, bom	[35,9 - 64,1]
Wdbc	Breast Cancer Wisconsin (Diagnostic)	Real	30	569	Tipo de tumor: maligno, benigno	[37,3 - 62,7]
St_heart	Statlog (Heart)	Real, Categórico	13	270	Doença cardíaca: presente, ausente	[44,4 - 55,6]
Sonar	Connectionist Bench (Sonar, Mines vs. Rocks)	Real	60	208	Identificação do objeto: rocha, mina	[46,6 - 53,4]
Monks3	MONK's Problems	Categórico	6	432	Classificação do exemplo: 0, 1	[47,2 - 52,8]
Chess	Chess (King-Rook vs. King-Pawn)	Inteiro, Categórico	36	3196	Pedra branca pode vencer: não vence, vence	[47,8 - 52,2]
Monks1	MONK's Problems	Categórico	6	432	Classificação do exemplo: 0, 1	[50 - 50]

O conjunto de dados *Bank* é descrito por Moro [50]; *Blood* é descrito por Yeh [51]; e *Planning* é descrito por Bhatt [52].

Os conjuntos de dados Monks1, Monks2, Monks2, Spect e St_heart possuem atributos categóricos representados somente por valores numéricos, esses atributos foram convertidos para valores alfanuméricos (adicionou-se a letra “a” em seu conteúdo original) para

permitir que todos algoritmos fizessem o correto tratamento para os atributos categóricos.

6.2 Considerações Iniciais

A análise de desempenho realizada neste trabalho utilizou a técnica de v-fold de validação cruzada. Em nossos testes, utilizamos $V = \min\{20, \lfloor N/30 \rfloor\}$, onde N denota a quantidade de exemplos do conjunto de dados. Dessa forma, garantiu-se que cada subconjunto tivesse no mínimo 30 exemplos [25].

Tomou-se também o cuidado de realizar a partição aleatória de cada conjunto de dados uma única vez, de modo a garantir que os treinamentos e testes dos diferentes algoritmos fossem realizados exatamente sobre os mesmos exemplos. Dessa forma, buscou-se evitar que as diferenças de desempenho entre algoritmos pudessem ser atribuídas às flutuações decorrentes da amostragem.

Como ocorre usualmente em problemas de diagnósticos médicos, assumimos a classe minoritária como Positiva, e a classe majoritária como Negativa.

As Tabelas 8 e 11 apresentam, para cada *dataset* e para cada algoritmo, a matriz de confusão média sobre as V iterações das validações cruzadas:

$$\begin{aligned}\overline{VN} &= \frac{1}{V} \sum_{v=1}^V VN^v, \\ \overline{FN} &= \frac{1}{V} \sum_{v=1}^V FN^v, \\ \overline{FP} &= \frac{1}{V} \sum_{v=1}^V FP^v, \\ \overline{VP} &= \frac{1}{V} \sum_{v=1}^V VP^v\end{aligned}$$

onde VN^v , FN^v , FP^v , VP^v são denotado pelas respectivas quantidades absolutas de cada um dos casos da matriz de confusão na V -ésima interação da validação cruzada.

Assim como na apresentação das matrizes de confusão (Tabelas 8 e 11), as Tabelas 9 e 12 apresentam os valores $\overline{PP_r}$, $\overline{FP_r}$, e $\overline{FN_r}$, que denotam as médias dessas taxas, obtidas nas V iterações dos testes realizados para os conjuntos de dados, dado por:

$$\begin{aligned}\overline{PP_r} &= \frac{1}{V} \sum_{v=1}^V PP_r^v \\ \overline{FP_r} &= \frac{1}{V} \sum_{v=1}^V FP_r^v \\ \overline{FN_r} &= \frac{1}{V} \sum_{v=1}^V FN_r^v\end{aligned}$$

onde PP_r^v , FP_r^v , FN_r^v são denotado pelas respectivas taxas obtidas na V -ésima interação da validação cruzada.

Nas Tabelas 9, 12 (taxas de desempenho) e 10, 13 (Análise do F-Score e do AUC)

as células demarcadas e destacadas com a cor verde representam os melhores resultados obtidos por conjunto de dados e por taxa e/ou indicador.

Na comparação do DDBT com os demais algoritmos, os resultados apresentados nas Tabelas 12 (taxas de desempenho) e Tabela 13 (Análise do F-Score e do AUC) foram submetidos ao teste t-pareado (com coeficiente de confiança de 90%) para identificar quais resultados são estatisticamente equivalentes. Para a realização desse teste, para cada taxa/indicador e para cada conjunto de dados, tomamos o melhor resultado obtido e comparamos com resultado dos demais algoritmos. As células destacadas em azul correspondem aos resultados que apresentaram equivalência estatística com o melhor desempenho.

6.3 Análise do Desempenho comparativa entre DDBT e REAL

Nesta seção é analisada a diferença de desempenho entre os algoritmos DDBT e REAL, nessa análise é apresentada a matriz de confusão as principais taxas de desempenho e uma análise dos indicadores F-Score e AUC.

6.3.1 Matriz de Confusão

A Tabela 8 apresenta a Matriz de Confusão referente ao processamento dos algoritmos DDBT e REAL. Comparando as duas matrizes é possível notar que, para muitos conjuntos de dados, o DDBT obteve os melhores resultados. Entretanto, analisando a classe minoritária, para alguns conjuntos de dados, o REAL obteve os melhores resultados, inclusive para conjuntos de dados com alto desbalanceamento entre as classes, como é o caso do conjunto de dados Spect.

6.3.2 Taxas de Desempenho

A Tabela 9 apresenta os resultados da taxa de: Precisão (PP_r); Falso Positivo (FP_r); e Falso Negativo (FN_r). Essas taxas são obtidas a partir dos dados da Matriz de Confusão acima e serão utilizadas mais adiante nos indicadores de desempenho.

A taxa de Precisão ($\overline{PP_r}$) corresponde a taxa de acerto das predições reais para a classe positiva (para o nosso trabalho, a classe minoritária). Na Tabela 9 podemos observar que, para esta taxa, o DDBT apresentou todos os melhores resultados.

Tabela 8 – Matriz de Confusão comparativa do DDBT e REAL por conjunto de dados.

Conj. Dados	Classe Min. (%)	DDBT				REAL			
		VN	FP	FN	VP	VN	FP	FN	VP
PageBlock	6,3	241,9	3,8	1,0	15,5	235,5	10,2	1,7	14,8
Bank	11,5	140,8	59,2	4,3	21,7	141,0	59,1	7,6	18,5
Spect	20,6	20,6	5,5	2,4	4,5	15,5	10,6	0,9	6,0
SpectF	20,6	22,5	3,9	3,6	3,0	16,1	10,3	1,4	5,3
Blood	23,8	19,4	8,9	3,3	5,6	17,3	10,9	2,4	6,4
Haberman	26,5	17,5	4,5	4,5	3,5	16,2	5,8	4,0	4,0
Planning	28,6	15,0	6,5	5,2	3,3	12,0	9,5	5,5	3,0
St_credit	30,0	24,4	10,6	4,8	10,3	17,8	17,2	5,3	9,7
St_credit_n	30,0	24,8	10,2	4,9	10,2	19,9	15,2	3,0	12,0
Column_2C	32,3	16,7	4,3	1,5	8,5	16,1	4,9	1,0	9,0
Monks2	32,9	10,1	10,1	4,4	5,4	5,0	15,2	2,9	6,9
TicTacToe	34,7	23,5	7,3	6,9	9,4	23,5	7,3	6,9	9,4
Magic	35,2	523,2	93,4	87,1	247,3	460,8	155,8	93,0	241,5
Ionosphere	35,9	18,2	1,6	1,2	10,0	15,9	3,9	1,3	9,9
Wdbc	37,3	18,9	0,7	1,4	10,0	17,7	1,9	1,3	10,1
St_heart	44,4	13,1	3,6	3,9	9,4	12,9	3,8	3,7	9,7
Sonar	46,6	14,5	3,8	7,2	8,5	11,3	7,0	5,2	10,5
Monks3	47,2	15,1	0,8	0,0	14,1	15,1	0,8	0,0	14,1
Chess	47,8	80,5	2,5	6,8	69,3	77,0	6,0	6,2	69,9
Monks1	50,0	7,6	7,3	0,0	15,1	7,6	7,3	0,0	15,1

Tabela 9 – Comparativo DDBT e REAL - Taxas de Precisão; Falso Negativo e Falso Positivo

Conj. Dados	Classe Min. (%)	PP_r (%)		FP_r (%)		FN_r (%)	
		DDBT	REAL	DDBT	REAL	DDBT	REAL
PageBlock	6,3	80,4	59,2	1,5	4,1	5,5	9,8
Bank	11,5	27,0	24,0	29,6	29,5	15,8	28,3
Spect	20,6	45,7	38,0	20,8	39,9	33,5	12,9
SpectF	20,6	41,6	33,9	14,7	39,1	53,7	18,6
Blood	23,8	38,9	37,5	31,5	38,6	36,9	26,6
Haberman	26,5	42,6	42,1	20,5	26,1	54,4	47,0
Planning	28,6	37,3	22,9	29,6	44,6	60,2	66,0
St_credit	30,0	48,8	35,9	30,1	48,8	32,3	32,3
St_credit_n	30,0	50,1	44,4	29,1	43,1	32,2	19,9
Column_2C	32,3	66,8	65,8	21,1	23,2	15,4	9,3
Monks2	32,9	35,9	31,3	50,1	75,0	42,2	27,9
TicTacToe	34,7	56,5	56,5	23,8	23,8	42,3	42,3
Magic	35,2	72,6	60,8	15,1	25,3	26,0	27,8
Ionosphere	35,9	86,2	71,8	8,2	19,3	11,2	11,3
Wdbc	37,3	94,3	85,0	3,5	9,7	12,2	10,8
St_heart	44,4	73,0	71,7	22,3	22,5	28,5	27,2
Sonar	46,6	68,5	56,6	21,5	39,7	44,6	36,6
Monks3	47,2	94,9	94,9	5,5	5,5	0,0	0,0
Chess	47,8	96,6	91,8	3,0	7,2	9,0	8,2
Monks1	50,0	67,3	67,3	49,2	49,2	0,0	0,0

A taxa de Falso Positivo ($\overline{FP_r}$) retrata os erros na classificação da exemplos da classe negativa (para o nosso trabalho, a classe majoritária). Novamente o DDBT em relação ao REAL apresentou todos os melhores resultados.

A taxa de Falso Negativo ($\overline{FN_r}$) é a taxa de erro na classificação dos exemplos da classe positiva (para o nosso trabalho, a classe minoritária). Para essa a taxa o REAL em relação ao DDBT obteve o melhor resultado para onze conjuntos de dados.

6.3.3 Análise do F-Score e do AUC

Na Tabela 10 são apresentados dois indicadores que associam mais de uma taxa, obtidas a partir da matriz de confusão, para poder analisar o desempenho dos algoritmos testados.

O primeiro indicador é o F-score (indicador de precisão do teste). Para a análise dos resultados obtidos por este indicador, foi somente utilizado o valor de $\beta = 1$, que representa o equilíbrio entre a precisão e a sensibilidade.

O segundo indicador é o *AUC* (*Area Under Curve*), cuja formulação não necessita de parâmetro de calibração, sendo apurada a média entre sensibilidade (VP_r) e especificidade (VN_r). Na Tabela 13 as células demarcadas e destacadas com a cor verde representam os melhores resultados obtidos por conjunto de dados e por indicador.

Tabela 10 – Comparativo DDBT e REAL - F-Score com $\beta = 1$ e do AUC para cada conjunto de dados

Conj. Dados	Classe Min. (%)	F-Score $\beta = 1$ (%)		AUC (%)	
		DDBT	REAL	DDBT	REAL
PageBlock	6,3	43,3	35,6	96,5	93,0
Bank	11,5	20,2	17,7	77,3	71,1
Spect	20,6	26,2	25,6	72,8	73,6
SpectF	20,6	20,8	23,4	65,8	71,2
Blood	23,8	23,5	24,4	65,8	67,4
Haberman	26,5	21,3	22,3	62,5	63,5
Planning	28,6	18,3	13,6	55,1	44,7
St_credit	30,0	28,0	22,8	68,8	59,4
St_credit_n	30,0	28,5	28,2	69,4	68,5
Column_2C	32,3	37,2	37,7	81,8	83,7
Monks2	32,9	21,3	21,4	53,9	48,5
TicTacToe	34,7	28,2	28,2	66,9	66,9
Magic	35,2	36,6	33,0	79,4	73,5
Ionosphere	35,9	43,6	39,4	90,3	84,7
Wdbc	37,3	45,2	43,2	92,2	89,7
St_heart	44,4	36,0	35,9	74,6	75,1
Sonar	46,6	30,3	29,8	66,9	61,9
Monks3	47,2	48,7	48,7	97,2	97,2
Chess	47,8	46,9	45,9	94,0	92,3
Monks1	50,0	40,0	40,0	75,4	75,4

Os algoritmos DDBT e REAL apresentaram o mesmo desempenho para os conjuntos de dados Monks1, Monks3 e TicTacToe, tanto para o indicador F-Score como para o AUC.

Para o indicador F-Score o DDBT apresentou desempenho melhor que o REAL para doze conjuntos de dados, enquanto o REAL foi melhor para os outros cinco conjuntos de dados. Para o indicador AUC o DDBT apresentou desempenho melhor que o REAL para

onze conjuntos de dados e o Real foi melhor para os outros seis conjuntos de dados.

6.4 Análise do Desempenho comparativa entre DDBT e demais algoritmos

Nesta seção é analisada e comparado o desempenho do DDBT com o desempenho dos algoritmos CTREE, J48, LMT RForest e RPART.

6.4.1 Matriz de Confusão

A Tabela 11 apresenta a Matriz de Confusão referente ao processamento dos algoritmos DDBT e os cinco outros algoritmos utilizados na avaliação deste trabalho. As taxas e indicadores de desempenho apresentadas adiante são obtidas a partir destas matrizes.

Tabela 11 – Matriz de Confusão por algoritmo e por conjunto de dados.

Conj. Dados	Classe Min. (%)	Ctree				DDBT				J48				LMT				RForest				RPART			
		VN	FN	FP	VP	VN	FN	FP	VP	VN	FN	FP	VP	VN	FN	FP	VP	VN	FN	FP	VP	VN	FN	FP	VP
PageBlock	6,3	243	1	2	15	242	1	4	15	244	2	2	15	244	1	2	15	245	2	1	15	244	2	2	15
Bank	11,5	192	15	8	11	141	4	59	22	193	16	7	10	196	18	5	8	193	16	7	10	194	17	6	9
Spect	20,6	24	6	3	1	21	2	6	5	24	4	3	3	24	4	2	3	24	4	2	3	25	4	2	3
SpectF	20,6	24	5	3	2	23	4	4	3	22	4	4	3	24	4	3	2	26	5	1	2	22	4	5	3
Blood	23,8	25	5	3	4	19	3	9	6	26	5	3	3	26	6	2	3	25	6	3	3	26	6	2	3
Haberman	26,5	19	6	3	3	18	5	5	4	19	5	3	3	21	6	1	2	19	6	3	2	20	5	2	3
Planning	28,6	22	9	-	-	15	5	7	3	22	9	-	-	22	9	-	-	20	8	2	0	15	7	6	2
St_credit	30,0	30	8	5	7	24	5	11	10	30	9	5	6	31	8	4	7	32	9	3	6	31	9	4	6
St_credit_n	30,0	31	9	4	6	25	5	10	10	30	8	5	7	31	8	4	7	32	8	3	7	30	8	5	7
Column_2C	32,3	19	4	2	6	17	2	4	9	19	3	2	7	18	2	3	8	19	2	2	8	18	3	3	7
Monks2	32,9	20	10	-	-	10	4	10	5	20	10	-	-	19	8	1	2	20	4	0	6	18	5	3	5
TicTacToe	34,7	29	2	2	15	24	7	7	9	27	4	4	12	31	1	0	15	31	0	0	16	29	2	2	14
Magic	35,2	571	98	46	236	523	87	93	247	570	95	47	240	578	90	39	244	579	75	38	259	573	128	43	207
Ionosphere	35,9	19	2	1	9	18	1	2	10	19	2	1	9	20	2	0	9	19	1	0	10	19	2	1	9
Wdbc	37,3	19	1	1	10	19	1	1	10	19	1	1	11	19	1	0	11	19	1	1	11	18	2	1	10
St_heart	44,4	15	4	2	9	13	4	4	9	14	3	3	10	15	3	2	10	15	3	2	10	15	3	2	10
Sonar	46,6	13	4	5	12	15	7	4	9	15	5	3	11	14	4	4	12	16	4	2	12	14	5	5	11
Monks3	47,2	16	-	-	14	15	-	1	14	16	-	-	14	16	-	0	14	16	-	-	14	16	-	-	14
Chess	47,8	82	2	1	74	81	7	2	69	83	0	0	76	83	0	0	76	82	1	1	75	80	2	3	74
Monks1	50,0	8	-	7	15	8	-	7	15	13	-	2	15	13	0	2	15	15	-	-	15	11	1	4	14

6.4.2 Taxas de Desempenho

Assim como a Tabela 9, a Tabela 12 apresenta os resultados da taxa de: Precisão (PP_r); Falso Positivo (FP_r); e Falso Negativo (FN_r).

Tabela 12 – Média dos valores obtidos para as Taxas de Precisão; Falso Negativo e Falso Positivo

Conj. Dados	Classe Min. (%)	PP_r (%)						FP_r (%)						FN_r (%)					
		Ctree	DDBT	J48	LMT	RForest	RPART	Ctree	DDBT	J48	LMT	RForest	RPART	Ctree	DDBT	J48	LMT	RForest	RPART
PageBlock	6,3	86,2	80,4	89,6	89,9	93,2	90,0	1,0	1,5	0,7	0,7	0,4	0,6	8,6	5,5	9,4	8,6	10,1	11,1
Bank	11,5	58,6	27,0	57,7	64,8	56,8	59,8	3,9	29,6	3,7	2,3	3,6	3,0	57,4	15,8	61,6	68,9	63,4	64,8
Spect	20,6	11,8	45,7	59,8	67,1	62,4	65,6	9,7	20,8	9,3	6,4	8,2	6,0	78,3	33,5	53,5	56,7	53,5	53,9
SpectF	20,6	19,7	41,6	38,5	45,5	75,0	38,5	10,4	14,7	16,9	9,8	2,9	17,4	73,6	53,7	56,1	66,2	70,7	56,1
Blood	23,8	56,2	38,9	57,1	67,9	52,3	66,5	9,9	31,5	9,0	6,4	10,6	6,4	59,4	36,9	62,2	63,1	68,0	66,7
Haberman	26,5	33,9	42,6	43,3	62,3	46,1	56,3	12,8	20,5	12,5	4,7	14,0	9,8	66,8	54,4	62,3	75,4	67,2	65,5
Planning	28,6	-	37,3	-	-	25,0	22,9	-	29,6	-	-	8,3	29,0	100	60,2	100	100	96,6	77,4
St_credit	30,0	59,0	48,8	52,1	60,7	66,6	56,9	13,9	30,1	15,5	12,5	9,7	12,1	54,1	32,3	61,9	55,3	56,6	61,7
St_credit_n	30,0	62,1	50,1	56,5	66,1	69,0	61,3	11,4	29,1	14,2	10,4	8,7	13,4	58,3	32,2	55,6	53,5	54,3	52,0
Column_2C	32,3	77,2	66,8	78,6	74,8	77,3	72,0	9,6	21,1	10,1	13,2	10,8	13,9	37,1	15,4	35,8	21,6	24,3	31,3
Monks2	32,9	-	35,9	-	31,6	96,4	63,1	-	50,1	-	5,0	1,3	12,6	100	42,2	100	79,9	39,9	55,7
TicTacToe	34,7	87,3	56,5	77,6	99,4	99,4	88,5	7,5	23,8	11,5	0,3	0,3	6,3	10,2	42,3	24,6	4,8	2,7	12,7
Magic	35,2	83,7	72,6	83,7	86,4	87,3	82,7	7,4	15,1	7,5	6,2	6,1	7,0	29,4	26,0	28,3	27,0	22,6	38,2
Ionosphere	35,9	91,0	86,2	87,8	97,6	96,0	90,7	4,0	8,2	6,4	1,3	2,4	5,5	23,1	11,2	20,3	19,5	12,8	20,8
Wdbc	37,3	90,8	94,3	93,4	98,4	96,0	89,6	5,4	3,5	4,1	0,8	2,8	6,6	8,6	12,2	8,4	4,6	5,9	12,4
St_heart	44,4	82,4	73,0	77,6	86,9	85,6	85,5	11,8	22,3	18,1	9,5	11,1	11,2	29,6	28,5	25,6	22,5	21,8	22,1
Sonar	46,6	68,2	68,5	80,3	72,5	85,7	69,4	28,8	21,5	17,8	22,5	11,6	25,7	25,9	44,6	32,1	27,1	22,4	31,5
Monks3	47,2	100	94,9	100	99,5	100	100	-	5,5	-	0,4	-	-	-	-	-	-	-	-
Chess	47,8	99,0	96,6	99,4	99,8	99,0	96,6	0,9	3,0	0,5	0,2	0,9	3,1	2,1	9,0	0,5	0,4	1,7	3,1
Monks1	50,0	67,3	67,3	91,2	88,6	100	76,8	49,2	49,2	10,4	11,9	-	27,1	-	-	-	1,2	-	6,3

Para a taxa de Precisão ($\overline{PP_r}$), os algoritmos CTREE, DDBT, J48 e RPART não apresentaram bons resultados: o DDBT, CTREE e o RPART cada um apresentou a melhor taxa para apenas um conjunto de dados; o J48 teve desempenho um pouco melhor que o DDBT, apresentando a melhor taxa para dois conjuntos de dados. Os melhores resultados para essa taxa foram apresentados pelos algoritmos LMT (nove conjuntos de dados) e RFOREST (dez conjuntos de dados). Cabe lembrar que não necessariamente esses dois algoritmos estão classificando bem a classe minoritária, pois exemplos da classe positiva podem ter sido classificados como classe negativa, e tal situação não é capturada pela taxa de precisão.

Em relação à taxa de Falso Positivo ($\overline{FP_r}$), assim como para a taxa de Precisão, os

melhores valores (menor taxa) foram obtidos pelos os algoritmos RFOREST e LMT. O RFOREST apresentou o melhor resultado para nove conjunto de dados e o LMT apresentou o melhor resultado para dez conjuntos de dados. Para os demais algoritmos o DDBT não apresentou a melhor taxa para nenhum conjunto de dados; o CTREE apresentou a melhor taxa para quatro conjuntos de dados; o J48 e o RPART cada um apresentou a melhor taxa para três conjuntos de dados.

Para a taxa de Falso Negativo ($\overline{FN}_r$) o DDBT foi o algoritmo que apresentou os melhores resultados, especialmente sobre os conjuntos de dados com maior desbalanceamento entre as classes. O DDBT apresentou o melhor resultado para treze conjuntos de dados, o RFOREST (segundo melhor algoritmo) apresentou o melhor resultado para 7 conjunto de dados, o LMT, J48, CTREE e RPART apresentaram o melhor resultado para 3, 2, 2, e 1 conjuntos de dados respectivamente.

6.4.3 Análise do F-Score e do AUC

Na Tabela 13 são apresentados os indicadores F-score (indicador de precisão do teste) e *AUC* (*Area Under Curve*), como dito anteriormente, esses indicadores associam mais de uma taxa, obtidas a partir da matriz de confusão, para poder analisar o desempenho dos algoritmos testados.

Analisando os resultados da Tabela 13 podemos observar que nos dez primeiros conjuntos de dados, aqueles com maior desbalanceamento entre as classes, de forma geral o DDBT foi o algoritmo que apresentou o melhor desempenho, tanto para o indicador F-Score como para o AUC. Pelo critério F-Score, dentre esses 10 datasets o DDBT obteve o melhor resultado ou resultado estatisticamente equivalente ao melhor em 8 casos, contra 7 casos no LMT, 6 casos no RForest, 4 casos no RPART e 2 casos no CTree. Note-se também que o DDBT obteve o melhor F-Score médio em 7 casos, número bastante superior aos demais. Pelo critério AUC, o DDBT obteve a melhor média em 8 casos, e foi estatisticamente equivalente ao melhor em 2 casos. Para os demais algoritmos, apresentaram um certo equilíbrio, observando-se no máximo 6 casos com AUC satisfatória. Um aspecto importante é que, comparando os resultados do indicador AUC com as taxas da Tabela 12, podemos observar que a menor taxa de erros da classe minoritária ($\overline{FN}_r$) não implicou em incremento expressivo das taxas de erros na classe majoritária ($\overline{FP}_r$).

Com relação aos dez últimos conjuntos de dados, aqueles que apresentam pequeno desbalanceamento entre as classes, podemos observar que o algoritmo RFOREST foi o que apresentou os melhores resultados, tanto para o indicador F-Score como para o AUC.

Tabela 13 – Resultado do F-Score com $\beta = 1$ e do AUC para cada conjunto de dados

Conj. Dados	Classe Min. (%)	F-Score $\beta = 1$ (%)						AUC (%)					
		Ctree	DDBT	J48	LMT	RForest	RPART	Ctree	DDBT	J48	LMT	RForest	RPART
PageBlock	6,3	88,4	86,5	89,8	90,4	91,2	89,2	95,2	96,5	94,9	95,4	94,7	94,1
Bank	11,5	48,7	40,4	45,2	41,6	43,9	43,7	69,4	77,3	67,3	64,4	66,5	66,1
Spect	20,6	15,3	52,3	49,7	48,5	49,6	52,0	56,0	72,8	68,6	68,5	69,1	70,0
SpectF	20,6	22,3	41,6	40,0	36,2	38,9	38,7	58,0	65,8	63,5	62,0	63,2	63,3
Blood	23,8	45,7	47,1	44,1	45,0	38,0	41,7	65,3	65,8	64,4	65,3	60,7	63,5
Haberman	26,5	32,5	42,6	32,5	34,6	35,9	39,9	60,2	62,5	62,6	60,0	59,4	62,3
Planning	28,6	-	36,6	-	-	5,9	21,7	50,0	55,1	50,0	50,0	47,5	46,8
St_credit	30,0	50,4	56,0	43,1	50,4	50,9	45,0	66,0	68,8	61,3	66,1	66,9	63,1
St_credit_n	30,0	48,3	56,9	49,0	52,9	53,9	52,1	65,1	69,4	65,1	68,1	68,5	67,3
Column_2C	32,3	68,2	74,4	68,4	75,5	75,7	69,4	76,6	81,8	77,1	82,6	82,5	77,4
Monks2	32,9	-	42,6	-	21,8	70,2	50,4	50,0	53,9	50,0	57,5	79,4	65,8
TicTacToe	34,7	88,1	56,4	75,9	97,2	98,3	87,6	91,2	66,9	81,9	97,4	98,5	90,5
Magic	35,2	76,6	73,3	77,2	79,1	82,1	70,7	81,6	79,4	82,1	83,4	85,7	77,4
Ionosphere	35,9	83,0	87,1	82,8	87,5	91,0	83,5	86,4	90,3	86,6	89,6	92,4	86,8
Wdbc	37,3	90,8	90,4	92,2	96,7	94,8	88,1	93,0	92,2	93,8	97,3	95,7	90,5
St_heart	44,4	75,1	72,0	75,7	81,6	81,6	80,8	79,3	74,6	78,1	84,0	83,6	83,4
Sonar	46,6	70,6	60,7	72,9	72,4	81,3	68,2	72,6	66,9	75,0	75,2	83,0	71,4
Monks3	47,2	100,0	97,4	100,0	99,7	100,0	100,0	100,0	97,2	100,0	99,8	100,0	100,0
Chess	47,8	98,4	93,7	99,4	99,7	98,7	96,8	98,5	94,0	99,5	99,7	98,7	96,9
Monks1	50,0	80,0	80,0	94,9	93,0	100,0	83,5	75,4	75,4	94,8	93,5	100,0	83,3

Para esses conjuntos de dados a quantidade de resultados com equivalência estatística foi muito pequena, e a distribuição dos melhores resultados do indicador F-Scores é bastante semelhante ao do indicador AUC. Observar que tanto para o indicador F-Score como para o AUC, o algoritmo DDBT não apresentou o melhor resultado para nenhum conjunto de dados, e somente para o conjunto de dados Ionosphere e indicador F-Score o DDBT apresentou resultado com equivalência estatística.

Tanto para o indicador F-Score como para o AUC, nos dez primeiros conjuntos de dados o algoritmo DDBT apresentou melhor desempenho, e para o últimos dez conjunto de dados o RFOREST apresentou melhor desempenho. Entretanto, os dois indicadores apresentam diferenças significativas nos resultados obtidos para diversos conjuntos de dados. A seguir apresentamos as principais divergências obtidas.

Os dois primeiros conjuntos de dados (PageBlock e Bank) apresentaram comportamento bastante diferente: para o indicador AUC o melhor algoritmo foi o DDBT e nenhum outro algoritmo conseguiu apresentar equivalência estatística de resultados pelo teste t-

pareado, porém, para o indicador F-Score o DDBT apresentou o pior resultado entre os seis algoritmos. A característica comum entre esses dois conjuntos de dados para DDBT é que, olhando a Tabela 12 vamos perceber a taxa de Falso Positivo FP_r muito alta e a taxa de Falso Negativo FN_r muito baixa.

Para o conjunto de dados Planning, tanto para o indicador F-Score como para o AUC, o melhor resultado foi do algoritmo DDBT, porém para o F-Score nenhum algoritmo apresentou resultado com equivalência estatística, enquanto para o indicador AUC o algoritmo CTREE, J48 e LMT apresentaram equivalência estatística.

No conjunto de dados Haberman, para o indicador F-Score o algoritmo J48 apresentou o pior desempenho, enquanto para o indicador AUC o J48 apresentou o melhor resultado.

Em relação ao conjunto dados Column.2C, para o indicador F-Score o RFOREST apresentou o melhor resultado e o LMT ficou entre os algoritmos com equivalência estatística, enquanto na análise do indicador AUC a situação inverteu-se: o LMT apresentou o melhor resultado e o RFOREST ficou entre os algoritmos que apresentaram equivalência estatística.

Para os conjunto de dados Sonar, na análise pelo teste t-pareado, o F-Score apresentou um algoritmo com equivalência estatística, enquanto para o AUC, não encontrou resultados com equivalência estatística.

6.4.4 Análise do Gráfico ROC e Teste t-Pareado por Conjunto de Dados

Nas Figuras 12 e 13 são apresentados os gráficos ROC dos seis algoritmos, sobre cada conjunto de dados. Em cada gráfico temos: os resultados dos seis algoritmos avaliados; a linha de isodesempenho, linha preta; e a linha de equilíbrio na predição das classes, linha pontilhada em azul. A linha de isodesempenho passa pelo ponto que representa o melhor resultado obtido na avaliação, e sua inclinação foi dada pela proporção entre as classes no conjunto de dados original. O equilíbrio no desempenho das classificações da classe positiva e negativa é representado pela linha diagonal descendente (0,100 - 100,0), vide seção 3.4.

Analisando os classificadores em relação ao desempenho das predições entre as classes, observamos que para a maioria dos conjuntos de dados os classificadores apresentaram desempenho melhor para a classe negativa, ou seja, a maioria dos resultados dos algoritmos estão abaixo da linha de equilíbrio. Para poucos conjuntos de dados os resultados dos

algoritmos foram acima da linha de equilíbrio. Podemos observar que o DDBT foi o algoritmo que apresentou seus resultados mais próximos da linha de equilíbrio, mesmo para os conjuntos de dados onde ele não apresentou o melhor resultado.

Analizando o desempenho dos classificadores em relação ao ponto ótimo (0,100) do gráfico, encontramos resultados semelhantes aos obtidos na análise do indicador AUC da Tabela 13, com a diferença que aqui é possível fazer a comparação visual dos resultados dos classificadores.

Nos dez primeiros conjuntos de dados, onde o DDBT apresentou os melhores resultados na análise do F-Score e AUC, em alguns conjuntos de dados o DDBT também classificou melhor a classe negativa, como é observado nos conjuntos de dados Pageblock, SpectF, Haberman e Planning. Ainda assim, para esses conjuntos de dados o DDBT foi o algoritmo mais próximo do equilíbrio, sendo que o J48 para o SpecF apresentou equilíbrio semelhante.

Nos dez últimos conjuntos de dados, apesar do DDBT não ter apresentado os melhores resultados na análise do F-Score e AUC, ainda assim para alguns conjuntos de dados foi o algoritmo que apresentou o maior equilíbrio entre as classes na predição dos conjuntos de dados Monks2 Magic e Ionosphere. Para os conjuntos de dados Monks1 e Monks3 todos os algoritmos classificaram melhor a classe negativa. Para o conjunto de dados St_Heart o DDBT e o J48 apresentaram os melhores equilíbrios. Para os demais conjuntos de dados o DDBT não apresentou equilíbrio entre as classes e não ficou próximo a linha do isodesempenho.

Para cada conjunto de dados, o gráfico ROC é apresentado nas Figuras 12 e 13.

Figura 12 – Gráficos ROC para os conjuntos de dados - 1ª parte

Figura 13 – Gráficos ROC para os conjuntos de dados - 2ª parte

Capítulo 7

Conclusões

Neste trabalho, propomos um novo algoritmo indutor de árvores de classificação para problemas com dados desbalanceados. Esse algoritmo, denominado atualmente DDBT (*Dynamic Discriminant Bounds Tree*), utiliza um critério de partição de nós que, ao invés de se basear em frequências absolutas de classes, compara as proporções das classes nos nós com as proporções do conjunto de treinamento original, buscando formar subconjuntos com maior discriminação de classes em relação ao conjunto de dados original. Para a rotulação de nós terminais, o algoritmo atribui a classe com maior prevalência relativa no nó em relação à prevalência no conjunto original. Essas características fornecem ao algoritmo a flexibilidade para o tratamento de conjuntos de dados com desbalanceamento de classes, resultando em um maior equilíbrio entre as taxas de erro em classificação de objetos entre as classes.

Para problemas de classificação em que a prevalência da classe minoritária era inferior a 30%, o desempenho do DDBT mostrou-se superior aos demais algoritmos, sob os indicadores *AUC* (área sob a curva ROC) e *F-Score*. Para esses conjuntos de dados, além de apresentar as melhores taxas de classificação da classe minoritária (FN_r - taxa de falso negativos), o DDBT não penalizou significativamente o desempenho na classificação da classe majoritária (VN_r - taxa de verdadeiro negativo), em comparação aos demais algoritmos analisados. Esses resultados indicam que o DDBT pode ser um bom competidor em domínios nos quais a classe minoritária representa eventos raros ou possui elevados custos de erro de classificação.

Uma vantagem adicional da abordagem proposta é a boa adaptabilidade do algoritmo, sem necessidade de calibrações manuais de parâmetros, e sem a necessidade de utilização de métodos para o balanceamento das classes no conjunto de treinamento. Essa característica confere ao algoritmo a possibilidade de sua utilização por usuários menos experientes.

Por outro lado, em problemas de classificação com dados mais balanceados (proporções da classe minoritária superior a 30%, outros algoritmos tiveram desempenho superior ao DDBT.

7.1 Trabalhos Futuros

A versão atual do DDBT é voltada exclusivamente para problemas de classificação binária. Ademais, não incorpora métodos para tratamento de dados faltantes (*missing values*). Assim, extensões desse algoritmo devem ser orientadas para tratar problemas multi-classes e com *missing values*.

O baixo desempenho na classificação para problemas que envolvam conjunto de dados com baixo desbalanceamento entre as classes é também uma oportunidade de extensão deste trabalho.

Trabalhos futuros também podem serem desenvolvidos na: investigação do desempenho do algoritmo sob *ensembles* dos tipos *bagging* e *boosting*; análises teóricas das funções de convicção; e análises comparativas envolvendo números maiores de e algoritmos.

Referências

- 1 MONARD, M. C.; PRATI, R. C. Aprendizado de máquina simbólico para mineração de dados. In: WRCÓPIAS. *XIII Escola Regional de Informática (ERI 2005)*. WRCópias, 2005. p. 1–26. Disponível em: <<http://www.labic.icmc.usp.br/pub/mcmonard/eri05book.pdf>>.
- 2 WEBB, A. R.; COPSEY, K. D. *Statistical pattern recognition*. 3nd. ed. [S.l.]: A John Wiley Sons, Ltd., 2011.
- 3 WITTEN, I. H.; FRANK, E.; HALL, M. A. *Data Mining Practical Machine Learning Tools and Techniques*. 3nd. ed. [S.l.]: Morgan Kaufmann Publishers, 2011.
- 4 PRATI, R. C.; BATISTA, G. E. A. P. A.; MONARD, M. C. Evaluating classifiers using roc curves. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, v. 6, p. 215–222, 2008.
- 5 QUINLAN, J. *C4.5: Programs for Machine Learning*. [S.l.]: Morgan Kaufmann Publishers, 1993. (C4.5 - programs for machine learning / J. Ross Quinlan). ISBN 9781558602380.
- 6 BREIMAN, L. et al. *Classification and Regression Trees*. [S.l.]: Chapman and Hall, 1984.
- 7 BATISTA, G. E. A. P. A.; PRATI, R. C.; MONARD, M. C. A study of the behavior of several methods for balancing machine learning training data. *Sigkdd Explorations*, v. 6, p. 20–29, 2004.
- 8 LAURETTO, M. S. *Árvores de Classificação para Escolha de Estratégias de Operação em Mercados de Capitais. -In Portuguese*. Dissertação (Mestrado) — Universidade de São Paulo - Instituto de Matemática e Estatística., 1996.
- 9 STERN, J. M. et al. Algoritmo de aprendizagem para atributos reais e estratégias de operação em mercado. *IBERAMIA*, 1998.
- 10 XU, R.; WUNSCH-II, D. Survey of clustering algorithms. *IEEE Tansactions on Neural Network*, v. 65, n. 3, 2005.
- 11 ZHU, X. *Semi-Supervised Learning with Graphs*. Dissertação (Mestrado) — School of Computer Science - Carnegie Mellon University, 2005.
- 12 BASGALUPP, M. P. *LEGAL Tree: Um algoritmo genérico multi-objetos lexicográfico para indução de árvores de decisão*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação ICMC - USP, 2010.

- 13 MAIMOM, O.; ROKACH, L. *Data Mining and Knowledge Discovery Handbook*. 2nd. ed. [S.l.]: Springer New York, 2010.
- 14 QUINLAN, J. R. Induction of decision trees. *Machine Learning*, v. 1, p. 81–106, 1986.
- 15 LANDWEHR, N.; HALL, M.; FRANK, E. Logistic model trees. *Machine Learning*, Kluwer Academic Publishers, v. 59, n. 1-2, p. 161–205, 2005. ISSN 0885-6125. Disponível em: <<http://dx.doi.org/10.1007/s10994-005-0466-3>>.
- 16 STROBL, C.; MALLEY, J.; TUTZ, G. An introduction to recursive partitioning: Rationale, application and characteristics of classification and regression trees, bagging and random forests. *Psychol Methods*, v. 14, n. 4, p. 323–348, 2009.
- 17 THERNEAU, T.; ATKINSON, B.; RIPLEY, B. *rpart: Recursive Partitioning*. [S.l.], 2012. R package version 3.1-55. Disponível em: <<http://CRAN.R-project.org/package=rpart>>.
- 18 QUINLAN, J. R. Improved use of continuous attributes in c4.5. *J. Artif. Int. Res.*, v. 4, n. 1, p. 77–90, mar. 1996.
- 19 FRANK, E. *Pruning Decision Trees and Lists*. Dissertação (Mestrado) — University of Waikato - Department of Computer Science, 2000.
- 20 ESPOSITO, F. et al. The effects of pruning methods on the predictive accuracy of induced decisions trees. *Applied Stochastic Models in Business and Industry*, v. 15, p. 277–299, 1999.
- 21 QUINLAN, J. R. *Simplifying Decision Trees*. 1986.
- 22 MILARE, C. R.; BATISTA, G. E.; CARVALHO, A. C. A hybrid approach to learn with imbalanced classes using evolutionary algorithms. *Logic Jnl IGPL*, v. 19, p. 293–303, 2011.
- 23 CASTRO, C. L. de; BRAGA, A. P. Aprendizado supervisionado com conjuntos de dados desbalanceados. *Sba Controle Automação [online]*, v. 22, p. 441–466, 2011. Disponível em: <<http://dx.doi.org/10.1590/S0103-17592011000500002>>.
- 24 WIKIPÉDIA. *Validação cruzada – Wikipédia a enciclopédia livre*. 2013. [Online; accessed 11-maio-2013]. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Validacao_cruzada&oldid=349508000>.
- 25 MITCHELL, T. M. *Machine Learning*. [S.l.]: McGraw-Hill, 1997.
- 26 KOHAVI, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In: *International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 1995. p. 1137–1145.
- 27 RIJSBERGEN, C. J. V. *Information Retrieval*. 2nd. ed. Newton, MA, USA: Butterworth-Heinemann, 1979. ISBN 0408709294.

- 28 SOKOLOVA, M.; JAPKOWICZ, N.; SZPAKOWICZ, S. Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In: *Proceedings of the 19th Australian joint conference on Artificial Intelligence: advances in Artificial Intelligence*. Berlin, Heidelberg: Springer-Verlag, 2006. (AI'06), p. 1015–1021. ISBN 3-540-49787-0, 978-3-540-49787-5. Disponível em: <http://dx.doi.org/10.1007/11941439_14j>.
- 29 POWERS, D. M. W. Evaluation: From precision, recall and f-measure to roc, informedness, markedness correlation. *Journal of Machine Learning Technologies*, v. 2, p. 37–63, 2011. Disponível em: <<http://www.bioinfo.in/contents.php?id=51>>.
- 30 QIAO, X.; LIU, Y. Adaptive weighted learning for unbalanced multicategory classification. *Biometrics*, v. 65, p. 159168, 2009.
- 31 DEGROOT, M. H.; SCHERVISH, M. J. *Probability and Statistics*. 2nd. ed. [S.l.]: Addison-Wesley, 1986.
- 32 VÊNCIO, R. Z.; BRENTANI, H.; PEREIRA, C. A. Using credibility intervals instead of hypothesis tests in sage analysis. *Bioinformatics*, v. 19, p. 2461–2464, 2003.
- 33 PHAM-GIA, T. Distributions of the ratios of independent beta variables and applications. *Communications in Statistics - Theory and Methods*, v. 29, p. 2693–2715, 2000.
- 34 R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2012. ISBN 3-900051-07-0. Disponível em: <<http://www.R-project.org/>>.
- 35 HOTHORN, T.; HORNIK, K.; ZEILEIS, A. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, v. 15, n. 3, p. 651–674, 2006.
- 36 MOLNAR, C. *Recursive partitioning by conditional inference*. [S.l.], 2013. Disponível em: <<https://github.com/christophM/overview-ctrees/blob/master/main.pdf>>.
- 37 HALL, M. et al. *The WEKA Data Mining Software: An Update*. 2009. Issue 1.
- 38 HORNIK, K.; BUCHTA, C.; ZEILEIS, A. Open-source machine learning: R meets weka. *Computational Statistics*, v. 24, n. 2, p. 225–232, 2009.
- 39 AITCHISON, J. *The Statistical Analysis of Compositional Data*. [S.l.]: Chapman and Hall, 1986.
- 40 D.W.HOSMER. *Applied Logistic Regression*. 2nd. ed. [S.l.]: Wiley, 2000.
- 41 FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, v. 28, n. 2, p. 337–407, 2000.
- 42 LIAW, A.; WIENER, M. Classification and regression by randomforest. *R News*, v. 2, n. 3, p. 18–22, 2002. Disponível em: <<http://CRAN.R-project.org/doc/Rnews/>>.
- 43 BREIMAN, L.; CUTLER, A. *Random Forests*. Disponível em: <http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm>.

- 44 BREIMAN, L. Random forests. *Machine Learning*, v. 45, p. 5–32, 2001.
- 45 EFRON, B. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, v. 7, p. 1–26, 1979. Disponível em: <<http://www.jstor.org/stable/2958830>>.
- 46 EFRON, B.; TIBSHIRANI, R. J. *An Introduction to the Bootstrap*. [S.l.]: CHAPMAN HALL, 1993.
- 47 BREIMAN, L. Bagging predictors. *Machine Learning*, v. 24, p. 123–140, 1996.
- 48 BASTOS, D. G. D.; NASCIMENTO, P. S.; LAURETTO, M. S. Proposta e análise de desempenho de dois métodos de seleção de características para random forests. *SBSI Sociedade Brasileira de Sistemas*, Maio 2013.
- 49 BACHE, K.; LICHMAN, M. *UCI Machine Learning Repository*. 2013. Disponível em: <<http://archive.ics.uci.edu/ml>>.
- 50 MORO, S.; LAUREANO, R.; CORTEZ, P. Using data mining for bank direct marketing: An application of the crisp-dm methodology. In: AL., P. N. et (Ed.). *Proceedings of the European Simulation and Modelling Conference - ESM'2011*. Guimaraes, Portugal: EUROSIS, 2011. p. 117–121.
- 51 YEH, I.-C.; YANG, K.-J.; TING, T.-M. Knowledge discovery on rfm model using bernoulli sequence. *Expert Systems with Applications*, 2008.
- 52 BHATT, R. *Planning-Relax Dataset for Automatic Classification of EEG Signals*. UCI Machine Learning Repository. Disponível em: <<http://archive.ics.uci.edu/ml/datasets/Planning+Relax>>.