Relações entre Ranking, Análise ROC e Calibração em Aprendizado de Máquina

Edson Takashi Matsubara

Relações entre Ranking, Análise ROC e Calibração em Aprendizado de Máquina[1]

Edson Takashi Matsubara

Orientadora: *Prof^a Dr^a Maria Carolina Monard*

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP como parte dos requisitos necessários à obtenção para do título em Doutor em Ciências de Computação e Matemática Computacional.

USP - São Carlos
julho/2008

[1]Trabalho Realizado com Auxílio da FAPESP Proc. No: 2005/03792-9

Aos meus pais,
Ryuiti e Ritie,

Aos meus irmãos,
Koiti e Sayuri,

À Maria Carolina Monard.

Agradecimentos

Gostaria de agradecer a Deus por estar sempre ao meu lado, sempre me indicando caminhos que me levam a sua bondade, por nunca ter desistido de mim, por mostrar que nenhuma passagem pode ser sem esforço, por me dar coragem, por mostrar os valores que realmente valem, por colocar pessoas tão boas e especiais em meu caminho, por cuidar de cada simples detalhe em minha vida.

Como diria o poeta, mesmo que tivesse em minhas mãos todo o perfume das rosas, toda a beleza do céu, toda a pureza dos anjos, toda a inocência das crianças, toda a grandeza do mar, toda a força das ondas, mesmo que eu tivesse todas as coisas belas da vida e todos os belos lugares do mundo nada teria sentido se eu não tivesse o presente mais valioso, mais nobre e mais sagrado que o Senhor pode me dar... minha família. Agradeço meu pai Ryuiti, por me ensinar muito, mas muito mesmo, sobre todas as coisas da vida. Agradeço minha mãe Ritie por seu imenso carinho e amor. Agradeço aos meus irmãos Koiti, Sayuri, e agora a mais nova irmã Miwa, por serem os melhores irmãos que alguém pode ter.

Mestres nos conduzem, não somente ao conhecimento, mas também ao saber. Agradeço à professora Carolina, por tudo que tem me ensinado, pelos conselhos e pelas conversas, talvez a senhora nem saiba mas sem você, eu não teria feito pós-graduação. Tenho muito carinho por você. Gostaria de agradecer também ao professor Peter Flach e sua esposa Lisa por terem me ensinado muito no ano em que passei pela Universidade de Bristol. Por terem me acolhido em sua casa e pelas horas de utilizávamos os ladrilhos da cozinha me explicando sobre curvas ROC. Gostaria de agradecer aos professores Huei e Paulo, por serem exemplos de dedicação e ensino na formação de pessoas. Gostaria de agradecer também à professora Solange e ao professor André pelo apoio ao meu trabalho.

Além dessas pessoas que considero grandes mestres, existem as pessoas que dão um toque muito especial a minha vida. Considero essas pessoas

grandes amigos. O rio é a mistura de pequenos encontros e quando unidas possuem grande força. Assim considero as amizades que faço nos caminhos da vida. Muitas vezes essas amizades de distanciam de nós, mas elas estão sempre lá, sempre agregando força a esse rio, querendo o nosso bem gratuitamente. Gostaria muito de agradecer aos amigos que fiz aqui em São Carlos. Aos amigos de república Mauro, Ronaldo, Sidão, Danielzinho e Marcio. À dois grandes amigos, Gustavo Batista e Richardson pelos bons momentos projetando e fazendo aviõezinhos e pelas conversas das mais diversas coisas. Pelos amigos que fiz na graduação, Marcio, Alex, Kleber, Testa e Zóid. Agradeço a todos vocês por momentos onde a alegria, descontração e amizade se fizeram presentes.

Cada um que passa em nossa vida, passa sozinho, pois cada pessoa é única e nenhuma substitui outra. Também gostaria de agradecer aos amigos que fiz na Inglaterra: Sebastian, Virgínia, Tarek, Bill, Ksenia, Susanna e Rob. Ao pessoal do LABIC: André Maletzke, André Rossi, Andrés Ferrero, Brunos Ferres, Caneca, Capoli, Camila, Claudia Martins, Claudia Milaré, Christiane, Damiance, Débora, Edson Melanda, Eduardo Spinosa, Evandro, Fabiano, Flávia, Igor, Jaqueline, Jean, Katti, Leonardo, Lorena, Magaly, Marcio Basgalupp, Marcos Cintra, Marcos Quiles, Mariza, Merley, Murilo, Patrícia Rufino, Rafael Giusti, Renatinho, Roberta, Robson, Rodrigo Bianchi, Rodrigo Calvo e Valmir. Gostaria de agradecer à Pâmela, por ser tão única em minha vida e ter me ensinado coisas que somente você poderia ter me ensinado.

Agradeço ao pessoal da pós-graduação do ICMC, à Beth, à Laura, à Ana Paula, por serem tão atenciosas a cada um de nós pós-graduandos. É incrível como vocês decoram o nome de todos nós. Também à Marília por seus maravilhosos coffe breaks. Você tem um papel determinante na presença dos alunos e professores nas palestras.

Agradeço também as pessoas que mantém a USP funcionando, como Paulinho, Dotta, Sonia, Dagoberto, Cabral, Seu Arly e tantos outros que vão do setor administrativo ao faxineiro e jardineiro.

Finalmente gostaria de agradecer à FAPESP pela minha bolsa de doutorado, à CAPES pela minha bolsa de doutorado sandwich e ao ICMC-USP, pelo suporte e estrutura disponibilizados para o desenvolvimento de minha formação.

Abstract

Supervised learning has been used mostly for classification. In this work we show the benefits of a welcome shift in attention from classification to ranking. A ranker is an algorithm that sorts a set of instances from highest to lowest expectation that the instance is positive, and a ranking is the outcome of this sorting. Usually a ranking is obtained by sorting scores given by classifiers. In this work, we are concerned about novel approaches to promote the use of ranking. Therefore, we present the differences and relations between ranking and classification followed by a proposal of a novel ranking algorithm called LEXRANK, whose rankings are derived not from scores, but from a simple ranking of attribute values obtained from the training data. One very important field which uses rankings as its main input is ROC analysis. The study of decision trees and ROC analysis suggested an interesting way to visualize the tree construction in ROC graphs, which has been implemented in a system called PROGROC. Focusing on ROC analysis, we observed that the slope of segments obtained from the ROC convex hull is equivalent to the likelihood ratio, which can be converted into probabilities. Interestingly, this ROC convex hull calibration method is equivalent to Pool Adjacent Violators (PAV). Furthermore, the ROC convex hull calibration method optimizes Brier Score, and the exploration of this measure leads us to find an interesting connection between the Brier Score and ROC Curves. Finally, we also investigate rankings build in the selection method which increments the labelled set of CO-TRAINING, a semi-supervised multi-view learning algorithm.

Resumo

Aprendizado supervisionado tem sido principalmente utilizado para classificação. Neste trabalho são mostrados os benefícios do uso de *rankings* ao invés de classificação de exemplos isolados. Um *rankeador* é um algoritmo que ordena um conjunto de exemplos de tal modo que eles são apresentados do exemplo de maior para o exemplo de menor expectativa de ser positivo. Um *ranking* é o resultado dessa ordenação. Normalmente, um *ranking* é obtido pela ordenação do valor de confiança de classificação dado por um classificador. Este trabalho tem como objetivo procurar por novas abordagens para promover o uso de *rankings*. Desse modo, inicialmente são apresentados as diferenças e semelhanças entre *ranking* e classificação, bem como um novo algoritmo de *ranking* que os obtém diretamente sem a necessidade de obter os valores de confiança de classificação, esse algoritmo é denominado de LEXRANK. Uma área de pesquisa bastante importante em *rankings* é a análise ROC. O estudo de árvores de decisão e análise ROC é bastante sugestivo para o desenvolvimento de uma visualização da construção da árvore em gráficos ROC. Para mostrar passo a passo essa visualização foi desenvolvido uma sistema denominado PROGROC. Ainda do estudo de análise ROC, foi observado que a inclinação (coeficiente angular) dos segmentos que compõem o feixe convexo de curvas ROC é equivalente a razão de verossimilhança que pode ser convertida para probabilidades. Essa conversão é denominada de calibração por feixe convexo de curvas ROC que coincidentemente é equivalente ao algoritmo PAV que implementa regressão isotônica. Esse método de calibração otimiza Brier Score. Ao explorar essa medida foi encontrada uma relação bastante interessante entre Brier Score e curvas ROC. Finalmente, também foram explorados os *rankings* construídos durante o método de seleção de exemplos do algoritmo de aprendizado semi-supervisionado multi-descrição CO-TRAINING.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AD Árvore de Decisão

AM Aprendizado de Máquina

AUC *Area Under ROC Curve*

EM *Expectation Maximization*

FN Falso Negativo

FP Falso Positivo

IA Inteligência Artificial

KDD *Knowledge Discovery in Databases*

LABIC Laboratório de Inteligência Computacional

LR *Likelihood Ratio*

MAP *Maximum a Posteriori*

MCAR *Missing Completely at Random*

MT Mineração de Texto

NB *Naive Bayes*

OD *Odds Ratio*

PAV *Pool Adjacent Violators*

PET *Probability Estimation Trees*

ROC *Receiver Operating Characteristic*

SVM *Support Vector Machines*

TFP Taxa de Falso Positivo

TVP Taxa de Verdadeiro Positivo

VN Verdadeiro Negativo

VP Verdadeiro Positivo

Lista de Algoritmos

CAPÍTULO

1

Introdução

Neste capítulo é apresenta uma descrição geral desta tese, com o objetivo de fornecer uma visão geral dos problemas tratados e dos objetivos principais do trabalho de pesquisa realizado. O capítulo está organizado da seguinte maneira: na Seção 1.1 é apresentada a motivação sobre o tema de pesquisa tratado nesta tese; na Seção 1.2 são apresentados os objetivos do trabalho em forma de questionamentos, que serão respondidos no decorrer dos capítulos; na Seção 1.3 são apresentados brevemente as principais contribuições deste trabalho; por fim, na Seção 1.4 é apresentada a organização da tese, com uma descrição resumida do conteúdo abordado em cada capítulo.

1.1 Motivação

Por muito anos, os algoritmos de aprendizado de máquina supervisionado tem sido avaliados com medidas simples como a precisão de classificação. Com o amadurecimento da área, verificou-se que analisar a qualidade de um classificador (ou hipótese) apenas com essas medidas não é suficiente para atestar a sua qualidade. Por exemplo, problemas do mundo real apresentam frequentemente grande desproporção entre as classes. Em casos mais comuns, pode-se ter um exemplo positivo a cada 100 exemplos negativos. Em casos extremos, pode-se encontrar um exemplo positivo a cada um milhão de exemplos negativos (Fawcett, 2006). Nessas situações, é muito fácil para qualquer algoritmo de aprendizado supervisionado induzir um classificador com alta precisão, simplesmente classificando novos exemplos com a classe majoritária. No primeiro caso, a precisão da classificação seria de 99% e no segundo caso de praticamente 100%. Porém, essa medida de precisão oculta

um fato muito importante: a precisão de classificação dos exemplos positivos, que na maioria dos problemas do mundo real representam os exemplos mais importantes para a classificação, é de praticamente 0%.

Assim, verificou-se que uma pergunta muito pertinente ao verificar a precisão de um classificador deveria ser: *qual é o erro da classe majoritária do classificador?* Em outras palavras, qual é o erro de classificação quanto todos os exemplos são classificados com a classe de maior freqüência. Ou seja, a precisão de um classificador precisa vir acompanhada de alguma informação sobre a proporção das classes para que o usuário do classificador não seja levado a acreditar em uma qualidade de classificação que não reflete a real qualidade dos classificadores.

Com esse objetivo, a comunidade de aprendizado começou a utilizar medidas, muitas delas provenientes de outras áreas de pesquisa com problema semelhante, para avaliar melhor os classificadores. Boa parte dessas medidas não tinham interpretação tão direta e natural como a precisão, requerendo sempre uma explicação dessas medidas para serem interpretadas corretamente.

Em problemas de duas classes, uma das maneiras mais completas de se analisar o desempenho de um classificador é por meio de análise ROC (Provost and Fawcett, 1997, 1998). Apesar de ser um modo de analisar o desempenho de classificação desde de a segunda guerra mundial, a análise ROC somente tem tido mais atenção da comunidade de aprendizado de máquina nos últimos 10 anos. Essa tendência de utilizar análise ROC (*Receiver Operating Characteristic*) vem motivando pesquisadores a utilizar e a explorar cada vez mais o valor de confiança de classificação. Valor esse que tem sido ignorado em grande parte dos trabalhos que avaliam algoritmos de aprendizado. Esse valor de confiança, muitas vezes, pode ser dado quase que gratuitamente ao usuário do classificador. Em outras palavras, o esforço computacional para rotular um exemplo, não é muito diferente de rotular um exemplo e fornecer o valor de confiança dessa rotulação. Desafortunadamente, são raros os trabalhos que exploram esse valor de confiança apropriadamente. Neste trabalho, tenta-se valorizar um pouco mais o que é possível fazer com esse valor de confiança por meio de *rankings*.

De modo sucinto, um *ranking* de exemplos nada mais é do que uma ordenação de um conjunto de exemplos utilizando esse valor de confiança. Os *rankings* são a principal entrada para desenhar uma curva ROC. Apesar de conceitualmente simples, as curvas ROC possuem algumas nuances não muito óbvias que podem levar a interpretações incorretas. Neste trabalho são mostradas e discutidas em detalhes as diversas informações que a curva ROC pode fornecer. É também mostrada uma maneira diferente de visualizar a in-

dução de uma árvore de decisão, que é uma linguagem simbólica de descrição de hipótese bastante simples, e que pode apresentar muito bons resultados utilizando curvas ROC.

Curvas ROC, podem também ser utilizadas para calibrar probabilidades, *i.e.*, transformar o valor de confiança de classificação em probabilidades. Os ganhos dessa transformação são bastante evidentes em problemas que requerem uma interpretação probabilística da classificação dos exemplos. Por exemplo, um médico pode querer saber qual é a probabilidade do seu paciente estar com gripe, sarampo ou catapora. Saber a probabilidade das possíveis doenças pode auxiliá-lo na tomada de decisão sobre o que fazer com o paciente.

Além dessas transformações e análises, existem uma infinidade de outras que podem ser exploradas utilizando *rankings* em aprendizado de máquina.

1.2 Objetivos

Como mencionado, os classificadores podem fornecer o valor de confiança da classificação, o qual pode ser utilizado para produzir *ranking* de exemplos. A obtenção do *ranking* possibilita o uso de outros técnicas, como calibração e análise ROC, que podem ser utilizadas para facilitar e melhorar a análise dos resultados obtidos. Aplicar essas técnicas ao resultado da classificação de um conjunto de exemplos agrega informações bastante interessantes. Com o objetivo de explorar ainda mais o que se pode fazer com um *ranking* de exemplos, neste trabalho são mostradas as relações existentes em *ranking*, análise ROC e calibração em aprendizado de máquina e são propostas contribuições para cada uma dessas relações. Mais especificamente, neste trabalho, nos concentramos em encontrar soluções para os seguintes problemas:

1. *Quais são as diferenças entre ranking e classificação? É possível desenvolver um algoritmo específico para ranking? Quais os benefícios que um algoritmo específico para ranking pode oferecer?*

2. *Quais são as informações que uma gráfico ROC pode fornecer? Existe alguma maneira de utilizar a curva ROC para ilustrar o processo de indução de hipótese de algoritmos de aprendizado? Quais algoritmos? Como?*

3. *Como a calibração se relaciona com curvas ROC? Como é essa relação com rankings? Quais ganhos podem ser obtidos com essas relações?*

4. *Por que o algoritmo* CO-TRAINING *é interessante para ranking e análise ROC? Como podem ser utilizadas em* CO-TRAINING*? Quais são os ganhos dessa união?*

1.3 Principais Contribuições

Diversas contribuições, descritas a seguir, foram realizadas durante a exploração dos objetivos mencionados:

1. É possível obter *rankings* a partir do valor de confiança de classificação de árvores de decisão e NAIVE BAYES. Foi observado que é possível desenvolver um algoritmo que cria uma estrutura lexicográfica que pode ser considerada como um algoritmo de aprendizado de *rankings*. A partir desse estudo de *rankings* foi desenvolvido um algoritmo denominado LEXRANK (Flach and Matsubara, 2007b);

2. A análise ROC, muitas vezes, deixa de ser utilizada por não ser bem compreendida. No capítulo sobre análise ROC são descritas diversas particularidades da análise ROC bem como as importantes informações que ela oferece. Além disso, foi observado que curvas ROC podem ser interessantes para serem estudadas em conjunto com árvores de decisão. Desse estudo resultou o desenvolvimento da ferramenta PROGROC (Flach, 2007) que ilustra a construção da árvore de decisão em curvas ROC;

3. A calibração utilizando o gráfico ROC é equivalente ao algoritmo que implementa regressão isotônica (Zadrozny and Elkan, 2002). Essa equivalência é bastante interessante e é explicada em detalhes neste trabalho. Uma outra contribuição é a decomposição que esses métodos de calibração otimizam. Essa decomposição possui interpretações diretas em gráficos ROC (Flach and Matsubara, 2007a);

4. A exploração do parâmetro que controla a proporção das classes de CO-TRAINING, um algoritmo de aprendizado semi-supervisionado multi-descrição, é importante para melhorar o seu desempenho, como mostrado em (Matsubara et al., 2006). Uma outra contribuição é a proposta de um algoritmo, que é uma variação de CO-TRAINING, para aliviar o problema de conjunto de exemplos com atributos com valores faltantes (Matsubara et al., 2008).

1.4 Organização

Os capítulos seguintes deste trabalho estão organizados de seguinte maneira:

Capítulo 2: Aprendizado de Máquina

Neste capítulo são apresentados conceitos de aprendizado de máquina e

é introduzida a notação e os termos utilizados. São também apresentados dois algoritmos de aprendizado bastante conhecidos na comunidade: NAIVE BAYES e árvores de decisão, os quais são utilizados no desenvolvimento do trabalho.

Capítulo 3: Ranking

Neste capítulo é definido *ranking* e suas diferenças com a classificação, bem como algumas maneiras simples de obter o valor de confiança de classificação de alguns algoritmos de aprendizado de máquina. Também é apresentado o conceito de *ranking lexicográfico* que pode ser utilizado tanto em NAIVE BAYES quanto em árvores de decisão. Esse conceito é a base do algoritmo LEXRANK.

Capítulo 4: Análise ROC

Neste capítulo são apresentado os conceitos básicos de análise ROC, mostrando as diversas informações que podem-se obter dessa técnica visual para avaliação, organização e seleção de classificadores. São também estabelecidas relações interessantes entre árvores de decisão e curvas ROC, e é apresentada a ferramenta PROGROC desenvolvida para mostrar graficamente essas relações.

Capítulo 5: Calibração

Neste capítulo são descritos dois métodos de calibração: a calibração de Platt e a regressão isotônica. É mostrado que a regressão isotônica é equivalente ao método de calibração utilizando gráficos ROC. Essa relação é explicada em detalhes, assim como uma decomposição do *Brier score* que tem interpretações em gráficos ROC.

Capítulo 6: Aplicando Análise ROC e Rankings em CO-TRAINING

Neste capítulo é apresentado um conjunto de experimentos com CO-TRAINING para explorar o parâmetro que controla a proporção das classes para rotular exemplos em cada iteração do algoritmo, bem como uma variação de CO-TRAINING, denominado CORAI, desenvolvido para a aplicação em problemas de conjunto de exemplos com valores faltantes em atributos.

Capítulo 7: Conclusões

Neste capítulo são apresentadas as conclusões deste trabalho e as propostas de trabalhos futuros.

CAPÍTULO

2

Aprendizado de Máquina

Neste capítulo são apresentados conceitos introdutórios de Aprendizado de Máquina (AM), tema ao qual está relacionado esta tese. O capítulo está organizado da seguinte maneira: na Seção 2.1 são feitas algumas considerações gerais sobre o aprendizado. Nos algoritmos tratados neste trabalho, o aprendizado é realizado utilizando um conjunto de exemplos. A linguagem de descrição desses exemplos é apresentada na Seção 2.2. Para ilustrar como alguns algoritmos de aprendizado fazem uso desses exemplos são apresentados dois algoritmos de aprendizado, *árvores de decisão* na Seção 2.3 e NAIVE BAYES na Seção 2.4, já bastante estudados e conhecidos pela comunidade. Finalmente, na Seção 2.5, são apresentadas algumas considerações finais deste capítulo.

2.1 Aprendizado de Máquina

Inteligência é um termo utilizado para descrever as diversas capacidades, muitas delas únicas, da mente humana. Essas capacidades tem sido objeto de discussões, tanto técnicas quanto filosóficas, há vários séculos. Entre essas capacidades, uma das mais fascinantes é o aprendizado. É essa capacidade que possibilita o homem adquirir novos conhecimentos. Segundo Dietterich (1990), o aprendizado pode ser definido como:

"Aprendizado é o aumento de conhecimento"

O aumento de conhecimento assume que o aprendiz já possui algum conhecimento, e aprender é agregar algo ao conhecimento já existente. Essa definição, apesar de ser bastante ampla, dá uma noção geral de aprendizado.

Uma definição um pouco mais específica é a de Michalski et al. (1986), que define o aprender da seguinte maneira:

> *"Aprender é construir ou modificar representações sobre o que está sendo experienciado"*.

Nessa definição existem dois termos chave: *representação* e *experiência*. A experiência pode acontecer de duas maneiras: por meio de estímulos sensoriais ou por processos internos de pensamentos. Os estímulos são meios utilizados pelo sistema de aprendizado para perceber a realidade, a qual está representada de alguma maneira no aprendiz. O processo interno de pensamento pode ser também considerado experiência, pois ela é capaz de criar representações novas a partir de conhecimento já existente.

Quando um conhecimento é bem representado se tem o aprendizado (Michalski et al., 1986). A qualidade dessa representação influencia a qualidade do aprendizado. Por exemplo, o comportamento inteligente está relacionado as ações tomadas utilizando essa representação. Quando o conceito está representado de modo adequado, as ações tomadas utilizando essa representação normalmente são melhores, *i.e.* o aprendiz executa sua tarefa melhor e pode-se dizer que ele aprendeu melhor. Avaliar objetivamente a qualidade dessa representação é difícil. Porém, é possível avaliar a qualidade das ações tomadas.

Ambos os termos chave, experiência e representação, podem ocorrer de modo bastante complexo nos seres humanos. As experiências humanas podem ocorrer por meio dos estímulos sensoriais, visão, audição, tato e paladar, que são combinadas das mais diferentes maneiras com diferentes graus de intensidade, além de uma infinidade de outras variações que podem ocorrer simultaneamente ou em tempos diferentes. A outra maneira de experimentação, o processo interno de pensamento, pode ser dada pelo raciocínio que pode ocorrer de modo bastante complexo em nossas mentes. A experiência humana tem início antes do próprio nascimento da pessoa e a cada dia vivido são acumuladas cada vez mais experiências.

Compreender como a mente humana representa essas experiências na forma de conhecimento é objeto de estudo de diversas áreas de pesquisa que vão de física à ciências cognitivas. Os cientistas que atuam nessas áreas raramente concordam sobre os modelos de representação do conhecimento utilizados em cada caso pelos seres humanos. Existem diversos modelos e cada um possui pontos fortes e fracos. Normalmente, diferentes problemas requerem diferentes representações (Markman, 1998). Provavelmente não existe um modelo único que seja ótimo para todos os casos.

Compreender como o aprendizado acontece é importante para reproduzi-lo, e tornar isso realidade constitui um dos grandes desafios atuais. A área

de pesquisa que enfrenta esse desafio é conhecida como Aprendizado de Máquina. Originalmente, essa área de pesquisa surgiu na tentativa de reproduzir o aprendizado humano. Nos dias atuais busca-se por métodos de aprendizado não necessariamente inspirados na mente humana. Muitas vezes, em busca de soluções pragmáticas para problemas reais, diversas teorias elegantes sobre o aprendizado acabaram por competir com idéias *ad hoc*, intuições e suposições que são muitas vezes mais eficientes e mais simples (Kuncheva, 2004).

Geralmente, os algoritmos de aprendizado de máquina tem como entrada um conjunto de *exemplos* que podem ser considerados como representações dos estímulos sensoriais para a máquina. Um *exemplo* é como uma unidade individual e independente para aprender o conceito desejado. Uma criança que está aprendendo o conceito da fruta manga, a cada manga apresentada, os sentidos da criança podem distinguir o formato geométrico, cheiro e sabor de cada fruta. Cada manga apresentada é uma unidade individual e independente, a qual pode ser considerada como um exemplo para a criança aprender o conceito. O conceito que se deseja aprender é geralmente denominado de classe.

Considerando a inferência realizada sobre as informações disponíveis, como os exemplos, existem diversas estratégias de aprendizado de máquina, tais como: hábito, instrução, dedução, analogia e indução. A estratégia de aprendizado por hábito é a mais simples, enquanto que as estratégias de aprendizado por analogia e indução são as de maior complexidade, ou seja, o aprendiz necessita de maior esforço para o aprendizado.

A indução é a forma de inferência lógica que permite obter conclusões genéricas partindo de um conjunto particular de exemplos ou casos previamente observados. É caracterizada como o raciocínio que parte do específico para o geral, do particular para o universal, da parte para o todo. O resultado da inferência indutiva nem sempre preserva a verdade e por isso é normalmente denominado de *hipótese*. Mesmo assim, a inferência indutiva é um dos principais métodos utilizados para descobrir conhecimento novo e predizer eventos futuros. A maioria dos algoritmos de aprendizado utilizam a indução para a construção de *hipóteses*.

2.2 Linguagem de Descrição dos Exemplos e Terminologia

Diversas linguagens podem ser utilizadas para representar os exemplos usados pelos algoritmos de aprendizado, as quais apresentam diferente complexidade e poder de descrição. A linguagem proposicional é amplamente uti-

lizada pela maioria dos algoritmos de aprendizado. Nessa linguagem, utilizada neste trabalho, os exemplos são descritos utilizando um conjunto de atributos, os quais podem assumir valores diferentes, e cada exemplo é representado em uma linha de uma tabela no formato atributo-valor.

Na Tabela 2.1 é apresentado o formato geral de uma tabela no formato atributo-valor com n_{ex} exemplos $\mathbf{x}_i$ com $i = 1,...,n_{ex}$, na forma $\{(\mathbf{x}_1, y_1),...,(\mathbf{x}_{n_{ex}}, y_{n_{ex}})\}$. Os $\mathbf{x}_i$ são vetores da forma $(x_{i1}, x_{i2},..., x_{in_{at}})$, com valores contínuos, discretos ou booleanos. Assim, x_{ij} refere-se ao valor do atributo j, denominado A_j, do exemplo $\mathbf{x}_i$. O espaço de exemplos sobre um conjunto de atributos $\{A_1,\ldots,A_{n_{at}}\}$ é definido como $X = A_1 \times \ldots \times A_{n_{at}}$. Os valores y_i, os quais podem ou não existir, referem-se ao valor do atributo A_{classe}, denominado atributo *classe*. No caso de classificação, os valores do atributo classe A_{classe} pertencem a um conjunto Y de classes discretas y_v com $v = 1,...,n_{cl}$.

	A_1	A_2	$\ldots$	$A_{n_{at}}$	A_{classe}
$\mathbf{x}_1$	x_{11}	x_{12}	$\vdots$	$x_{2n_{at}}$	y_1
$\mathbf{x}_2$	x_{21}	x_{22}	$\vdots$	$x_{2n_{at}}$	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$\mathbf{x}_{n_{ex}}$	$x_{n_{ex}1}$	$x_{n_{ex}2}$	$\vdots$	$x_{n_{ex}n_{at}}$	$y_{n_{ex}}$

Tabela 2.1: Exemplos no formato atributo valor

Geralmente o valor do atributo classe é obtido baseado em observações passadas ou é fornecido por um especialista de domínio. Quando o conjunto de exemplos está rotulado, *i.e.*, possui o atributo classe, ele é denominado de *conjunto rotulado*, caso contrário é denominado de *conjunto não rotulado*.

Dado um conjunto de exemplos, é importante distinguir os seguintes três conjuntos utilizados no processo de aprendizado:

Conjunto de treinamento: esse conjunto é a principal entrada dos algoritmos de aprendizado. É a partir dele que são construídos os modelos (hipóteses) e, portanto, ele deve ser representativo da distribuição da população para realizar inferência sobre esses dados. Na literatura, esse conjunto também é conhecido como *seen cases*, pois refere-se aos exemplos que foram "vistos" pelo algoritmo de aprendizado durante a construção do modelo.

Conjunto de teste: esse conjunto é utilizado para avaliar o modelo construído. Esse conjunto, também conhecido como *unseen cases*, não deve ser apresentado ao algoritmo de aprendizado durante a construção do modelo. Idealmente, esse conjunto não deveria ter exemplos em comum com o conjunto de treinamento.

Conjunto de validação: em alguns casos, pode ser necessário utilizar exemplos para realizar ajustes no modelo construído pelo algoritmo de aprendizado. Esses exemplos não são utilizados diretamente na construção do modelo, mas são utilizados para o seu ajuste. Dessa maneira, esses exemplos são indiretamente "vistos" durante o processo de aprendizado, o que obriga que os exemplos de validação sejam distintos dos exemplos de teste.

Muitos algoritmos de aprendizado têm sido propostos na literatura, os quais podem ser agrupados utilizando diversos critérios (Mitchell, 1997). Neste trabalho usamos o critério do grau de supervisão presente nos dados para agrupar esses algoritmos.

Como mencionado, o conjunto de exemplos fornecido a um algoritmo de aprendizado pode estar constituído de exemplos rotulados com uma classe. No caso de conjunto de exemplos rotulados, um outro aspecto importante é a quantidade de exemplos para os quais o valor do atributo classe é conhecido. Quanto maior é a quantidade/proporção de exemplos rotulados, maior é o grau de supervisão. De acordo com esse critério, que leva em consideração o grau de supervisão nos dados, três tipos de aprendizado podem ser distinguidos, os quais são ilustrados na Figura 2.1:

- **Supervisionado**: utilizado quando existe um número expressivo de exemplos rotulados;
- **Não-supervisionado**: utilizado quando os exemplos não estão rotulados;
- **Semi-supervisionado**: utilizado quando existem exemplos rotulados e não-rotulados e ambos são utilizados para construir um classificador.

No aprendizado semi-supervisionado, o conjunto de treinamento consiste de um conjunto de exemplos rotulados e não rotulados enquanto que o aprendizado supervisionado utiliza apenas o conjunto de exemplos rotulados. O classificador obtido pelo aprendizado semi-supervisionado é induzido para predizer a classe de qualquer exemplo pertencente ao espaço de exemplos X, *i.e.*, o algoritmo deve ser capaz de predizer a classe de qualquer exemplo de $A_1 \times \ldots \times A_{n_{at}}$ (Chapelle et al., 2006). Existe também um tipo de aprendizado parecido com o aprendizado semi-supervisionado, conhecido como aprendizado *transdutivo*, proposto por Vapnik (1998), no qual o conjunto de treinamento também consiste de exemplos rotulados e não rotulados. Entretanto, o classificador é induzido para predizer a classe dos exemplos do conjunto de exemplos não rotulados dados como exemplos de treinamento, ao invés de quaisquer exemplos de X.

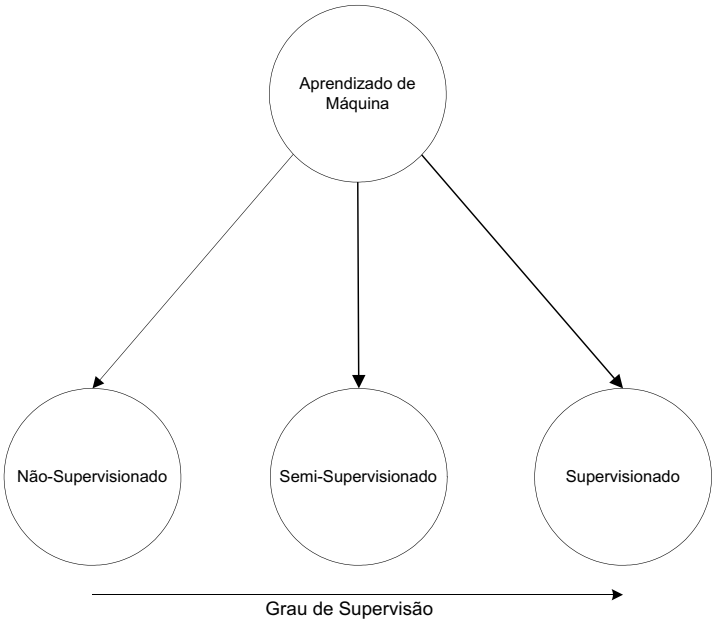


Figura 2.1: Hierarquia do aprendizado segundo o grau de supervisão dos dados

Existem alguns termos utilizados neste trabalho, descritos a seguir, que devem ser distinguidos (Russell and Norvig, 2002; Flach and Matsubara, 2007b; Mitchell, 1997; Witten and Frank, 2005):

Indutor o indutor é um algoritmo de aprendizado que utiliza um processo indutivo para gerar a sua hipótese ou modelo. Essa hipótese é utilizada para classificar exemplos. Exemplos de indutores são NAIVE BAYES, árvores de decisão e regressão logística.

Score algoritmos indutores podem gerar classificadores que utilizam um valor de confiança na sua predição para determinar a classe de um exemplo, esse valor de confiança é denominado de *score*. Exemplos de *scores* são a probabilidade a posteriori fornecida por NAIVE BAYES, a porcentagem de exemplos de treinamento positivos em uma folha de uma árvore de decisão e a soma das distâncias dos k vizinhos mais próximos no algoritmo k-NN, entre outros.

Ranking dados os *scores* de um conjunto de exemplos, pode-se ordená-los. A ordenação desse conjunto de exemplos é denominada *ranking*. O *ranking* determina o posicionamento dos exemplos nesse *ranking*, independentemente do valor do *score*.

Classificador *Score* a principal saída do classificador *score* são os *scores*, relacionados à classificação de seus exemplos. Exemplo de classificador *score* é o induzido por NAIVE BAYES, o qual utiliza sua estimativa de probabilidade a posteriori para determinar a classe dos exemplos.

Classificador diferentemente do classificador *score*, a principal saída do classificador é a classe dos exemplos. Um classificador *score* pode ser facilmente convertido em um classificador. Para isto, basta determinar um limiar que define a classe. Por exemplo, classificar como positivo todo exemplo cujo *score* > 0.5, negativo caso contrário.

Overfitting quando um classificador é induzido, é possível que ele seja muito específico para o conjunto de treinamento, *i.e.* apresenta um alto grau de precisão para o conjunto de exemplos de treinamento e uma alta taxa de erro para o conjunto de teste. Nesse caso, pode-se dizer que o classificador "decorou" os dados, não conseguindo generalizar o conceito. Essa situação é conhecida como *overfitting*.

underfitting por outro lado, também é possível que o conjunto de treinamento seja composto por exemplos pouco representativos, ou que o usuário pré defina o tamanho do classificador como muito pequeno (por exemplo, um alto fator de poda[1] para uma árvore de decisão) ou uma combinação de ambos os casos. Nesse caso pode-se dizer que o classificador não conseguiu abstrair o conceito, apresentando baixa performance no conjunto de treinamento e no conjunto de teste. Essa situação é conhecida como *underfitting*.

Existem dois algoritmos de aprendizado de máquina supervisionado, árvores de decisão e NAIVE BAYES, que inspiraram o algoritmo de aprendizado proposto neste trabalho, além de serem utilizados em combinação com outros métodos. Com o objetivo de facilitar a compreensão dessas propostas, árvores de decisão e NAIVE BAYES, são apresentados em detalhes.

2.3 Árvores de Decisão

A construção da árvore de decisão pode ser descrita por um procedimento recursivo, no qual a idéia é dividir o conjunto de dados em subconjuntos de exemplos cada vez mais "puros" em termos de classe. Essa divisão é feita utilizando em cada nível da árvore os atributos que melhor separam as classes.

Para ilustrar a construção de uma árvore de decisão, considere o conjunto de dados *palestra*, descrito na Tabela 2.3, que representa alguns atributos sobre palestras e a decisão de um aluno assistir ou não a essas palestras oferecidas em seu departamento. Esse conjunto possui os seguintes atributos: relevância da palestra (alta ou baixa), se foi oferecida comida após a palestra

[1] Poda em árvores de decisão consiste na exclusão de algumas ramificações da árvore, tornando-a mais "simples".

(sim ou não), a distância da sala onde a pessoa se encontra e o local da palestra (longe ou perto), se o apresentador sorteou brindes no final (sim ou não), e o atributo classe (sim ou não) que representa a decisão ou não do aluno assistir a palestra.

Tabela 2.2: Conjunto de dados *palestra*

palestra	*relevância*	*comida*	*distância*	*brinde*	*classe*
1	alta	sim	perto	sim	sim
2	alta	não	perto	não	sim
3	alta	sim	perto	sim	sim
4	alta	não	perto	não	sim
5	alta	sim	perto	sim	sim
6	alta	não	perto	não	sim
7	alta	sim	perto	sim	não
8	alta	não	longe	não	sim
9	alta	sim	longe	sim	não
10	alta	não	longe	não	não
11	baixa	sim	perto	sim	sim
12	baixa	sim	longe	não	sim
13	baixa	sim	perto	sim	sim
14	baixa	sim	longe	não	não
15	baixa	não	perto	sim	não
16	baixa	não	longe	não	não
17	baixa	não	perto	sim	não
18	baixa	não	longe	não	não
19	baixa	não	perto	sim	não
20	baixa	não	longe	não	não

Na Figura 2.2 é ilustrada uma possível árvore de decisão que pode ser obtida utilizando o conjunto de dados *palestra*. A árvore classifica os exemplos testando as condições começando pelo nó raiz da árvore até chegar em um nó folha que classifica o exemplo.

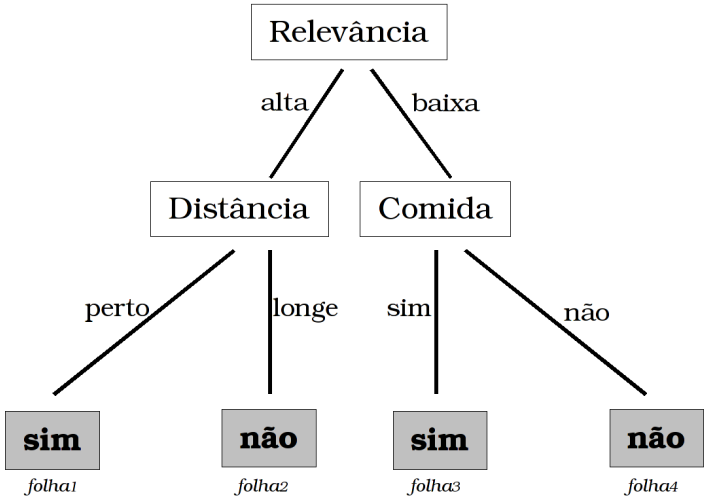


Figura 2.2: Árvore de decisão induzida utilizando o conjunto de dados *palestra*

Um caminho começando pelo nó raiz até um nó folha corresponde a uma conjunção de restrições. O caminho entre o nó raiz e a *folha1*, por exemplo, corresponde a expressão:

$$\textit{relevância} = \text{alta} \wedge \textit{distância} = \text{perto}$$

De modo geral, as árvores de decisão representam disjunções de conjunções de restrições. Por exemplo, a classe "sim" da árvore de decisão ilustrada na Figura 2.2 corresponde a expressão:

$$(\textit{relevância} = \text{alta} \wedge \textit{distância} = \text{perto}) \vee (\textit{relevância} = \text{baixa} \wedge \textit{comida} = \text{sim})$$

Esse tipo de representação é bastante útil, pois permite representar uma árvore de decisão como disjunções de regras, facilitando a leitura e interpretação da árvore induzida.

Árvores de decisão particionam o espaço de exemplos de tal modo que os subconjuntos de exemplos se tornem cada vez mais "puros", *i.e.*, possuindo a mesma classe. Isso é feito particionando o espaço de exemplos escolhendo estrategicamente o atributo que "melhor" separa os exemplos. Observe que no conjunto de exemplos *palestra*, o atributo *relevância* pode separar os exemplos em dois subconjuntos determinados pelos valores "alta" e "baixa", um composto pelas palestras de 1 a 10 e outro pelas palestras de 11 a 20, respectivamente. No subconjunto determinado pelos valores "alta" existem 70% dos exemplos da classe sim, no segundo subconjunto, 70% dos exemplos pertencem a classe não. O atributo *brinde* também pode separar os exemplos em dois subconjuntos, mas ambos os subconjuntos possuem 50% de exemplos pertencentes a classe sim. Desse modo, o atributo *relevância* separa exemplos em subconjuntos com maior porcentagem de pertencer a uma classe (mais "puros") do que o atributo *brinde*.

A "pureza" dos subconjuntos obtidos pelos atributos é medida comparando a impuridade do nó pai com a impuridade ponderada dos filhos. A medida mais conhecida para impuridade é a baseada na entropia de Shannon (Shannon, 1948). A entropia para problemas de classe binária para um conjunto de exemplos S pode ser definida pela Equação 2.1

$$\textit{Entropia}(S) = -p_+ log_2 p_+ - p_- log_2 p_- \tag{2.1}$$

na qual p_+ é a proporção de exemplos positivos em S e $p_- = 1 - p_+$ é a proporção de exemplos negativos.

Uma explicação clássica para entropia binária pode ser dada da seguinte maneira: considere que a probabilidade de uma moeda, não necessariamente justa, de ser cara ou coroa, seja conhecida. A entropia do próximo lançamento de uma moeda é maximizada quando a moeda é justa, *i.e.*, 50% de chance de ser cara e 50% de chance de ser coroa. Nessa situação, a incerteza é máxima

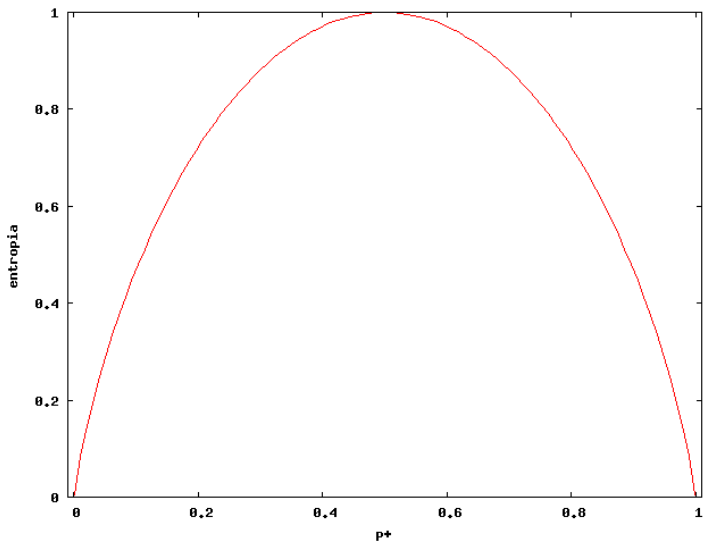


Figura 2.3: Entropia

e, desse modo, é a situação mais difícil de prever. Para codificar a informação de qual será o próximo lançamento é necessário 1 bit (entropia igual a 1). Nos casos extremos, quando a moeda possui a mesma face dos dois lados, o grau de incerteza é zero, pois cada lançamento da moeda não contém nenhuma informação, requer 0 bit (entropia igual a zero). A entropia é uma medida de incerteza associada a uma variável randômica. Quando um conjunto de exemplos é "puro", a incerteza é zero. Por isso, a entropia também é conhecida como uma medida de impuridade.

A Figura 2.3 ilustra a Equação 2.1 na qual é variada a proporção de exemplos positivos. Observe que a entropia atinge seu valor máximo quando a p_+ em um conjunto de exemplos é igual a 0.5 e atinge o seu valor mínimo quando a p_+ é igual a 0 ou 1.

O conjunto de exemplos *palestra* possui 20 exemplos: 10 exemplos positivos e 10 exemplos negativos. A seguir é adotada a notação [10+,10-] para sumarizar o número de exemplos por classe. Assim, a entropia para esse conjunto é dada por

$$\begin{aligned} Entropia([10+,10-]) &= -(10/20) \times log_2(10/20) - (10/20) \times log_2(10/20) \\ &= -0.5 \times -1 - 0.5 \times -1 \\ &= 1 \end{aligned}$$

Utilizando o conceito de entropia, o ganho de informação mede a redução esperada da entropia causada pelo particionamento dos exemplos. O ganho de informação é definido pela Equação 2.2 onde; S é um conjunto de exemplos; A_i é um atributo; v é um valor do atributo e $atributo_v$ é o subconjunto de S no

qual o valor desse *atributo* é *v*.

$$GanhoInfo(S, atributo) = Entropia(S) - \sum_{v \in Dom(atributo)} \frac{|atributo_v|}{|S|} \times Entropia(atributo_v) \tag{2.2}$$

Inicialmente, para a construção da árvore de decisão, o algoritmo avalia cada atributo como ilustrado na Figura 2.4. Cada valor de atributo divide os exemplos em subconjuntos. Por exemplo, o subconjunto $distância_{perto}$ possui [8+,5-] exemplos e o subconjunto $distância_{longe}$ possui [2+,5-]. O cálculo do ganho de informação para esse atributo é dado por:

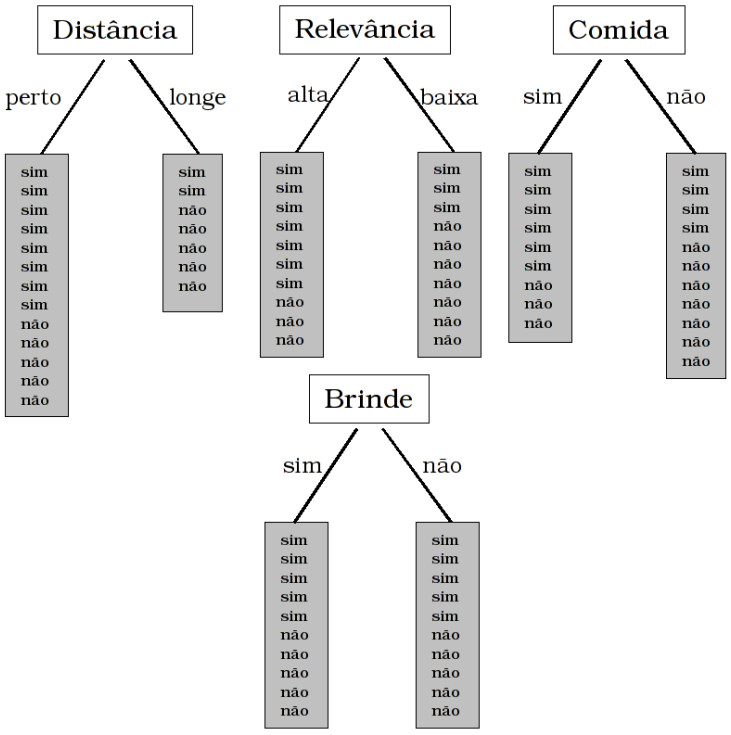


Figura 2.4: Conjunto *palestra*: árvore de decisão parcial (*decision stumps*)

$$\begin{aligned} GanhoInfo([10+,10-], \textit{distância}) =& Entropia([10+,10-]) - \sum_{v \in \{perto,longe\}} \frac{|S_v|}{|S|} \times Entropia(S_v) \\ =& Entropia([10+,10-]) - (13/20) \times Entropia([8+,5-]) \\ & - (7/20) \times Entropia([2+,5-]) \\ =& 1 - (13/20) \times 0.961 - (7/20) \times 0.863 \\ =& 0.07 \end{aligned}$$

Calculando para os outros atributos obtém-se

$$GanhoInfo([10+,10-], \textit{relevância}) = 0.22$$
$$GanhoInfo([10+,10-], \textit{comida}) = 0.06$$
$$GanhoInfo([10+,10-], \textit{brinde}) = 0.00$$

O atributo com maior ganho de informação é *relevância*. Assim, o atributo *relevância* é utilizado para fazer o particionamento. Ao expandir o nó que representa $\textit{relevância}_{alta}$, devem ser realizados os mesmos cálculos com os exemplos que possuem o valor $alta$ no atributo *relevância*.

$$GanhoInfo([7+,3-], \textit{distância}) = 0.19$$
$$GanhoInfo([7+,3-], \textit{comida}) = 0.03$$
$$GanhoInfo([7+,3-], \textit{brinde}) = 0.03$$

O atributo com maior ganho de informação é *distância*. A árvore com o ramo $\textit{relevância}_{alta}$ expandido é ilustrado na Figura 2.5. O próximo ramo a ser expandido é $\textit{relevância}_{baixa}$.

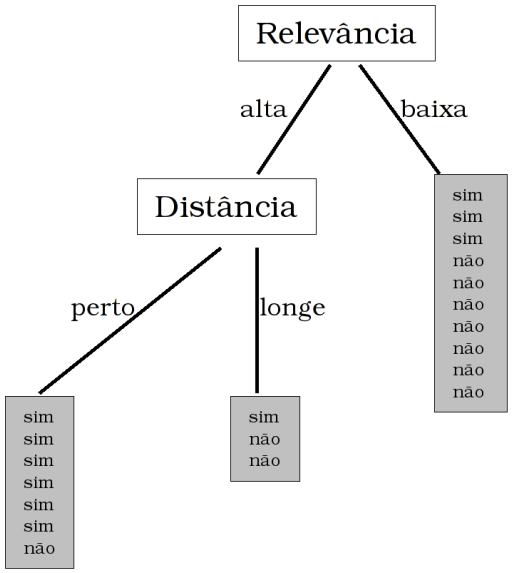


Figura 2.5: Conjunto *palestra*: expandindo a sub-árvore esquerda

Calculando os ganhos de informação para expandir $\textit{relevância}_{baixa}$ obtém-se:

$$GanhoInfo([3+,7-], \textit{distância}) = 0.03$$
$$GanhoInfo([3+,7-], \textit{comida}) = 0.55$$
$$GanhoInfo([3+,7-], \textit{brinde}) = 0.03$$

O atributo com maior ganho de informação é *comida*. Expandindo o ramo *comida* obtém-se a árvore ilustrada na Figura 2.6. Observe que a cada divisão os nós ficam mais "puros", com apenas exemplos de uma classe, ou com a maioria dos exemplos de uma classe.

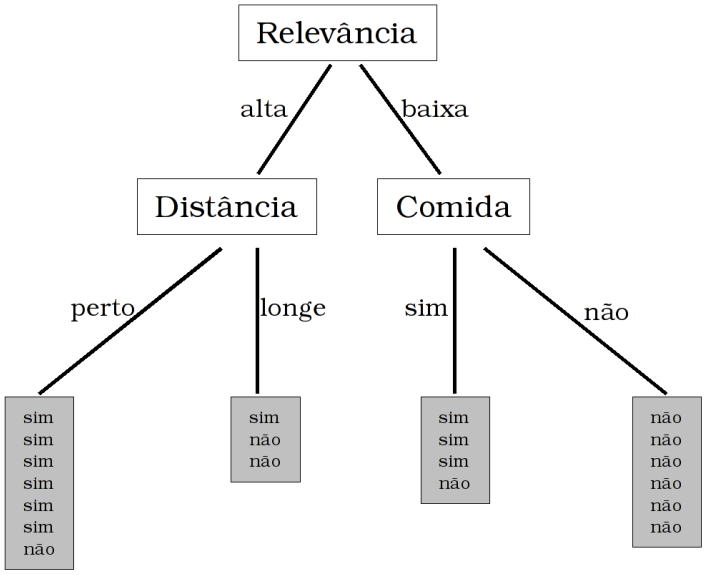


Figura 2.6: Conjunto *palestra*: expandindo a sub-árvore direita

A construção da árvore pode continuar indefinidamente até que o ganho de informação praticamente seja zero. No entanto, quando isso acontece a árvore tende a tornar-se muito específica para o conjunto de treinamento. Nesse caso, como mencionado anteriormente, a árvore apresenta um alto grau de precisão para os exemplos de treinamento e uma alta taxa de erro para o conjunto de teste, caracterizando o *overfitting*. Em geral, é definido um limiar do critério de particionamento para determinar se a construção da árvore continua ou interrompe naquele ponto. Mesmo assim, pode ocorrer *overfitting* e após a árvore construída pode-se fazer uma poda. Uma técnica conhecida para evitar o *overfitting* é a poda. A poda consiste em fazer uma validação cruzada interna que poda a árvore enquanto o erro do conjunto de teste for diminuindo, quando o erro começa a aumentar o processo de poda é interrompido.

Existem diversas outras medidas de impuridades como o índice Gini (Breiman et al., 1984), DKM (Kearns and Mansour, 1996), razão de chances (*odds ratio*), AUC (Ferri et al., 2002) e diversos outros. Flach (2003) fez um estudo comparando ganho de informação, Gini-split e DKM-split e mostra que o ganho de informação é mais sensível a problemas de classes desbalanceadas que o Gini-split. Mostra também e que é possível derivar versões insensíveis à desproporção das classes das medidas DKM e Gini. Isso pode ser visualizado por meio de curvas de iso-desempenho em gráficos ROC, ilustrado mais adiante.

Uma das grandes vantagens de árvore de decisão é a facilidade de interpretação do conceito (hipótese) por ela representado, sendo assim amplamente utilizada em mineração de dados e em tarefas que requerem a validação semântica da hipótese obtida. Uma outra vantagem é que a classificação de exemplos usando uma árvore de decisão não requer a avaliação de todos os atributos que compõem o exemplo, mas requer apenas a avaliação dos atributos que compõem a árvore. Devido a isso, o classificador expresso na linguagem de árvores de decisão é um dos mais rápidos entre os algoritmos de aprendizado existentes.

A árvore de decisão pode ser convertida em um estimador de probabilidade ao utilizar a proporção de exemplos positivos nos nós folhas. Essas árvores são conhecidas na literatura como PETs (*Probability Estimation Trees*). Entretanto, a probabilidade estimada por essas árvores deixa a desejar se comparada com outros algoritmos que também estimam probabilidades. Isso ocorre devido ao viés indutivo do algoritmo de árvore de decisão que privilegia árvores com altura limitada. Porém, existem algumas pequenas alterações que podem tornar a árvore em um bom estimador de probabilidades. Provost and Domingos (2003) descobriram que o uso de suavizadores na freqüência das folhas pode melhorar o desempenho da árvore em termos de estimação de probabilidades. Ferri et al. (2002) sugerem algumas outras modificações para tornar a árvore em um melhor estimador, tais como um critério de particionamento baseado em curvas ROC. Uma outra estratégia é desativar o mecanismo de poda da árvore, com isso evita-se árvores de pequena altura.

2.4 NAIVE BAYES

Os métodos Bayesianos oferecem uma abordagem probabilística para a inferência de hipóteses. O aprendizado bayesiano utiliza evidências durante a inferência que permitem associar uma probabilidade à hipótese gerada. Uma vez que cada exemplo pode reforçar ou refutar uma hipótese, esse tipo de aprendizado possibilita um modo flexível para, ao invés de simplesmente descartar ou aceitar a hipótese, diminuir ou aumentar a probabilidade associada a essa hipótese. Os algoritmos de aprendizado de máquina geralmente estão interessados em encontrar a *melhor* hipótese dado um conjunto de exemplos de treinamento. A *melhor* hipótese em aprendizado bayesiano é a hipótese mais provável (Mitchell, 1997).

O NAIVE BAYES é um método de aprendizado Bayesiano no qual a classificação de um novo exemplo é dada pela probabilidade máxima a posteriori (*maximum a posteriori* ou simplesmente MAP) v_{MAP} considerando os valores dos atributos que descrevem os exemplos a serem classificados — Equação

2.4.

$$v_{MAP} = \arg\max_{y_v \in Y} P(y_v|\mathbf{x}_i) \tag{2.3}$$

$$= \arg\max_{y_v \in Y} \frac{P(\mathbf{x}_i|y_v).P(y_v)}{P(\mathbf{x}_i)} \tag{2.4}$$

$P(y_v)$ pode ser estimada observando a freqüência com que cada valor $y_v \in Y$ ocorre nos exemplos de treinamento. Por outro lado, estimar $P(x_{i1}, x_{i2} \ldots x_{in_{at}}|y_v)$ com base na sua freqüência no conjunto de exemplos de treinamento não é viável, uma vez que é necessário que se conheça a probabilidade condicional para quaisquer possíveis valores de cada atributo. Por exemplo, em um conjunto de dados com 10 atributos binários, é necessário estimar $2^{10} - 1$ probabilidades condicionais. Assim, para o caso geral de um conjunto de treinamento com n_{at} atributos binários a complexidade da estimativa das probabilidades condicionais é $O(2^{n_{at}})$. Ainda que o conjunto de exemplos seja muito grande, de maneira que se possa estimar cada uma dessas probabilidades com alguma confiança, essa abordagem é inviável.

Para a determinação da classe mais provável de um exemplo qualquer, o algoritmo NAIVE BAYES considera que os valores dos atributos são condicionalmente independentes. Isto é, dado um valor alvo da classe de um exemplo, o NB supõe que a probabilidade de se observar o exemplo $\mathbf{x}_i = (x_{i1}, x_{i2} \ldots x_{in_{at}})$, é dada pelo produto das probabilidades de cada atributo individual:

$$\begin{aligned} P(x_{i1}, x_{i2} \ldots x_{in_{at}}|y_v) &= P(x_{i1}|y_v)p(x_{i2}|y_v) \ldots P(x_{in_{at}}|y_v) \\ &= \prod_{j=1}^{n_{at}} P(x_{ij}|y_v) \end{aligned}$$

Assim, a hipótese gerada por NAIVE BAYES pode ser definida pela Equação 2.5.

$$v_{NB} = \arg\max_{y_v \in Y} P(y_v) \prod_{j=1}^{n_{at}} P(x_{ij}|y_v) \tag{2.5}$$

Quando a independência condicional assumida pelo NAIVE BAYES é satisfeita pelo conjunto de exemplos de treinamento, a classificação v_{NB} é idêntica à classificação v_{MAP}. Como pode ser observado na Equação 2.5, o número distinto de termos $P(x_{ij}|y_v)$ que deve ser calculado usando os exemplos de treinamento, é dado apenas pelo número de atributos e valores desses atributos multiplicado pelo número de classes. Portanto, muito menor que a complexidade para estimar $P(x_{i1}, x_{i2} \ldots x_{in_{at}}|y_v)$.

Considere um novo exemplo do domínio do conjunto *palestra*, tal que, o orientador do aluno gostaria de saber se seu orientando irá ou não assistir essa nova palestra:

$$(\textit{relevância} = \text{baixa}, \textit{comida} = \text{sim}, \textit{distância} = \text{longe}, \textit{brinde} = \text{sim})$$

Instanciando os x_{ij} da Equação 2.5 para calcular v_{NB} obtém-se:

$$v_{NB} = \arg\max_{y_v \in Y} P(y_v) \times P(\textit{relevância} = \text{baixa}|y_v) \times P(\textit{comida} = \text{sim}|y_v) \times P(\textit{distância} = \text{perto}|y_v) \times P(\textit{brinde} = \text{sim}|y_v)$$

Utilizando o conjunto *palestra* (Tabela 2.3 na página 14) como conjunto de treinamento para NAIVE BAYES, as probabilidades a priori podem ser estimadas utilizando a freqüência obtida dos 20 exemplos da tabela:

$$P(\textit{classe} = \text{sim}) = \frac{10}{20} = 0.5$$
$$P(\textit{classe} = \text{não}) = \frac{10}{20} = 0.5$$

De modo similar, pode-se estimar as probabilidades condicionais $P(x_{ij}|y_v)$ utilizando as freqüências obtidas do conjunto *palestra*. Por exemplo, para calcular $\textit{relevância} = \text{baixa}$, tem-se:

$$P(\textit{relevância} = \text{baixa}|\textit{classe} = \text{sim}) = \frac{3}{10} = 0.3$$
$$P(\textit{relevância} = \text{baixa}|\textit{classe} = \text{não}) = \frac{7}{10} = 0.7$$

Utilizando essas probabilidades e estimando as probabilidades para os atributos restantes (os atributos estão na ordem apresentada na Tabela 2.3 e os nomes dos atributos foram omitidos para facilitar a leitura) tem-se:

$$P(\text{sim}) \times P(\text{baixa}|\text{sim}) \times P(\text{sim}|\text{sim}) \times P(\text{perto}|\text{sim}) \times P(\text{sim}|\text{sim}) = \frac{10}{20} \times \frac{3}{10} \times \frac{6}{10} \times \frac{8}{10} \times \frac{5}{10} = 0.36$$

$$P(\text{não}) \times P(\text{baixa}|\text{não}) \times P(\text{sim}|\text{não}) \times P(\text{perto}|\text{não}) \times P(\text{sim}|\text{não}) = \frac{10}{20} \times \frac{7}{10} \times \frac{3}{10} \times \frac{4}{10} \times \frac{5}{10} = 0.02$$

Como $0.36 > 0.021$, o NAIVE BAYES atribui a $\textit{classe} = \text{sim}$ para o novo exemplo apresentado. Uma estratégia bastante utilizada nas implementações de NAIVE BAYES é a normalização dos valores obtidos. Isso é feito para se ter uma

melhor idéia comparativa do exemplo ser positivo ou negativo. Para o exemplo considerado o resultado normalizado seria $\frac{0.36}{0.36+0.02} = 0.94$ para o exemplo pertencer a $classe = \text{sim}$ e 0.06 para o exemplo pertencer a $classe = \text{não}$.

Como todos os $P(x_{ij}|y_v)$ são valores menores que um, a multiplicação de dois valores menores que um é sempre um valor menor. Assim, quando existem muitos atributos, o resultado da multiplicação desses valores se torna muito pequeno. Nesses casos, normalmente, quando os valores são normalizados, é bastante comum que um dos valores fique muito próximo dos extremos zero ou um. Algumas implementações realizam cálculos em espaço logarítmico para que essa multiplicação se torne uma soma, e o valor não tende a ser tão pequeno, facilitando a comparação.

Em suma, o método de aprendizado usado pelo NAIVE BAYES estima várias probabilidades, tais como $P(y_v)$ e $P(x_{ij}|y_v)$, usando as freqüências observadas nos exemplos de treinamento. O conjunto dessas probabilidades estimadas corresponde ao aprendizado de uma hipótese, a qual é usada para classificar novos exemplos.

É importante ressaltar que uma característica do NAIVE BAYES é que seu espaço de hipótese é restrito, pois não são consideradas as hipóteses com dependência entre os atributos. O espaço de busca, nesse caso, é o espaço de possíveis valores que podem ser atribuídos aos termos $P(y_v)$ e $P(\mathbf{x}_i|y_v)$.

2.5 Considerações Finais

Neste capítulo foram apresentados alguns conceitos introdutórios sobre aprendizado de máquina, a notação utilizada para descrever os exemplos bem como dois algoritmos utilizados para explorar algumas idéias propostas neste trabalho.

Apesar de originalmente o aprendizado de máquina tem surgido como uma na tentativa de reproduzir o aprendizado humano, são poucos os algoritmos de AM que tentam fazê-lo atualmente. Apesar dos grandes avanços em termos de equipamentos para analisar o cérebro humano, atualmente não se conhece em detalhes como o aprendizado ocorre em nosso cérebro. Boa parte dos algoritmos atuais já não buscam imitar o cérebro humano, mas tentam aprender utilizado formalismos matemáticos bem conhecidos como as árvores de decisão e NAIVE BAYES. Como mencionado, esses dois algoritmos são bem conhecidos pela comunidade de aprendizado de máquina e serviram como ponto de partida para algumas das contribuições propostas neste trabalho.

CAPÍTULO

3

Ranking

Durante muitos anos, o problema de classificação tem sido foco de pesquisas em aprendizado de máquina. Nesse tipo de aprendizado, muitas vezes, ignora-se a informação do valor de confiança da classificação proposta pelo algoritmo, importando-se apenas com o valor da classe. Porém, o valor de confiança é muito importante na geração de *rankings* e, dependo do domínio de aplicação, o *ranking* torna-se mais atrativo que a própria classificação. Um exemplo disso são as páginas de internet retornadas por um buscador de internet. Nesse caso, mais interessante que retornar os endereços de internet relevantes, é retornar esses endereços na ordem de relevância.

Neste capítulo são abordados conceitos de classificação e *ranking*. O capítulo está organizado da seguinte maneira: na Seção 3.1 são explicadas as relações e diferenças entre classificação e *ranking*; na Seção 3.2 é mostrado como é possível obter *ranking* dos mais diversos algoritmos de aprendizado de máquina; na Seção 3.3 é apresentado um algoritmo de *ranking*, proposto neste trabalho, denominado LEXRANK. Finalmente, na Seção 3.5 são apresentados algumas considerações finais.

3.1 Relação de Classificação, Ranking e Probabilidade

Relembrando a notação definida no Capítulo 2, seja $X = A_1 \times \ldots \times A_{n_{at}}$ o espaço de exemplos sobre um conjunto de atributos discretos $A_1, \ldots, A_{n_{at}}$. Um *classificador* é o mapeamento $\hat{c} : X \to Y$, onde Y é o conjunto de rótulos. Quando $Y = \{+, -\}$ o *classificador* é denominado de *classificador binário*.

O *ranking* é uma relação de *ordem total* possivelmente com empates. O empate é representado como uma relação de equivalência sobre X (Grätzer,

1971). Neste trabalho, um conjunto de exemplos "empatados" é denominado *segmento*. Por conveniência notacional, um *rankeador* é representado como uma função $\hat{r} : X \times X \to \{>, =, <\}$, que define para qualquer par de exemplos se o primeiro é mais provável ($>$), igualmente provável ($=$) ou menos provável ($<$) de ser positivo que o segundo (abusando um pouco da notação, os símbolos $>$ e $<$ também são utilizados para a ordem total nos *segmentos* de X).

Se $X_1, X_2 \subseteq X$ são *segmentos* tal que $X_1 > X_2$ e não existe um *segmento* X_3 tal que $X_1 > X_3 > X_2$, pode-se dizer que X_1 e X_2 são *adjacentes*. Um *rankeador* pode ser transformado em um classificador binário selecionando um par de *segmentos adjacentes* $X_1 > X_2$ e atribuir para cada exemplo em X_1 ou qualquer $X' > X_1$ a classe positiva e, para qualquer exemplo em X_2 ou qualquer $X' < X_2$ a classe negativa. Além disso, dado um *rankeador* $\hat{r}$, é possível construir outro *rankeador* $\hat{r}'$ pela união de dois *segmentos adjacentes* X_1 e X_2. Pode-se dizer que $\hat{r}'$ é mais *grosseiro* que $\hat{r}$, ou equivalentemente, que $\hat{r}$ é mais *refinado* que $\hat{r}'$. O *ranking* trivial é o *ranking* mais *grosseiro* que mapeia todos os exemplos no mesmo segmento e não representa nenhuma ordem de preferência. Em contrapartida, o *ranking* mais *refinado* é aquele que não possui empate.

Um classificador *score* é um mapeamento $\hat{s} : X \to \mathbb{R}$ atribuindo um *score* numérico $\hat{s}(\mathbf{x}_i)$ para cada exemplo $\mathbf{x}_i$. Será adotada a convenção de que *scores* mais altos representam preferência pela classe positiva. Um *estimador de probabilidade* é um classificador *score* que atribui probabilidades, *i.e.*, um mapeamento $\hat{P} : X \to [0,1]$. $\hat{P}(\mathbf{x}_i)$ deve estimar a probabilidade a posteriori $P(+|\mathbf{x}_i)$, *i.e.*, a verdadeira probabilidade de um exemplo pertencer a classe positiva.

Dado um classificador *score* $\hat{s}$ (ou um *estimador de probabilidade*) é possível construir um *rankeador* $\hat{r}$ da seguinte maneira:

$$\hat{r}(\mathbf{x}_i, \mathbf{x}_{i+1}) = \begin{cases} > & \text{if} \quad \hat{s}(\mathbf{x}_i) > \hat{s}(\mathbf{x}_{i+1}) \\ = & \text{if} \quad \hat{s}(\mathbf{x}_i) = \hat{s}(\mathbf{x}_{i+1}) \\ < & \text{if} \quad \hat{s}(\mathbf{x}_i) < \hat{s}(\mathbf{x}_{i+1}) \end{cases}$$

Também é possível transformar um classificador *score* em um classificador transformando um *rankeador* associado em um classificador como descrito anteriormente, ou equivalentemente, definindo um limiar $t \in \mathbb{R}$ que atribui a classe positiva a todos os exemplo $\mathbf{x}_i$ tal que $\hat{s}(\mathbf{x}_i) \geq t$, e atribui a classe negativa aos exemplos restantes.

Exemplo 3.1 *Considere um conjunto com 10 exemplos positivos e 10 exemplos negativos, com dois atributos binários A_1 e A_2 ilustrados na Figura 3.1. Ao induzir uma árvore de decisão utilizando esse conjunto de treinamento obtém-se a árvore ilustrada na Figura 3.2.*

Ferri et al. (2002) e Provost and Domingos (2003) mostram que árvores de decisão podem ser utilizadas como estimadores de probabilidades, calculando

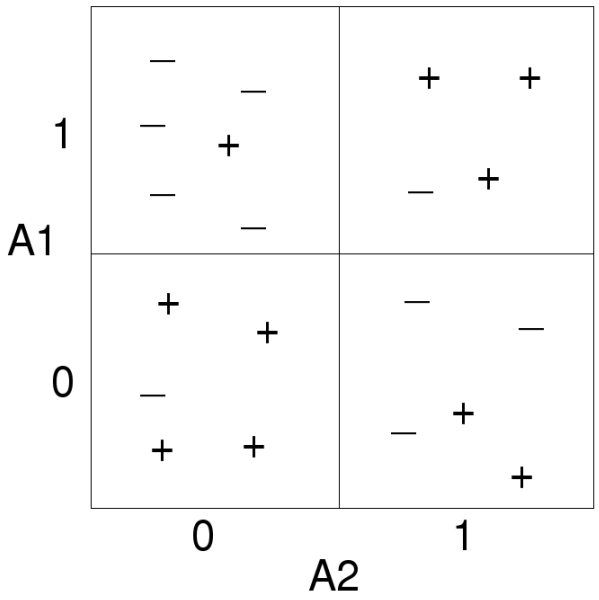


Figura 3.1: Exemplo 3.1 - conjunto de treinamento

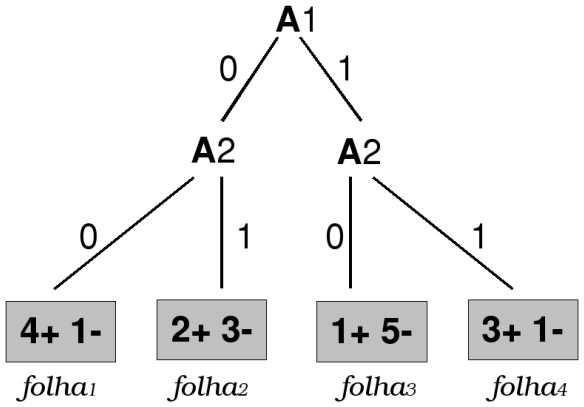


Figura 3.2: Exemplo 3.1 - árvore de decisão

$P(+|folha_i) = \frac{n_i^+}{n_i^+ + n_i^-}$, onde ($n_i^+$) é o número de exemplos positivos e (n_i^-) é o número de exemplos negativos pertencentes a i-ésima folha. Deste modo, as probabilidades das folhas serem positivas na árvore ilustrada na Figura 3.2 são 0.80, 0.40, 0.16, 0.75 para as folhas 1, 2, 3 e 4, respectivamente. Ao ordenar as folhas de acordo com as probabilidades de serem positivas, obtém-se o ranking dado por $folha_1 - folha_4 - folha_2 - folha_3$. Em cada folha existe um conjunto de exemplos empatados (segmentos). No segmento da $folha_1$ existem 4 exemplos positivos e 1 negativo; esses 5 exemplos possuem a mesma probabilidade (0.80) de serem positivos e, portanto, estão empatadas no ranking. Os segmentos das $folha_1$ e $folha_4$ são adjacentes, pois não existe folha com probabilidade entre 0.80 e 0.75 na árvore apresentada. Com as folhas ordenadas em ordem decrescente das probabilidades das folhas serem positivas ($folha_1 - folha_4 - folha_2 - folha_3$), pode-se obter um classificador rotulando as k primeiras folhas

como positivas, e as restantes como negativas.

3.2 Obtendo Scores

A maioria dos algoritmos de aprendizado supervisionado podem ser convertidos em *rankeadores*. Classificadores frequentemente podem fornecer um *score* numérico que representa o valor de confiança da classificação, podendo ser considerados classificadores *score*. Como ilustrado anteriormente, classificadores *score* podem ser transformados em *rankeadores*.

Essa conversão de classificadores em *rankings* só é possível graças aos *scores* calculados pelos algoritmos. Para alguns algoritmos a obtenção desse *score* é bastante natural, para outros nem tanto. Nesta seção são ilustrados algumas maneiras de obter *score* dos algoritmos de aprendizado mais conhecidos (Russell and Norvig, 2002).

NAIVE BAYES: apresentado no capítulo anterior, é um dos diversos algoritmos de aprendizado que utilizam a regra de Bayes para estimar a probabilidade a posteriori $P(+|\mathbf{x}_i)$. O *score* desse tipo de algoritmos pode ser obtido diretamente utilizando $\hat{P}(+|\mathbf{x}_i)$.

Regressão Logística: a regressão logística também estima a probabilidade $\hat{P}(+|\mathbf{x})$, o *score* pode ser obtido dessa estimação.

Redes Neurais: quando utilizado em aprendizado supervisionado, o algoritmo retorna uma função que é a combinação de neurônios. A rede neural geralmente retornar um número real que pode ser considerado um *score*.

Árvores de Decisão: os nós folhas de uma árvore de decisão possuem informação sobre a quantidade de exemplos positivos e negativos, nelas contidas, encontradas durante a fase de treinamento. A proporção de exemplos positivos nos nós folhas pode ser considerado um *score*.

Máquinas de Vetores de Suporte: a distância entre um exemplo e a margem pode ser considerado um *score*. Quando o exemplo está do lado dos exemplos positivos do limiar de decisão, o *score* é a distância da margem com sinal positivo, quando o exemplo está do lado dos exemplos negativos, o *score* é a distância, mas com sinal negativo.

Vizinhos Mais Próximos: considerando apenas os vizinhos mais próximos, o *score* pode ser representado pela proporção de exemplos positivos dentro da vizinhança.

Indução de Regras: o *score* pode ser obtido pela proporção de exemplos positivos cobertos pela(s) regra(s) que determinaram a classificação do exemplo.

Existem outras maneiras de se obter *scores* desses algoritmos além dos descritos aqui. O intuito é apenas mostrar que é possível obter *scores* de uma maneira relativamente simples, sem a necessidade de grandes alterações nas implementações desses algoritmos. O *score* tem um papel importante em *rankeadores*, sem eles não é possível determinar a relação de ordem total entre os exemplos. Entretanto, é possível desenvolver algoritmos *rankeadores* sem a necessidade de *score*. Na seção a seguir é descrito o algoritmo LEXRANK, proposto neste trabalho, um dos primeiros algoritmos com essa propriedade.

3.3 LEXRANK

A ordenação lexicográfica, muito utilizada em ordenação de palavras, possui características bastante interessantes que podem ser utilizadas em algoritmos de aprendizado de máquina. A ordem lexicográfica pode ser representada como uma estrutura de árvore, podendo ser utilizada para construir árvores de decisão, além de ser um *rankeador* natural pois ordena seus elementos por definição. Fazendo uso dessas idéias, juntamente com conceitos de árvore de decisão e NAIVE BAYES, foi desenvolvido um algoritmo legitimamente *rankeador* (não provém de um classificador *score*), no qual obtém-se a ordem diretamente de sua estrutura.

Exemplo 3.2 *Considere os seguintes quatro exemplos com cinco atributos binários* $\boldsymbol{x}_1 = (0,1,0,0,0)$, $\boldsymbol{x}_2 = (0,0,1,0,0)$, $\boldsymbol{x}_3 = (0,0,0,1,0)$ *e* $\boldsymbol{x}_4 = (0,0,1,0,1)$. *Ao converter os exemplos em strings e ordená-los lexicograficamente, obtém-se a seguinte ordem*

$$\begin{aligned} \boldsymbol{x}_3 &= (0,0,0,1,0) \\ \boldsymbol{x}_2 &= (0,0,1,0,0) \\ \boldsymbol{x}_4 &= (0,0,1,0,1) \\ \boldsymbol{x}_1 &= (0,1,0,0,0) \end{aligned}$$

Em uma situação particular, essa ordenação lexicográfica pode representar os exemplos do "mais positivo" para o "mais negativo", como em rankeadores.

Essa simples idéia de ordenação lexicográfica foi considerada no desenvolvimento do algoritmo de aprendizado de máquina LEXRANK (Flach and Matsubara, 2007b). Antes de definir LEXRANK, são apresentados alguns conceitos de *ranking lexicográfico* e sua relação com NAIVE BAYES e árvore de decisão.

3.3.1 Relações entre Ranking Lexicográfico, Árvores de Decisão e NAIVE BAYES

Por conveniência de notação, nesta seção também são assumidos atributos binários, lembrando que atributos nominais com t valores podem ser convertidos em t atributos binários.

Os seguintes dois conceitos devem ser distinguidos para a compreensão de *ranking lexicográficos*:

ordem de preferência de atributos: é a ordem imposta sobre os atributos utilizando uma medida que separa exemplos positivos e negativos. Por exemplo, ordenar os atributos por *ganho de informação* de tal modo que os primeiros atributos separam melhor os exemplos positivos dos negativos.

valor de preferência de um atributo: em um atributo booleano, um dos valores, zero ou um, representa melhor a classe positiva, o *valor de preferência* é aquele que melhor representa a classe positiva.

Utilizando esses dois conceitos pode-se definir um *ranking lexicográfico*.

Definição 3.1 (Ranking Lexicográfico) *Seja $A_1,\ldots,A_{n_{at}}$ um conjunto de atributos booleanos, tal que os índices representam uma ordem de preferência. Seja v_{j+} o valor de preferência do atributo A_j. O rankeador lexicográfico corresponde a uma ordem de preferência nos atributos e nos valores dos atributos definido como:*

$$\hat{r}_{lex}(\boldsymbol{x}_i,\boldsymbol{x}_{i+1}) = \begin{cases} > & \text{if} \quad A_j(\boldsymbol{x}_i) = v_{k+} \\ < & \text{if} \quad A_j(\boldsymbol{x}_i) \neq v_{k+} \end{cases}$$

onde k denota o menor índice de atributo tal que $\boldsymbol{x}_i$ e $\boldsymbol{x}_{i+1}$ possuem diferentes valores (caso não exista um índice que satisfaça essas condições, os dois exemplos estão empatados).

Exemplo 3.3 *Dado os exemplos $\boldsymbol{x}_1 = (0,0,1,0,0)$ e $\boldsymbol{x}_2 = (0,0,0,1,0)$, onde os atributos estão ordenados por uma ordem de preferência A_1,A_2,A_3,A_4,A_5, e o valor de preferência $v_{j+} = 0$ para todos os j. Os valores dos atributos dos dois exemplos vão sendo comparados até que se encontre uma diferença. Nesse par de exemplos, essa diferença é encontrada no atributo A_3, onde $A_3(\boldsymbol{x}_1) = 1$ e $A_3(\boldsymbol{x}_2) = 0$ e, portanto, $A_j = A_3$. Como o valor $v_{j+} = 0$ é o valor de preferência (indicativo de positivo), o exemplo $\boldsymbol{x}_2$ é mais positivo que $\boldsymbol{x}_1$, i.e., $\hat{r}_{lex}(\boldsymbol{x}_1,\boldsymbol{x}_2)$ retorna $<$. Assim, quando esses exemplos são submetidos a um ranking lexicográfico, o exemplo $\boldsymbol{x}_1$ é "menos positivo" que $\boldsymbol{x}_2$.*

Qualquer *ranking lexicográfico* pode ser representado como uma árvore de decisão binária com as seguintes propriedades:

1. O atributo A_j somente ocorre na profundidade j da árvore, *i.e.*, em um caminho do nó raiz até um nó folha, os atributos sempre aparecem em uma ordem de preferência.

2. em cada partição (*split*) da árvore, v_{j+} é sempre o valor da aresta para o filho da esquerda.

Conseqüentemente, a ordem do *ranking* é representada pela ordem da esquerda para a direita das folhas. Uma árvore com essas características é denominada *árvore lexicográfica de ranking*.

Exemplo 3.4 *Considere os exemplos apresentados na Figura 3.1. Sabendo que A_1 tem maior preferência que A_2, 0 é o valor de preferência de A_1 e 1 é o valor preferência de A_2. Com essas informações e utilizando a Definição 3.1, é possível construir o seguinte ranking lexicográfico.*

#	A_1	A_2	# pos(+) e neg(-)
1	0	1	2+ 3-
2	0	0	4+ 1-
3	1	1	3+ 1-
4	1	0	1+ 5-

Vale ressaltar que o valor de preferência de A_2 é 1 ao invés de 0, isso faz com que os exemplos pareçam não estar ordenados lexicograficamente, mas segundo a Definição 3.1 os exemplos estão em ordem lexicográfica. Esse ranking lexicográfico pode ser representado com uma árvore de decisão (Figura 3.3), na qual o atributo A_1 tem maior preferência que A_2 e é o nó raiz (profundidade 1), e o atributo A_2 representa os nós de profundidade 2. Note que, diferentemente da árvore de decisão tradicional, onde em uma mesma altura pode-se ter diferentes atributos, pela definição, na profundidade i pode existir apenas o atributo A_j. Em toda a profundidade i, os zeros e uns sempre se alternam, começando sempre com v_{j+}.

A *árvore lexicográfica de ranking* posiciona as folhas da árvore da mais positiva para a mais negativa. Assim, na Figura 3.3, da esquerda para a direita, a $folha_1$, é mais positiva que a $folha_2$, que é mais positiva que a $folha_3$, que é mais positiva que a $folha_4$.

Visualmente, a *árvore lexicográfica de ranking* é bem intuitiva. Entretanto, o tamanho dessa árvore é exponencial no número de atributos e, na prática,

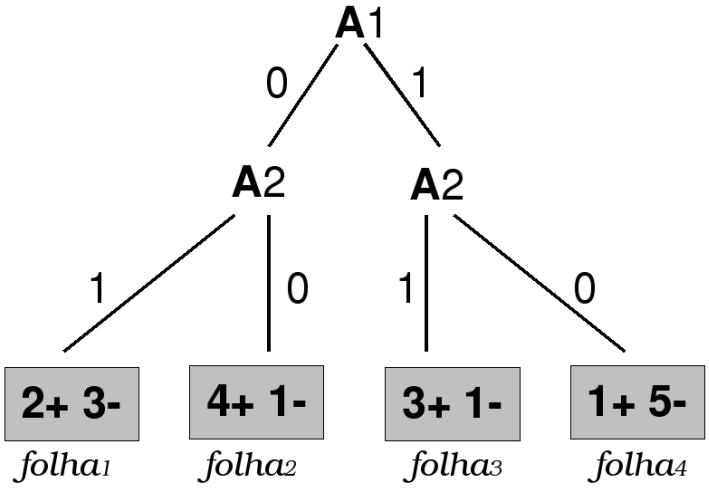


Figura 3.3: Árvore lexicográfica de *ranking*

torna-se inviável a construção de uma estrutura desse tipo. Essa árvore apenas é apresentada para mostrar que todo *ranking* lexicográfico corresponde a uma árvore lexicográfica de *ranking*, mas o contrário nem sempre é verdade.

De modo similar a essa árvore, é possível verificar um *ranking* lexicográfico utilizando conceitos de NAIVE BAYES.

Exemplo 3.5 *O rankeador de* NAIVE BAYES *obtido dos dados do Exemplo 3.1 pode ser definido da seguinte maneira. Calculando as razões de verossimilhança LR (likelihood ratios) dos atributos A_1 e A_2, obtém-se:*

$$LR(A_1 = 0) = \frac{p(A_1=0|+)}{p(A_1=0|-)} = \frac{6}{4}$$

$$LR(A_1 = 1) = \frac{p(A_1=1|+)}{p(A_1=1|-)} = \frac{4}{6}$$

$$LR(A_2 = 0) = \frac{p(A_2=0|+)}{p(A_2=0|-)} = \frac{5}{6}$$

$$LR(A_2 = 1) = \frac{p(A_2=1|+)}{p(A_2=1|-)} = \frac{5}{4}$$

A chance a priori (prior odds) não afeta o ranking, então basta multiplicá-los

$$LR(A_1 = 0) \times LR(A_2 = 1) = \frac{30}{16} >$$
$$LR(A_1 = 0) \times LR(A_2 = 0) = \frac{30}{24} >$$
$$LR(A_1 = 1) \times LR(A_2 = 1) = \frac{20}{24} >$$
$$LR(A_1 = 1) \times LR(A_2 = 0) = \frac{20}{36}$$

Essa ordem é um ranking lexicográfico, que é equivalente a árvore lexicográfica de ranking, apresentada na Figura 3.3, na qual $LR(A_1 = 0) \times LR(A_2 = 1)$ representa a $folha_1$, $LR(A_1 = 0) \times LR(A_2 = 1)$ representa a $folha_2$, $LR(A_1 = 1) \times LR(A_2 = 1)$ representa a $folha_3$ e $LR(A_1 = 1) \times LR(A_2 = 0)$ representa a $folha_4$.

3.3.2 Descrição do Algoritmo LEXRANK

Com isso pode-se estreitar ainda mais a conexão entre NAIVE BAYES e *ranking lexicográfico*, utilizando razão de chances como critério para ordenar os atributos.

Definição 3.2 LEXRANK *é um ranking lexicográfico que utiliza o seguinte critério: o valor de preferência* v_{j+} *para um atributo* A_j *é definido como o valor que satisfaz* $LR(A_j = v_{j+}) \geq 1$. *A ordem de preferência nos atributos é definida em ordem decrescente utilizando razão de chance (Odds Ratio)* $OR(A_j) = \frac{LR(A_j=v_{j+})}{LR(A_j=v_{j-})}$, *onde* v_{j-} *denota o valor de menor preferência.*

A fase de treinamento de LEXRANK é ilustrada no Algoritmo 1. Inicialmente são calculados os $LR(A_j)$ para ambos os valores, os LR determinam os valores de preferência de cada atributo v_{j+}. Em seguida são calculados $OR(A_j)$. Para que os atributos sejam mapeados em ordem decrescente de $OR(A_j)$ é utilizada uma lista ordenada. O cálculo de LR é de complexidade $O(n_{ex})$, a complexidade da inserção de $OR(A_j)$ na lista ordenada é de $O(log(n_{at}))$. Portanto, a complexidade do algoritmo é de $O(n_{at} \times n_{ex} \times log(n_{at}))$.

Algoritmo 1: LEXRANK

```
for j = 0 to n_at do
    calcular LR(A_j) ;
    if LR(A_j = 0) ≥ 1 then
        v_j+ = 0 ;
        v_j− = 1 ;
    else
        v_j+ = 1 ;
        v_j− = 0 ;
    end
    calcular OR(A_j);
    inserir OR(A_j em uma lista ordenada de acordo com o valor de OR(A_j).
end
```

O diferencial do algoritmo está na obtenção do *ranking*, pois, como o algoritmo não atribui *scores*, o *ranking* pode ser obtido diretamente. Desse modo, a complexidade do algoritmo para obtenção do *ranking* é a complexidade de um algoritmo de ordenação $O(n_{ex} \times log(n_{ex}))$. Qualquer outro algoritmo que atribui *scores* possui complexidade maior pois $O(n_{ex} \times log(n_{ex})) < O(m \times n_{ex} \times log(n_{ex}))$, onde m é a complexidade para calcular o *score* em um classificador *score*.

Como pode ser observado, o algoritmo executa os seguintes 4 passos:

1. calcular $LR(A_j)$
2. determinar v_{j+}

3. calcular $OR(A_j)$

4. ordenar os atributos de acordo com os $OR(A_j)$

O exemplo a seguir ilustra esses 4 passos na execução de LEXRANK.

Exemplo 3.6 *Considere o conjunto de exemplos na Tabela 3.1.*

Tabela 3.1: Conjunto de exemplos de treinamento

A_3	A_1	A_2	Classe
1	1	0	-
0	0	0	+
1	0	1	+
0	1	1	-
1	0	0	+
1	1	1	-

1. *Calculando as razões de verosimilhança $LR(A_j)$ com suavizador de Laplace*[1] *obtém-se:*

$$LR(A_3 = 0) = \frac{2}{2}$$
$$LR(A_3 = 1) = \frac{3}{3}$$
$$LR(A_1 = 0) = \frac{4}{1}$$
$$LR(A_1 = 1) = \frac{1}{4}$$
$$LR(A_2 = 0) = \frac{3}{2}$$
$$LR(A_2 = 1) = \frac{2}{3}$$

2. *Para cada um dos atributos com valores $A_j = 0$ ou $A_j = 1$, o valor de preferência para A_1, A_2 e A_3 é $v_{j+} = 0$ e $v_{j-} = 1$.*

3. *Calculando $OR(A_j)$:*

$$OR(A_3) = \frac{\frac{2}{2}}{\frac{3}{3}} = 1$$
$$OR(A_1) = \frac{\frac{4}{1}}{\frac{1}{4}} = 16$$
$$OR(A_2) = \frac{\frac{3}{2}}{\frac{2}{3}} = \frac{9}{4} = 2.25$$

[1]Também conhecida como lei de sucessão de Laplace, simplesmente consiste em somar 1 no final da contagem. Isso evita que eventos raros tenham probabilidade zero.

4. *Ordenando os atributos em ordem decrescente de OR (A_1,A_2 e A_3) e os exemplos lexicograficamente (000 < 001< 011 < 101 < 110 < 111) obtém-se o ranking ilustrado na Tabela 3.2.*

Tabela 3.2: Conjunto de exemplos ordenados lexicograficamente utilizando LEXRANK

A_1	A_2	A_3	Classe
0	0	0	+
0	0	1	+
0	1	1	+
1	0	1	-
1	1	0	-
1	1	1	-

A ordem lexicográfica cobre todos os exemplos possíveis, mesmo quando o exemplo não faz parte do conjunto de treinamento. Considere o exemplo $(0,1,0)$, apesar de não estar no conjunto de treinamento, ele pode ser colocado no *ranking* lexicográfico entre $(0,0,1)$ e $(0,1,1)$. Quando aplicado a um conjunto de teste, LEXRANK apenas ordena os exemplos; a classificação pode ser realizada definindo um limiar nesse *ranking*.

A simplicidade de LEXRANK o torna um algoritmo bastante interessante. É possível demonstrar que para dois atributos binários LEXRANK e NAIVE BAYES sempre produzem o mesmo *ranking*. Entretanto, para mais de dois atributos binários, existem modelos induzidos por NAIVE BAYES que não podem ser representados como uma ordem de preferência nos atributos, como em LEXRANK. Com isso, pode-se concluir que LEXRANK possui um bias mais forte que NAIVE BAYES. Apesar disso, nos resultados experimentais, LEXRANK tem resultados tão bons quanto NAIVE BAYES e a árvore de decisão *J48* em termos de desempenho de *ranking*, como mostrado a seguir.

3.4 Resultados Experimentais

Os experimentos foram conduzidos utilizando 27 conjuntos de dados da UCI (Asuncion and Newman, 2007). Os conjuntos de dados foram selecionados dando preferência para conjuntos com atributos nominais, poucos valores faltantes (*missing values*) e duas classes. Alguns conjuntos multi-classe foram convertidos para duas classes selecionando uma das classes como positiva e o restante como negativa. Exemplos com valores faltantes foram removidos. Atributos contínuos foram discretizados usando discretização não supervisionado de dez compartimentos (*ten-bin*).

Na Tabela 3.3 são apresentados os conjuntos de dados utilizados (Conjunto), o número de atributos (#Atrib), o número de exemplos (#Exem) e a

classe majoritária (Classe Maj.). Conjuntos com menos de 270 exemplos, indicados com "*", foram analisados utilizando validação cruzada de 5 partições, o restante dos conjuntos com validação cruzada de 10 partições. Os colchetes indicam que o conjunto possui mais de duas classes e o valor entre colchetes indica a classe escolhida para ser positiva e a classe negativa são as classes restantes (um *versus* todos).

Tabela 3.3: Conjunto de exemplos da UCI utilizados nos experimentos na avaliação de LEXRANK

#	Conjunto	#Atrib	#Exem	Classe Maj.
1	anneal[3]	145	898	76.169
2	breast-cancer	49	277	70.758
3	breast-w	91	683	65.007
4	car[unacc]	22	1728	70.023
5	credit-a	98	653	54.671
6	credit-g	125	1000	70.000
7	dermatology[1]	139	358	68.994
8	diabetes	81	768	65.104
9	*glass	91	214	76.168
10	haberman	33	306	73.529
11	*hayes-roth[1]	41	160	59.375
12	heart-statlog	131	270	55.556
13	ionosphere	332	351	64.103
14	kr-vs-kp	41	3196	52.222
15	letter[A]	161	20000	96.055
16	liver-disorders	61	345	57.971
17	*lymph[met]	66	148	56.081
18	mfeat-pixel[one]	1649	2000	90.000
19	*molec-bio	229	106	50.000
20	monks-1	16	556	50.000
21	monks-2	16	601	65.724
22	monks-3	16	554	51.986
23	*postoper-pat[A]	22	87	71.264
24	sonar	601	208	53.365
25	*spect	23	267	79.401
26	splice[EI]	288	3190	75.956
27	tic-tac-toe	28	958	65.344

Foram criadas versões binárias e não binárias dos conjuntos de dados, já que NAIVE BAYES e árvores de decisão podem tratar atributos não binários. A versão binária mostrou uma pequena diferença, mas insignificante estatisticamente, no desempenho de NAIVE BAYES e árvores de decisão. Desse modo, por simplicidade, decidiu-se utilizar em todos os experimentos apenas conjuntos de dados binários. Assim, atributos com t valores nominais foram transformados em t atributos binários em todos os conjuntos.

Foram comparados os seguintes algorítimos: o algoritmo proposto LEXRANK, NAIVE BAYES e a árvore de decisão *J48*. O LEXRANK foi desenvolvido dentro do ambiente Weka (Witten and Frank, 2005) e foram utilizadas as implementações do NAIVE BAYES e *J48* existentes no ambiente Weka. *J48* foi executado sem poda e com correção de Laplace, para se obter melhor desem-

penho de *rankings* (Provost and Domingos, 2003). Como o interesse é avaliar LEXRANK em termos de *ranking*, os resultados experimentais foram avaliados utilizando AUC (*Area Under ROC Curve*) (Fawcett, 2006). Foram feitos testes de significância, um paramétrico e um não paramétrico, o teste-*t* pareado e o teste de Friedman, respectivamente.

Na Tabela 3.4 são mostrados a média das AUC dos algoritmos com os respectivos desvios padrão entre parênteses. +/- indicam diferença significativa comparando LEXRANK como referência utilizando teste-*t* pareado com 0.05 nível de significância (+ indica que LEXRANK é significativamente superior). A última linha g/e/p indica o número de vezes que LEXRANK é significativamente superior (g), não há diferença significativa (e) ou é significativamente inferior(p) que NAIVE BAYES e *J48*.

Tabela 3.4: Média de AUC com desvio padrão e teste de significância t-pareado comparando LEXRANK com os outros dois algoritmos (+ significa LEXRANK é melhor). A última coluna soma ganhador/empate/perdedor para LEXRANK

# Conjunto	LexRank	NB		J48	
1	0.985 0.027	0.948 0.036	+	0.994 0.017	-
2	0.707 0.075	0.727 0.079		0.677 0.084	
3	0.994 0.007	0.994 0.007		0.977 0.018	+
4	0.941 0.013	0.989 0.005	-	0.997 0.005	-
5	0.919 0.028	0.897 0.046	+	0.899 0.034	+
6	0.750 0.067	0.788 0.042	-	0.713 0.049	
7	0.999 0.002	1.000 0.000		0.987 0.024	
8	0.826 0.056	0.828 0.048		0.771 0.066	+
9	0.954 0.020	0.976 0.012		0.948 0.032	
10	0.708 0.116	0.698 0.107		0.634 0.103	+
11	0.861 0.061	0.914 0.079	-	0.879 0.092	
12	0.841 0.055	0.912 0.045	-	0.840 0.044	
13	0.960 0.050	0.928 0.041	+	0.898 0.063	+
14	0.978 0.010	0.931 0.015	+	0.999 0.001	-
15	0.965 0.012	0.946 0.020	+	0.984 0.012	-
16	0.633 0.085	0.648 0.076		0.663 0.078	
17	0.855 0.047	0.911 0.039	-	0.879 0.077	
18	0.996 0.003	0.970 0.034	+	0.994 0.004	
19	0.922 0.027	0.971 0.040	-	0.867 0.070	
20	0.708 0.046	0.731 0.067	-	0.996 0.004	-
21	0.472 0.083	0.531 0.095	-	0.998 0.004	-
22	0.983 0.020	0.983 0.023		0.988 0.014	-
23	0.542 0.103	0.559 0.084		0.566 0.062	
24	0.759 0.076	0.850 0.068	-	0.725 0.043	
25	0.809 0.097	0.846 0.081	-	0.762 0.083	+
26	0.991 0.004	0.995 0.003	-	0.987 0.008	
27	0.723 0.019	0.750 0.020	-	0.977 0.013	-
g/e/p		6/9/12		6/13/8	

Na Tabela 3.5 são mostrados os resultados correspondente ao teste-*t* pareado. Para cada linha são ilustrados os ganhos/empates/perdas do algoritmo na linha *versus* o algoritmo na coluna. Pode ser observado que NAIVE BAYES tem um desempenho um pouco superior que a média, mas de modo geral os algoritmos tem desempenho comparável.

Tabela 3.5: Resultados comparando teste pareado de significância utilizando AUC

	LexRank	NB	J48
LexRank	-	6/9/12	6/13/8
NB	12/9/6	-	11/8/8
J48	8/13/6	8/8/11	-

Na Tabela 3.6 são mostradas as médias da validação cruzada com o *ranking* do algoritmo entre parênteses. Calculando-se a média do *ranking* pode-se ter uma idéia do desempenho de cada algoritmo e também utilizá-lo para o cálculo da estatística F, utilizada no teste de Friedman. A estatística F é de 1.39 para os resultados obtidos. O valor crítico utilizando a estatística F com 2 e 52 graus de liberdade a 95% é de 3.18. Desse modo, não se pode descartar a hipótese nula, *i.e.*, não existe diferença significativa entre os algoritmos.

Tabela 3.6: Média das AUCs, rank dos algoritmos (entre parênteses) e rank médio utilizado para realizar o teste de Friedman

# Conjunto	LexRank	naive Bayes	*J48*
1	0.985(2.0)	0.948(3.0)	0.994(1.0)
2	0.707(2.0)	0.727(1.0)	0.677(3.0)
3	0.994(1.5)	0.994(1.5)	0.977(3.0)
4	0.941(3.0)	0.989(2.0)	0.997(1.0)
5	0.919(1.0)	0.897(3.0)	0.899(2.0)
6	0.750(2.0)	0.788(1.0)	0.713(3.0)
7	0.999(2.0)	1.000(1.0)	0.987(3.0)
8	0.826(2.0)	0.828(1.0)	0.771(3.0)
9	0.954(2.0)	0.976(1.0)	0.948(3.0)
10	0.708(1.0)	0.698(2.0)	0.634(3.0)
11	0.861(3.0)	0.914(1.0)	0.879(2.0)
12	0.841(2.0)	0.912(1.0)	0.840(3.0)
13	0.960(1.0)	0.928(2.0)	0.898(3.0)
14	0.978(2.0)	0.931(3.0)	0.999(1.0)
15	0.965(2.0)	0.946(3.0)	0.984(1.0)
16	0.633(3.0)	0.648(2.0)	0.663(1.0)
17	0.855(3.0)	0.911(1.0)	0.879(2.0)
18	0.996(1.0)	0.970(3.0)	0.994(2.0)
19	0.922(2.0)	0.971(1.0)	0.867(3.0)
20	0.708(3.0)	0.731(2.0)	0.996(1.0)
21	0.472(3.0)	0.531(2.0)	0.998(1.0)
22	0.983(2.5)	0.983(2.5)	0.988(1.0)
23	0.542(3.0)	0.559(2.0)	0.566(1.0)
24	0.759(2.0)	0.850(1.0)	0.725(3.0)
25	0.809(2.0)	0.846(1.0)	0.762(3.0)
26	0.991(2.0)	0.995(1.0)	0.987(3.0)
27	0.723(3.0)	0.750(2.0)	0.977(1.0)
rank médio	2.148	1.741	2.111

Na Tabela 3.7 são reportados o tempo de execução dos algoritmos. O tempo de treinamento de LexRank é bastante semelhante ao de naive Bayes, já que suas complexidades para o treinamento são praticamente iguais. No teste J48 é o mais rápido LexRank possui tempo de teste um pouco mais rápido que naive Bayes. Vale ressaltar que dentro do ambiente Weka, não existe uma

maneira apropriada para avaliar o tempo de teste de LEXRANK, já que o sistema foi desenvolvido para classificação e não para *ranking*, o que obriga a implementação de LEXRANK fornecer um *score*. Num ambiente próprio para *ranking*, poderia-se mostrar LEXRANK com um tempo de teste tão rápido quanto árvores de decisão.

Tabela 3.7: Tempo de uma execução sobre os 27 conjunto de dados (sem validação cruzada).

	LexRank	NB	J48
treino	20.28	16.32	142.74
teste	26.44	33.18	1.29
total	46.72	49.50	144.03

3.5 Considerações Finais

Nos últimos anos, a tarefa de classificação tem dominado boa parte dos esforços em desenvolvimentos da área de aprendizado de máquina. Atualmente, existe um grande apelo em utilizar aprendizado de máquina em tarefas diferentes à classificação. Neste capítulo propomos o uso de *rankings* e ilustramos algumas diferenças e semelhanças entre a classificação e *ranking*. Mostramos também como se pode transformar um problema de *ranking* em classificação, e o inverso, e como é possível obter *rankings* dos principais algoritmos de classificação.

Além disso, propomos o algoritmo de aprendizado LEXRANK que consiste basicamente em uma ordenação de atributos e a ordenação lexicográfica dos exemplos. Apesar da sua simplicidade, LEXRANK pode obter resultados tão bons quanto NAIVE BAYES e *J48*.

Uma maneira de tornar *ranking* ainda mais interessante, é utilizá-lo em conjunto com análise ROC. Essa união possibilita fazer ajustes em algoritmos de classificação, construir algoritmos mais robustos (Provost and Fawcett, 2001), analisar problemas de classes desbalanceadas e algumas outras características, descritas no próximo capítulo.

CAPÍTULO

4

Análise ROC

A maior parte da pesquisa realizada, bem como as propostas apresentadas nos próximos capítulos, fazem uso da análise ROC. Com o intuito de facilitar a compreensão dos trabalhos realizados, neste capítulo são apresentados os conceitos de análise ROC utilizados neste trabalho. O capítulo está organizado da seguinte maneira: na Seção 4.1 são apresentados alguns conceitos de análise ROC, como as interpretações de um gráfico ROC, a construção de uma curva ROC, linhas de iso-desempenho, feixe convexo e área abaixo da curva. Na Seção 4.2 são apresentados os conceitos de árvores de decisão aplicados a análise ROC, bem como um sistema desenvolvido para facilitar a visualização da construção de uma árvore de decisão. Finalmente, na Seção 4.3 são apresentadas as considerações finais deste capítulo.

4.1 Análise ROC

Análise ROC (*Receiver Operating Characteristic*) é uma técnica visual para avaliação, organização e seleção de classificadores de duas classes (Fawcett, 2006). Cada vez mais utilizada em aprendizado de máquina nos dias atuais, acredita-se que as curvas ROC foram desenvolvidas durante a segunda guerra mundial para auxiliar na análise de sinais de radares. Desde então, a análise ROC tem sido utilizada em diversos domínios como medicina, radiologia e ciências sociais. Particularmente em medicina, a análise ROC possui uma literatura bastante extensa relacionada a sua aplicação a teste de diagnósticos (Zou, 2007).

O trabalho de Spackman (1989) é considerado um dos primeiros trabalhos que se tem conhecimento a utilizar análise ROC em aprendizado de máquina

com o objetivo de avaliar e comparar algoritmos de aprendizado. Nos últimos anos, com a necessidade de métricas mais ricas para realizar avaliações mais apuradas de algoritmos de aprendizado (Provost and Fawcett, 2001), a análise ROC vem ganhando bastante atenção da comunidade de aprendizado de máquina.

Existem duas situações nas quais a análise ROC é especialmente interessante,

1. classes desbalanceadas, isto é, quando uma classe tem uma proporção de exemplos maior ou muito maior que a outra classe, e

2. custo de classificação diferente para uma das classes.

Essas duas situações são explicadas no decorrer do capítulo.

4.1.1 Medidas Utilizadas

Dado um classificador binário que classifica exemplos em positivo ou negativo e um conjunto de exemplos rotulados, ao predizer a classe dos exemplos desse conjunto e compará-los com a classe verdadeira, pode-se distinguir quatro diferentes situações:

verdadeiro positivo	: exemplo positivo	predito como positivo
falso positivo	: exemplo negativo	predito como positivo
verdadeiro negativo	: exemplo negativo	predito como negativo
falso negativo	: exemplo positivo	predito como negativo

Quando um conjunto de exemplos é classificado, pode-se contar quantos exemplos se enquadram em cada uma dessas categorias. Com essa contagem pode-se compor uma matriz de contingência, ilustrada na Tabela 4.1, onde *VP*, *FP*, *VN* e *FN* representam, respectivamente, o número de exemplos verdadeiros positivos, falso positivos, verdadeiros negativos e falso negativos. *Pos* é o total de exemplos positivos; *Neg* é o total de exemplos negativos; *PPos* é o total de exemplos preditos positivos; *PNeg* é o total de exemplos preditos negativos e *Total* é o número total de exemplos no conjunto de exemplos considerado.

Tabela 4.1: Matriz de contingência

	preditos positivo	preditos negativo	
exemplos positivo	*VP*	*FN*	*Pos*
exemplos negativo	*FP*	*VN*	*Neg*
	PPos	*PNeg*	*Total*

Com esses valores é possível definir as seguintes medidas, onde *TVP* indica a taxa de verdadeiros positivos e *TFP* a taxa de falsos positivos:

$$TVP = \frac{\text{positivos classificados corretamente}}{\text{total de positivos}} = \frac{VP}{Pos} \tag{4.1}$$

$$TFP = \frac{\text{negativos classificados incorretamente}}{\text{total de negativos}} = \frac{FP}{Neg} \tag{4.2}$$

$$acurácia = \frac{VP}{PPos} \tag{4.3}$$

$$precisão = \frac{VP+VN}{Total} \tag{4.4}$$

$$erro = 1 - precisão \tag{4.5}$$

Algumas dessas medidas podem ter outros nomes na literatura, como a *TVP* que também é conhecida como *recall* ou sensibilidade (*sensitivity*). Também, *TFP* é equivalente a 1 - especificidade (*specificity*).

Entre as medidas apresentadas, a *precisão* é a mais conhecida e talvez a mais utilizada. No entanto, quando utilizada em problemas de classes desbalanceadas, ela pode "esconder" o erro de predição da classe minoritária. Isso pode ser verificado ao observar o termo *Total* que soma indistintamente exemplos positivos e negativos na fórmula de *precisão*. O exemplo a seguir ilustra essa característica indesejada da medida de precisão.

Exemplo 4.1 *Considere um conjunto de 1000 exemplos de teste, dos quais 100 são da classe positiva e 900 da classe negativa. Ao classificar os 1000 exemplos obteve-se a matriz de contingência ilustrado na Figura 4.2.*

Tabela 4.2: Exemplo 4.1: matriz de contingência

	preditos positivo	preditos negativo	
exemplos positivo	10	90	100
exemplos negativo	10	890	900
	20	980	1000

Dos 100 exemplos positivos, quase todos (90) foram rotulados incorretamente, enquanto que, na classe negativa, dos 900 exemplos quase todos (890) foram rotulados corretamente. O classificador, apesar de ser muito bom para classificar a classe negativa, deixa muito a desejar na classificação da classe positiva. Calculando[1] as medidas definidas anteriormente obtém-se:

[1]Neste trabalho adotou-se o padrão americano de representação decimal que utiliza ponto ao invés de virgula, o qual é também utilizado pelos softwares gráficos e as implementações realizadas neste trabalho

- $TVP = \frac{10}{100} = 0.1 = 10\%$
- $TFP = \frac{10}{900} = 0.011 = 1.1\%$
- $acurácia = \frac{10}{20} = 0.5 = 50\%$
- $precisão = \frac{10+890}{100+900} = 0.9 = 90\%$

Com esse simples exemplo, pode-se notar algumas pequenas nuances de cada medida:

- a *TVP* expressa a qualidade de classificação da classe positiva. Como esperado, essa medida, para o exemplo considerado, indica um resultado que deixa a desejar (10%).
- a *TFP* expressa a qualidade de classificação da classe negativa como uma taxa de erro e, portanto, quanto menor, melhor é a classificação da classe negativa. Assim, o resultado pode ser considerado muito bom (1.1%).
- a *acurácia* expressa a proporção de exemplos preditos como positivos corretamente. Dos 20 exemplos preditos como positivos, somente a metade (50%) deles foram corretamente classificados.
- a *precisão* expressa a percentagem de exemplos rotulados corretamente, indistintamente da classe. Indica um resultado aparentemente bom (90%), apesar de não ser um bom classificador para a classe positiva.

Deve ser observado que a análise ROC desenha pontos sempre utilizando a *TVP* e a *TFP*. Um modo bastante conveniente para verificar como algumas medidas se comportam é reescrevê-las utilizando essas duas taxas. Por exemplo, a equação de *precisão* pode ser reescrita em termos da *TVP* e *TFP* da seguinte maneira:

$$precisão = \frac{Pos \times TVP + Neg \times (1 - TFP)}{Total} \tag{4.6}$$

Desse modo, pode-se observar claramente como a *precisão* se relaciona com a *TVP* e *TFP*. Observe que tanto a *TVP* quando a *TFP* possuem em seus denominadores a quantidade de exemplos positivos e exemplos negativos, respectivamente, o que as torna sensível à variação da proporção de classes. Ao reescrever a equação de *precisão* como mostra a Equação 4.6, os produtos $Pos \times TVP$ e $Neg \times (1 - TFP)$ removem, respectivamente, a quantidade de exemplos positivos e negativos dos denominadores de *TVP* e *TFP*.

4.1.2 Gráfico ROC

O gráfico ROC é um gráfico bidimensional no qual os eixos Y e X do gráfico sempre representam as medidas *TVP* e *TFP* respectivamente. Na Figura 4.1 é ilustrado um gráfico ROC identificando quatro regiões importantes do gráfico, descritas a seguir, e uma linha diagonal que representa classificadores aleatórios:

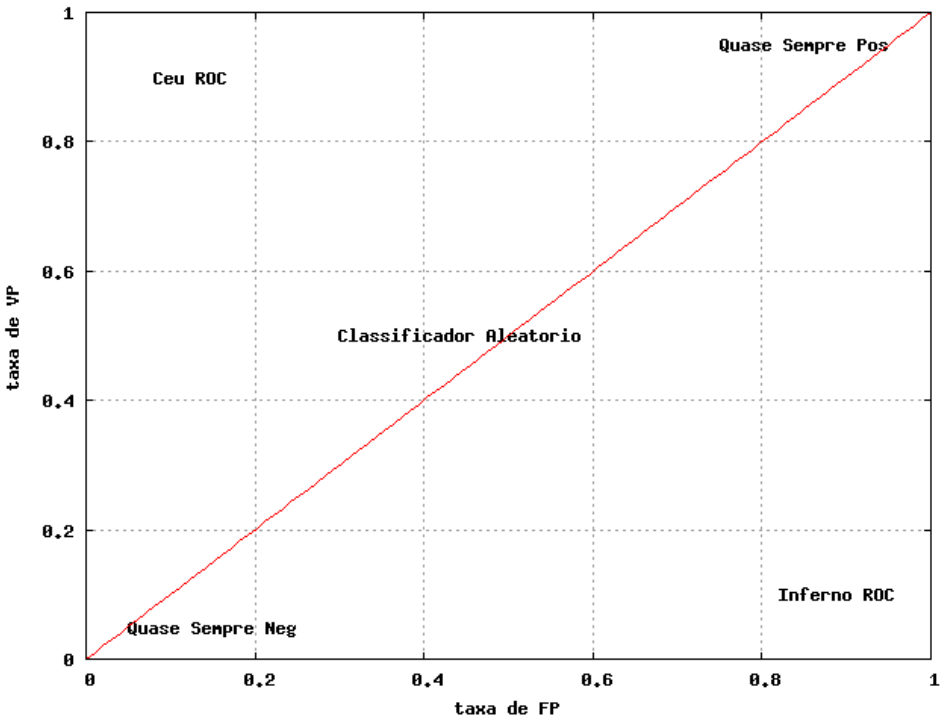


Figura 4.1: Gráfico ROC

Céu ROC: O ponto (0,1) representa uma classificação perfeita, na qual todos os exemplos positivos e negativos são rotulados corretamente. Na região *Céu ROC* encontram-se os pontos mais próximos da classificação perfeita e representam bons resultados.

Inferno ROC: região localizada no lado oposto ao *Céu*, pode ser considerada uma região na qual são encontrados os resultados "ruins". No entanto, classificadores representados nessa região possuem informações com capacidade de distinguir as classes, mas não as utilizam corretamente (Flach and Wu, 2005). Observe que, ao inverter os rótulos, o ponto que era (x,y) passa a ser (y,x), *i.e.*, passa do inferno para o céu ROC.

Quase Sempre Neg: classificadores que são representados nessa região rotulam quase sempre os exemplos como negativos. Assim, o número de exemplos negativos rotulados errados normalmente é baixo (*TFP* próximo de 0) e número de exemplos positivos rotulados corretamente também é baixo (*TVP* próximo de 0).

Quase Sempre Pos: classificadores que são representados nessa região rotulam quase sempre os exemplos como positivos. Assim, quase todos os exemplos positivos são rotulados corretamente (*TVP* próximo de 1), e quase todos os exemplos negativos incorretamente (*TFP* próximo de 1).

Normalmente, os classificadores representados por pontos próximos a linha diagonal são considerados classificadores aleatórios, *i.e.*, não possuem informação sobre a classe. Classificadores aleatórios sorteiam aleatoriamente a classe que atribuem aos exemplos.

Exemplo 4.2 *Suponha que seja fornecido um conjunto de dados de 100 exemplos no qual a proporção das classes seja igual (50% positivos e 50% negativos). Considere os seguintes exemplos de classificadores aleatórios:*

50% positivos e 50% negativos *Se o classificador rotula aleatoriamente exemplos como positivo na metade das vezes (50 exemplos), então é esperado que ele acerte metade desses exemplos (25 exemplos) obtendo TVP* = $\frac{25}{50} = 0.5$. *De modo similar, dos outros 50 exemplos negativos, espera-se que o classificador erre 25 exemplos obtendo TFP* = $\frac{25}{50} = 0.5$. *Isso representa o ponto (0.5,0.5) que está na diagonal do gráfico ROC.*

80% positivos e 20% negativos *Se o classificador rotula aleatoriamente 80 exemplos como positivos, é esperado que ele acerte 80% desses exemplos (64 exemplos) obtendo TVP* = $\frac{64}{80} = 0.8$. *De modo similar, dos 20 exemplos rotulados como negativos espera-se que se erre 80% (16 exemplos) obtendo TFP* = $\frac{16}{20} = 0.8$. *Isso representa o ponto (0.8,0.8), que também está na diagonal do gráfico ROC.*

Observe que os pontos (0,0) e (1,1) também representam pontos na diagonal e são os casos extremos de *Quase Sempre Neg* e *Quase Sempre Pos*, nos quais, sempre se rotula como negativo todos os exemplos (*TVP* = 0, *TFP* = 0) e sempre se rotula positivamente todos os exemplos (*TVP* = 1, *TFP* = 1), respectivamente, sendo também exemplos de classificadores aleatórios.

O gráfico ROC compara diferentes classificadores de acordo com suas *TVP* e *TFP* em um espaço bidimensional. Esse espaço bidimensional também é conhecido como *espaço ROC*. Entretanto, em algumas situações, como árvore de decisão e aprendizado de regras, as vezes é mais conveniente pensar em termos absolutos do número de exemplos. Desse modo, ao invés de plotar *TVP* e *TFP*, são plotados *VP* e *FP* que são os números absolutos de exemplos classificados como positivo. Em aprendizado por regras isso é equivalente aos exemplos coberto pela(s) regra(s). Esse espaço é denominado de *espaço de cobertura* (Fürnkranz and Flach, 2005).

Na literatura, é bastante comum encontrar problemas nos quais o gráfico ROC é utilizado para estudar um classificador *score*. Porém, muitas vezes, essa análise é confundida com a análise ROC comparando diversos classificadores induzidos utilizando diferentes algoritmos de aprendizado. A fim de não confundir estes dois problemas, neste trabalho o estudo de análise ROC é dividido em dois casos:

Caso 1 gráficos ROC nos quais os pontos representam classificadores induzidos utilizando diferentes algoritmos indutores, e

Caso 2 gráficos ROC nos quais a curva representa classificadores obtidos de diferentes limiares de um classificador *score*.

No primeiro caso, apresentado na Seção 4.1.3, os classificadores são representados por pontos; no segundo caso, apresentado na Seção 4.1.4, os classificadores *score* são representados por uma curva.

4.1.3 Caso 1: Classificadores versus Classificadores

Uma maneira bastante interessante de comparar e analisar classificadores é desenhar os pontos respectivos a cada classificador no gráfico ROC. No gráfico ROC é possível visualizar quais classificadores privilegiam alguma das classes, quais os classificadores potencialmente ótimos e quais classificadores podem ser descartados da análise.

Em linha gerais, pontos que aparecem do lado esquerdo do gráfico ROC, próximos ao eixo Y, podem ser mais conservadores pois classificam apenas quando possuem forte evidência dos exemplos serem positivos. Desse modo, poucos exemplos negativos são classificados como positivos, *i.e.*, baixa *taxa de FP*. Em contrapartida, os pontos que aparecem do lado superior do gráfico podem representar classificadores menos conservadores ao classificar positivos e por isso apresentam altas *taxas de VP*.

Em alguns domínios de aplicação, o custo de se classificar errado um exemplo positivo é mais alto que classificar um exemplo como negativo. Em aplicações médicas isso é bastante comum. Enviar um paciente para sala de cirurgia sem necessidade é financeiramente custoso além dos traumas naturais que um paciente pode sofrer ao ser submetido a um procedimento cirúrgico sem a menor necessidade. Um outro domínio na qual o custo de classificação é diferente são os filtros de *spam*. Classificar um e-mail importante como *spam* é bastante grave, enquanto que classificar um *spam* como não *spam* é bem menos grave.

Na Figura 4.2 é ilustrado um exemplo de gráfico ROC com diversos classificadores que classificam os exemplos em *spam* e não *spam*. Entre os diversos pontos apresentados, os mais próximos à classificação ideal (0,1) são

os classificadores *B* e *C*. O classificador *A* é o mais conservador entre todos o classificadores apresentados. O classificador *D* é o classificador que consegue classificar corretamente todos os positivos, mas classifica erroneamente metade dos negativos. Qual seria o melhor classificador de e-mails?

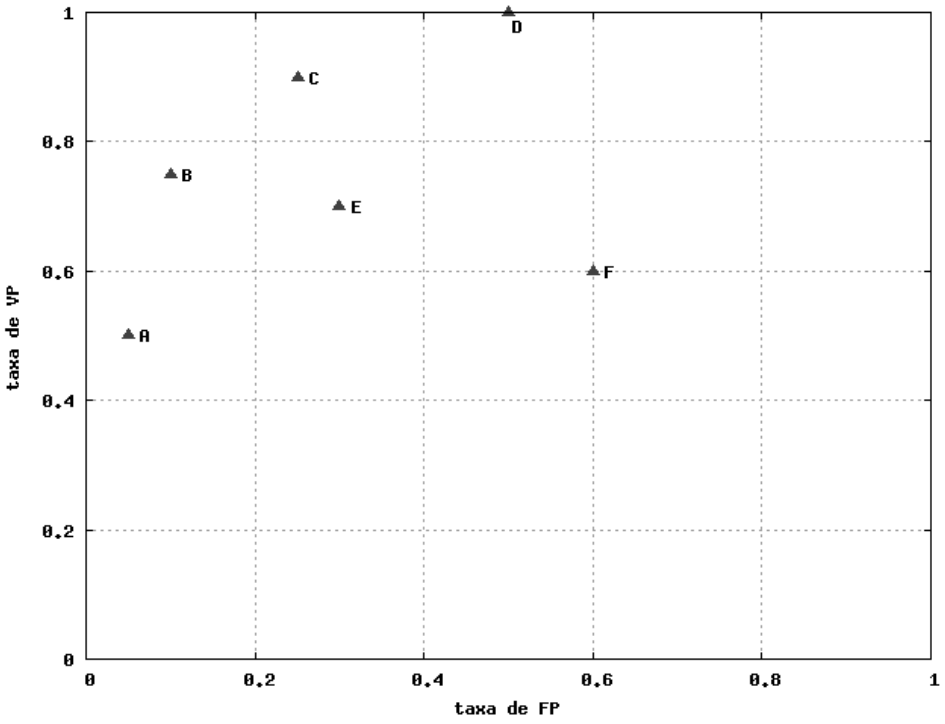


Figura 4.2: Representação de múltiplos classificadores em gráficos ROC

Ao considerar exemplos positivos não *spam* — e-mail importantes —, é preferível um classificador que não perca nenhum exemplo positivo (alta *taxa de VP*) mesmo que sejam rotulados alguns *spam* como não *spam*. Assim, o classificador escolhido para essa tarefa é o *D* que não erra na classificação de exemplos positivos, mesmo classificando vários *spam* como não *spam*.

Assim, pode ser observado que a resposta sobre qual seria o melhor classificador ao analisar um gráfico ROC é bastante dependente do domínio de aplicação. Em linhas gerais, quando um erro de classificação de exemplos positivos é grave, deve-se dar preferência para os pontos na parte superior do gráfico (alta *taxa de VP*). Quando um exemplo classificado como positivo, mas que é negativo, é grave, deve-se dar preferência para os pontos mais próximos ao eixo *Y* (baixa *taxa de FP*).

4.1.4 Caso 2: ROC de um Classificador Score

Como já mostrado, os pontos da curva ROC representam diferentes *taxas de VP* e *taxas de FP* que são obtidas a partir de diferentes matrizes de contingência. No caso de classificadores *score* surgem várias questões: *Como é possível obter diferentes matrizes de contingência a partir de um classificador*

Tabela 4.3: Classificação utilizando diferentes limiares

exemplo	A	B	V	C	D	W	X	Y	E	Z
posição	1	2	3	4	5	6	7	8	9	10
classe	p	p	n	p	p	n	n	n	p	n
score	0.9	0.8	0.4	0.39	0.38	0.37	0.36	0.2	0.2	0.05
limiar = 0.5	p	p	n	n	n	n	n	n	n	n
limiar = 0.9	p	n	n	n	n	n	n	n	n	n
limiar = 0.8	p	p	n	n	n	n	n	n	n	n
limiar = 0.4	p	p	p	n	n	n	n	n	n	n
limiar = 0.39	p	p	p	p	n	n	n	n	n	n
limiar = 0.38	p	p	p	p	p	n	n	n	n	n
limiar = 0.37	p	p	p	p	p	p	n	n	n	n
limiar = 0.36	p	p	p	p	p	p	p	n	n	n
limiar = 0.2	p	p	p	p	p	p	p	p	p	n
limiar = 0.05	p	p	p	p	p	p	p	p	p	p

score? O que eles significam e para que eles servem? Quando esse tipo de análise pode ser potencialmente útil?

Considere um classificador *score* induzido utilizando o algoritmo NAIVE BAYES, para o qual é posteriormente submetido um conjunto de teste com 10 exemplos. Em um primeiro momento, apenas são utilizados os *scores*. Os exemplos classificados são ordenados utilizando os *scores* ilustrados na Tabela 4.3. Na primeira linha dessa tabela são apresentados os identificadores de cada um dos 10 exemplos de teste; na segunda linha a posição obtida pela ordenação dos scores; na terceira linha a classe verdadeira de cada exemplo de teste e, após, o *score* fornecido pelo classificador.

Vale ressaltar que em análise ROC, os *scores* são normalmente valores de confiança dos exemplos a serem classificados como positivos. No caso de NAIVE BAYES, o *score* de um exemplo ser negativo pode ser obtido por 1 - *score* de ele ser positivo, já que geralmente os valores são normalizados, como visto no Capítulo 2. Assim, quando o *score* é maior ou igual que 0.5 o exemplo é classificado como positivo e quando o *score* é menor que 0.5 é classificado como negativo. Na Tabela 4.3 são ilustrados diferentes limiares que podem ser utilizados. O limiar padrão (*default*) utilizado por NAIVE BAYES é 0.5, ilustrado na linha 5 da tabela.

Para cada limiar pode-se construir uma matriz de contingência diferente. Dessa maneira, pode-se fazer uma análise mais detalhada e escolher um limiar melhor para o classificador apresentado. Note que na Tabela 4.3, os valores estão bastante concentrados em 0.3 e o limiar que possui a melhor *precisão* é igual a 0.38. Apesar de ser um exemplo artificial, é bastante comum encontrar situações reais nas quais o melhor limiar é diferente de 0.5.

Para ter uma melhor visualização dos possíveis limiares, pode-se usar vários limiares, como mostrado na Tabela 4.3, gerar as matrizes de contingencia correspondentes, obter os valores de *TVP* e *TFP* e plotar os pontos no gráfico

ROC. Na Tabela 4.4 é mostrado, em cada linha, os valores de VP, FN, FP e VN para construir uma matriz de contingência e nas duas últimas colunas a *TVP* e a *TFP* necessárias para plotar um ponto no gráfico ROC.

Tabela 4.4: Resumo das matrizes de contingência obtidas utilizando diferentes limiares

	VP	FN	FP	VN	TVP	TFP
limiar = 0.9	1	4	0	5	0.2	0.0
limiar = 0.8	2	3	0	5	0.4	0.0
limiar = 0.4	2	3	1	4	0.4	0.2
limiar = 0.39	3	2	1	4	0.6	0.2
limiar = 0.38	4	1	1	4	0.8	0.2
limiar = 0.37	4	1	2	3	0.8	0.4
limiar = 0.36	4	1	3	2	0.8	0.6
limiar = 0.2	5	0	4	1	1.0	0.8
limiar = 0.05	5	0	5	0	1.0	1.0

Observe que ao decrementar o limiar, quando não existem empates, um exemplo que estava em FN passa para VP, ou um exemplo que estava em VN passa para FP. Isso sugere um método simples para desenhar a curva ROC, o qual consiste em avaliar cada posição do *ranking* da seguinte maneira: começando sempre da posição (0,0) do gráfico, para cada exemplo, desenhe:

1. exemplo positivo: caso a classe verdadeira do exemplo seja positiva move-se $\frac{1}{Pos}$ para cima, lembrando que *Pos* é o número total de exemplos positivos;

2. exemplo negativo: caso a classe verdadeira do exemplo seja negativa move-se $\frac{1}{Neg}$ para direita, lembrando que *Neg* é o número total de exemplos negativos;

3. em caso de empate no *ranking*, contar o número de positivos (n^+) e número de negativos (n^-) que compõem o empate e desenhar uma linha diagonal que liga o ponto atual (x,y) e $(x+n^- \times \frac{1}{Neg}, y+n^+ \times \frac{1}{Pos})$.

Assim, partindo da posição (0,0) no gráfico ROC e seguindo a ordem de exemplos determinada pelos *scores*, move-se para cima ou para direita dependendo da classe ou, em caso de empate, move-se na diagonal. A junção desses segmentos é a *curva ROC*

Em um *espaço de cobertura*, o procedimento é praticamente o mesmo, a única diferença está no tamanho dos passos, que ao invés de ser $\frac{1}{Pos}$ e $\frac{1}{Neg}$ é de tamanho 1. Desse modo, o céu ROC fica em (0,Pos) ao invés de (0,1); o inferno ROC fica em (Neg,0) ao invés de (1,0) e o ponto (1,1) no gráfico ROC é (Neg,Pos) no *espaço* de cobertura.

Na Figura 4.3 é ilustrada a curva ROC construída com o *ranking* obtido na Tabela 4.3, onde cada segmento é identificado pelo par "seg:class". Por exemplo "3:n" significa terceiro segmento, classe negativa.

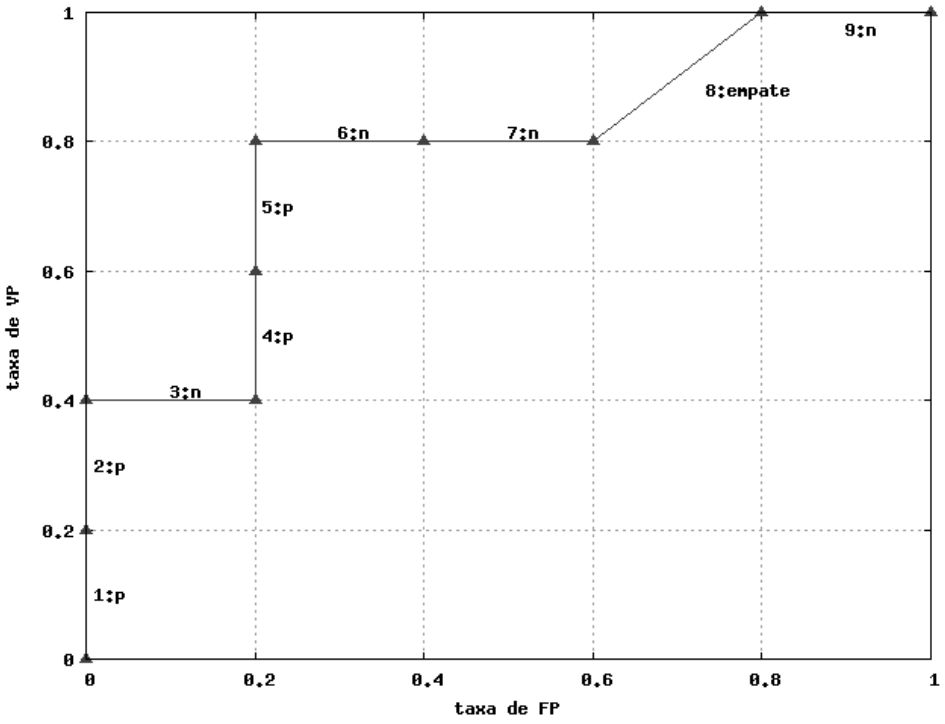


Figura 4.3: Construção de uma curva ROC

No contexto de teste considerado , tem-se 5 exemplos positivos e 5 exemplos negativos. Assim, cada passo para cima terá tamanho $\frac{1}{5} = 0.2$ e cada passo para direita terá tamanho $\frac{1}{5} = 0.2$. Seguindo essa ordem do *ranking* e partindo de (0,0) tem-se;

1. um exemplo positivo, move-se um passo para cima, (0,0+0.2);
2. um exemplo positivo, move-se um passo para cima, (0,0.2+0.2);
3. um exemplo negativo, move-se um passo para direita, (0+0.2,0.4);
4. um exemplo positivo, move-se um passo para cima, (0.2,0.4+0.2);
5. um exemplo positivo, move-se um passo para cima, (0.2,0.6+0.2);
6. um exemplo negativo, move-se um passo para direita, (0.2+0.2,0.8);
7. um exemplo negativo, move-se um passo para direita, (0.4+0.2,0.8);
8. empate de um exemplo positivo e um exemplo negativo, move-se na diagonal que liga um passo para cima e um passo para direita, (0.6+0.2,0.8+0.2);
9. e finalmente o último exemplo que é negativo, move-se um passo para direita, (0.8+0.2,1).

Dessa maneira, além de ser bastante prático e rápido o procedimento para construir uma curva ROC, ele gera exatamente a mesma curva se comparado

com o procedimento de construção no qual se calcula a matriz de contingência para cada limiar. Todas as interpretações realizadas sobre gráficos ROC na Seção 4.1.3 na página 47, podem também ser feitas aqui. Além de prático e rápido, esse procedimento de construção da curva permite verificar alguns aspectos interessantes da curva ROC.

Geometricamente, quando um exemplo negativo aparece antes de um exemplo positivo no *ranking*, a curva ROC vai em direção à direita prematuramente. Quando isso ocorre, o exemplo negativo causa um aumento na área sobre a curva, afastando a curva do ponto ideal (0,1). Por exemplo, observe o segmento 3:n na Figura 4.3. Caso o exemplo negativo que representa o segmento 3:n estivesse após 4:p e 5:p, a área sobre a curva seria menor.

Para que seja desenhada uma curva ROC perfeita, passando pelo ponto (0,1), é necessário que todos os exemplos positivos apareçam antes dos negativos. Seguindo a idéia de que para cada exemplo positivo desenha-se um segmento para cima e para cada exemplo negativo desenha-se um segmento para a direita, se todos os exemplos positivos aparecerem antes dos negativos, então é desenhada uma reta vertical até o ponto (0,1), seguida por segmentos para a direita, até atingir o ponto (1,1). Isso é bastante interessante pois, para que a curva ROC seja perfeita, a ordem interna dos positivos ou negativos não é relevante. Em outras palavras, não importa se os *scores* dos exemplos positivos estão embaralhados entre si, um *score* alto para um exemplo positivo aparece antes ou depois de um exemplo positivo com *score* mais baixo, importa apenas que todos os exemplos positivos apareçam antes dos negativos.

Como a curva ROC é desenhada inteiramente baseada na posição do *ranking* dos exemplos, a escala em que os *scores* estão não é relevante, importando somente a ordem imposta pelos *scores* e os empates. Os valores numéricos obtidos da estimação da probabilidade a posteriori, da distância da margem de decisão ou da contagem de vizinhos mais próximos, podem ser utilizados sem maiores problemas para desenhar a curva ROC.

4.1.5 Iso-desempenho

Um outro conceito bastante útil para auxilar na escolha do classificador mais apropriado são as linhas ou curvas de *iso-desempenho*. Essas linhas são desenhadas no gráfico ROC de tal modo que todos os pontos que compõem essa linha tenham o mesmo desempenho para uma determinada medida. Por exemplo, pode-se ter uma linha no gráfico ROC que representa todos os classificadores com *precisão* igual a 90%. É interessante observar que em casos de classes desbalanceadas pode-se ter pontos em lugares muitas vezes inesperados. Pode-se obter linhas de *iso-desempenho* para diferentes medidas como *precisão* e *acurácia*.

Para obter essas linhas, reescreve-se uma medida em termos de *TVP* e *TFP* como apresentado anteriormente na Equação 4.6 na página 44 para a *precisão*. Dividindo-se o numerador e o denominador por $\frac{1}{Pos}$ obtém-se a seguinte equação:

$$precisão = \frac{TVP + c \times (1 - TFP)}{1 + c} \tag{4.7}$$

onde $c = \frac{Neg}{Pos}$ representa a taxa de desbalanceamento que é a inversa da chance a priori (*prior odds*) a qual é dada por $\frac{\hat{P}(+)}{\hat{P}(-)} = \frac{Pos}{Neg} = \frac{1}{c}$

Na Figura 4.4 são apresentas linhas de *iso-desempenho* para valores de *precisão* iguais a 0.1, 0.2, 0.3, 0.4, ..., 0.9. As linhas contínuas representam linhas de *iso-desempenho* para problemas com classes balanceadas ($c = 1$), as linhas pontilhadas representam linhas de *iso-desempenho* para um problema no qual existe o dobro de exemplos positivos, $c = \frac{1}{2}$. A taxa de desbalanceamento atua como um coeficiente angular para a medida de *precisão*, na qual, quanto maior a proporção de exemplos positivos, menor o coeficiente angular da linha de *iso-desempenho*, *i.e.*, mais horizontal é a linha.

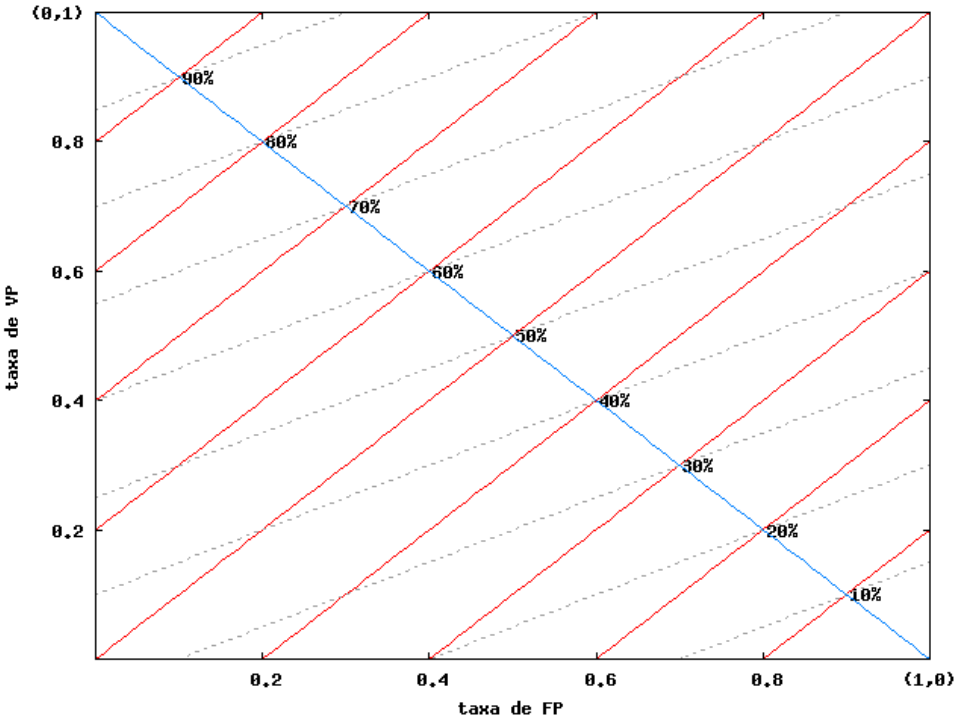


Figura 4.4: Linhas de *iso-desempenho* para *precisão*. As linhas contínuas representam *precisão* para problemas de classes balanceadas $c = 1$. As linhas pontilhadas para problemas nos quais existe o dobro de exemplos positivos $c = \frac{1}{2}$

Pode-se verificar que a diagonal que liga (0,1) e (1,0) contém pontos que são invariáveis ao valor de c e, mais interessante ainda, o mapeamento des-

ses pontos na coordenada no eixo Y representa a *precisão*. Veja a Figura 4.5 que ilustra um classificador A que toca a linha de iso-desempenho de *precisão* para $c = \frac{1}{2}$. Como a *precisão* pode ser obtida diretamente no eixo Y, sabe-se que a *precisão* do classificador A é igual a 80%. Esse é um artifício bastante interessante para obter a *precisão* diretamente da curva ROC e pode ser utilizado para responder perguntas como: *otimizar ranking é o mesmo que otimizar classificação (precisão)?*

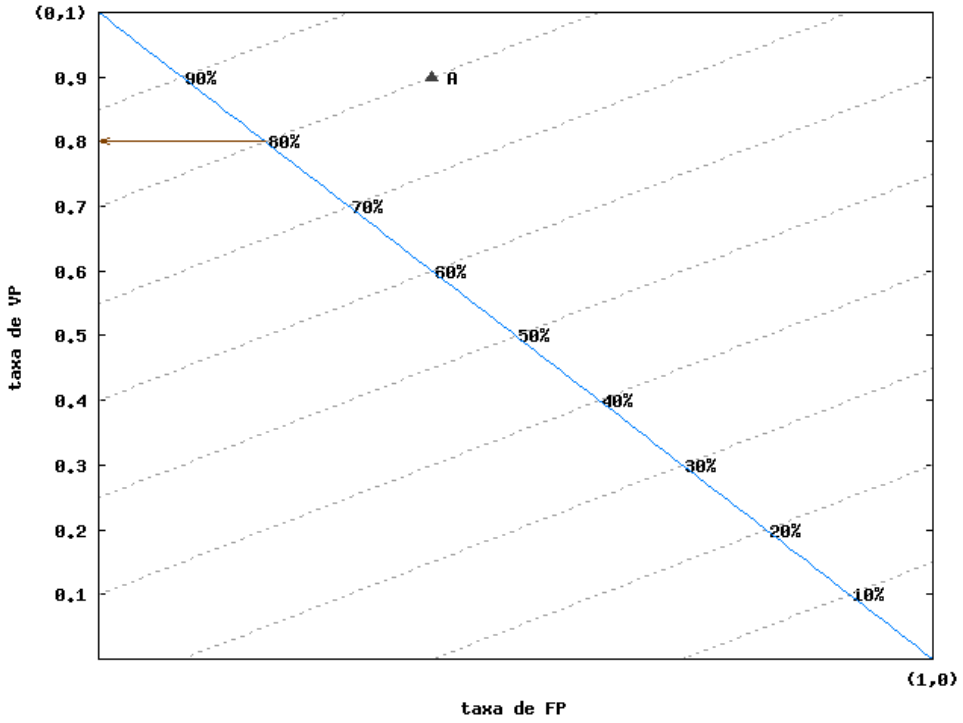


Figura 4.5: Obtenção da *precisão* com linhas de *iso-desempenho*

Flach (2003) mostra curvas de *iso-desempenho* para medidas como ganho de informação, DKM, GINI e *Odds Ratio* utilizadas para particionar uma árvore de decisão. Essas curvas de *iso-desempenho* permitem mostrar a sensibilidade ao problema de classes desbalanceadas de cada medida observando as distorções causadas na curva.

As curvas/linhas de *iso-desempenho* podem ser utilizadas para auxiliar na escolha do melhor classificador quando se conhece c. Esse método consiste em:

1. desenhar uma linha de *iso-desempenho* com o c fixo no ponto (0,1).
2. mover a linha para baixo, sem mudar seu coeficiente angular, até que se atinja um ponto que compõe o gráfico ROC.

Como mostrado em (Provost and Fawcett, 2001), esse ponto é sempre ótimo. Esse método sempre busca a região mais ao noroeste do gráfico ROC que intersecta o ponto de um classificador.

Variar c para obter todas as proporções de classes possíveis de tal modo que se consiga obter um conjunto de pontos ótimos para qualquer c é equivalente a obter o feixe convexo do gráfico ROC, o qual é descrito a seguir.

4.1.6 Feixe Convexo do Gráfico ROC

O feixe convexo de gráficos ROC (*ROC Convex Hull*), é um método que combina técnicas de análise ROC, geometria computacional e análise de decisões (Provost and Fawcett, 2001). Aparentemente simples, consistindo apenas em obter o feixe convexo dos pontos que compõem um gráfico ROC, é um método bastante poderoso, pois permite obter os pontos no gráfico ROC que são potencialmente ótimos.

Considere um gráfico ROC no qual são comparados diferentes classificadores de diferentes indutores (caso 1 descrito na Seção 4.1.3). Na Figura 4.6 é ilustrado o feixe convexo obtido pelos pontos da Figura 4.2. Ao variar c e desenhar as *iso-desempenho* de *precisão* para procurar pelos melhores pontos, obtém-se sempre os pontos que compõem o feixe convexo da curva ROC. Fawcett (2006) mostra que o feixe convexo indica não apenas os pontos potencialmente ótimos para *precisão*, mas para diversas medidas. Essas implicações são bastante úteis já que somente os classificadores que estão no feixe convexo são potencialmente ótimos e não existe motivo para manter os pontos (classificadores) que estão fora do feixe (Flach, 2003; Fawcett, 2006).

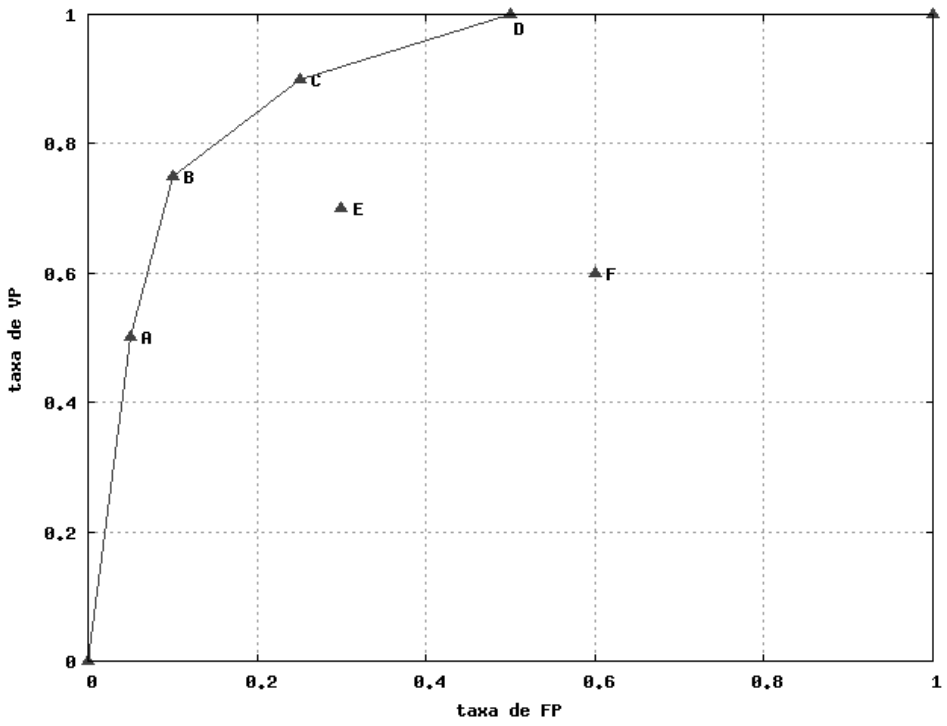


Figura 4.6: Feixe convexo de gráficos ROC

Existem diversas outras propriedades do feixe convexo. Um exemplo inte-

ressante pode ser encontrado em (Provost and Fawcett, 2001), que constroem um classificador híbrido, obtendo diversos feixes convexos de diversos classificadores *score*. Esse classificador híbrido é considerado robusto para aplicações que mudam o custo de classificação, ou a proporção das classes, com o passar do tempo. Uma outra aplicação interessante é o uso de feixe convexo para calibrar probabilidades, descrita no Capítulo 5.

4.1.7 Área Abaixo da Curva ROC

Quando diversas curvas ROC são obtidas de diferentes classificadores *score*, é normal e bastante comum tentar compará-las. No entanto, curvas ROC são bidimensionais e são representadas por segmentos. *Como comparar algo que é bidimensional ou que é representado por segmentos de curvas?*

Como já mencionado, um exemplo negativo que aparece antes de um exemplo positivo faz com que a curva vá para a direita prematuramente, aumentando a área sobre a curva e diminuindo a área abaixo da curva. A área abaixo da curva, denominada AUC (*Area Under ROC Curve*), sendo uma grandeza escalar de apenas uma variável, é uma medida bastante utilizada para comparação de múltiplas curvas.

Para ilustrar, considere o exemplo da Tabela 4.3. Na curva ROC da Figura 4.7 estão representados os 10 exemplos, 5 positivos (A, B, C, D, E) e 5 negativos (V, W, X, Y, Z). Com esses 10 exemplos é possível obter $5 \times 5 = 25$ pares de exemplos positivos e negativos {(A,V), (A,W), ..., (E,Y), (E,Z)}. Observe que existem 25 quadrados demarcados por linhas pontilhadas. Cada quadrado representa um par de exemplos positivo e negativo.

Os quadrados da primeira linha representam todos os pares do primeiro exemplo positivo A com os 5 exemplos negativos (A,V), (A,W), (A,X), (A,Y),(A,Z). Nessa linha existe apenas o segmento 1:p que está a esquerda de todos os quadrados, isso mostra que todos os pares estão corretos. O mesmo acontece na segunda linha para o exemplo B.

Na terceira linha existem dois segmentos. O primeiro deles, 3:n, representa o par (V,C). Observe na Tabela 4.3 que o exemplo V (negativo) precede o exemplo C (positivo). Isso cria uma concavidade na curva afastando-a do ponto ótimo (0,1). O segundo segmento, 4:p, representa o restante dos pares {(C,W), (C,X), (C,Y), (C,Z)} que estão corretos.

Quando o par de exemplo está correto, *i.e.*, o exemplo positivo vem antes do negativo no *ranking*, existe um segmento a esquerda do quadrado. Quando ocorre o inverso, *i.e.*, o exemplo negativo vem antes do positivo no *ranking*, existe um segmento abaixo do quadrado. Quando existe empate, existe uma linha diagonal que corta o quadrado. Assim, os quadrados que estão abaixo da curva representam pares que estão corretos e os quadrados acima repre-

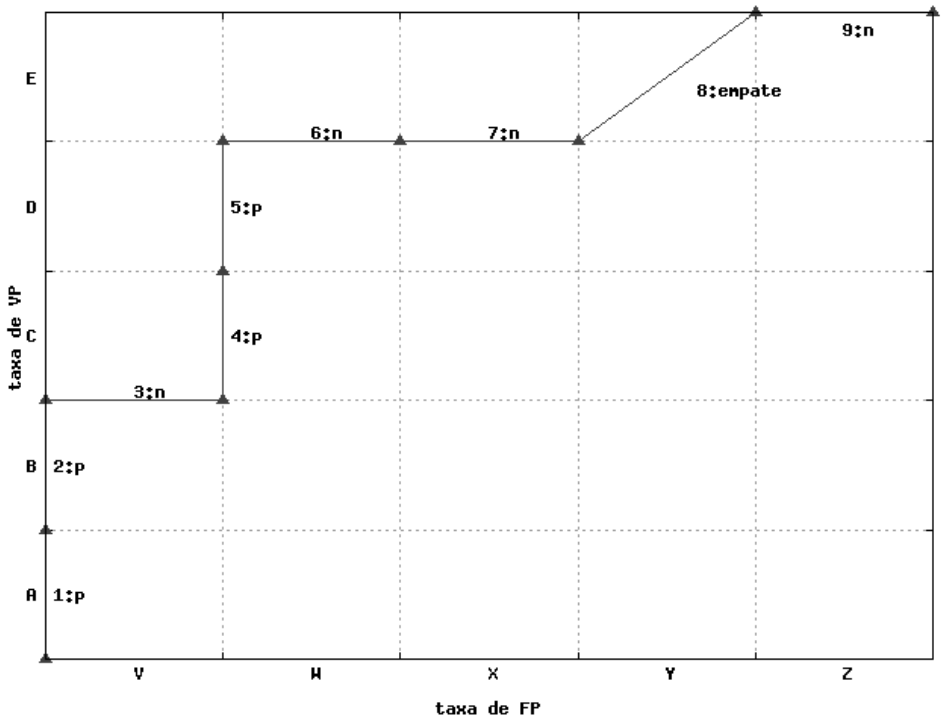


Figura 4.7: Construção de uma curva ROC

sentam pares para os quais existe uma inversão de posição no *ranking*, *i.e.*, um exemplo negativo precede pelo menos um exemplo positivo. A área abaixo da curva ROC representa a proporção de pares que estão corretos.

Uma outra maneira de descrever o significado da área abaixo da curva é considerar a escolha aleatória de dois exemplos do *ranking*, um positivo e um negativo. A probabilidade do exemplo positivo aparecer antes do negativo no *ranking* é equivalente à AUC. Descrito dessa maneira, a AUC é equivalente a estatística Wilcoxon-Mann-Whitney (Hanley and McNeil, 1982).

A análise ROC é bastante rica e pode ser utilizada das mais diferentes formas. A seguir é descrito um modo de visualizar como árvores de decisão escolhem o atributo para dividir o espaço de exemplos a cada divisão da árvore.

4.2 PROGROC

Como visto na Seção 2.3 na página 13, algoritmos de indução de árvores de decisão escolhem um valor de atributo para particionar o espaço de exemplos em hiper retângulos. Recursivamente, os ramos da árvore vão sendo refinados até que um critério de parada seja atingido. Tanto as partições quanto os diversos valores de atributos que particionam o espaço de exemplos podem ser representados em gráficos ROC. Visualizar cada passo da construção da árvore de decisão no gráfico ROC pode ser bastante interessante. Essa

idéia foi explorada, em um trabalho conjunto com o Professor Peter Flach e o Tarek Abudawood, da Universidade de Bristol. Como resultado, foi desenvolvido um sistema denominado PROGROC que permite fazer essa visualização, ilustrando passo a passo a construção da árvore em um gráfico ROC.

4.2.1 Relações entre Árvores de Decisão e ROC

Para explicar os conceitos utilizados por PROGROC, nesta seção são utilizados os exemplos apresentados em (Flach, 2007). Inicialmente, considere o conjunto de exemplos apresentado na Figura 4.8 que representa um conjunto de dados com 10 exemplos positivos e 10 exemplos negativos descritos por dois atributos binários A_1 e A_2 que dividem o espaço de exemplos em quatro regiões. Na Figura 4.9 é mostrada a respectiva árvore de decisão com quatro nós folha. As folhas 1, 2, 3 e 4 representam as regiões 1, 2, 3, 4 da Figura 4.8.

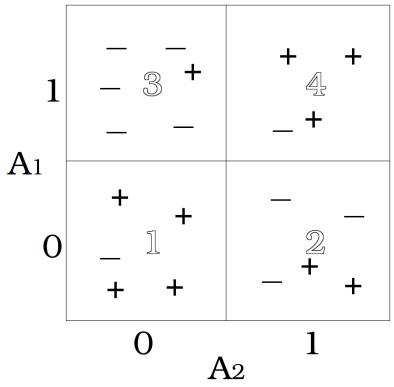


Figura 4.8: PROGROC: conjunto de exemplos

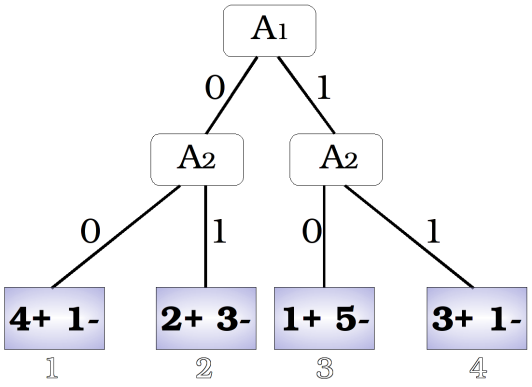


Figura 4.9: PROGROC: árvore de decisão

Cada nó folha da árvore ilustrada na Figura 4.11 pode ser diretamente representado no espaço ROC, como ilustrado na Figura 4.10. Os segmentos obedecem o procedimento de construção da curva ROC. Por exemplo, na folha 1, existem 4 positivos e 1 negativo, então move-se quatro passos para cima e um para direita e, como esses exemplos estão empatados, desenha-se uma linha diagonal que representa o retângulo. Observe na curva ROC que isso representa o primeiro segmento da curva. A ordem das folhas, 1, 3, 4, 2 na Figura 4.11é determinada pela proporção de exemplos positivos. A folha 1 possui 80% de exemplos positivos; a folha 2 possui 75%; a folha 3 possui 40% e a folha 4 possui 17% de exemplos positivos.

No procedimento para desenhar a curva ROC, o primeiro passo é ordenar os exemplos de acordo com o *score*. Variando o limiar de classificação obtido por essa ordenação, os pontos que compõem a curva são desenhados. Observe a Figura 4.13 na qual são ilustradas as rotulações obtidas variando esse limiar. A *rotulação 0* representa o ponto (0,0) do gráfico ROC na qual todas as

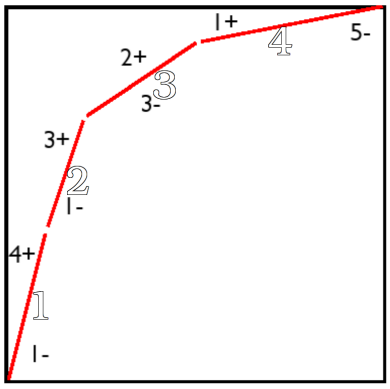


Figura 4.10: PROGROC: ROC da arvore de decisão

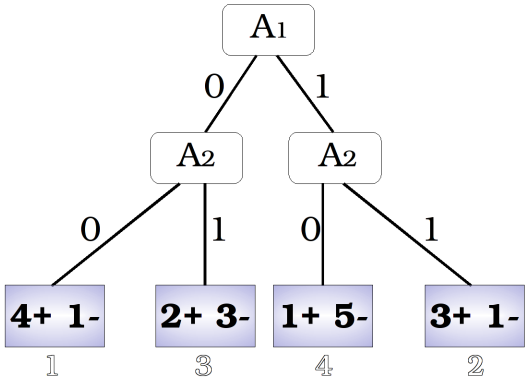


Figura 4.11: PROGROC: árvore de decisão e ROC

folhas (todos os exemplos) são rotuladas como negativo; a *rotulação 1* é obtida quando o limiar de classificação é 80% (scores maiores ou iguais a 80% são rotulados como positivo); a *rotulação 2* é obtida quando o limiar de classificação é 75%; a *rotulação 3* é obtida quando o limiar de classificação é 40%; a *rotulação 4* é obtida quando todos os exemplos são rotulados como positivos. Essas cinco rotulações representam os pontos que compõem o gráfico ROC para a árvore de decisão.

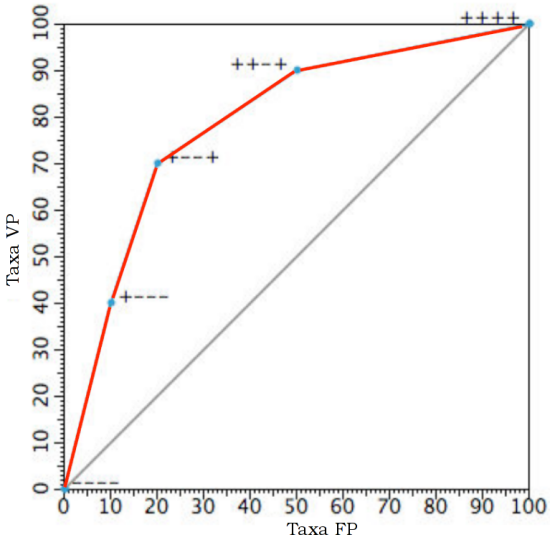


Figura 4.12: PROGROC: ROC das possíveis rotulações

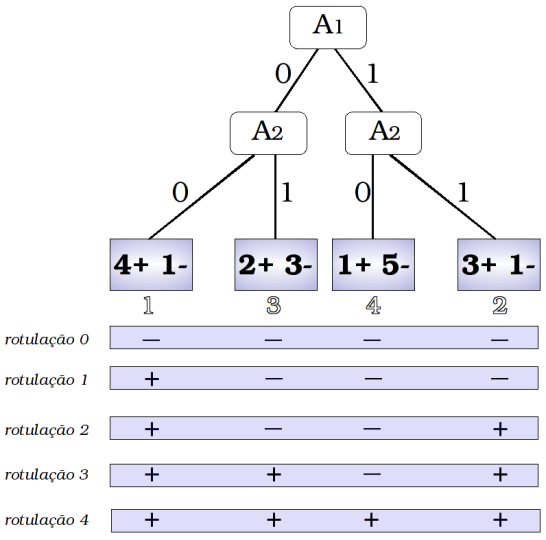


Figura 4.13: PROGROC: possíveis rotulações variando o limiar

Desse modo, com as diferentes rotulações das 4 folhas, ao analisar todas as combinações é possível obter $2^4 = 16$ maneiras diferentes de realizar rotulações. Na Figura 4.14 são ilustradas as 16 rotulações que essa árvore de decisão pode fornecer. A diagonal que representa o classificador aleatório divide o espaço ROC em dois triângulos, um superior e outro inferior. O triângulo superior é equivalente ao triângulo inferior com os rótulos invertidos. Para encontrar os pontos correspondentes, basta rotacionar o triângulo superior

em 180 graus e assim se obtém o triângulo inferior com os rótulos invertidos. Veja que a relação do triângulo superior e inferior é dada por esse rotacionamento e não pelo espelhamento. Observe o ponto + − − − que está na parte superior, ao rotacionar em 180 graus e inverter os rótulos encontra-se o seu correspondente na parte inferior − + + +.

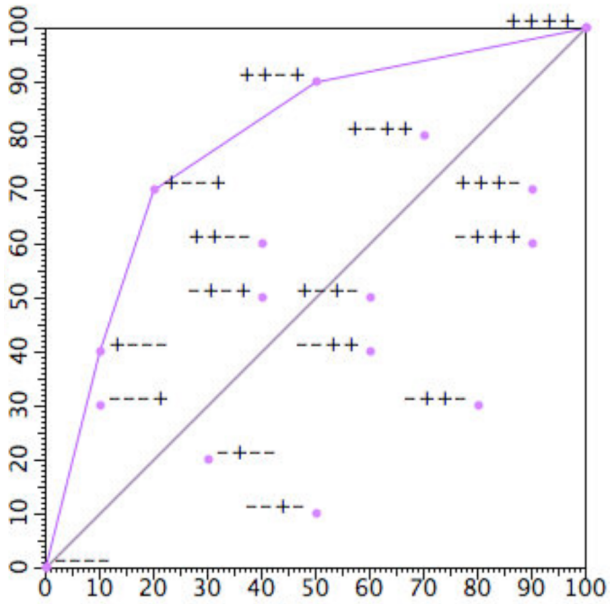


Figura 4.14: PROGROC: possíveis rotulações

As possíveis rotulações permitem visualizar os particionamentos no espaço ROC que a árvore de decisão faz nos exemplos de treinamento. Observe a árvore de decisão da Figura 4.16 e o gráfico ROC na Figura 4.15. A rotulação dos exemplos com essa árvore é dada por + + − − que é representada pelo ponto $(0.40, 0.60)$ no gráfico[2]. Observe que a coordenada $(0.40, 0.60)$ tem relação direta com a porcentagem de positivos e negativos da folha à esquerda [6+,4-].

Esse particionamento divide o gráfico em 4 regiões, duas brancas e duas mais escuras. Como a árvore de decisão utiliza uma estratégia gulosa e não troca nós já construídos na árvore, os exemplos que estão na folha à direita não influenciam os exemplos da folha à esquerda. Durante a construção da árvore isso também ocorre no gráfico ROC. As duas partes brancas não influenciam uma na outra e cada parte branca é como se fosse um novo gráfico ROC no espaço de cobertura[3].

Na parte inferior a esquerda, parte branca, representa um gráfico ROC no espaço de cobertura. Assim, na divisão do nó a esquerda o ponto utilizado para particionar o gráfico é + − − − como ilustrado na Figura 4.17. Ao utilizar esse ponto para particionar o gráfico obtém-se a árvore ilustrada na

[2]Portanto 40 no gráfico ROC representa 0.40. A escala desses gráficos é 10^{-2}

[3]Lembrando que o espaço de cobertura considera números absolutos de exemplos positivos e negativos ao invés de *TVP* e *TFP*.

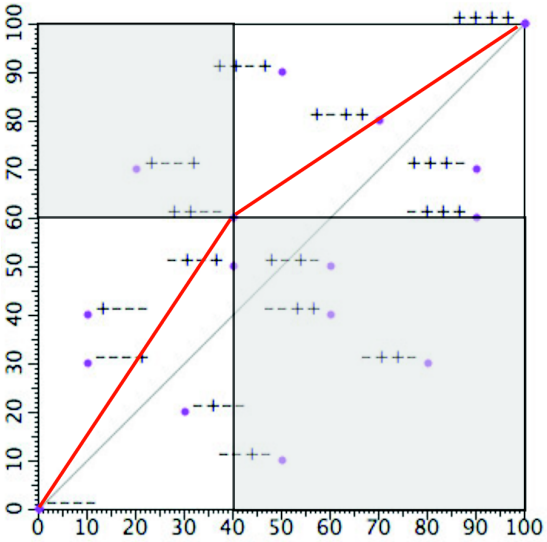


Figura 4.15: PROGROC: primeiro particionamento

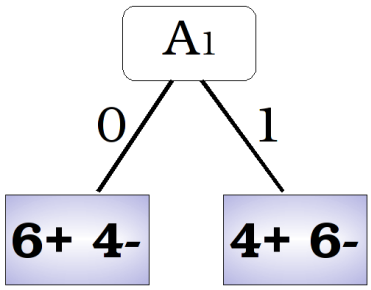


Figura 4.16: PROGROC: árvore de decisão com profundidade 1

Figura 4.18. Observe que a folha [4+,1-] é representada pelo segmento entre $(0,0)$ e $(0.10,0.40)$ e a folha [2+,3-] pelo segmento entre $(0.10,0.40)$ e $(0.40,0.60)$.

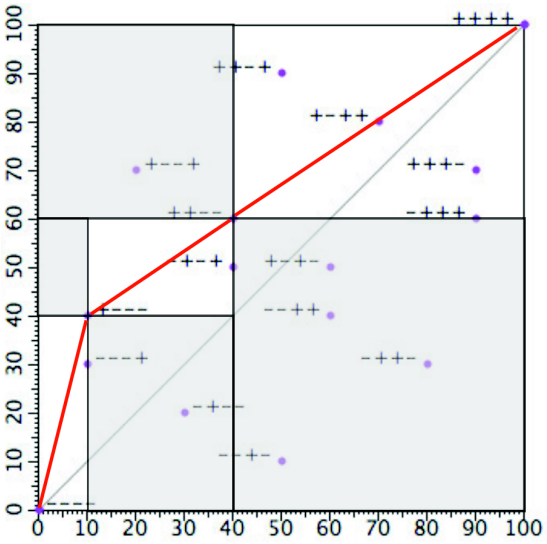


Figura 4.17: PROGROC: segundo particionamento

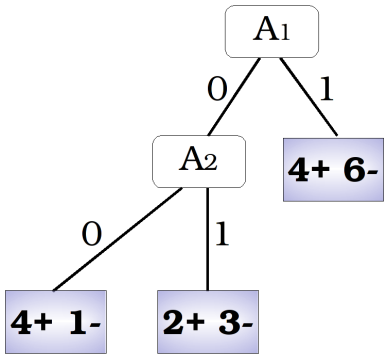


Figura 4.18: PROGROC: árvore de decisão com profundidade 2, ramo à esquerda

A divisão do nó [4+,6-] é representada na Figura 4.20 e esse particionamento pode ser visualizado no gráfico ROC como ilustrado na Figura 4.19. As folhas 1, 2, 3 e 4 aparecem na ordem dos segmentos do gráfico ROC, segmentos 1, 2, 3 e 4 respectivamente. Lembrando que no procedimento de construção de curvas ROC os empates são representados por linhas diagonais. Por exemplo, o segmento 3 que representa a folha 3 com [1+,5-], é a diagonal do retângulo desenhado com 1 passo para cima e 5 para esquerda.

A curva ROC mostrada no início desta seção é convexa e a curva obtida pelos passos ilustrados aqui é concava. Isso ocorre pois quando a curva ROC

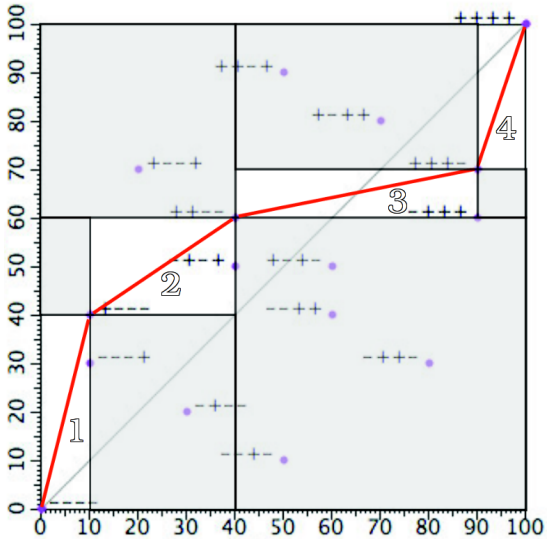


Figura 4.19: PROGROC: terceiro particionamento

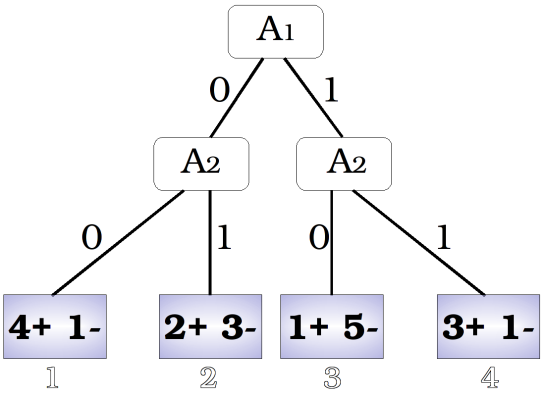


Figura 4.20: PROGROC: árvore de decisão com profundidade 2, ramo à direita

de uma árvore de decisão é construída, são utilizados os *scores* obtidos pela porcentagem de exemplos positivos nos nós folhas, *i.e.*, 80%, 40%, 17% e 75%. Quando esses *scores* são ordenados utilizando essas porcentagens, os segmentos também são ordenados como ilustrado na Figura 4.21.

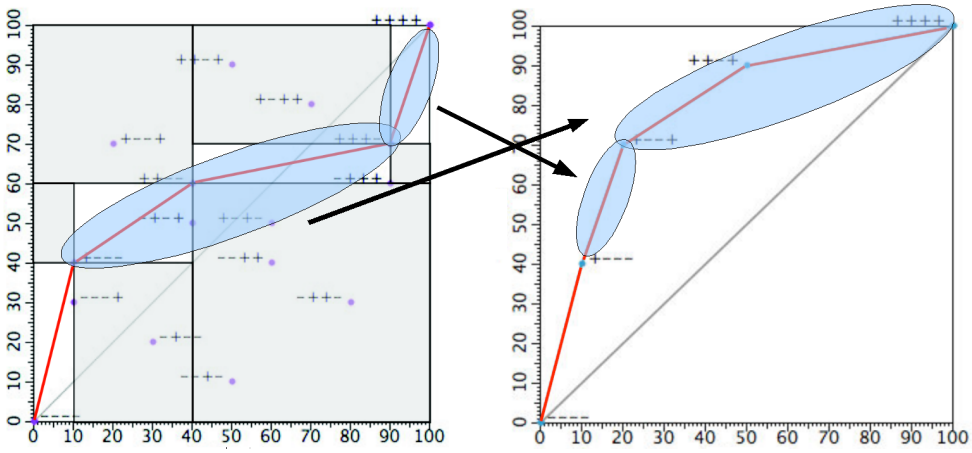


Figura 4.21: PROGROC: ordenando segmentos

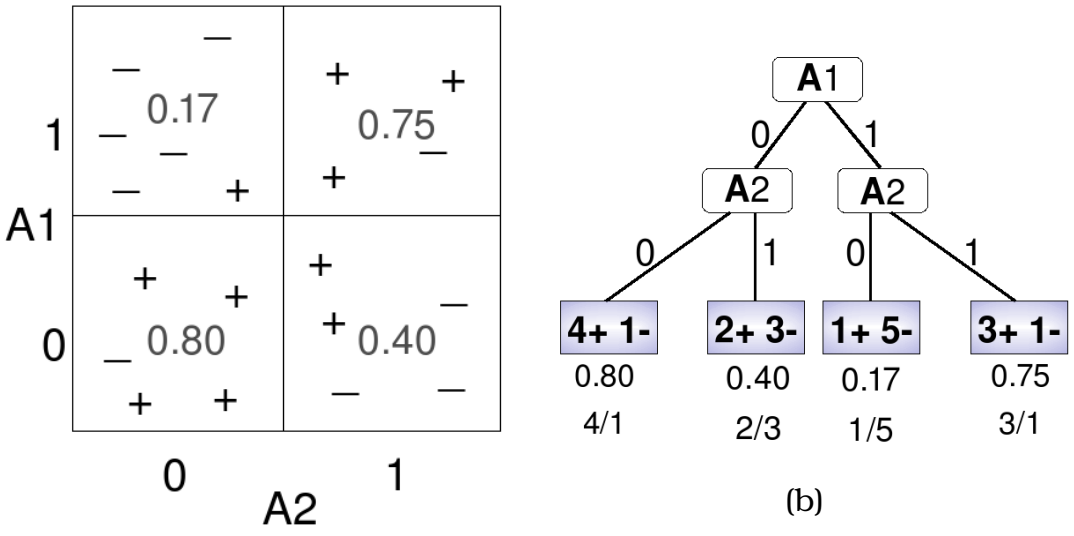
Esse processo de construção é bastante interessante pois permite visualizar os diferentes espaços de cobertura (retângulos brancos) durante a construção da árvore. Cada espaço de cobertura é como se fosse um novo gráfico ROC no qual a árvore procura obter pontos próximos ao céu ROC ou ao inferno ROC (ele inverte os rótulos quando cai no inferno ROC). Assim, pode-se avaliar o critério de particionamento em termos de análise ROC facilitando a compreensão desses critérios.

Um outro algoritmo bastante interessante de ser analisado dessa maneira é LEXRANK, descrito na Seção 3.3 na página 29. A construção do gráfico ROC é bastante semelhante, já que praticamente todos os passos mostrados para

árvore de decisão são válidos para LEXRANK. Existem apenas duas diferenças: na árvore lexicográfica, o atributo A_j somente ocorre na profundidade j da árvore. Coincidentemente o exemplo da árvore aqui apresentada como exemplo também satisfaz essa restrição; a segunda diferença é que LEXRANK não reordena os segmentos. Isso faz com que o gráfico ROC tenha, algumas vezes, concavidades na curva.

NAIVE BAYES também pode ser analisado como uma árvore construída de uma maneira um pouco diferente. NAIVE BAYES consiste da multiplicação das verosimilhanças com a probabilidade a priori. Uma árvore na qual um atributo A_j somente ocorre na profundidade j da árvore, pode representar todas as possíveis multiplicações que NAIVE BAYES pode fazer. Cada ramo da árvore representa a razão $\frac{P(A_j=v|+)}{P(A_j=v|-)}$ onde v é o valor do ramo. Desse modo, um conjunto com n_{at} atributos tem profundidade n_{at} com $2^{n_{at}}$ folhas. Embora essa representação seja impraticável, ela pode ser utilizada para facilitar a compreensão das relações de NAIVE BAYES com análise ROC.

Nas Figuras 4.23 e 4.22 são ilustradas, respectivamente, as estatísticas obtidas por NAIVE BAYES e árvore de decisão. Na Figura 4.23 (b) é ilustrada a representação de NAIVE BAYES como uma árvore, e a multiplicação dos ramos partindo do nó raiz até um nó folha representa $\frac{P(A_1|+)}{P(A_1|-)} \times \frac{P(A_2|+)}{P(A_2|-)} \times \frac{P(+)}{P(-)}$. Por exemplo, o filho a esquerda de $A_1 = 0$ representa a parte inferior da Figura 4.23 (a) que possui 6 exemplos positivos e 4 exemplos negativos, assim, a razão de verosimilhança é dada por $\frac{P(A_1=0|+)}{P(A_1=0|-)} = \frac{6}{4} \times \frac{10}{10}$. O filho a esquerda de $A_2 = 0$ representa a parte a direita da Figura 4.23 (a) que possui 5 exemplos positivos e 6 exemplos negativos, assim, a razão de verosimilhança é dada por $\frac{P(A_2=0|+)}{P(A_2=0|-)} = \frac{5}{6} \times \frac{10}{10}$.



(a) (b)

Figura 4.22: Estatísticas de árvore de decisão

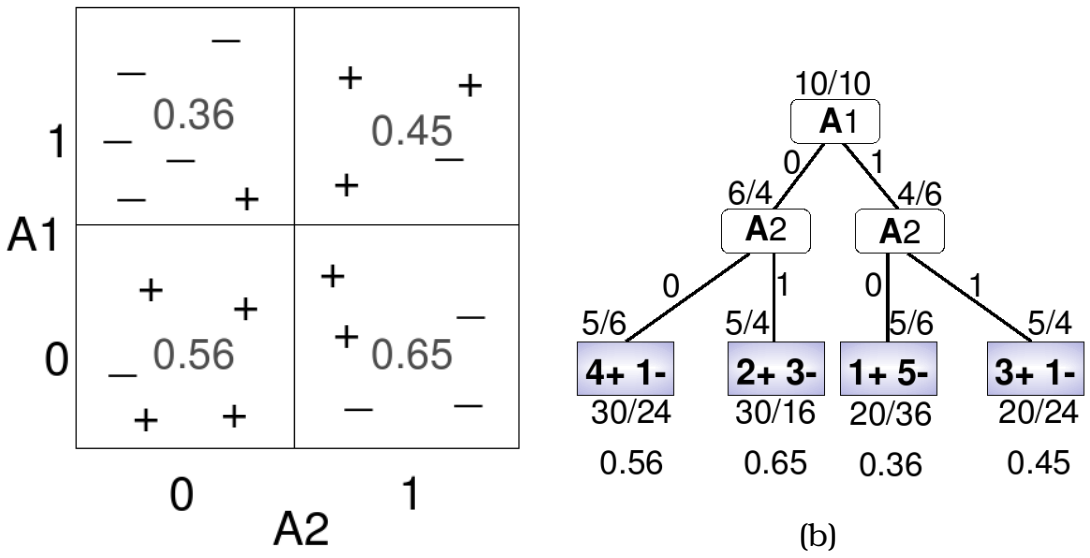


Figura 4.23: Estatísticas de NAIVE BAYES

Na Figura 4.23, NAIVE BAYES multiplica as razões de verosimilhança obtidas dos atributos A_1 e A_2 para cada um dos possíveis valores. Ainda na primeira folha a esquerda, multiplica-se a razão de verosimilhança dos atributos A_1 e A_2, $\frac{P(A_1=0|+)}{P(A_1=0|-)} \cdot \frac{P(A_2=0|+)}{P(A_2=0|-)} = \frac{6}{4} \cdot \frac{5}{6} = \frac{30}{24}$. Convertendo esse valor para probabilidade obtém-se 0.56. Para o mesmo nó, na árvore de decisão ilustrada na Figura 4.22, o *score* é obtido pela razão da probabilidade conjunta, $\frac{P(A_1=0,A_2=0|+)}{P(A_1=0,A_2=0|-)} = \frac{4}{1} = 4$. Convertendo razão de chances em probabilidades obtém-se $\frac{4}{5} = 0.80$. Veja que no conjunto de treinamento essa folha possui [4+,1-] e 0.80 do exemplo ser positivo é uma estatística bem melhor (mais semelhante as estatísticas obtidas do conjunto de treinamento) que 0.56 do exemplo ser positivo.

Desse modo, pode-se dizer que a árvore de decisão tem acesso a estatísticas melhores do que NAIVE BAYES. Os atributos possuem uma certa dependência e NAIVE BAYES tenta estimar $\frac{P(A_1=0,A_2=0|+)}{P(A_1=0,A_2=0|-)}$ assumindo que os atributos são independentes, *i.e.*,considera $\frac{P(A_1=0,A_2=0|+)}{P(A_1=0,A_2=0|-)} = \frac{P(A_1=0|+)}{P(A_1=0|-)} \cdot \frac{P(A_2=0|+)}{P(A_2=0|-)}$, enquanto a árvore de decisão consegue obter diretamente $\frac{P(A_1=0,A_2=0|+)}{P(A_1=0,A_2=0|-)}$. Caso os atributos fossem independentes, ambos os algoritmos obtém o mesmo resultado.

4.2.2 Screenshots e Explicação do Sistema

PROGROC foi implementando em Java utilizando as bibliotecas existentes no Weka (Witten and Frank, 2005), para importar os indutores já implementados no sistema. Várias bibliotecas disponíveis no *Prefuse* (Heer et al., 2005) foram utilizadas para plotar o gráfico ROC e fazer as animações. A interface do sistema é bastante simples, consistindo de seis itens, como ilustrado na Figura 4.24.

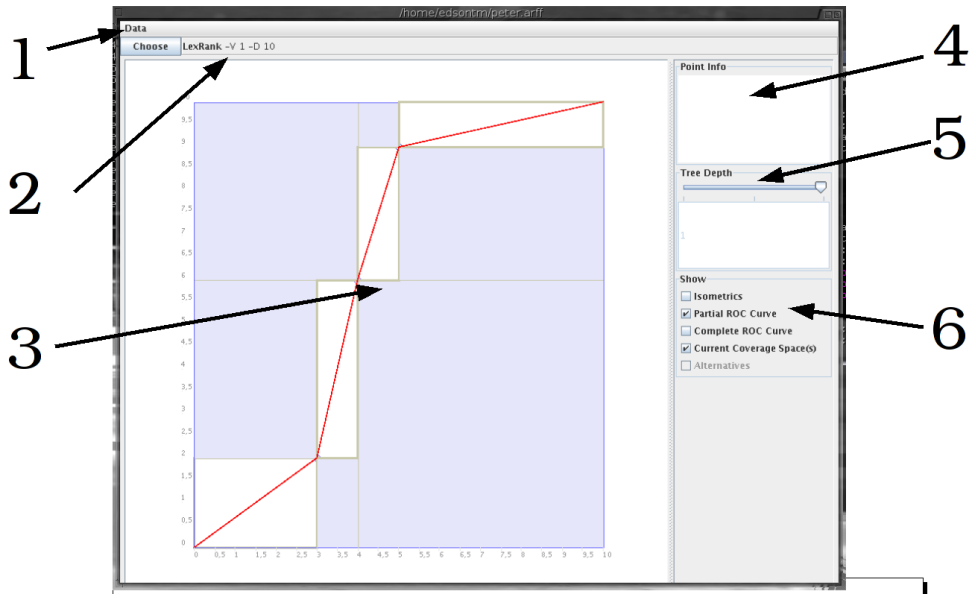


Figura 4.24: PROGROC: comandos e telas de informação

Cada um desses itens realizam as seguintes tarefas:

1. seleciona o conjunto de exemplos a ser utilizado;
2. seleciona o algoritmo a ser utilizado. Na versão atual o sistema consegue construir curvas para *J48* (árvore de decisão), NAIVE BAYES e LEXRANK;
3. principal tela de visualização. Nessa região é desenhado o gráfico ROC em um espaço de cobertura;
4. tela de visualização que mostra informações sobre os pontos que compõem a curva ROC;
5. define a profundidade atual da árvore;
6. opções de visualização.

Os itens 5 e 6 estão relacionados aos maiores pontos de iteração entre o usuário e o sistema. O item 5 possibilita ao usuário alterar a profundidade da árvore a ser plotada no item 3. O item 6 possui as seguintes cinco opções de visualização:

1. *Isometrics*: quando essa opção está ativa, a ferramenta mostra as curvas de iso-desempenho da medida razão de chances (*odds ratio*) utilizando como critério de escolha o melhor atributo para o particionamento dos exemplos;
2. *Partial ROC Curve*: quando essa opção está ativa, a ferramenta ilustra a curva ROC obtida até a profundidade definida no item 5;

3. *Complete ROC Curve*: mostra a curva ROC com os segmentos reordenados (para a árvore de decisão);

4. *Current Coverage Space*: mostra o espaço de cobertura ativo (retângulos brancos);

5. *Alternatives*: mostra as alternativas de partição (atributos) no espaço ROC.

Para ilustrar a utilização da ferramenta, considere a Figura 4.25 que ilustra a ferramenta com o conjunto de dados *harberman* da UCI utilizando *J48* como algoritmo indutor. Inicialmente as opções de visualização estão desativadas.

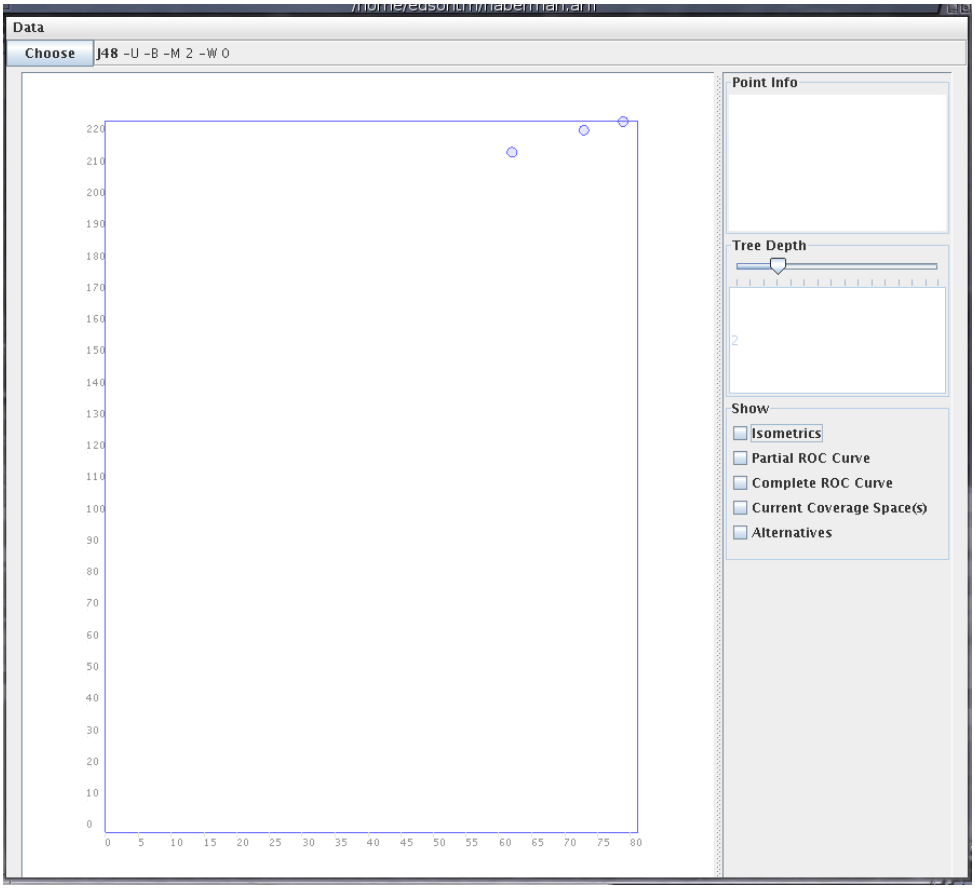


Figura 4.25: PROGROC: opções de visualização desativadas

Para ilustrar os espaços de cobertura a opção de visualização *Current Coverage Space* é ativada, como ilustrado na Figura 4.26. As regiões brancas, dentro do gráfico ROC, ilustram os espaços de cobertura válidos e a região mais escura os inválidos. As regiões inválidas podem ser ignoradas para a análise, já que as partições somente acontecem nas regiões brancas.

Dentro dos espaço de cobertura válidos (regiões brancas), ao ativar a opção *Alternatives*, os outros possíveis pontos de partição são apresentados na forma de quadrados, como ilustrado na Figura 4.27. Desse modo, pode-se

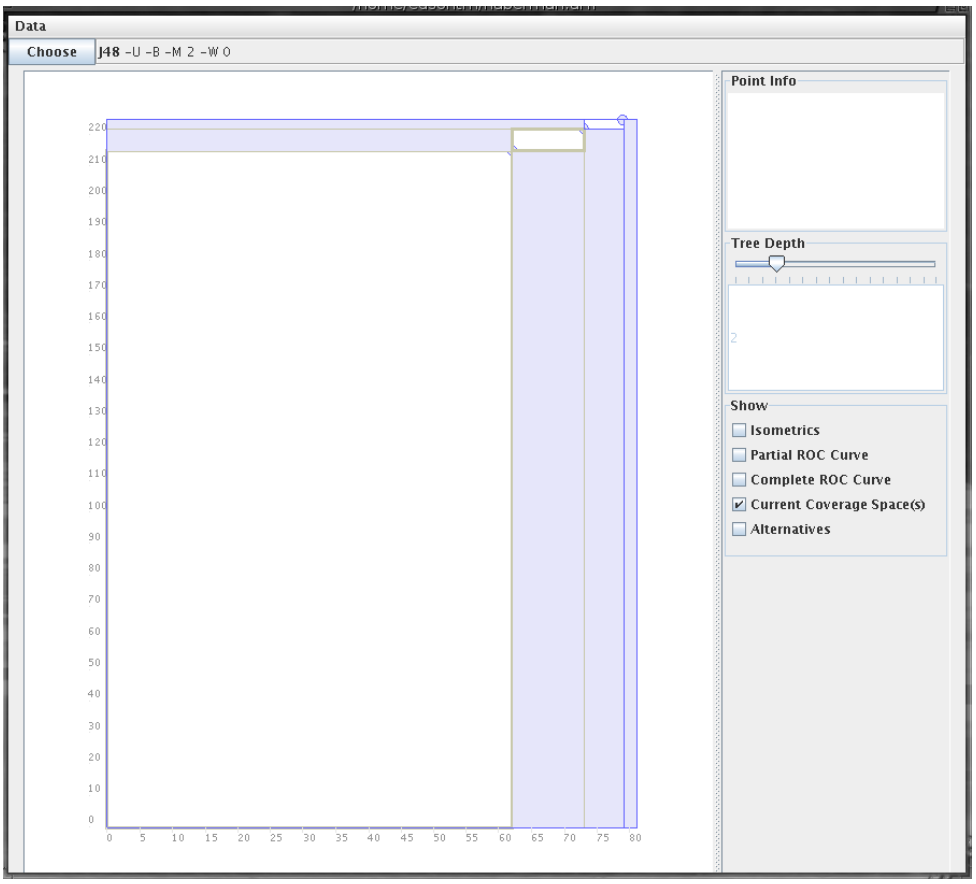


Figura 4.26: PROGROC: ilustração da opção *Current Coverage Space* ativada

verificar quais pontos a árvore de decisão pode utilizar para dividir os espaços de cobertura.

A opção *Alternatives* é ainda mais interessante quando utilizado em conjunto com a opção *Isometrics* (iso-desempenho). Com a opção de *isometrics*, é possível visualizar a curva de iso-desempenho para o ponto que é escolhido na próxima partição. Com essa opção ativada é possível visualizar se os outros pontos de partição são muito diferentes do ponto de partição escolhido. Na Figura 4.28 é ilustrada a curva de iso-desempenho juntamente com os pontos de particionamento alternativos (que não são escolhidos pelo algoritmo). Observe que no canto inferior esquerdo existem 3 quadrados. Apenas um deles toca o curva de iso-desempenho, esse ponto é o ponto escolhido para fazer o próximo particionamento do gráfico ROC.

Uma outra opção bastante interessante é o *Partial ROC Curve*. Essa opção ilustra a curva ROC caso o processo de construção parasse na profundidade atual. Veja que a curva ROC sempre corta os espaços de cobertura na sua diagonal.

A opção *Complete ROC Curve* é bastante semelhante a *Partial ROC Curve*. A opção *Complete* mostra a curva ROC quando a árvore está completamente construída e não a curva ROC da profundidade atual. Essa opção é interessante pois permite visualizar o quanto a curva ROC atual está longe da curva

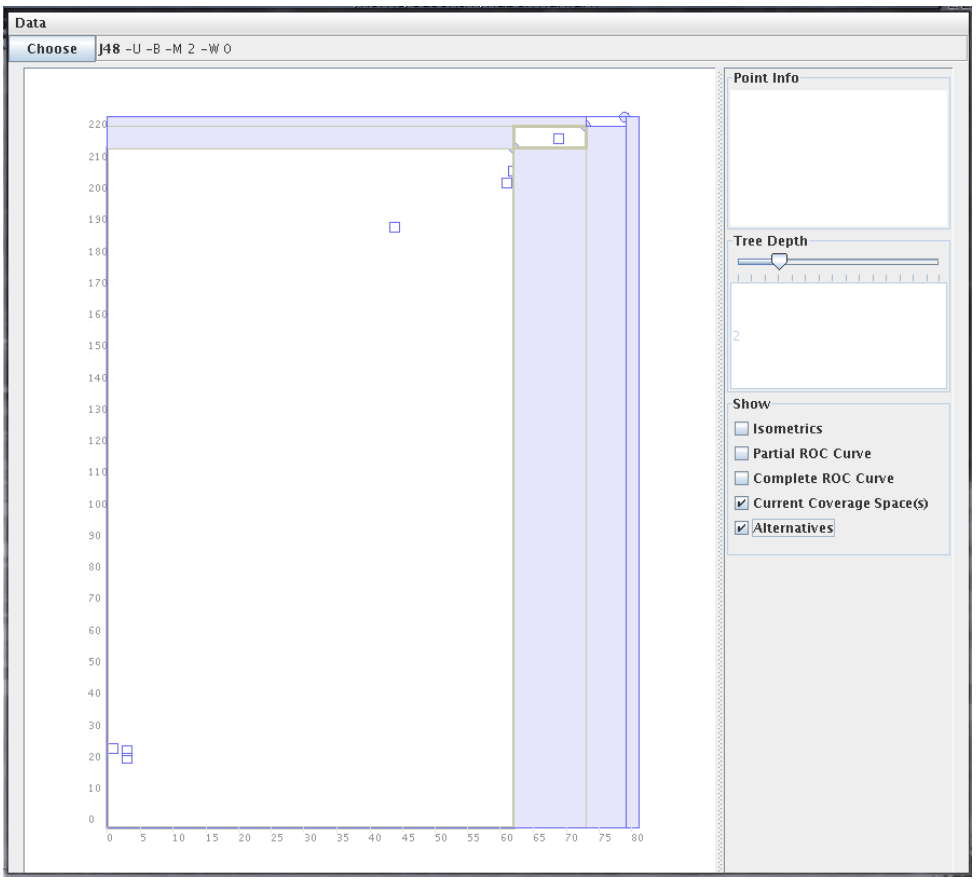


Figura 4.27: PROGROC: ilustração da opção *Alternatives* ativada

com a árvore inteira construída. Na Figura 4.30 é ilustrado o mesmo gráfico mas com a opção *Complete* ativada. Note que a curva é mais convexa que a curva do *Partial*, ilustrando, nesse caso, como a árvore mais completa pode ser melhor que a árvore parcial.

Como pode ser observado, cada profundidade da árvore possui diferentes espaços de cobertura (retângulos brancos). Existem espaços de cobertura maiores que outros, e muitas vezes os atributos começam a competir em regiões com espaço de cobertura pequenos. Quando isso acontece, significa que não haverá uma grande melhora de desempenho da árvore. Normalmente, é em grandes espaços de cobertura que espera-se uma grande melhoria, pois nessas regiões é possível colocar pontos cada vez mais perto do céu ROC. Observe a Figura 4.31 que ilustra essas características. Diversos pontos competem na parte superior do gráfico, e poucos na região inferior. Na região superior, a curva ROC completa é bem mais próxima do eixo superior e, desse modo, essa árvore de decisão, quando prediz exemplos com *score* baixo, é bem provável que ela raramente erre na classificação, enquanto que para os exemplos de *score* mais alto, provavelmente erre mais, pois tem uma quantia razoável de exemplos positivos e negativos misturados com *scores* mais altos.

Existem também visualizações para NAIVE BAYES e LEXRANK que são praticamente as mesmas que os mostrados para *J48*. A principal diferença é que,

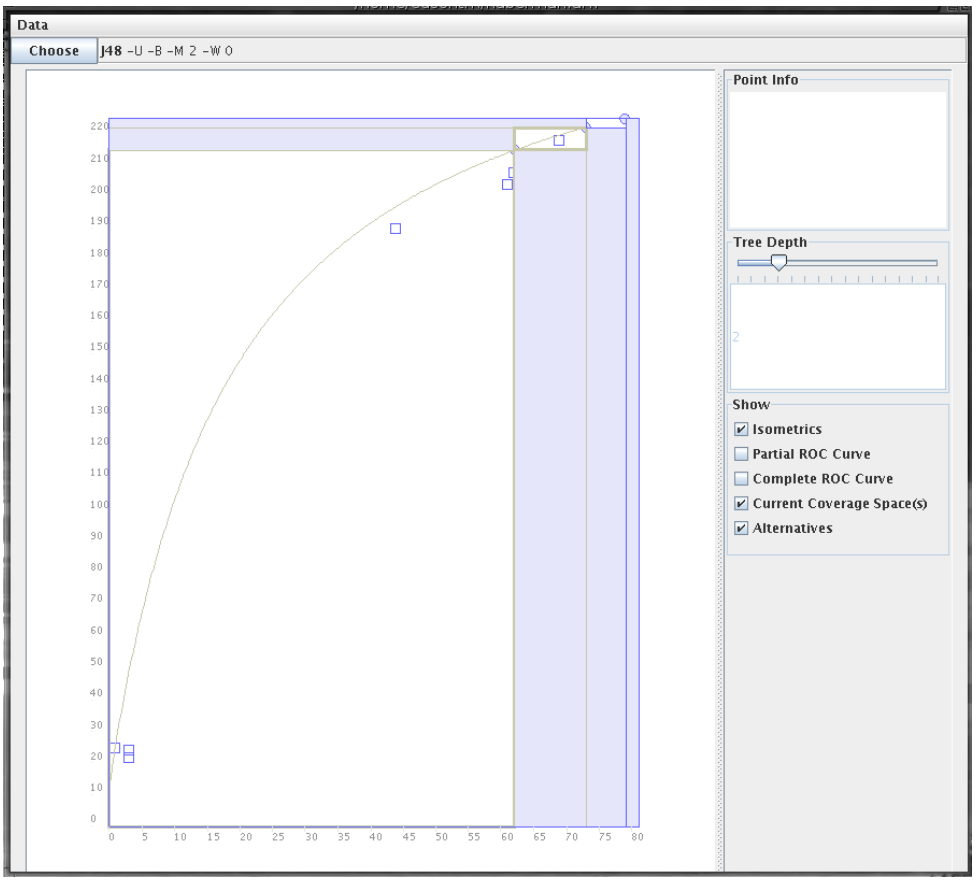


Figura 4.28: PROGROC: ilustração da opção *Isometrics* ativada

para esses dois algoritmos, não existe a opção de plotar os pontos de partição alternativos (opção *Alternatives*). Desse modo, essa opção é desabilitada para esses algoritmos.

Além da análise descrita, diversas outras análises podem ser feitas com a ferramenta PROGROC. Essa ferramenta traz um modo inédito de analisar os três algoritmos de aprendizado, *J48*, NAIVE BAYES e LEXRANK em termos de análise ROC. Isso permite fazer uma análise detalhada de cada passo da construção da árvore para esses algoritmos de aprendizado. Futuramente, pretendemos utilizar a ferramenta para alterar a árvore de decisão, tal que seja possível fazer uma escolha mais refinada dos pontos de partição na árvore. Dessa maneira, será possível construir gráficos ROC mais convexos e, portanto, árvores de decisão mais precisas.

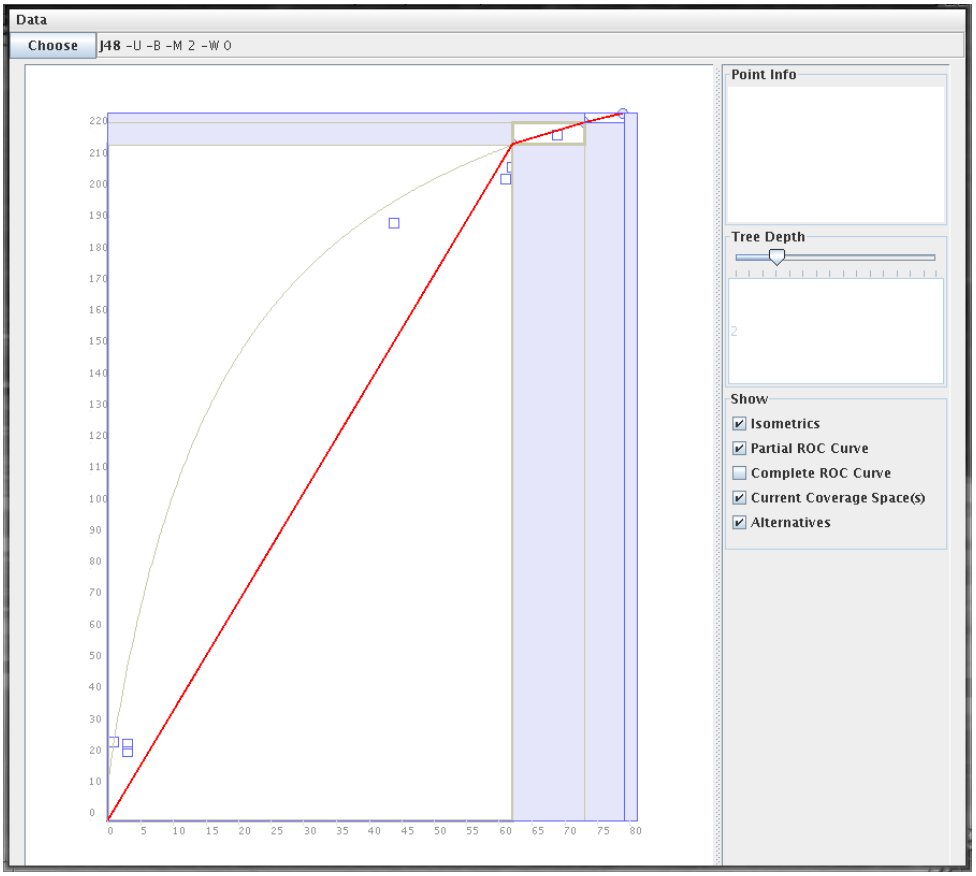


Figura 4.29: PROGROC: ilustração da opção *Partial ROC Curve* ativada

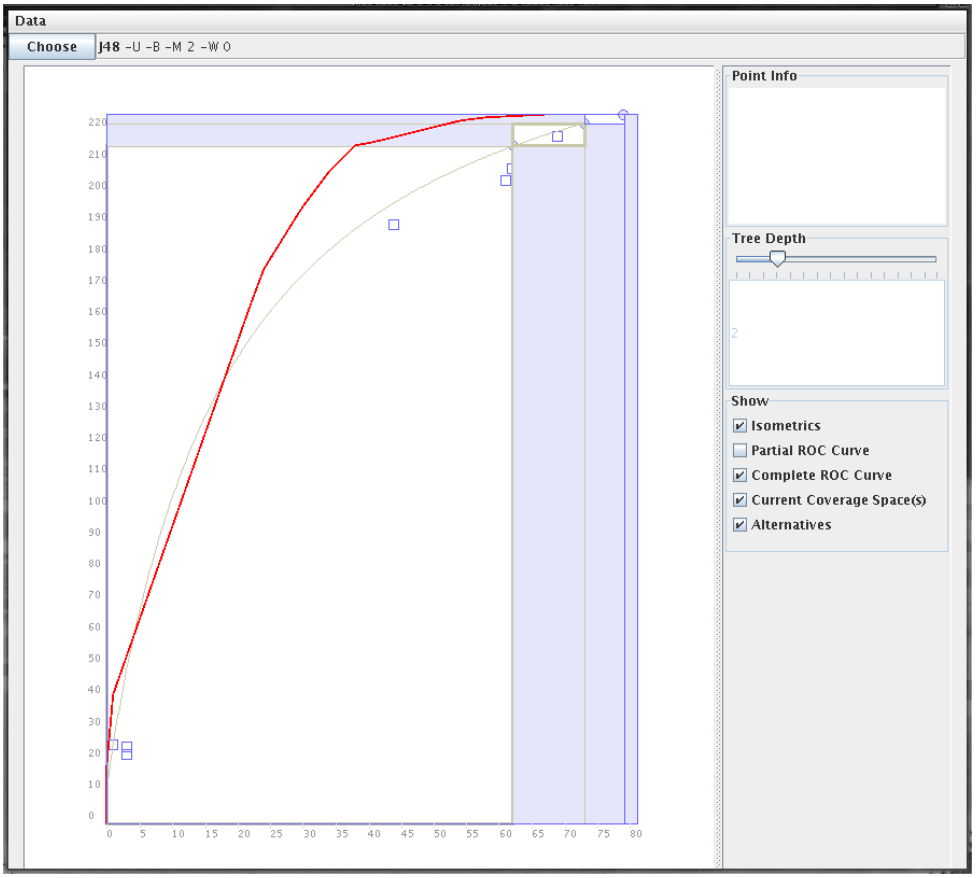


Figura 4.30: PROGROC: ilustração da opção *Complete* ativada

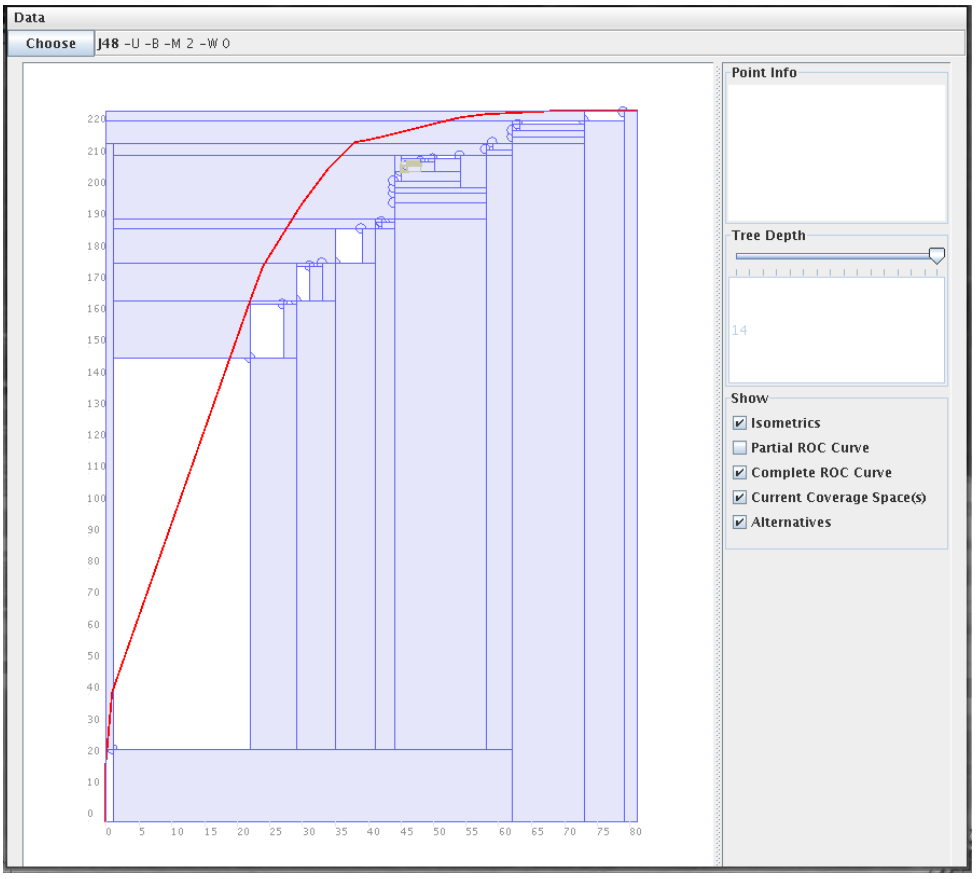


Figura 4.31: PROGROC: opções de visualização ativadas na última profundidade da árvore

4.3 Considerações Finais

Neste capítulo foram apresentados os conceitos de análise ROC necessários para a compreensão dos métodos e algoritmos propostos neste trabalho. Inicialmente foram apresentadas as medidas básicas que compõem os eixos do gráfico ROC, bem como uma breve explicação do que é um gráfico ROC. Foram também apresentados os dois modos mais comuns de utilização de gráficos ROC em aprendizado de máquina e três conceitos bastante utilizados na literatura, iso-desempenho, feixe convexo e área abaixo da curva. Finalmente, esses conceitos foram aplicados à árvores de decisão utilizando a ferramenta PROGROC, a qual possibilita uma gama que visualizações do gráfico para os três algoritmos de aprendizado.

Um dos conceitos apresentados neste capítulo, o feixe convexo de gráficos ROC, pode ser utilizado para calibrar probabilidades. Esse método de calibração é equivalente ao algoritmo PAV de regressão isotônica (Fawcett and Niculescu-Mizil, 2007). As relações entre esses métodos são discutidas no próximo capítulo.

CAPÍTULO

5

Calibração

Scores normalizados entre 0 e 1 e probabilidades podem ser facilmente confundidos. NAIVE BAYES é um exemplo bastante comum disso. Quando a assunção de independência entre os atributos do conjunto de exemplos se verifica, NAIVE BAYES estima probabilidades. Porém, quando essa assunção não se verifica, NAIVE BAYES é melhor qualificado como um classificador *score* do que um estimador de probabilidade. Uma maneira de fazer com que NAIVE BAYES estime probabilidades é utilizar algum método de calibração que mapeia *scores* na probabilidade a posteriori $P(y_v|\mathbf{x}_i)$.

Este capítulo apresenta conceitos introdutórios dos dois métodos de calibração mais conhecidos em aprendizado de máquina, bem como o relacionamento de um desses métodos com Curvas ROC. O capítulo está organizado da seguinte maneira: na Seção 5.1 são descritos brevemente os dois métodos de calibração, *calibração de Platt* e *regressão isotônica*; na Seção 5.2 são apresentadas as relações de regressão isotônica, as quais minimizam uma medida conhecida como *Brier score*, e feixe convexo de curvas ROC; na Seção 5.3 é apresentada uma decomposição do *Brier score* que possui interpretações geométrica em curvas ROC e na Seção 5.4 são apresentadas algumas considerações finais.

5.1 Métodos de Calibração Utilizados em Aprendizado de Máquina

Rankings e alguns estimadores de probabilidade são baseados em algoritmos de aprendizado de máquina supervisionado. Apesar de *ranking* e probabi-

lidades serem facilmente confundidos, eles possuem diferenças significativas. As saídas desses algoritmos levam a interpretações bastante diferentes e cada problema de mundo real deve ser analisado com cuidado para verificar a adequação do melhor algoritmo para cada problema.

As diferenças entre *rankeadores* e estimadores de probabilidades são mais nítidas quando apresentadas as suas definições. Como já definido no Capítulo 3, um *rankeador* é uma função $\hat{r} : X \times X \rightarrow \{>, =, <\}$ e os parâmetros dessa função requerem dois exemplos. Note que requerer dois exemplos é significativamente diferentemente de um estimador de probabilidade que requer apenas um exemplo como parâmetro. *Rankings* são utilizados para um conjunto de exemplos mas não para um exemplo isoladamente, pois a ordenação somente pode ser feita com pelo menos dois exemplos. Já no estimador de probabilidade, pode-se fornecer somente um exemplo para estimar sua probabilidade. O objetivo do *ranking* é ordenar e, por isso, uma vez obtida a ordem dos exemplos o *score* deixa de ser importante.

Os *rankings* podem ser obtidos de classificadores *score*, assim como os *estimadores de probabilidade*. Apenas relembrando, um classificador *score* é uma função $\hat{s} : X \rightarrow \mathbb{R}$ e um *estimador de probabilidade* é uma função $\hat{P} :\rightarrow [0,1]$, onde $\hat{P}$ estima a probabilidade a posteriori $P(+|\mathbf{x})$. A calibração é um modo de obter *estimadores de probabilidade* utilizando um classificador *score* que pode ser definida como uma função $\hat{m} : \mathbb{R} \rightarrow [0,1]$ que converte *scores* numéricos em $P(+|\mathbf{x})$. Pode-se unir um método de calibração com um classificador *score* fazendo a composta das funções $\hat{m}(\hat{s}) : X \rightarrow [0,1]$.

Os *scores* diferem de probabilidade, pois a probabilidade representa a freqüência relativa de ocorrência dos exemplos pertencerem a classe, enquanto o *score* é apenas um valor numérico que quanto maior "mais positivo" é o exemplo e quanto menor "mais negativo". Desse modo, quando apresentado um exemplo $\mathbf{x}_i$ com probabilidade 0.70 de ser positivo, pode-se inferir que quando apresentado uma amostra representativa de exemplos do domínio ao qual $\mathbf{x}_i$ pertence, 70% deles serão positivos e 30% serão negativos. Entretanto, quando apresentado um exemplo com *score* 0.70, esse valor pode não representar nada isoladamente. Em outras palavras, não se pode nem afirmar se existe uma chance maior desse exemplo ser positivo ou negativo. O *score* apenas começa a fazer sentido quando apresentado em conjunto com outros exemplos com seus respectivos *scores*. Somente nesses casos pode-se fazer uma relação com a classe e o *score*.

O intervalo e a escala de valores dos *scores*, quando desconhecidos, podem dificultar a interpretação dos resultados. Por exemplo, sabe-se que é verdade que um *score* de valor igual a 11 representa um exemplo positivo, quando apresentado outro *score* igual a 10, eles podem, a princípio parecerem pró-

ximos, pois *scores* podem variar de $[-\infty, +\infty]$. Entretanto, com somente dois exemplos é difícil conhecer o intervalo de valores e saber se um *score* igual a 10 também pode ser considerado positivo. Quando apresentado com um número maior de exemplos pode-se definir um intervalo, mas, mesmo assim, não é possível saber se a distância entre 10 e 11 é significativa, pois os valores podem estar em uma escala linear, logarítmica, exponencial ou alguma outra escala desconhecida.

Um outro problema com *scores* é saber qual é o limiar que separa exemplos positivos e negativos. Um *score* igual a 10 pode ser mais representativo para um exemplo negativo do que para um positivo. Como visto em análise ROC, o limiar ótimo pode ser bem diferente do que se imagina, principalmente em problemas com classes desbalanceadas. Desse modo, o uso do *score* deve ser feito com muito cuidado para que o usuário não seja levado a uma interpretação errada sobre o valor do *score*.

Uma alternativa para facilitar a análise do *score* é encontrar um mapeamento que possa transformá-lo em probabilidade. A primeira vantagem é saber onde no intervalo de valores estão as regiões que representam exemplos positivos e negativos, pois o limiar de decisão entre positivo e negativo deve ficar em 0.50. Com os *scores* calibrados, *i.e.*, mapeados em probabilidade, o intervalo dos valores e a escala dos valores deixam de ser um problema e possibilita o uso de toda a teoria estatística já desenvolvida, ampliando as possibilidades do que fazer com os exemplos preditos em termos probabilísticos. Desse modo, um calibrador de probabilidade pode ser de grande utilidade, principalmente em tarefas nas quais se deve interpretar o valor de confiança da classificação.

Uma pergunta pertinente para o desenvolvimento de um calibrador de probabilidade é saber o que é um bom calibrador. Uma boa maneira de responder essa pergunta é definir uma medida de erro. Medir o erro de um *estimador de probabilidade* consiste em medir a diferença entre $\hat{P}(+|\mathbf{x}_i)$ e $P(+|\mathbf{x}_i)$. Para simplificar a notação, considere $\hat{P}(\mathbf{x}_i) = \hat{P}(+|\mathbf{x}_i)$. Desafortunadamente, $P(+|\mathbf{x}_i)$ normalmente é desconhecido. Assim, o erro pode ser calculado utilizado diferença quadrática entre $\hat{P}(\mathbf{x}_i)$ e a classe verdadeira do exemplo (1 e 0 para positivo e negativo respectivamente) como definido na Equação 5.1. Essa medida de erro é conhecida como *Brier Score* (Brier, 1950).

$$bs(y, \hat{P}) = \frac{1}{n_{ex}} \sum_{i}^{n_{ex}} (\hat{P}(\mathbf{x}_i) - y_i)^2 \tag{5.1}$$

onde y_i é a classe verdadeira do exemplo $\mathbf{x}_i$. Uma outra medida utilizada é a

entropia cruzada (*Cross Entropy*), definida pela Equação 5.2:

$$entropiaCruzada(y,\hat{P}) = -\sum_{i}^{n_{ex}}(y_i log(\hat{P}(\mathbf{x}_i)) + (1-y_i) log(\hat{P}(\mathbf{x}_i))) \tag{5.2}$$

Ambas as medidas possuem seus valores mínimos quando $y_i = \hat{P}(\mathbf{x}_i)$ e são utilizadas pela calibração de Platt (Platt, 1999) e a regressão isotônica (Zadrozny and Elkan, 2002). O método de calibração de Platt ajusta os dados em uma sigmóide e utiliza entropia cruzada como critério para minimização, enquanto que a regressão isotônica ajusta os dados de tal modo que ela se torne isotônica (monotonicamente crescente) e utiliza o *Brier score* como critério de minimização. Esses dois métodos de calibração somente se aplicam a problemas de duas classes. Uma conversão para problemas multi-classe é apresentada por Zadrozny and Elkan (2002).

5.1.1 Calibração de Platt

John Platt, observou empiricamente que, para alguns conjuntos de dados, o mapeamento entre *scores* de SVM e a probabilidade empírica poderia ser feito com uma função sigmóide. Assim, no artigo (Platt, 1999) é desenvolvido o método de calibração paramétrico para SVM que ajusta *scores* $\hat{s}(x)$ em uma sigmóide com dois parâmetros A e B, como definido pela Equação 5.3.

$$\hat{P}(+|\hat{s}(\mathbf{x})) = \frac{1}{1+exp(-A\times\hat{s}(\mathbf{x})+B)} \tag{5.3}$$

Graficamente, essa equação pode ser representada como uma curva com formato de "S", como ilustrado na Figura 5.1. Quando o $A > 0$ é garantido que a sigmóide é monotônica e crescente. Esse parâmetro A controla a suavidade da curva. Quanto mais próximo de 0, mais a curva tende a ser uma reta na horizontal, e quanto maior mais a curva tende a se tornar uma reta na vertical. O parâmetro B desloca a curva para direita (valores positivos) e para esquerda (valores negativos).

Os parâmetros A e B são determinados por meio da estimação de máxima verosimilhança utilizando um conjunto de treinamento. Utilizando gradiente descendente para encontrar os valores de A e B da sigmóide tal que a entropia cruzada seja mínima tem-se:

$$argmin_{A,B}\{-\sum_{i}^{n_{ex}} y_i log(z_i) + (1-y_i) log(1-z_i)\} \tag{5.4}$$

onde

$$z_i = \frac{1}{1+exp(-A\hat{s}(\mathbf{x}_i)+B)} \tag{5.5}$$

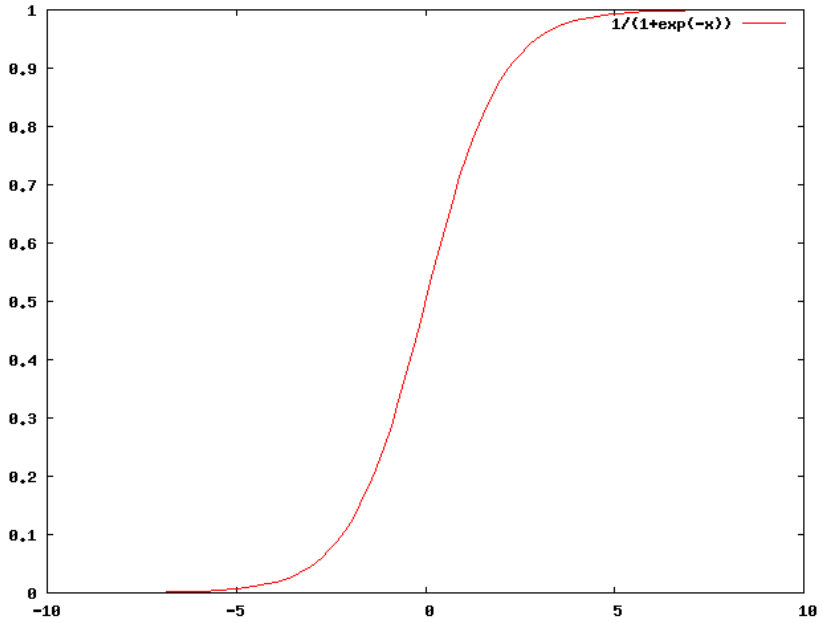


Figura 5.1: Exemplo de sigmóid

Como apontado por Niculescu-Mizil and Caruana (2005), a calibração de Platt é relativamente efetiva quando existem poucos exemplos para construir o mapeamento. Quando existe uma quantidade maior de exemplos, a regressão isotônica, descrita a seguir, é mais indicada.

5.1.2 Regressão Isotônica

A regressão isotônica (Zadrozny and Elkan, 2002) é uma técnica não paramétrica que pode ser utilizada para construir um mapeamento entre *scores* e probabilidades utilizando uma função isotônica (monotônica crescente) $\hat{m} : \mathbb{R} \rightarrow [0,1]$, que deve minimizar o *Brier score*, *i.e.*, o erro quadrático entre a classe estimada e a classe verdadeira do exemplo. Um dos algoritmos utilizados para encontrar $\hat{m}$ é conhecido como algoritmo PAV (*Pool Adjacent Violators*)(Ayer et al., 1955).

Antes de explicar o algoritmo, considere um conjunto de treinamento/validação em ordem crescente de *score* tal que os $\mathbf{x}_i$ satisfaçam $\hat{s}(\mathbf{x}_{i-1}) \leq \hat{s}(\mathbf{x}_i)$ e o atributo classe (A_{classe}) com valores 0 e 1 para negativo e positivo respectivamente. Esses exemplos são sempre adicionados a um conjunto denominado de $pool_k$ que contém n_k^+ exemplos positivos e n_k^- exemplos negativos. A proporção de exemplos positivos em um $pool_k$ é denotado por $p_k = \frac{n_k^+}{n_k^+ + n_k^-}$.

O algoritmo pode ser definido da seguinte maneira: percorrendo os exemplos sempre na ordem definida pelo *score*, cada exemplo é inserido em um $pool_k$ onde k é um identificador. Um $pool_k$ é criado quando são encontrados pares de exemplos consecutivos onde $A_{classe}(\mathbf{x}_{i-1}) < A_{classe}(\mathbf{x}_i)$, ou seja, zero seguido de um. Como *pool* é criado quando um zero é seguido de um (par crescente),

nas iterações seguintes quando os pares forem iguais ou decrescentes eles são adicionados dentro do mesmo *pool*. O objetivo é manter a proporção p_k sempre monotônica crescente ($p_{k-1} \leq p_k$). Portanto, quando $pool_{k-1}$ e $pool_k$ quebram a monotonicidade da seqüência *i.e.*, $p_{k-1} > p_k$, esses dois *pools* devem ser unidos. No final desse processo são atribuídas a probabilidade p_k aos exemplos no $pool_k$. Desse modo, é possível fazer um mapeamento dos *scores* com as probabilidades p_k definindo uma função isotônica $\hat{m}$.

Exemplo 5.1 *Na Tabela 5.1 são ilustrados 10 exemplos de validação/treinamento, 5 positivos (1) e 5 negativos (0). Os exemplos estão em ordem crescente de score (coluna score) com suas respectivas classes (coluna A_{classe}). Cada passo é representado por uma coluna que começa em m1 e termina em m10. Em cada coluna é apresentado o p_k de cada $pool_k$.*

Tabela 5.1: Calibrando exemplos com PAV: ilustração passo a passo

#	*score*	A_{classe}	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	$\hat{m}$
1	0.02	0	0	0	0	0	0	0	0	0	0	0	0.00
2	0.20	1		1	1/2	1/3	1/3	1/3	1/3	1/3	1/3	1/3	0.33
3	0.24	0			1/2	1/3	1/3	1/3	1/3	1/3	1/3	1/3	0.33
4	0.28	0				1/3	1/3	1/3	1/3	1/3	1/3	1/3	0.33
5	0.30	1					1	1/2	1/2	1/2	1/2	1/2	0.50
6	0.43	0						1/2	1/2	1/2	1/2	1/2	0.50
7	0.73	1							1	1	2/3	2/3	0.66
8	0.75	1								1	2/3	2/3	0.66
9	0.78	0									2/3	2/3	0.66
10	0.90	1										1	1.00

m1: *considera-se o exemplo que antecede o primeiro exemplo como zero, desse modo, a leitura da seqüencia seria, zero seguido de zero, o exemplo 1 é adicionado no $pool_0$ e $p_0 = 0/1$;*

m2: *zero seguido de um, o $pool_1$ é criado, o exemplo 2 é adicionado no $pool_1$ e $p_1 = 1/1$;*

m3: *um seguido de zero, o exemplo 3 é adicionado ao $pool_1$ e $p_1 = 1/2$;*

m4: *zero seguido de zero, o exemplo 4 é adicionado ao $pool_1$ e $p_1 = 1/3$;*

m5: *zero seguido de um, o $pool_2$ é criado, o exemplo 5 é adicionado ao $pool_2$ e $p_2 = 1/1$;*

m6: *um seguido de zero, o exemplo 6 é adicionado ao $pool_2$ e $p_2 = 1/2$;*

m7: *zero seguido de um, o $pool_3$ é criado, o exemplo 7 é adicionado ao $pool_3$ e $p_3 = 1/1$;*

m8: *um seguido de um, o exemplo 8 é adicionado ao $pool_3$ e $p_3 = 2/2$;*

m9: *um seguido de zero, o exemplo 9 é adicionado ao* $pool_3$ *e* $p_3 = 2/3$*;*

m10: *zero seguido de um, o* $pool_4$ *é criado, o exemplo 10 é adicionado ao* $pool_4$ *e* $p_4 = 1/1$.

Desse modo, o mapeamento entre scores e probabilidade pode ser feito definindo a função $\hat{m}$*:*

$$\hat{m}(score) = \begin{cases} 0.00 & \text{if} \quad score < 0.20 \\ 0.33 & \text{if} \quad 0.20 \leq score < 0.30 \\ 0.50 & \text{if} \quad 0.30 \leq score < 0.73 \\ 0.66 & \text{if} \quad 0.73 \leq score < 0.90 \\ 1.00 & \text{if} \quad score \geq 0.90 \end{cases} \tag{5.6}$$

Na literatura, o algoritmo é apresentado de uma maneira equivalente, mas dando enfoque aos casos nos quais um é seguido de zero. Nesses casos, os pares de exemplos são denominados de *adjacent violators* e a remoção desses *adjacent violators* garante que $\hat{m}$ seja isotônico.

5.2 Calibração Utilizando o Feixe Convexo de Curvas ROC

Nesta seção são apresentados os conceitos envolvidos na calibração utilizando o feixe convexo de curvas ROC e sua relação com o algoritmo PAV, seguido por uma breve avaliação experimental com bases de dados da UCI.

5.2.1 Relações de PAV com Análise ROC

O feixe convexo de curvas ROC, do mesmo modo que em PAV, empata exemplos. Assim, as seguintes perguntas podem ser bastante pertinentes: *O que significa o empate em curvas ROC? Quais os ganhos que o empate pode levar?*

Para responder essas perguntas, considere, como definido na Seção 3.1, um *segmento* como um conjunto de exemplos empatados no *ranking*. Esse *segmento*, em curvas ROC, pode ser visualizado como um segmento de reta que compõe uma curva ROC. Na curva ROC, um segmento k contém n_k^+ exemplos positivos e n_k^- exemplos negativos. O tamanho vertical e horizontal do *segmento* k é dado por $\frac{n_k^+}{Pos}$ e $\frac{n_k^-}{Neg}$ respectivamente. A inclinação (coeficiente angular) de cada *segmento* é dado pela divisão do tamanho vertical pelo tamanho horizontal do segmento, que é dado por $l_k = \frac{n_k^+}{n_k^-} c^{-1}$, onde $c = \frac{Pos}{Neg}$ é a chance a

priori (*prior odds*) da classe positiva. Do mesmo modo que em *pools*, a proporção de exemplos positivos em um *segmento* é denotado por $p_k = \frac{n_k^+}{n_k^+ + n_k^-}$. Com essa notação é possível definir p_k em termos de l_k, onde $p_k = \frac{l_k}{l_k + 1/c}$.

A relação entre feixe convexo e PAV torna-se mais evidente quando os pontos são plotados em um *espaço de cobertura* ao invés do espaço ROC, lembrando que no espaço ROC os eixos são compostos por *TVP* e *TFP* e no espaço de cobertura os eixos são compostos pelo número absoluto de exemplos positivos e negativos. O algoritmo PAV, assim como a inclinação em um espaço de cobertura, estima a probabilidade a posteriori $P(+|\mathbf{x}_i)$, enquanto que a probabilidade obtida da inclinação em um espaço ROC retorna a verosimilhança $P(\mathbf{x}_i|+)$. Desse modo, a probabilidade obtida da inclinação do espaço de cobertura é igual a probabilidade obtida de um *pool* no algoritmo PAV e, portanto, torna-se mais fácil visualizar a relação do feixo convexo e PAV. Uma outra vantagem de visualizar os pontos no espaço de cobertura é poder visualizar mais facilmente o número de exemplos que compõe cada segmento.

Regiões concavas no gráfico ROC são equivalentes a pontos que caem no inferno ROC. Apenas relembrando, os pontos que caem no inferno ROC são considerados ruins e, quando é possível fazer a inversão dos rótulos, um ponto que cai nessa região pode ir do inferno ROC para o céu ROC. Quando essa inversão não é feita, um classificador aleatório tem um desempenho melhor que o classificador no inferno ROC. Uma região concava na curva ROC é bastante similar ao inferno ROC. Os exemplos estão em ordem inversa, negativos na frente dos positivos e a inversão dos exemplos dessa região pode corrigir a ordem dos exemplos. Quando essa inversão não é feita, um *ranking* aleatório tem um desempenho melhor que o *ranking* invertido.

É interessante notar que o classificador aleatório é representado por uma linha diagonal que corta o gráfico ROC assim como os empates dos *rankings* onde nas regiões com concavidades também são representadas por linhas diagonais. Quando o feixe convexo desenha uma linha diagonal em uma região concava da curva, ela empata exemplos e torna regiões *rankings* no inferno ROC em *rankings* aleatórios.

Considere a curva ROC ilustrada na Figura 5.2 que possui uma concavidade. As concavidades representam inversão no *ranking* de exemplos positivos e negativos. Na primeira concavidade, dois exemplos negativos precedem dois exemplos positivos. Empatar esses 4 exemplos, *i.e.*, tornar a classificação aleatória entre esses 4 exemplos, melhora o *ranking*, pois entre um *ranking* invertido e um *ranking* aleatório, o *ranking* aleatório erra menos.

Uma outra característica do feixe convexo de curvas ROC é que, partindo do (0,0), os segmentos possuem inclinação l_k sempre monotônico decrescente. Quando vista de traz para frente, partindo do ponto (Neg, Pos), l_k é sempre

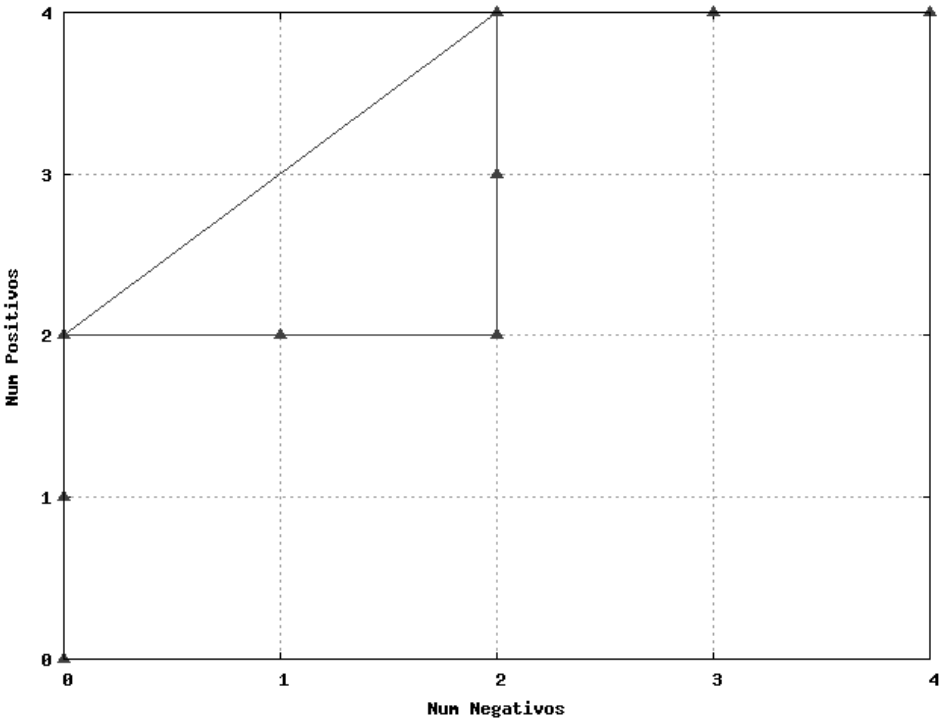


Figura 5.2: Explicação de concavidades

monotônico crescente (isotônico).

Observe a curva ROC da Figura 5.3 que representa os exemplos da Tabela 5.1. Cada segmento do feixe convexo da curva ROC é equivalente a um *pool* do algoritmo PAV. Observe a curva partindo do ponto (Neg, Pos) para (0,0). Seguindo o algoritmo PAV, um novo *pool* é criado toda vez que aparece um zero seguido de um. Na curva ROC isso significa sempre começo de uma concavidade e, portanto, começo de um *segmento* é começo de um *pool*. Em termos de análise ROC, é interessante empatar os exemplos após a ocorrência de zero seguido de um, pois ali começa uma concavidade.

Entretanto, criar *pools* quando aparece um zero seguido de um não garante que p_k seja monotônica crescente. Observe a Figura 5.4 que possui duas concavidades. No algoritmo PAV é necessário uma correção que garanta que os $pool_k$ sejam sempre isotônicos. Isso é feito unindo $pool_{k-1}$ e $pool_k$ quando $p_{k-1} > p_k$. Quando essa união ocorre, a função torna-se isotônica para essa região, como ilustrado na Figura 5.5.

5.2.2 Resultados Experimentais

Os resultados experimentais têm por objetivo mostrar a qualidade da probabilidade estimada dos algoritmos NAIVE BAYES e *J48*. Eles têm também por objetivo verificar o quão efetivo é calibrar as probabilidades de NAIVE BAYES e LEXRANK. Neste trabalho houve um maior interesse em testar a regressão isotônica do que a calibração de Platt, devido a suas relações com as curvas

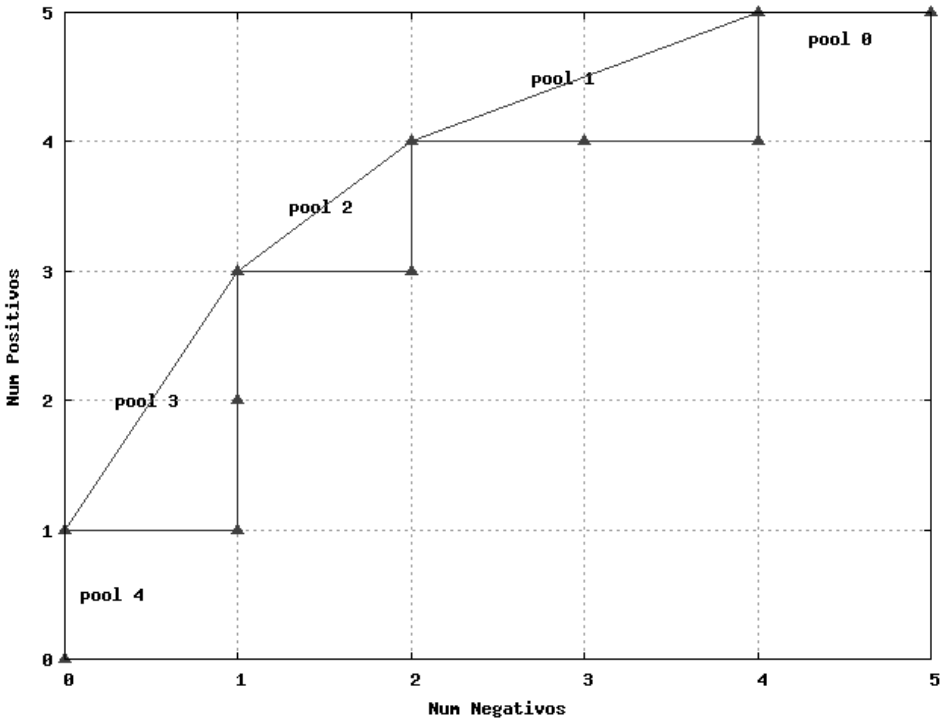


Figura 5.3: Exemplo da Tabela 5.1 representado em uma curva ROC

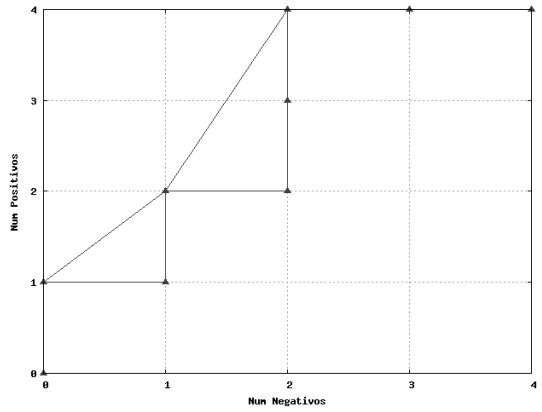


Figura 5.4: Sem união de *pools*

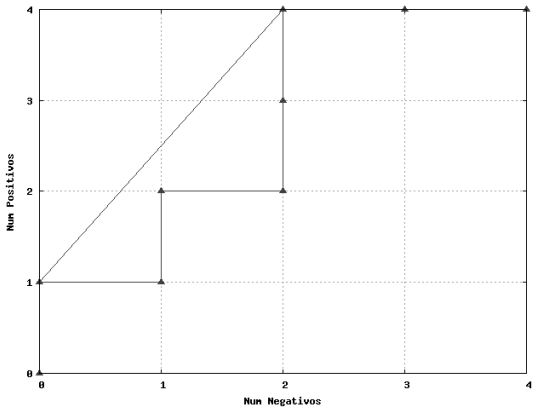


Figura 5.5: Com união de *pools*

ROC.

As versões calibradas de NAIVE BAYES e LEXRANK são denominadas de CALINB e LEXPROB respectivamente. Para esses experimentos foram utilizados os mesmos 27 conjuntos pré-processados na Seção 3.4 obtidos da UCI. Todos os algoritmos foram avaliados em termos do *Brier score* (Equação 5.1 na página 75). Nesses experimentos também foram feitos testes de significância, um paramétrico e um não paramétrico, o teste-t pareado e o teste de Friedman, respectivamente (Demšar, 2006).

Na Tabela 3.5 são mostrados os resultados do teste-t pareado. Para cada linha são ilustrados os ganhos/empates/perdas do algoritmo na linha *versus* o algoritmo na coluna. LEXRANK apenas ordena exemplos e não são obtidos *scores* e muito menos probabilidades. Com os exemplos ordenados por LEX-

Tabela 5.2: Média do *Brier score* de validação cruzada com seus respectivos desvios padrões. Aplicando teste pareado com 5% significância e LEXPROB como algoritmo base (+ significa que LEXPROB é melhor). A última linha soma as ganhos/empates/perdas de LEXPROB em comparação com os outros algoritmos

set	LexRank		LexProb	NB		CaliNB		J48	
1	0.152 0.007	+	0.032 0.008	0.080 0.007	+	0.072 0.006	+	0.009 0.002	-
2	0.244 0.007	+	0.186 0.005	0.205 0.007	+	0.180 0.005	-	0.214 0.006	+
3	0.067 0.003	+	0.034 0.004	0.032 0.013		0.028 0.009	-	0.052 0.003	+
4	0.152 0.002	+	0.077 0.003	0.051 0.002	-	0.033 0.003	-	0.010 0.001	-
5	0.125 0.005	+	0.103 0.006	0.144 0.013	+	0.134 0.008	+	0.129 0.004	+
6	0.234 0.006	+	0.181 0.006	0.201 0.010	+	0.167 0.005	-	0.209 0.011	+
7	0.125 0.007	+	0.012 0.005	0.002 0.002	-	0.008 0.008		0.013 0.007	
8	0.194 0.006	+	0.166 0.005	0.197 0.008	+	0.163 0.005	-	0.221 0.018	+
9	0.166 0.004	+	0.070 0.001	0.052 0.013	-	0.049 0.006	-	0.069 0.013	
10	0.208 0.008	+	0.182 0.013	0.212 0.025	+	0.192 0.015	+	0.207 0.012	+
11	0.177 0.018	+	0.119 0.026	0.122 0.010		0.094 0.013	-	0.106 0.028	-
12	0.163 0.007	+	0.146 0.007	0.129 0.015	-	0.124 0.009	-	0.176 0.008	+
13	0.129 0.003	+	0.082 0.020	0.094 0.010	+	0.083 0.006		0.115 0.008	+
14	0.094 0.004	+	0.044 0.003	0.106 0.007	+	0.103 0.006	+	0.008 0.002	-
15	0.294 0.006	+	0.019 0.001	0.021 0.001	+	0.010 0.001	-	0.007 0.000	-
16	0.291 0.023	+	0.247 0.019	0.268 0.027	+	0.253 0.026	+	0.259 0.017	+
17	0.161 0.008		0.165 0.009	0.150 0.018		0.128 0.011	-	0.145 0.010	-
18	0.248 0.005	+	0.011 0.001	0.021 0.006	+	0.014 0.002	+	0.010 0.002	
19	0.140 0.009	+	0.127 0.007	0.081 0.012	-	0.126 0.019		0.143 0.015	+
20	0.224 0.004	+	0.177 0.003	0.176 0.004	-	0.182 0.005	+	0.032 0.007	-
21	0.333 0.012	+	0.223 0.005	0.232 0.004	+	0.213 0.004	-	0.022 0.003	-
22	0.099 0.006	+	0.011 0.005	0.046 0.007	+	0.014 0.005	+	0.011 0.004	
23	0.308 0.027	+	0.216 0.014	0.243 0.012	+	0.210 0.003		0.220 0.024	
24	0.212 0.018		0.211 0.018	0.234 0.017	+	0.185 0.021	-	0.262 0.012	+
25	0.251 0.037	+	0.116 0.018	0.172 0.025	+	0.112 0.010		0.142 0.011	+
26	0.156 0.004	+	0.027 0.002	0.024 0.001	-	0.022 0.001	-	0.028 0.002	+
27	0.231 0.002	+	0.175 0.005	0.192 0.004	+	0.174 0.003		0.056 0.004	-
w/t/l	25/2/0			17/3/7		8/6/13		13/5/9	

RANK, foram atribuídos valores arbitrários entre [0,1]. Assim, quando avaliado como estimador de probabilidade LEXRANK representa o pior caso.

Tabela 5.3: Resultados comparados com o teste t pareado de significância utilizando *Brier score*

	LexRank	LexProb	NB	CaliNB	J48
LexRank	-	0/2/25	3/3/21	2/0/25	4/2/21
LexProb	25/2/0	-	17/3/7	8/6/13	13/5/9
NB	21/3/3	7/3/17	-	3/2/22	10/5/12
CaliNB	25/0/2	13/6/8	22/2/3	-	13/5/9
J48	21/2/4	9/5/13	12/5/10	9/5/13	-

Com pode ser observado, LEXPROB ganha de NAIVE BAYES em 17 conjuntos de dados, empata em 3 e perde em 7, indicando um resultado um pouco melhor para LEXPROB. Com relação à versão calibrada de NAIVE BAYES, CALINB, LEXPROB ganha em 8, empata 7 e perde em 13, sendo comparável a CALINB, mas perdendo mais que ganhando. Quando comparado com *J48*, LEXPROB também parece ter desempenho bastante similar. CALINB comparado a *J48*

também parece ter resultados bastante comparáveis.

Aplicando o teste de Friedman, obteve-se a estatística F de 17.20. O valor crítico da estatística F com 4 e 104 graus de liberdade e 5% de significância é de 2.46. De acordo com o teste de Friedman, a hipótese nula pode ser rejeitada (existe diferença significativa entre os algoritmos) e, assim a pode-se prosseguir com a análise *post-hoc*.

De acordo com o teste *post-hoc* de Nemenyi, médias dos rankings com diferenças maiores que 1.17 podem ser consideradas significativamente diferentes. Na Figura 5.6 é ilustrado um gráfico de diferença crítica. No eixo principal está representado a média dos rankings dos algoritmos testados (quanto menor, melhor o algoritmo). Quando dois algoritmos possuem diferenças entre as suas médias inferiores a 1.17, esses dois algoritmos são ligados por uma linha grossa. Quando isso ocorre, *i.e.*, dois algoritmos são ligados, não existe diferença significativa entre esses algoritmos.

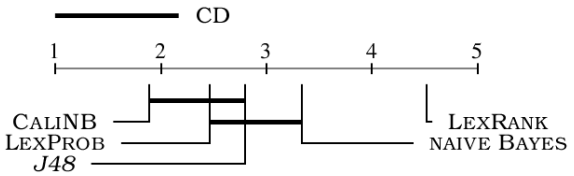


Figura 5.6: Gráfico de diferença crítica comparando CALINB, LEXPROB, *J48*, NAIVE BAYES e LEXRANK em termos de *Brier score*

A análise experimental revela que a calibração para NAIVE BAYES, CALINB, mostra-se significativamente melhor que NAIVE BAYES, mostrando que é bastante válido calibrar probabilidades para NAIVE BAYES. LEXRANK, como atribui *scores* arbitrários e foi considerado nos experimentos apenas para ilustrar o pior caso, a melhora significativa já era esperada. *J48* está situado no meio do gráfico de diferença crítica, e mesmo não podando a árvore e utilizando Laplace, tem um desempenho mediano entre os algoritmos comparados. Uma explicação mais detalhada sobre as probabilidades de árvores de decisão pode ser consultada em (Provost and Domingos, 2003).

5.3 Decomposição de Brier Score

A calibração utilizando regressão isotônica tem como critério de minimização o *Brier score* (Brier, 1950). Pode-se fazer uma interessante relação entre o *Brier score* e curvas ROC. Essa relação é encontrada quando o *Brier score* é decomposto em dois termos: *erro por calibração* e *erro por refinamento* (Flach and Matsubara, 2007b). Uma decomposição similar é conhecida em teoria de previsão (Cohen and Goldszmidt, 2004), entretanto, essa decomposição uti-

liza probabilidades estimadas obtidas pela proporção de exemplos positivos em uma discretização o que a torna uma aproximação. A decomposição proposta aqui utiliza os segmentos obtidos dos *rankings* e, desse modo, ela é mais precisa, além de ser mostrada a relação dessa decomposição com curvas ROC. A seguir são apresentados os conceitos sobre essa decomposição seguido pela visualização do efeito da calibração.

5.3.1 Apresentação da Decomposição de Brier Score

O *Brier score* utiliza a probabilidade estimada mas ignora o *ranking* (ele não requer a ordenação dos exemplos). Em contrapartida, curvas ROC utilizam *rankings* mas ignoram a probabilidade estimada. O *Brier score* e a AUC medem grandezas diferentes. Para ilustrar isso, considere o seguinte exemplo.

Exemplo 5.2 *Considere os exemplos apresentados na Tabela 5.4 que apresenta o score (score), o score calibrado (score cal) e a classe de cada exemplo (classe).*

Tabela 5.4: Calibração: conjunto de exemplos

Exemplo	*score*	*score* cal	classe
A	0.9	1.0	p
B	0.8	1.0	p
C	0.7	0.5	n
D	0.7	0.5	p
E	0.5	0.5	n
F	0.5	0.5	p
G	0.3	0.0	n
H	0.2	0.0	n

A curva ROC desse conjunto de exemplo é ilustrada na Figura 5.7. A curva ROC do score e a curva ROC do score calibrado é exatamente a mesma. Em outras palavras, a calibração é uma transformação monotônica e, desse modo, o ranking do score e o ranking do score calibrado é o mesmo. No entanto, o Brier score do score é diferente do Brier score do score calibrado.

Teorema 5.1 *Dado um curva ROC produzida por um rankeador em um conjunto de teste, seja n_k^+ e n_k^- o número de exemplos positivos e negativos no k-ésimo segmento da curva ROC, $n_k = n_k^+ + n_k^-$, $p_k = \frac{n_k^+}{n_k}$, e $\hat{P}_k$ a probabilidade predita nesse segmento. O Brier score é igual a*

$$BS = \frac{1}{n_{ex}} \sum_k n_k (\hat{P}_k - p_k)^2 + \frac{1}{n_{ex}} \sum_k n_k p_k (1 - p_k)$$

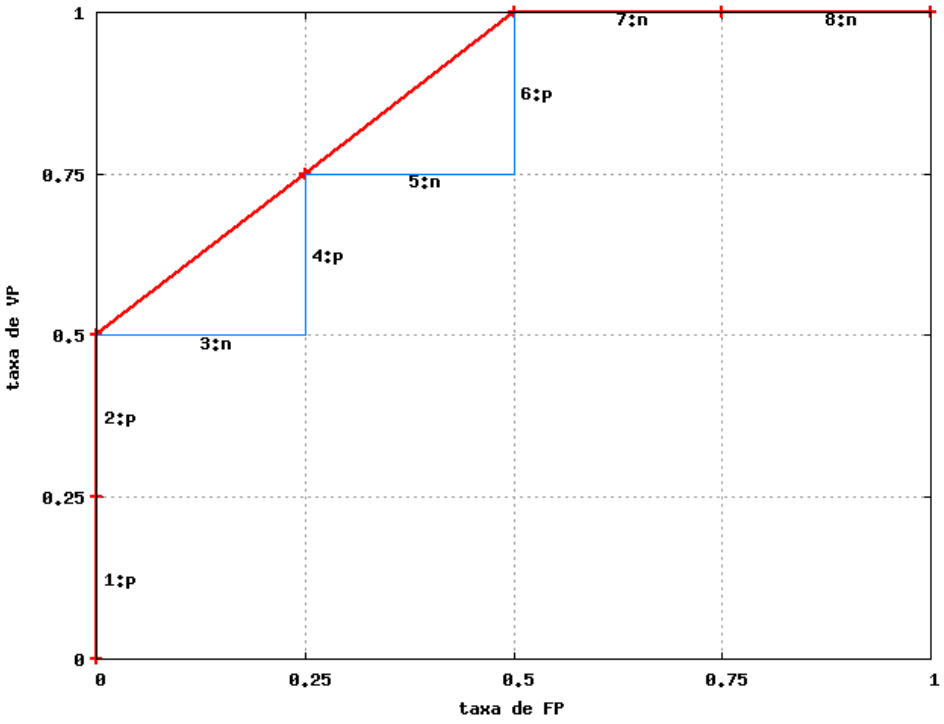


Figura 5.7: Calibração: exemplos de curva ROC

Prova:

$$
\begin{aligned}
BS &= \frac{1}{n_{ex}}\sum_{i}^{n_{ex}}(\hat{P}(\mathbf{x}_i)-y_i)^2 \\
&= \frac{1}{n_{ex}}\sum_{k}[n_k^+(\hat{P}_k-1)^2+n_k^-\hat{P}_k^2] \\
&= \frac{1}{n_{ex}}\sum_{k}[n_k\hat{P}_k^2-2n_k^+\hat{P}_k+n_k^+] \\
&= \frac{1}{n_{ex}}\sum_{k}[n_k\hat{P}_k^2-2n_k^+\hat{P}_k+n_k^+\frac{n_k^+}{n_k}-n_k^+\frac{n_k^+}{n_k}+n_k^+] \\
&= \frac{1}{n_{ex}}\sum_{k}[n_k(\hat{P}_k-\frac{n_k^+}{n_k})^2+n_k^+(1-\frac{n_k^+}{n_k})] \\
&= \frac{1}{n_{ex}}\sum_{k}n_k[(\hat{P}_k-\frac{n_k^+}{n_k})^2+\frac{n_k^+}{n_k}(1-\frac{n_k^+}{n_k})] \\
&= \frac{1}{n_{ex}}\sum_{k}n_k[(\hat{P}_k-p_k)^2+p_k(1-p_k)] \\
&= \frac{1}{n_{ex}}\sum_{k}n_k(\hat{P}_k-p_k)^2+\frac{1}{n_{ex}}\sum_{k}n_kp_k(1-p_k)
\end{aligned}
$$

O primeiro termo é denominado de *erro por calibração* e o segundo de *erro por refinamento*. O erro por calibração mede a diferença entre a probabilidade empírica p_k e a probabilidade predita $\hat{P}_k$ *que não é necessariamente 0 ou 1*, lembrando que a probabilidade empírica é obtida com a classe verdadeira dos exemplos de treinamento e a probabilidade predita é a probabilidade retornada pelo método de calibração ou estimador de probabilidade.

A curva ROC perfeita é composta por dois segmentos, um vertical que vai de (0,0) a (0,1) e outro horizontal que vai de (0,1) a (1,1) e ambos os segmentos possuem exemplos de apenas uma classe. Nessa curva não existem diagonais, somente retas verticais e horizontais. O segundo termo, o *erro por refinamento*, aumenta quando aparecem diagonais na curva ROC. Desse modo, a termo de refinamento é 0 se e somente se os segmentos são totalmente verticais ou horizontais ($p_k = 0$ ou $p_k = 1$), e possui valor máximo (0.25) se $p_k = 0.5$, *i.e.*, o segmento não representa nenhuma ordem de preferência pois possui a mesma proporção de exemplos positivos e negativos. É interessante verificar que esse termo não depende das probabilidades preditas $\hat{P}_k$ e, quando o erro por refinamento é alto, não importa quanto a probabilidade predita é boa, o *Brier score* vai ser alto.

Essa decomposição da lugar ao seguinte teorema.

Teorema 5.2 *O erro de calibração é igual a zero somente se a curva ROC é convexa.*

Prova: Se uma curva ROC não é convexa, então existem dois segmentos não adjacentes tal que $\hat{P}_k > \hat{P}_j$, mas que $l_k < l_j$. Como $l_k < l_j$ então $p_k < p_j$. Para poder obter $\hat{P}_k > \hat{P}_j$ só é possível se pelo menos um dos termos do erro de calibração $(\hat{P}_k - p_k)^2$ ou $(\hat{P}_j - p_j)^2$ forem diferentes de zero.

O inverso não é verdade, *i.e.*, o feixe convexo não é condição suficiente para o erro de calibração ser igual a zero (veja o Exemplo 5.2). Os *scores* podem estar na ordem correta, mas isso não garante calibração com o Brier Score igual a zero.

Teorema 5.3 *Seja $\hat{P}$ um estimador de probabilidade com uma curva ROC convexa mas com erro por calibração diferente de zero. Seja $\hat{P}'$ um score calibrado obtido calculando p_i de $\hat{P}$. Então $\hat{P}'$ tem o mesmo AUC mas um Brier score menor.*

Prova: $\hat{P}'$ pode ser mais *grosseiro* que $\hat{P}$ pois se existirem segmentos adjacentes com $p_k = p_j$ eles serão unidos. Isso não afeta o formato da curva ROC, nem o AUC. Como $\hat{P}'$ tem *erro por calibração* igual a zero, seu *Brier score* é menor.

Exemplo 5.3 *Utilizando o Exemplo 5.2 e considerando $\hat{P}$ igual ao score, p igual ao score calibrado, e as classes positiva e negativa iguais a 1 e 0 respectivamente, obtém-se a Tabela 5.5. Como já mencionado, um segmento é um conjunto de exemplos empatados no ranking. Desse modo, os exemplos C e D pertencem a um segmento e os exemplos E e F pertencem a outro segmento. O erro por calibração é dado por $\frac{0.26}{8}$ e o erro por refinamento por $\frac{1}{8}$. Quando o score é calibrado, o erro por calibração torna-se igual a zero e o Brier score, nesse caso, é composto apenas pelo erro por refinamento $\frac{1}{8}$.*

Tabela 5.5: Calibração: decomposição do *Brier score*

Exemplo	$\hat{P}$	p	classe	$(\hat{P}(\mathbf{x}_i) - y_i)^2$	$n_k(\hat{P}_k - p_k)^2$	$n_k p_k(1-p_k)$
A	0.9	1.0	1	0.01	0.01	0.00
B	0.8	1.0	1	0.04	0.04	0.00
C	0.7	0.5	0	0.49	0.08	0.50
D	0.7	0.5	1	0.09		
E	0.5	0.5	0	0.25	0.00	0.50
F	0.5	0.5	1	0.25		
G	0.3	0.0	0	0.09	0.09	0.00
H	0.2	0.0	0	0.04	0.04	0.00
soma				1.26	0.26	1

Diversos indutores não garantem curvas ROC convexas no conjunto de treinamento. Para esses modelos, a calibração utilizando feixe convexo junta segmentos adjacentes que estão em ordem incorreta. Essa junção incrementa o erro por refinamento, mas em compensação diminui o erro por calibração.

Scores com poucos empates, como os fornecidos por NAIVE BAYES e LEXRANK, tendem a ter poucas diagonais, e por isso o erro por refinamento é próximo de zero. Algoritmos que tendem a empatar exemplos, como árvores de decisão, possuem erro por refinamento maior. A visualização desses conceitos em conjuntos de dados da UCI é ilustrada a seguir.

5.3.2 Visualização do Erro por Calibração e Erro por Refinamento

Algoritmos nas quais os *score* possuem poucos empates, como NAIVE BAYES, normalmente se beneficiam quando utilizado com regressão isotônica. Como já mostrado na Seção 5.2.2, NAIVE BAYES pode ter melhora de desempenho significativo quando calibrado com regressão isotônica. A calibração utilizando esse método empata exemplos e, com isso, aumenta o erro por refinamento (*refinement loss*) e diminui o erro por calibração (*calibration loss*).

Na Figura 5.8 são ilustrados os erros de calibração e refinamento obtida pelos algoritmos CALINB, *J48*, *J48* sem poda usando Laplace (*J48UL* - Unprunned with Laplace), LEXPROB, LEXRANK e NAIVE BAYES utilizando validação cruzada de 10 partições. Cada ponto representa o resultado obtido por uma partição. Pode-se notar que como NAIVE BAYES e LEXRANK praticamente não tem empates em seus *scores*, o erro por refinamento é praticamente zero. As versões calibradas de NAIVE BAYES e LEXRANK (CALINB e LEXPROB) aumentam o erro por refinamento mas diminuem o erro por calibração.

Uma outra observação interessante pode ser feita utilizando árvores de decisão. Quando induzidas sem poda usando Laplace, a árvore normalmente possui mais nós folhas e, desse modo, ocorrem menos empates (erro de refinamento diminui). Na Figura 5.9 são ilustrados os erros de calibração e refinamento. Pode-se observar que na árvore de decisão induzida sem poda usando Laplace, o erro de refinamento é um pouco menor e o erro de calibra-

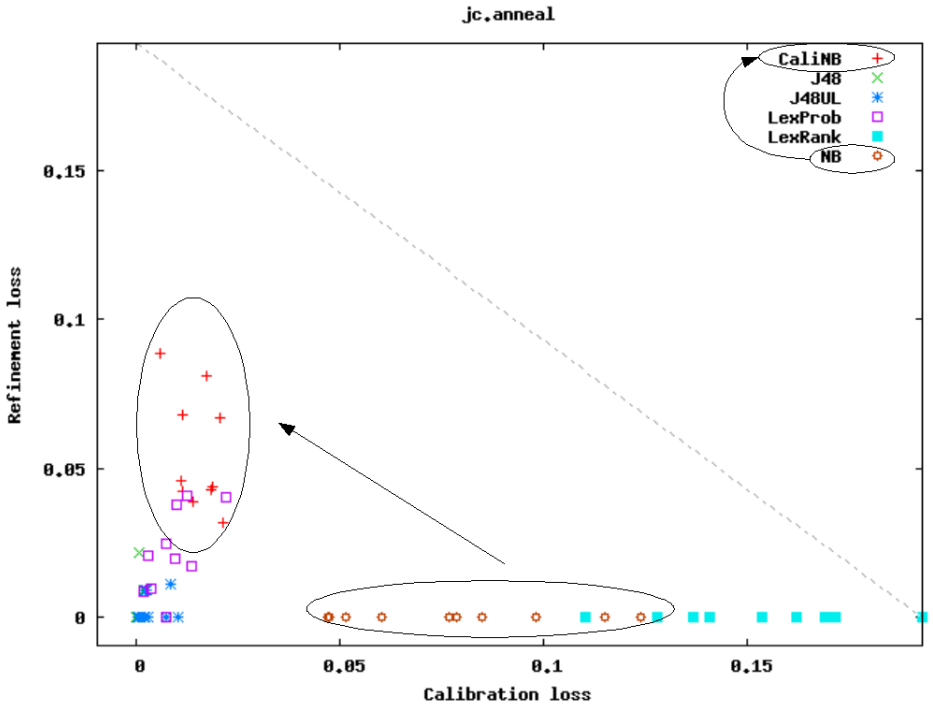


Figura 5.8: Ilustração do erro de calibração e refinamento no conjunto anneal da UCI

ção é um pouco maior do que com poda usando Laplace.

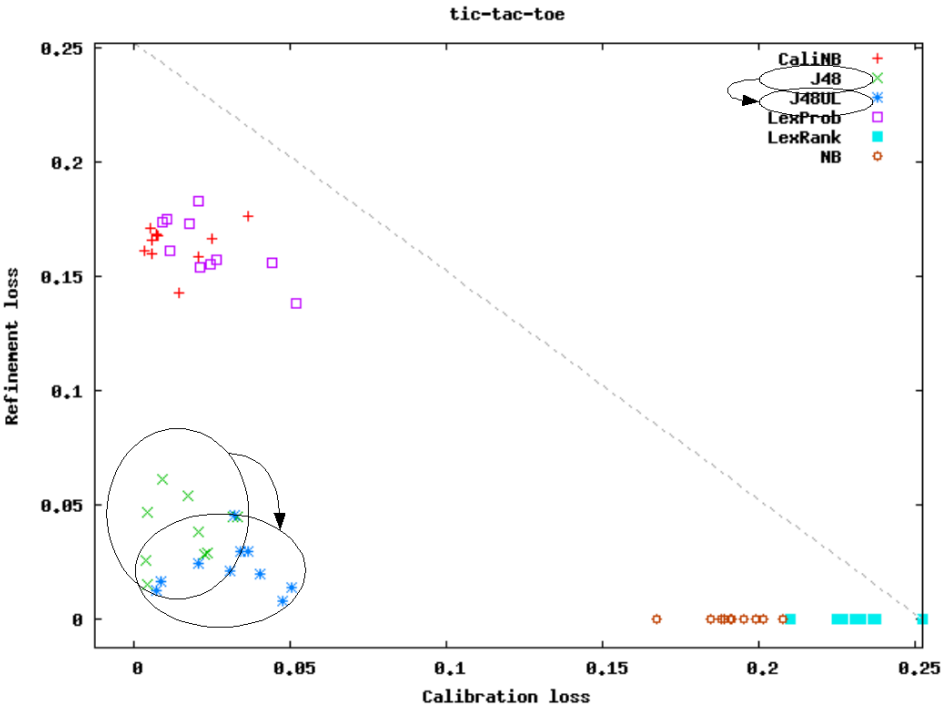


Figura 5.9: Ilustração do erro de calibração e refinamento no conjunto tic-tac-toe da UCI

5.4 Considerações Finais

Neste capítulo foram apresentados as duas técnicas de calibração de probabilidades mais utilizadas em aprendizado de máquina, a calibração de Platt e a regressão isotônica. A calibração de Platt é um método paramétrico e assume que a calibração pode ser feita através de uma sigmóide. A regressão isotônica procura um mapeamento isotônico entre *score* e probabilidade e possui uma relação interessante com o feixe convexo de curvas ROC. Essa relação possibilita interpretar a calibração em termos de análise ROC. A regressão isotônica utiliza como critério de minimização o *Brier score*, o qual pode ser decomposto em dois termos que possuem interpretações em curvas ROC. A contribuição apresentada neste capítulo está na relação do algoritmo PAV e o feixe convexo, bem como na relação entre o *Brier score* e as curvas ROC.

CAPÍTULO

6

Aplicando Análise ROC e Rankings em CO-TRAINING

Este capítulo apresenta a aplicação de alguns conceitos de análise ROC e *rankings* em um algoritmo de aprendizado semi-supervisionado multi-descrição denominado CO-TRAINING, proposto por Blum and Mitchell (1998), bastante conhecido pela comunidade. CO-TRAINING é um algoritmo interessante no contexto deste trabalho, pois a cada iteração do algoritmo são selecionados os melhores exemplos de cada classe para incrementar o conjunto de exemplos rotulados. Nessa seleção de exemplos, pode-se utilizar análise ROC (proporção de classes) e *rankings*. A análise ROC pode ser utilizada para visualizar os efeitos do parâmetro que define a proporção de exemplos de cada classe a ser inserida no conjunto de exemplos rotulados. Para a seleção dos melhores exemplos é necessário ordenar os exemplos, essa ordenação é equivalente a um *ranking*.

O capítulo esta organizado da seguinte maneira: na Seção 6.1 são apresentados os conceitos de aprendizado semi-supervisionado multi-descrição e o algoritmo CO-TRAINING: na Seção 6.2 é apresentado um estudo empírico de como a proporção das classes afeta CO-TRAINING; na Seção 6.3 é proposto uma variação de CO-TRAINING para o problema de imputação em atributos faltantes utilizando agregadores de *rankings* e, finalmente, na Seção 6.4 são apresentadas algumas considerações finais.

6.1 CO-TRAINING

Algoritmos de aprendizado semi-supervisionado, como visto no Capítulo 2, fazem uso de exemplos rotulados e não-rotulados para a indução de classificadores. Entre os diversos algoritmos de aprendizado semi-supervisionado (Zhu, 2005) existem os algoritmos multi-descrição (*multi-view*). A multi-descrição é caracterizada por exemplos que podem ser descritos de mais de uma maneira. Por exemplo, um instrumento musical pode ser descrito tanto pelo som quanto pelo formato geométrico; uma bebida pode ser descrita pela cor, gosto, textura ou cheiro; uma pessoa pode ser descrita tanto pelas características físicas como pelas características emocionais. Além desses casos, existem muitos outros problemas que admitem múltiplas descrições. Para fazer uso dessa característica, em (Blum and Mitchell, 1998) é proposto CO-TRAINING, considerado por muitos pesquisadores um dos primeiros algoritmos multi-descrição para aprendizado semi-supervisionado.

CO-TRAINING requer exemplos provenientes de domínios nos quais pode ser fornecida, pelo menos, duas descrições de cada exemplo. Para cada uma das descrições dos dados é induzido um classificador. Utilizando um conjunto de exemplos não-rotulado, rotula-se em cada iteração do algoritmo alguns desses exemplos e os exemplos rotulados diferentemente pelos classificadores são ignorados para a escolha dos melhores exemplos. Os exemplos para os quais os classificadores concordam com a rotulação são ordenados de acordo com o *score* obtido, *i.e.*, é construído um *ranking* da rotulação desses exemplos. Os exemplos no topo do *ranking* são inseridos no conjunto de exemplos rotulados de treinamento e, assim, iterativamente CO-TRAINING incrementa o conjunto de exemplos rotulados.

Apesar do procedimento parecer uma combinação de classificadores, CO-TRAINING faz assunções sobre a distribuição dos dados, o que torna o algoritmo similar aos algoritmos que fazem propagação de rótulos em grafos.

6.1.1 Criação das Duas Descrições

Para criar as duas descrições usadas por CO-TRAINING, é necessário particionar o conjunto de atributos $\mathbf{A}$ que descreve os exemplos em dois subconjuntos, $\mathbf{A}_{D_1}$ e $\mathbf{A}_{D_2}$, que constituem as duas descrições, tal que, $\mathbf{A} = \mathbf{A}_{D_1} \cup \mathbf{A}_{D_2}$ e $\mathbf{A}_{D_1} \cap \mathbf{A}_{D_2} = \emptyset$. Por simplicidade, é considerado que $\mathbf{A}_{D_1} = \{A_1, A_2, ..., A_j\}$ e $\mathbf{A}_{D_2} = \{A_{j+1}, A_{j+2}, ..., A_{n_{at}}\}$. Exemplos com valor de Y igual a "?" representam exemplos não rotulados.

Além da separação em duas descrições, o conjunto de exemplos $E = \{\mathbf{x}_1, ..., \mathbf{x}_{n_{ex}}\}$ deve ser separado em dois subconjuntos L e U, também conhecidos como conjunto de exemplos rotulados (*Labeled*) e não-rotulados (*Unlabeled*) respec-

tivamente. O subconjunto $L \subset E$ que contém exemplos que possuem o atributo classe conhecido é, por sua vez, dividido em dois conjuntos disjuntos L_{D_1} e L_{D_2} ($L = L_{D_1} \cup L_{D_2}$ e $L_{D_1} \cap L_{D_2} = \emptyset$). Analogamente, o subconjunto $U \subset E$ que contém os exemplos nos quais o atributo classe é desconhecido é também dividido em dois conjuntos U_{D_1} e U_{D_2}, tal que $U = U_{D_1} \cup U_{D_2}$ e $U_{D_1} \cap U_{D_2} = \emptyset$. Os dados de entrada do algoritmo de CO-TRAINING consistem desses quatro conjuntos de exemplos L_{D_1}, L_{D_2}, U_{D_1} e U_{D_2}.

6.1.2 Descrição do Algoritmo

No Algoritmo 2 são descritos os principais passos de CO-TRAINING, comentados a seguir.

Algoritmo 2: CO-TRAINING

Input: $L_{D_1}, L_{D_2}, U_{D_1}, U_{D_2}, k$
Output: L_{D_1}, L_{D_2}
Construir U'_{D_1} e U'_{D_2} como descrito anteriormente;
$U_{D_1} = U_{D_1} - U'_{D_1}$;
$U_{D_2} = U_{D_2} - U'_{D_2}$;
for $i = 0$ *to* k **do**
    induzir o classificador h_{D_1} a partir dos exemplos de treinamento L_{D_1};
    induzir o classificador h_{D_2} a partir dos exemplos de treinamento L_{D_2};
    R'_{D_1} = todos os exemplos de U'_{D_1} rotulados com o classificador h_{D_1};
    R'_{D_2} = todos os exemplos de U'_{D_2} rotulados com o classificador h_{D_2};
    $(R_{D_1}, R_{D_2}) = melhores(R'_{D_1}, R'_{D_2})$;
    if $R_{D_1} = \emptyset$ **then Return** L_{D_1}, L_{D_2} ;
    $L_{D_1} = L_{D_1} \cup R_{D_1}$;
    $L_{D_2} = L_{D_2} \cup R_{D_2}$;
    if $U_{D_1} = \emptyset$ **then Return** L_{D_1}, L_{D_2} **else**
        retirar aleatoriamente alguns exemplos de U_{D_1} e os exemplos respectivos de U_{D_2} e adicioná-los a U'_{D_1} e U'_{D_2} respectivamente;
    end
end
Return L_{D_1}, L_{D_2};

No primeiro passo do algoritmo antes do laço, são criados dois subconjuntos U'_{D_1} e U'_{D_2}, tal que $U' = U'_{D_1} \cup U'_{D_2}$ e $U'_{D_1} \cap U'_{D_2} = \emptyset$, sendo U' um subconjunto de, geralmente, poucos exemplos de U, ou seja, esses dois subconjuntos de exemplos não rotulados U'_{D_1} e U'_{D_2} são subconjuntos de U_{D_1} e U_{D_2} respectivamente. Os exemplos que compõem U'_{D_1} e U'_{D_2} são removidos de U_{D_1} e U_{D_2} a fim de verificar as condições $U'_{D_1} \cap U_{D_1} = \emptyset$ e $U'_{D_2} \cap U_{D_2} = \emptyset$, criando assim conjuntos disjuntos. Após cada iteração com os conjuntos de exemplos rotulados L_{D_1} e L_{D_2} são induzidos, respectivamente, dois classificadores h_{D_1} e h_{D_2} os quais são utilizados para rotular exemplos não rotulados de U'_{D1} e U'_{D2} respectivamente. No fim deste processo R'_{D_1} e R'_{D_2} contém, respectivamente, o conjunto de exemplos rotulados nesse passo da iteração, os quais são argumentos da função *melhores/2* que é responsável pela seleção de exemplos que foram "melhor" rotulados, e com o mesmo rótulo, pelos respectivos classificadores h_{D_1} e h_{D_2}.

A função *melhores/2* pode ser considerada como a mais importante na implementação do algoritmo CO-TRAINING. Após a execução da função *melhores/2*, os exemplos que foram "melhor" rotulados são adicionados, respectivamente, aos conjuntos de exemplos de treinamento L_{D_1} e L_{D_2}. No caso de ainda existirem exemplos não rotulados, são utilizados mais exemplos ainda não rotulados em U_{D1} e U_{D2} os quais são adicionados, respectivamente, aos conjuntos U'_{D1} e U'_{D2} e o processo é repetido.

Com os dois classificadores h_{D_1} e h_{D_2} é possível criar um terceiro classificador, também conhecido como classificador combinado. Esse terceiro classificador é criado para que seja possível comparar CO-TRAINING com outros algoritmos de aprendizado supervisionado ou semi-supervisionado que não utilizam o conceito multi-descrição.

O classificador combinado, descrito pelo Algoritmo 3, calcula a probabilidade $P(y_v|\mathbf{x}_i)$ multiplicando as probabilidades obtidas dos classificadores h_{D_1} e h_{D_2}. Para calcular a probabilidade $P(y_v|\mathbf{x}_i)$ considerando ambos os classificadores como se fosse um único classificador, cada exemplo $\mathbf{x}_i$ é considerado como $(\mathbf{x}_i^{D_1}, \mathbf{x}_i^{D_2})$ no qual $\mathbf{x}_i^{D_1}$ e $\mathbf{x}_i^{D_2}$ correspondem aos valores dos atributos da primeira e segunda descrição, respectivamente, do exemplo $\mathbf{x}_i$. Os dois classificadores, h_{D_1} e h_{D_2}, fornecem $P(y_v|\mathbf{x}_i^{D_1})$ e $P(y_v|\mathbf{x}_i^{D_2})$ ou seja, a probabilidade do exemplo $\mathbf{x}_i$ ser da classe y_v dado pelo classificador induzido a partir de L_{D_1} e a probabilidade de ser da classe y_v dado pelo classificador induzido a partir de L_{D_2}. Como v varia de 1 a n_{cl}, os classificadores fornecem as probabilidades do exemplo pertencer a todas as n_{cl} classes. Para obter $P(y_v|\mathbf{x}_i)$ são multiplicadas as probabilidades $P(y_v|\mathbf{x}_i^{D_1})$ e $P(y_v|\mathbf{x}_i^{D_2})$ calculadas por h_{D_1} e h_{D_2} para cada classe, como mostrado no Algoritmo 3.

Algoritmo 3: Classificador Combinado

```
for i = 1 to n_ex do
    for v = 1 to n_cl do
        P(Y_v|x_i) ← P(Y_v|x_i^{D1}) P(Y_v|x_i^{D2});
    end
end
```

Esse terceiro classificador permite representar CO-TRAINING como um único classificador, possibilitando assim comparações com outros algoritmos que utilizam somente exemplos especificados usando uma descrição.

6.2 Proporção de Classes em CO-TRAINING

CO-TRAINING rotula a cada iteração quantidades pré-definidas de exemplos de cada classe. Desse modo, o algoritmo pode controlar a distribuição dos exemplos que irão compor o conjunto de exemplos rotulados. Nesta seção, o

parâmetro do algoritmo que controla essa proporção é testado usando diversas proporções para avaliar o quanto CO-TRAINING é sensível a esse parâmetro.

6.2.1 Sensibilidade de CO-TRAINING à Proporção das Classes

Em aprendizado de máquina, assume-se que a distribuição de dados do conjunto de treinamento é a mesma dos exemplos a serem preditos. Desse modo, a proporção de exemplos de cada classe do conjunto de treinamento e dos exemplos a serem preditos é a mesma. Em aprendizado semi-supervisionado, o conjunto de exemplos rotulados é pequeno, é perigoso utilizar a proporção das classes nesse conjunto, já que a amostra desses exemplos é pequena pode não ser estatisticamente válida.

Por exemplo, suponha que CO-TRAINING seja utilizado para um problema de classificação de páginas de internet. Pode-se construir um mecanismo de busca para acessar algumas páginas da internet e armazena as páginas de interesse. Com essas páginas em mãos, pode-se solicitar a um especialista do domínio para rotular algumas páginas manualmente. Como geralmente não se conhece a proporção de exemplos, *i.e.* páginas, que compõem cada classe, pode-se solicitar ao especialista rotular a mesma quantidade de exemplos para cada classe. Um outra opção é apresentar ao especialista uma amostra pequena dessas páginas e requisitar que o especialista rotule essa amostra. Porém, em ambos os casos, não se pode ter uma estimação confiável da proporção da classes para a execução de CO-TRAINING.

Assim, em problemas reais de aprendizado, normalmente o usuário de CO-TRAINING não tem como saber a distribuição correta dos exemplos, o que pode ser bastante perigoso, pois pode fazer com que o usuário inicialize incorretamente o parâmetro que controla a proporção de classes. Os algoritmos base de CO-TRAINING podem ser sensíveis a classes desbalanceadas e alterar a proporção de exemplos no conjunto de treinamento desses algoritmos base pode fazer com que CO-TRAINING degrade o seu desempenho.

Na literatura observa-se que é bastante difícil caracterizar os efeitos de alterar a proporção das classes em algoritmos de aprendizado. Existem diversos estudos que avaliam esses efeitos em algoritmos bem conhecidos de aprendizado de máquina. Weiss and Provost (2003) realizaram uma avaliação experimental utilizando árvores de decisão induzidas pelo algoritmo de aprendizado C4.5 variando a proporção das classes de vários conjuntos de dados. Os autores mostram que ao utilizar a proporção das classes original são obtidos os melhores resultados. Zadrozny (2004) mostra que regressão logística e SVM com margens rígidas são menos afetados que NAIVE BAYES, SVM com margens suaves e árvores de decisão quando a proporção das classes no conjunto de treinamento são alterados. Fan et al. (2005) estendem esses resultados e

mostram que a sensibilidade a proporção das classes não pode ser atribuído somente aos indutores mas também aos conjuntos de exemplos. Como CO-TRAINING faz uso desses algoritmos como algoritmos base, essa sensibilidade é automaticamente herdada desses indutores. A seguir é descrita uma avaliação experimental, por nós conduzida (Matsubara et al., 2006), mostra como essa sensibilidade afeta os resultados de CO-TRAINING.

6.2.2 Avaliação Experimental

Os experimentos foram conduzidos utilizando três bases de documentos (textos), as quais foram pré-processadas utilizando a ferramenta PRETEXT (Matsubara et al., 2003). PRETEXT é uma ferramenta computacional que utiliza a abordagem *bag-of-words* para transformar textos, que são naturalmente não estruturados, em uma representação estruturada, mais especificamente, em uma tabela atributo-valor. Na abordagem *bag-of-words* cada documento é representado como um vetor das palavras que ocorrem no documento, *i.e.*, cada texto representa um exemplo e as palavras no texto constituem atributos. Essa representação produz uma tabela atributo-valor esparsa. Assim, é necessário utilizar métodos para reduzir a quantidade de atributos para uma melhor representação.

Vários métodos podem ser utilizados a fim de reduzir a quantidade de termos (atributos) necessário para representar uma coleção de documentos. A transformação de cada termo para o radical que o originou, por meio de algoritmos de *steamming*, é um método amplamente utilizado e difundido. Uma outra forma para reduzir a dimensionalidade dos atributos é buscando os termos que são mais representativos na discriminação dos documentos usando os cortes de Luhn (Luhn, 1958).

Stemming consiste em uma normalização lingüística, na qual as formas variantes de um termo são reduzidas a uma forma comum denominada *stem*. A conseqüência da aplicação de algoritmos de *steamming* consiste na remoção de prefixos ou sufixos de um termo, ou mesmo na transformação de um verbo para sua forma no infinitivo. Por exemplo, as palavras `trabalham`, `trabalhando`, `trabalhar`, `trabalho` e `trabalhos` podem ser transformados para um mesmo *stem* `trabalh`. Desse modo, algoritmos de *steamming* podem ser utilizados para reduzir a dimensão da tabela atributo-valor. PRETEXT possui implementações de *steamming* para português, espanhol e inglês.

Um outro modo para reduzir a dimensionalidade dos atributos é buscar os termos que são mais representativos na discriminação dos documentos. Luhn especifica dois pontos de corte, os quais denominou de superior e inferior, para excluir termos não relevantes (Luhn, 1958). Os termos que excedem o corte superior são os mais freqüentes e são considerados comuns por aparecer em

qualquer tipo de documento, como as preposições, conjunções e artigos. Já os termos abaixo do corte inferior são considerados raros e, portanto, não contribuem significativamente na discriminação dos documentos.

Uma das características da ferramenta é a construção de *stems* usando mais de um gram. No PRETEXT[1], 1-gram se refere a um *stem* simples, enquanto 2 e 3-gram referem-se a 2 ou 3 *stems*, cujas palavras ocorrem seqüencialmente no documento. A implementação original da ferramenta permite a concatenação de até 3 *stems*, ou seja, 3-gram. Atualmente a implementação esta sendo estendida para construir *stems* de qualquer ordem de grandeza.

Nesta seção é descrito em detalhes como foram feitas essas transformações utilizando PRETEXT para as três bases de textos utilizadas nos experimentos, denominadas NEWS, LNAI e COURSE descritas a seguir:

NEWS a base NEWS foi criada a partir da base *mini-news* (Asuncion and Newman, 2007) que contém textos relacionados a *news-groups*. Essa base foi criada com o objetivo de testar o CO-TRAINING com um conjunto balanceado de classes. A base original *mini-news* consiste de documentos classificados em 20 classes diferentes com 100 documentos de cada classe. A nova base NEWS foi construída agrupando as 4 classes da base *mini-news* relacionadas com `sci.crypt`, `sci.electronics`, `sci.med` e `sci.space` em uma única classe denominada `sci`, e as 4 classes relacionadas com `talk.politics.guns`, `talk.politics.mideast`, `talk.politics.misc` e `talk.religion.misc` em uma única classe denominada `talk`. Assim, a base NEWS consiste de 800 documentos classificados em duas classes, `sci` e `talk`, cada uma delas com 400 documentos. A primeira descrição é dada por 1-gram e a outra por 2-gram do conjunto de documentos.

LNAI Este conjunto de exemplos consiste dos títulos, resumos e referências de artigos de *Case-Based Reasoning* (CBR) e *Inductive Logic Programming* (ILP) retirados de *Lecture Notes in Artificial Intelligence* (LNAI) (Melo et al., 2003). A base LNAI tem um total de 396 artigos, dos quais 277 (70%) são da classe CBR e 119 (30%) são da classe ILP. Além das duas descrições dos dados, uma descrição referente a 1-gram e a outra referente a 2-gram, que podem ser extraídas desse conjunto de documentos, é possível extrair outras descrições considerando, por exemplo, o *bag-of-words* dos títulos, dos resumos e/ou das referências.

COURSE Este conjunto de exemplos foi construído a partir das bases de textos e links utilizado no artigo de Blum and Mitchell (1998) no qual é proposto

[1]É importante ressaltar que existem outras formas para representação de gram. Neste trabalho, *n*-gram significa que *n stems* são concatenados formando um único *stem*. Ainda, vale lembrar que, a palavra gram deve ser usada sempre no singular.

o algoritmo CO-TRAINING[2]. Essa base contém 1051 exemplos referentes a páginas de internet coletadas de quatro universidades: Universidade de Cornell, Universidade de Washington, Universidade de Wisconsin e Universidade do Texas. As páginas estão em formato html e estão dividas em duas classes: páginas referentes aos cursos oferecidos nessas universidade (`course`) e páginas que não se referem a cursos (`non-course`). As duas descrições são formadas pelos textos dessas páginas e pelos links que apontam para as mesmas. Neste trabalho, a primeira descrição, composta por textos, é sempre referida como TEXT, e a segunda descrição como LINKS. Foi observado que nem todos os exemplos (páginas de internet) na descrição LINKS continham algum conteúdo, portanto, ao realizar o pré-processamento dessa base com o PRETEXT, esses exemplos foram retirados do conjunto, o que reduziu o número de exemplos da base original para 1037 exemplos. Desses 1037 exemplos, 223 (21%) são da classe `course` e 817 (79%) da classe `non-course`.

Foram realizados os processos relacionados a *stemming* e cortes de Luhn. Para o conjunto NEWS na descrição composta por *2-gram* o corte inferior de Luhn foi fixado em 3, *i.e.*, palavras que aparecem menos de 3 vezes em toda a coleção são removidas. Para as outras descrições dos outros conjuntos de documentos, foram utilizados cortes de Luhn em 2. Na Tabela 6.1 é apresentado o resumo dos conjuntos de exemplos utilizados nesses experimentos. Na primeira coluna é apresentado o conjunto de dados (Conjunto); seguido pelo número de documentos no conjunto (#Doc); o número de *stems* obtidos (#Stem); o número de *stem* após os cortes de Luhn em cada descrição (#Atributos) e a distribuição das classes (%Classes).

Nesses experimentos foi utilizado NAIVE BAYES como algoritmo base de CO-TRAINING. Como os conjuntos de documentos estão rotulados, pode-se comparar os rótulos atributos por CO-TRAINING em cada iteração com os rótulos verdadeiros de cada documento. Apenas para verificar o desempenho de NAIVE BAYES usando os conjuntos rotulados, ele foi avaliado utilizando validação cruzada de 10 partições. Esse resultado pode ser considerado o limite superior (melhor caso) que CO-TRAINING poderia alcançar. A média do erro de NAIVE BAYES e o desvio padrão entre parênteses são mostrados na última coluna da Tabela 6.1.

Deve ser lembrado que CO-TRAINING utiliza duas descrições do conjunto de exemplos, os quais devem se corresponder um a um. Assim, o método de reamostragem de validação cruzada de 10 partições foi adaptado para essa particularidade do algoritmo, como ilustrado na Figura 6.1. Após o pré-processa-

[2] http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-51/www/co-training/data

Tabela 6.1: Descrição dos conjuntos e o erro de NAIVE BAYES

Conjunto	#Doc	Descrição	#Stem	#Atr.	Classe	%Classe	Erro NB	Erro
NEWS	800	1-gram	15711	8668	sci	50%	2.5 (1.7)	
					talk	50%	0.8 (1.2)	1.6 (1.0)
		2-gram	71039	4521	sci	50%	2.0 (2.0)	
					talk	50%	0.5 (1.1)	1.3 (1.2)
LNAI	396	1-gram	5627	2914	ILP	30%	1.7 (3.7)	
					CBR	70%	1.4 (1.9)	1.5 (1.8)
		2-gram	21969	3245	ILP	30%	1.8 (1.7)	
					CBR	70%	1.5 (1.9)	1.8 (1.7)
COURSE	1038	TEXT	13198	6870	course	20%	16.3 (5.4)	
					non-course	80%	3.8 (2.0)	6.5 (2.3)
		LINKS	1604	1067	course	20%	9.6 (7.6)	
					non-course	80%	16.0 (4.7)	14.6 (3.5)

mento das bases de documentos, cada uma das descrições do conjunto de exemplos foi particionado em 10 subconjuntos, sendo que cada subconjunto contém exemplos correspondentes a cada uma das descrições. Essas partições são sempre pareadas uma a uma, tal que as mesmas partições de ambas as descrições, em cada iteração, são utilizadas para treinamento (90%) e teste (10%). Todos os experimentos com CO-TRAINING, descrito a seguir, foram avaliados utilizando essa adaptação, na qual as 10 partições foram construídas como ilustrado na Figura 6.1.

Todos os experimentos foram executados utilizando um conjunto de 30 exemplos rotulados com 15 exemplos de cada classe (50% - 50%). Na implementação de CO-TRAINING realizada (Matsubara and Monard, 2004b) vários parâmetros podem ser definidos, tais como o *score* mínimo a ser utilizado; proporção das classes; diferentes métodos de seleção dos melhores exemplos; entre outros. Na execução dos experimentos CO-TRAINING descarta exemplos com *scores* inferiores que 0.6. Entre os exemplos não descartados são escolhidos os 10 melhores exemplos de cada classe para serem inseridos em L. Ou seja, em cada iteração, CO-TRAINING rotula até 10 exemplos, podendo ocorrer casos em que se rotule menos de 10 devido ao descarte de exemplos com *scores* inferiores a 0.6.

Na Tabela 6.2 é mostrada a média e o desvio padrão entre parênteses dos resultados obtidos. A primeira linha mostra o número máximo de exemplos selecionados por classe para serem inseridos em L a cada iteração. Os números são apresentados em pares no formato A/B que correspondem aos seguintes pares: `sci`/`talk` para NEWS, ILP/CBR para LNAI e `course`/`non-course` para COURSE. Por exemplo, a coluna 2/8 indica que a cada iteração do algoritmo são rotulados 2 exemplos da classe `sci` e 8 exemplos da classe `talk` para o conjunto NEWS, 2 exemplos da classe ILP e 8 exemplos da classe CBR para o conjunto LNAI e 2 exemplos da classe `course` e 8 exemplos da classe `non-course` para o conjunto COURSE.

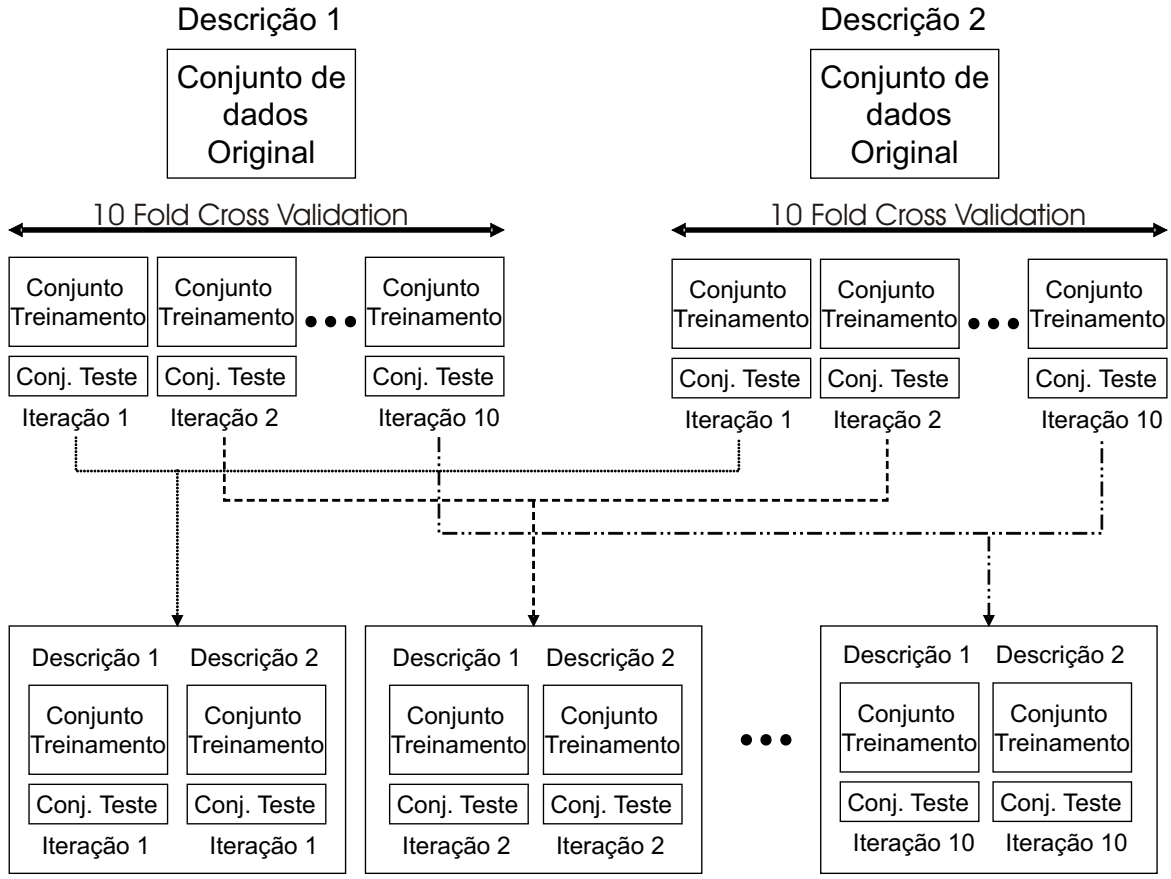


Figura 6.1: Representação gráfica da construção das 10 partições para CO-TRAINING

As primeiras quatro linhas de cada conjunto indicam quantos exemplos foram rotulados corretamente (C) ou incorretamente (I) para cada classe; LSize é o número de exemplos rotulados por CO-TRAINING, incluindo os 30 exemplos iniciais; U'Size é o número de exemplos não rotulados que não foram rotulados por CO-TRAINING; Erro e AUC são, respectivamente, a taxa de erro obtida e a área abaixo da curva ROC do classificador combinado e Errados é o número total de exemplos rotulados incorretamente. Os melhores resultados para essas três últimas medidas são apresentados em negrito.

Para todos os conjuntos, CO-TRAINING finalizou a execução devido ao critério de parada de conjunto U vazio, parando nas iterações 64, 28 e 86 para os conjuntos NEWS, LNAI e COURSE respectivamente. Como pode ser observado, os melhores resultados para os conjuntos NEWS e COURSE foram atingidos quando foram consideradas a proporção original de cada um desses conjuntos (5/5 para NEWS e 2/8 para COURSE). Para o conjunto LNAI, o melhor resultado não foi atingido utilizando a proporção original do conjunto (3/7), mas foi obtido com uma proporção muito similar à original (2/8). Para esse conjunto, note que LSize $\simeq$ 300 para 2/8 e 3/7. A principal diferença encontra-se em 3/7 que rotula corretamente todos os exemplos CBR enquanto que em 2/8

rotula corretamente todos os exemplos ILP.

Os erros do classificador combinado dos melhores resultados (negrito) são compatíveis com os erros obtidos utilizando todo o conjunto de exemplos rotulado ilustrado na Tabela 6.1.

	2/8	3/7	5/5	7/3	8/2
conjunto NEWS					
sci(I)	18.00 (26.45)	10.60 (15.47)	1.10 (1.85)	0.40 (0.52)	0.80 (0.42)
sci(C)	344.50 (2.72)	339.40 (2.50)	325.70 (11.51)	203.60 (0.52)	139.50 (1.51)
talk(I)	1.60 (1.17)	2.20 (0.63)	5.70 (10.03)	42.50 (30.34)	131.00 (18.89)
talk(C)	139.40 (1.17)	201.80 (0.63)	324.30 (10.03)	345.70 (1.89)	347.80 (3.08)
LSize	503.50 (26.53)	554.00 (15.30)	656.80 (9.77)	592.20 (30.07)	619.10 (17.00)
U'Size	206.50 (26.53)	156.00 (15.30)	53.20 (9.77)	117.80 (30.07)	90.90 (17.00)
Erro	3.00 (3.24)	2.38 (3.70)	**1.88 (2.14)**	6.25 (5.14)	19.00 (3.53)
AUC	0.98 (0.02)	0.98 (0.03)	**0.99 (0.02)**	0.97 (0.04)	0.92 (0.05)
Errados	19.80 (26.96)	12.80 (15.80)	**6.80 (11.77)**	43.70 (30.29)	133.50 (19.31)
conjunto LNAI					
ilp(I)	0.00 (0.00)	1.30 (1.25)	5.40 (1.71)	9.30 (3.23)	12.30 (5.10)
ilp(C)	69.00 (0.00)	94.20 (2.20)	101.00 (1.49)	100.80 (1.14)	101.70 (1.57)
cbr(I)	0.70 (0.95)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
cbr(C)	230.30 (0.95)	204.00 (0.00)	150.00 (0.00)	96.00 (0.00)	69.00 (0.00)
LSize	300.00 (0.00)	299.50 (1.08)	256.40 (2.41)	206.10 (3.54)	183.00 (5.10)
U'Size	50.00 (0.00)	50.50 (1.08)	93.60 (2.41)	143.90 (3.54)	167.00 (5.10)
Erro	**1.26 (1.33)**	2.02 (2.00)	2.03 (1.07)	3.28 (1.69)	4.80 (3.03)
AUC	**1.00 (0.00)**	1.00 (0.01)	0.99 (0.01)	0.99 (0.01)	0.99 (0.01)
Errados	**0.70 (0.95)**	1.30 (1.25)	5.60 (1.90)	9.30 (3.23)	12.50 (5.04)
conjunto COURSE					
course(I)	34.40 (29.73)	103.90 (66.05)	252.30 (72.89)	423.40 (27.35)	434.80 (112.58)
course(C)	146.00 (26.82)	132.80 (27.26)	155.50 (13.34)	175.40 (6.00)	179.30 (10.89)
ncourse(I)	5.30 (3.13)	7.20 (8.00)	4.20 (4.59)	1.50 (2.92)	2.40 (3.34)
ncourse(C)	505.20 (154.07)	307.10 (227.37)	146.80 (110.20)	81.60 (31.65)	81.30 (56.98)
LSize	690.90 (150.92)	551.00 (186.16)	558.80 (49.82)	681.90 (23.39)	697.80 (66.62)
U'Size	239.10 (150.92)	379.00 (186.16)	371.20 (49.82)	248.10 (23.39)	232.20 (66.62)
Erro	**14.11 (13.26)**	32.65 (20.15)	49.43 (15.95)	61.91 (8.07)	60.29 (17.28)
AUC	**0.92 (0.08)**	0.82 (0.11)	0.71 (0.09)	0.68 (0.07)	0.67 (0.07)
Errados	**40.20 (31.71)**	112.80 (67.28)	258.70 (72.08)	429.80 (25.59)	442.60 (111.98)

Tabela 6.2: Resultados de CO-TRAINING para os conjuntos NEWS, LNAI e COURSE

Ao analisar o comportamento de CO-TRAINING quando são alteradas a distribuição das classes, é possivel verificar alguns padrões interessantes. Para o conjunto de dados balanceado NEWS, alterando a proporção dos exemplos favorecendo a classe `talk`, *i.e.*, rotulando mais exemplos da classe `talk` (3/7 e 2/8), o desempenho do algoritmo não degrada tanto se comparado com 5/5. Entretanto, ao inverter a proporção (7/3 e 8/2) o erro aumenta significativamente (de 1.88 em 5/5 para 19.00 em 8/2) e o número de exemplo rotulados incorretamente também mostra esse comportamento (de 6.8% para 133.50%). Observe nos resultados da Tabela 6.1 para o conjunto NEWS que a classe `talk` (erros próximos a 2%) é um pouco mais "difícil" de se aprender que `sci` (erro menores que 0.8%). Ao rotular mais exemplos da classe mais "difícil" pode facilitar o aprendizado nesses casos. Para os outros dois conjuntos LNAI e COURSE, é bastante evidente que quando a proporção de exemplos vai em direção contrária a proporção de exemplos do conjunto original, a taxa de erro e o número de exemplo rotulados incorretamente aumenta.

Outro resultado interessante está relacionado à AUC. Para conjuntos com AUC alto, NEWS e LNAI, a degradação em termos de desempenho é menor que no conjunto COURSE. AUC próximo a 1 indica que os conjuntos de dados possuem uma boa separação entre as classes, o que torna mais fácil induzir classificadores com bom desempenho, mesmo com uma proporção de exemplos diferente do conjunto original.

O gráfico ROC para os conjuntos NEWS e LNAI para as diferentes proporções de classes são muito similares, já que a AUC desses conjuntos são praticamente 1.0 em todas as partições da validação cruzada. O conjunto COURSE, por sua vez, possui AUCs diferentes para diferentes proporções. Para visualizar essas diferentes curvas, na Figura 6.2 são ilustradas as curvas ROC para o classificador combinado na última iteração de CO-TRAINING (68^a), (positivo = `non-course`, negativo = `course`) para as proporções testadas. Cada linha representa a média das curvas ROC das 10 partições da validação cruzada para cada proporção.

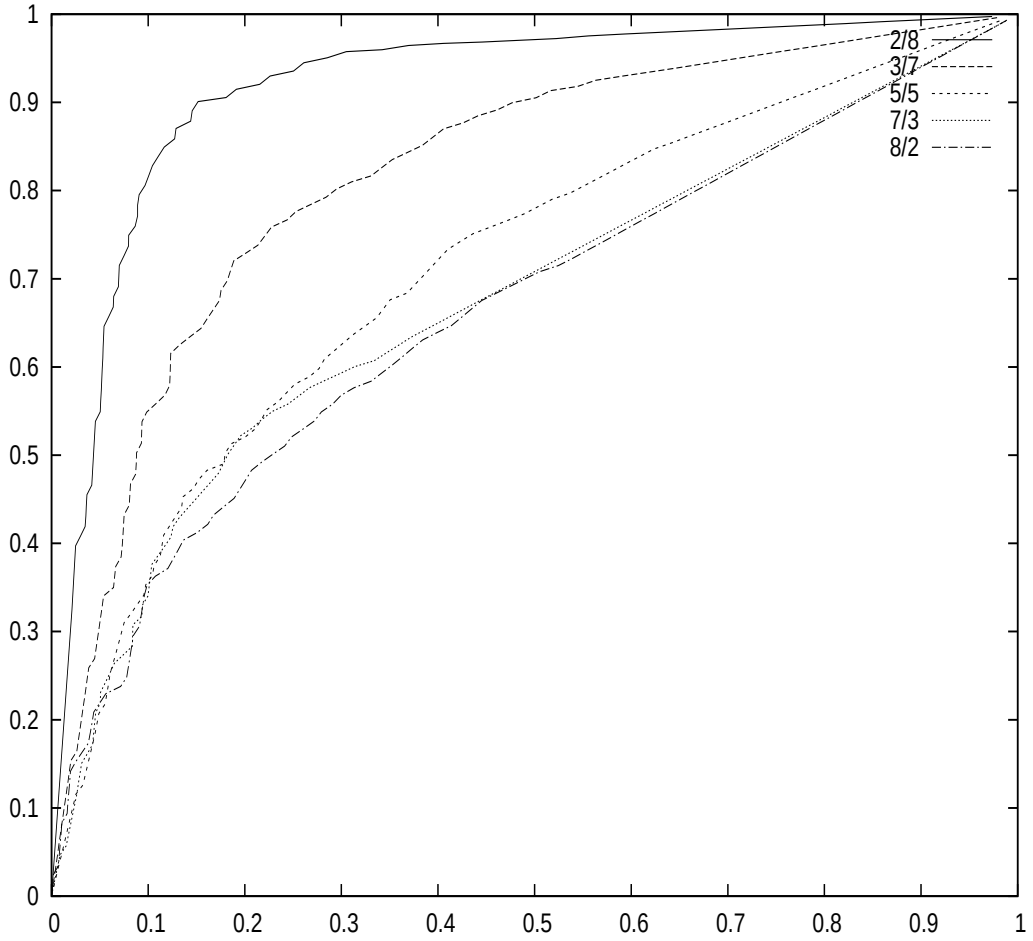


Figura 6.2: Gráfico ROC do conjunto COURSE para a última iteração de CO-TRAINING para diferente proporções de classes

A curva ROC para as proporções 2/8 e 3/7 não evidenciam classificadores mais específico para nenhuma das classes, já que as distâncias do eixo Y

a essas curvas e eixo superior a essas curvas são praticamente iguais. Outro modo de verificar isso é traçar uma reta que liga o céu ROC e o inferno ROC e verificar as curvas que são espelhadas no triangulo superior e inferior, se forem espelhadas indicam que o classificador não privilegia nenhuma das classes. Observando agora as proporções 7/3 e 8/2, a curva indica que o classificador é um pouco melhor para a classe positiva que para a negativa, já que no começo da curva, partindo de (0,0), ela é mais vertical que horizontal (praticamente uma diagonal). É interessante observar que a proporção 5/5 não evidencia privilégio para nenhuma das classe, sendo praticamente espelhada ao comparar os triângulos ao traçar a reta entre o céu e o inferno ROC.

Em linhas gerais, CO-TRAINING possui o melhor desempenho quando é utilizada a proporção correta das classes. Entretanto, como já mencionado, quando o conjunto de treinamento é pequeno, é perigoso utilizar a proporção desse conjunto. Como recomendação geral, e como ilustrado pela curva ROC, quando o domínio do problema não permite inferir uma proporção razoável de exemplos, é aconselhável rotular em cada iteração de CO-TRAINING a mesma proporção de exemplos para as classes envolvidas.

6.3 Agregação de Rankings em CO-TRAINING

Problemas de valores faltantes em conjuntos de exemplos possuem bastante similaridades com aprendizado semi-supervisionado. Quando o atributo com valores faltante é tratado como o atributo classe, podem existir poucos exemplos rotulados e muito exemplos não rotulados. Dessa maneira, problemas de atributos faltantes podem ser facilmente convertidos para problemas de aprendizado semi-supervisionado, com ganhos em termos de teoria e algoritmos para ambas as áreas. Assim, nesta seção é descrita a nossa proposta de uma variação de CO-TRAINING, denominada de CORAI (CO-TRAINING *ranking agregation*), adaptada para o problema de atributos com valores faltantes.

6.3.1 CORAI

As principais diferenças entre CORAI e CO-TRAINING são o uso de uma estratégia diferente para selecionar os melhores exemplos a serem rotulados em cada iteração e o uso de dois algoritmos indutores ao invés de duas descrições dos dados.

Usando a mesma notação utilizada para descrever CO-TRAINING, CORAI pode ser descrito da seguinte maneira: inicialmente, uma pequena amostra de exemplos não rotulados, conjunto de exemplos U', é criada a partir de U e o principal laço do Algoritmo 4 começa. O conjunto de exemplos rotulados L é utilizado para induzir dois classificadores h_1 e h_2 utilizando dois algorit-

mos indutores diferentes (NAIVE BAYES e C4.5 por exemplo). O próximo passo consiste em rotular o conjunto U' com h_1, inserir os exemplos rotulados em R'_1 e rotular novamente o conjunto U', mas agora com h_2, e inserir os exemplos rotulados em R'_2. Os dois conjuntos rotulados, R'_1 e R'_2, são dados como parâmetro à função *melhores*/2, que é responsável por produzir *rankings* dos exemplos utilizando algum critério e selecionar os melhores exemplos e inseri-los em L. Após, o laço é repetido até que um critério de parada seja atingido.

Algoritmo 4: CORAI

Input: L, U
Output: L
construir conjunto U' ;
$U = U - U'$;
while *não atingir critério de parada* **do**
  Induzir h_1 de L;
  Induzir h_2 de L;
  $R'_1 = h_1(U')$;
  $R'_2 = h_2(U')$;
  $R = melhores(R'_1, R'_2)$;
  $L = L \cup R$;
  if $U_1 = \emptyset$ **then return**(L) **else**
    Selecionar randomicamente exemplos de U e inserir em U';
  end
end
return(L_1);

A função *melhores*/2, é uma versão modificada que utiliza agregação de *rankings* e difere significativamente da função *melhores*/2 de CO-TRAINING. Originalmente, em CO-TRAINING, essa função filtra exemplos que quebram a compatibilidade das descrições, *i.e.*, $h_1(\mathbf{x}_i) \neq h_2(\mathbf{x}_i)$. Entretanto, atributos com valores faltantes podem possuir diversos valores e quando os exemplos são filtrados por $h_1(\mathbf{x}_i) \neq h_2(\mathbf{x}_i)$ quase todos os exemplos são filtrados. Isso acontece pois é menos provável que os classificadores concordem com a classificação dos exemplos em problemas multi-classe. Para evitar que muitos exemplos sejam filtrados, é proposto o uso de agregação de *rankings* para selecionar os melhores exemplos.

Inicialmente, o problema de valor faltante deve ser mapeado para um problema de aprendizado semi-supervisionado pela troca do atributo classe por um atributo com valor faltante A^*_j (A_{classe} é na realidade o atributo A^*_j). Utilizando NAIVE BAYES e C4.5 como classificadores *score* e partindo de R'_1 e R'_2 dado como parâmetro da função *melhores*/2, a agregação de *ranking* combina esses conjuntos em um mesmo conjunto para obter uma "melhor" ordenação. Seja y^1_v os *scores* de R'_1 e y^2_v os *scores* de R'_2 para a classe y_v com $v = 1, ..., n_{cl}$. Os exemplos são ordenados de acordo com os *scores* para cada classe e a posição do *ranking* é armazenada separadamente para cada classe em $rpos^1_v$ para *scores* obtidos de R'_1 e $rpos^2_v$ para *scores* obtidos de R'_2, onde v indica a classe.

Por exemplo, para calcular $rpos_1^1$ ordenam-se os *scores* da classe y_1^1, a posição do *ranking* obtido é armazenado em $rpos_1^1$. Isso é feito para todas as classes $y_1^1, ..., y_{n_{cl}}^1$ para obter $rpos_1^1, ..., rpos_{n_{cl}}^1$. Do mesmo modo, utilizando R_2' é calculado $rpos_1^2, ..., rpos_{n_{cl}}^2$. Finalmente, a posição combinada dos *rankings* obtidos de R_1' e R_2' é dada por $rpos_v = rpos_v^1 + rpos_v^2$ para cada classe y_v, e os exemplos selecionados são aqueles com baixo $rpos_v$. Utilizar os exemplos com baixo $rpos_v$ significa que esses exemplos possuem uma boa posição no *ranking*, *i.e.*, primeiros colocados no *ranking* em ambas as descrições.

6.3.2 Configuração dos Experimentos

Os experimentos foram conduzidos utilizando três conjuntos de dados sem valores faltantes do repositório de bases de dados UCI (Asuncion and Newman, 2007). Para um maior controle dos experimentos, os valores faltantes foram artificialmente inseridos nos conjuntos de dados. Na avaliação realizada, os valores faltantes foram inseridos randomicamente independente da classe ou dos valores dos atributos. Esse padrão de inserção também é conhecido como padrão *Missing Completely at Random* (MCAR). Os valores originais que foram transformados em valores faltantes são conhecidos e, desse modo, o erro de imputação pode ser medido. Essa configuração experimental possibilita que os dados faltantes sejam inseridos em diferentes taxas em diferentes atributos.

A Tabela 6.3 resume os conjuntos de dados utilizados neste estudo. Para cada conjunto são mostrados o número de exemplos (#Exemplos); o número de atributos (# Atributos), juntamente com o número quantitativo e qualitativo de atributos; número de classes (#Classes) e o erro da classe majoritária.

Tabela 6.3: CORAI: resumo do conjunto de exemplos utilizados

Conjuntos	# Exemplos	#Atributos (quanti., quali.)	#Classes	Majoritário error
CMC	1473	9 (2,7)	3	57.29%
German	1000	20 (7,13)	2	30.00%
Heart	270	13 (7,6)	2	44.44%

Para escolher os atributos utilizados para inserir os valores faltantes, foram conduzidos alguns experimentos para selecionar os atributos mais relevantes para predizer o atributo classe. Note que é importante imputar (preencher os valores faltantes) de atributos relevantes. Caso contrário, ao utilizar um atributo não representativo, o atributo pode não fazer parte da hipótese induzida. Como encontrar o atributo mais relevante não é um tarefa trivial, três métodos de seleção de atributos do sistema Weka (Witten and Frank, 2005) foram utilizados: *Chi-squared ranking filter*; *Gain ratio feature evaluator* e *ReliefF ranking filter*. Os três métodos de seleção de atributos geraram três *ranking* dos atributos mais relevantes para cada conjunto de exemplos. Fazendo a média

dos *rankings*, os três atributos mais relevantes de cada conjunto foram selecionados para serem imputados. Na Tabela 6.4 são mostrados os atributos nominais selecionados para cada conjunto de dados, assim como o número de valores diferentes em cada atributo (#Valores).

Tabela 6.4: Atributos selecionados para cada conjunto de exemplos

Conjunto	Atributos Selecionados		
	(posição)	nome	#Valores
CMC	1^o	(1) wife education	4
	2^o	(2) husband education	4
	3^o	(7) standard of living	4
German	1^o	(0) status	4
	2^o	(2) credit history	5
	3^o	(5) savings account	5
Heart	1^o	(12) thal	3
	2^o	(2) chest pain	4
	3^o	(8) angina	2

A validação cruzada de 10 partições foi utilizada para particionar os conjuntos de treinamento e teste. Os valores faltantes foram inseridos em 20%, 40%, 60% e 80% do número total de exemplos do conjunto de dados respectivo em cada um dos três atributos selecionados. Essa amostragem de valores faltantes foi realizada independentemente para cada atributo.

Na avaliação experimental foram utilizados três métodos para tratar valores faltantes: o método proposto CORAI; um método de imputação baseado em k vizinhos mais próximos conhecido como 9NNI (Batista and Monard, 2003) e a imputação utilizando a moda, que substitui os valores faltantes pelo valor de atributo mais freqüente. Para avaliar os métodos quando os valores faltantes ocorrem em mais de um atributo simultaneamente, CORAI foi executado independentemente para cada atributo faltante, *i.e.*, em cada execução, apenas um atributo faltante por vez é considerado como atributo classe, e os outros atributos com valores faltantes são mantidos no conjunto de dados.

6.3.3 Resultados Experimentais

O principal objetivo é avaliar o erro de imputação utilizando o método proposto comparado com 9NNI e moda. Como mencionado anteriormente, os valores faltantes foram inseridos artificialmente para poder comparar o verdadeiro valor dos atributos com o valor imputado. Para analisar estatisticamente esses métodos, os resultados foram analisados utilizando o teste de Friedman. O teste de Friedman foi realizado com as seguintes quatro diferentes hipóteses nula:

1. todos os métodos de imputação possuem desempenho similar para todos os resultados;

2. todos os métodos de imputação possuem desempenho similar para cada porcentagem de valores faltantes;

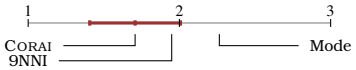


Figura 6.3: Resultado do teste Bonferroni-Dunn para todos os valores imputados. A linha mais grossa indica o intervalo de diferença crítica com 95% de confiança

3. todos os métodos de imputação possuem desempenho similar para diferentes quantidades de atributos com valores faltantes;

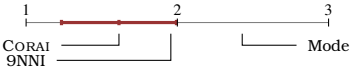
4. todos os métodos de imputação possuem desempenho similar para cada porcentagem de valores faltantes para diferentes quantidades de atributos com valores faltantes.

Quando a hipótese nula é rejeitada pelo teste de Friedman com 95% de confiança, é realizado um teste *post-hoc* para detectar quais diferenças são significativas entre os métodos comparados. O teste *post-hoc* escolhido foi o Bonferroni-Dunn e CORAI foi utilizado como controle do teste. Desse modo, o teste *post-hoc* pode mostrar se existe diferença significativa entre CORAI e os outros métodos de imputação envolvidos nas análises experimentais.

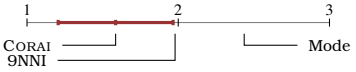
Na Figura 6.3 é ilustrado o resultado de Bonferroni-Dunn para a primeira hipótese nula: os métodos de imputação possuem desempenho similar (comparável) considerando todos os resultados. Este teste não faz nenhuma distinção entre a porcentagem de valores faltantes nem no número de atributos com valores faltantes. Como pode ser verificado na Figura 6.3, CORAI possui performance signitivamente melhor que MODE, mas que não existe diferença significativa entre CORAI e 9NNI.

A segunda hipótese nula verifica se existe diferença significativa entre os métodos de imputação para cada porcentagem de valores faltantes. O objetivo dessa avaliação é verificar se existe algum método que pode ter desempenho melhor para diferentes porcentagens de valores faltantes, ou quando a porcentagem de valores faltantes é alta. Na Figura 6.4 é ilustrado o resultado dessa análise. Pode-se observar que CORAI frequentemente possui desempenho melhor que outros métodos. Observe que quanto maior a porcentagem de exemplos faltantes, melhor é o desempenho de CORAI em relação aos outros dois métodos.

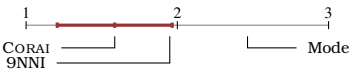
A terceira hipótese nula verifica se existe diferença significativa entre os métodos de imputação quando comparados com diferentes quantidades de atributos com valores faltantes. O objetivo dessa análise é verificar se algum método é privilegiado ao aumentar ou diminuir o número de atributos faltantes. Na Figura 6.5 é ilustrado o resultado do teste Bonferroni-Dunn. Como pode ser observado, CORAI possui o menor erro de imputação seguido por 9NNI e MODE. CORAI possui um desempenho melhor com 95% de confiança



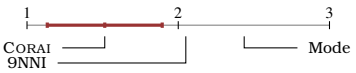
(a) 20% de valores faltantes



(b) 40% de valores faltantes

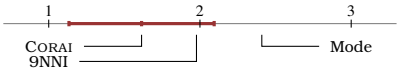


(c) 60% de valores faltantes

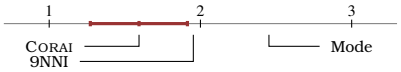


(d) 80% de valores faltantes

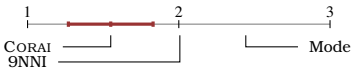
Figura 6.4: Resultados do teste Bonferroni-Duun considerando diferentes porcentagens de valores faltantes. As linhas grossas indicam o intervalo de diferença crítica com 95% de confiança



(a) 1 atributo com valores faltantes



(b) 2 atributos com valores faltantes



(c) 3 atributos com valores faltantes

Figura 6.5: Resultados do teste Bonferroni-Dunn considerando diferentes números de atributos com valores faltantes. As linhas grossas indicam o intervalo de diferença critica com 95% de confiança

quando valores faltantes são inseridos em dois ou três atributos. Quando valores faltantes são inseridos em apenas um atributo, CORAI possui desempenho melhor que MODE, mas não estatisticamente melhor que 9NNI.

Na Tabela 6.5 são mostrados os resultados do teste de Friedman e Bonferroni-Dunn para a quarta hipótese nula. Com essa hipótese é verificado se existe diferença significativa entre os métodos de imputação em combinações com diferentes de percentagens de valores faltantes e diferentes quantidades de atributos com valores faltantes. Nessa tabela, a coluna "%Faltantes" representa a percentagem de valores faltantes inseridos nos atributos selecionados;"#Atributos" representa o número de atributos com valores faltantes; as colunas "CORAI", "9NNI" e "MODE"representam o *ranking* médio utilizado para os teste de significância, e a coluna "CD" representa a diferença crítica para o teste Bonferroni-Dunn com 95% de confiança. Diferenças entre os *rankings* médios maiores que esses valores indicam que existe diferença significativa. Os resultados em cinza escuro indicam que CORAI é significativamente melhor que 9NNI e MODE, e os resultados em cinza claro indicam que CORAI é significativamente melhor que MODE. Como pode ser observado, CORAI sempre possui um menor valor nas médias dos *rankings* e, desse modo, possui

um desempenho na média melhor que os outros métodos.

Tabela 6.5: Resultado da média dos *rankings* para Bonferroni-Dunn para diferentes percentagens de valores faltantes e diferentes números de atributos com valores faltantes. Regiões em cinza indicam diferença significativa com 95% de confiança

%Faltantes	#Atributos	Métodos de Imputação			CD
		CORAI	9NNI	Mode	
m20	#At=1	1.727	1.864	2.409	0.96
	#At=2	1.625	1.958	2.417	0.65
	#At=3	1.571	1.986	2.443	0.54
m40	#At=1	1.545	2.045	2.409	0.96
	#At=2	1.625	1.917	2.458	0.65
	#At=3	1.571	2.000	2.429	0.54
m60	#At=1	1.636	1.955	2.409	0.96
	#At=2	1.583	1.938	2.479	0.65
	#At=3	1.571	1.957	2.471	0.54
m80	#At=1	1.545	2.045	2.409	0.96
	#At=2	1.542	2.000	2.458	0.65
	#At=3	1.486	2.086	2.429	0.54

6.4 Considerações Finais

Nesta capítulo foi apresentado o algoritmo CO-TRAINING, a exploração de um dos parâmetros do algoritmo que controla a proporção das classes e a aplicação de CO-TRAINING para problemas de atributos com valores faltantes. Verificou-se que CO-TRAINING é bastante sensível ao parâmetro que controla a quantidade de exemplos a ser inserido no conjunto de exemplos rotulados e, quando inicializado corretamente, o algoritmo pode apresentar uma melhora de desempenho significativa. Foi também proposto uma variação de CO-TRAINING, denominado CORAI, para tratar problemas de atributos com valores faltantes que mostrou-se melhor que 9NNI e MODE nos experimentos conduzidos neste trabalho.

CAPÍTULO

7

Conclusões

Neste capítulo são apresentadas as conclusões deste trabalho. Na Seção 7.1 é realizado um paralelo entre os objetivos desta tese e os resultados obtidos. Na Seção 7.2 são discutidas algumas limitações das soluções propostas e na Seção 7.3 são apresentadas algumas direções de trabalhos futuros.

7.1 Resumo dos Objetivos e Principais Resultados

O valor de confiança de classificação utilizado para produzir *rankings* de exemplos constitui uma rica fonte de pesquisas para a área de aprendizado de máquina. Neste trabalho, os *ranking* de exemplos foram explorados com o objetivo de mostrar algumas das informações que podem ser obtidas utilizando análise ROC e calibração, além de mostrar algumas das aplicações que ambas podem ter no algoritmo de aprendizado semi-supervisionado multi-descrição CO-TRAINING. A seguir, os questionamentos levantados na introdução relacionados aos objetivos deste trabalho são apresentados novamente juntamente com algumas das soluções encontradas.

1. *Quais são as diferenças entre ranking e classificação? É possível desenvolver um algoritmo específico para ranking? Quais os benefícios que um algoritmo específico para ranking pode oferecer?*

   O *ranking* requer como entrada um conjunto de exemplos, enquanto que o classificador requer um exemplo. O *ranking* estabelece uma relação de ordem entre os exemplos e essa relação requer pelo menos dois exemplos. Procurar pelo *ranking* ótimo requer um grau de refinamento maior

que procurar pelo classificador ótimo. Em outras palavras, pode-se ter um classificador ótimo que classifica corretamente todos os exemplos, mas a ordem dos exemplos pode estar errada. Inverter a ordem de dois exemplos adjacentes de mesma classe no *ranking* altera o *ranking* mas pode não alterar a classificação.

Desenvolver um algoritmo que ordena exemplos e torná-lo um algoritmo de aprendizado foi um dos desafios deste trabalho. Ao estudar os *rankings* obtidos usando NAIVE BAYES e árvores de decisão, encontrou-se uma maneira comum de representá-los em forma de *ranking*, o qual foi denominada *ranking* lexicográfico. Dessa representação surgiram algumas idéias interessantes que resultaram no algoritmo LEXRANK. A vantagem desse algoritmo é obter a ordenação dos exemplos sem a necessidade de *scores*. Deve ser observado que isso é significativamente diferente do que é realizado pelos algoritmos usuais de aprendizado de máquina, os quais atribuem *scores* aos exemplos e o *ranking* é obtido pela ordem dos *scores*. Dessa pesquisa resultaram as seguintes publicações: (Flach and Matsubara, 2007b,a; Flach, 2007).

2. *Quais são as informações que um gráfico ROC pode fornecer? Existe alguma maneira de utilizar a curva ROC para ilustrar o processo de indução de hipótese de algoritmos de aprendizado? Quais algoritmos? Como?*

   Gráficos ROC podem ser utilizados tanto para comparar classificadores, no qual cada classificador é representado como um ponto no gráfico ROC, como podem ser utilizados para avaliar em detalhes o *ranking* gerado por um classificador *score*. Gráficos ROC separam o desempenho por classes e também permitem observar o ganho de classificação considerando a distribuição dos exemplos nas classes, contrariamente a outras medidas escalares que somente conseguem mostrar o ganho para uma classe por vez. Em gráficos ROC, esse problema é resolvido plotando o desempenho em duas dimensões, uma classe para cada dimensão. Desse modo, pode-se analisar todas as variações de quando o classificador *score* ou os classificadores são bons para uma classe específica, sem o comprometimento da análise da outra classe, além de também mostrar situações nas quais os resultados são bons para ambas as classes.

   Normalmente, durante o processo de indução de hipótese por algoritmos de aprendizado de máquina, ao plotar o gráfico ROC para cada passo dessa indução, dificilmente encontra-se um padrão entre um passo e outro quando não se estabelece uma certa ordenação nos atributos utilizados. Talvez, um dos poucos algoritmos nos quais se encontra um certo

padrão de uma maneira mais direta é o que constrói árvores de decisão. Cada particionamento do espaço de exemplos é como um particionamento na árvore de decisão. A visualização desses particionamentos é interessante, principalmente quando aliada com curvas de iso-desempenho e os candidatos (valores de atributos) a particionamento. LEXRANK é outro algoritmo que pode ser plotado diretamente em curvas ROC, pois, como mostrado, ele pode ser representado como uma árvore de decisão. A pesquisa sobre esse tema resultou na implementação da ferramenta PROGROC que foi apresentada na conferência européia de aprendizado de máquina (ECML) na palestra convidada (Flach, 2007).

3. *Como a calibração se relaciona com curvas ROC? Como é essa relação com rankings? Quais ganhos podem ser obtidos com essas relações?*

   A inclinação (coeficiente angular) de cada segmento do feixe convexo de uma curva ROC é equivalente a razão de verossimilhança, a qual pode ser facilmente convertida na probabilidade a posteriori $P(+|\mathbf{x})$. Essa conversão é um método de calibração de probabilidades e, coincidentemente, é equivalente ao algoritmo PAV que implementa a regressão isotônica. Essa equivalência entre feixe convexo e PAV foi descoberta neste trabalho independente e paralelamente ao trabalho realizado por Fawcett and Niculescu-Mizil (2007).

   PAV tem como critério de minimização o *Brier score*. Durante a nossa pesquisa, foi encontrada uma decomposição que possui interpretações no gráfico ROC, as quais podem auxiliar a compreender algumas características da calibração e os motivos que fazem com que, em alguns casos, não se consiga obter uma calibração perfeita. Essa decomposição foi publicada em conjunto com LEXRANK no artigo (Flach and Matsubara, 2007b).

4. *Por que o algoritmo* CO-TRAINING *é interessante para ranking e análise ROC? Como podem ser utilizados em* CO-TRAINING*? Quais são os ganhos dessa união?*

   CO-TRAINING tem uma relação interessante com análise ROC (proporção das classes). CO-TRAINING iterativamente incrementa o conjunto de exemplos rotulados. Esse incremento pode ser feito usando diferentes proporções de classes e quantidade de exemplos, o que possibilita um certo controle sobre a proporção das classes no conjunto de exemplos rotulados. Desse modo, foram executados diversos experimentos variando essa proporção e verificou-se que CO-TRAINING melhora seu desempenho quando é utilizada a proporção igual ou semelhante ao do conjunto de exemplos rotulados. Entretanto, em um problema real não se conhece a

priori essa proporção. Como recomendação nesses casos, uma solução razoável é utilizar a mesma proporção de exemplos para todas as classes, como mostrado no artigo (Matsubara et al., 2006).

CO-TRAINING tem uma relação interessante com *rankings*. No critério de seleção dos exemplos que incrementam o conjunto de exemplos rotulados, em cada iteração podem ser construídos *rankings* de exemplos, um para cada classe, e podem ser considerados os exemplos do topo do *ranking* (os melhores exemplos de cada classe).

Na tentativa de usar essa idéia de aplicar CO-TRAINING a um problema de valores faltantes, verificou-se que seriam necessários algumas adaptações no algoritmo. Essa nova versão de CO-TRAINING foi denominada CORAI, a qual utiliza agregação de *rankings* para obter melhores resultados. CORAI gerou bons resultados que serão publicados em (Matsubara et al., 2008).

Deve ser observado que o algoritmo CO-TRAINING foi o principal tema da dissertação de mestrado do candidato, assim como o problema de pré-processamento de textos e sua conversão em uma tabela atributo valor. O estudo de CO-TRAINING foi bastante proveitoso e foram geradas algumas publicações sobre o assunto, bem como foi projetada e implementada a ferramenta PRETEXT para pré-processamento de textos a qual foi utilizada no desenvolvimento de diversos trabalhos relacionados (Matsubara et al., 2007; Matsubara and Monard, 2006; Matsubara et al., 2004, 2005b,a; Matsubara and Monard, 2004b,a; Matsubara, 2004; Matsubara et al., 2003; Martins et al., 2004, 2003b,a).

Também, no início do doutorado o candidato auxiliou no projeto e desenvolvimento de uma ferramenta para leitura de formulários médicos que resultaram nas seguintes publicações (Maletzke et al., 2007b,c,a, 2006).

7.2 Limitações

O uso correto de técnicas, métodos e algoritmos está relacionado com o conhecimento das limitações dos mesmos. Nesta seção são apresentadas algumas das limitações das soluções propostas neste trabalho:

1. *Rankings* são não paramétricos e, apesar de usarem os *scores*, são utilizados apenas para ordenar exemplos. Uma vez obtida a ordem, os *scores* podem ser descartados. Quando o *score* possui uma informação importante, por exemplo, uma probabilidade, deve-se procurar outras soluções além de *rankings*.

LexRank possui a limitação de poder ser aplicado apenas a conjunto de exemplos binários. Foram feitos testes preliminares com versões de LexRank que trabalham com atributos com valores numéricos (valores reais), usando mais de 30 conjuntos de dados da UCI, mas não foram encontrados ganhos significativo se comparados com os resultados obtidos com os mesmos conjuntos mas binarizados. Essa versão preliminar possui como vantagem apenas a de poder trabalhar diretamente com o conjunto de dados sem a necessidade da binarizar os atributos do conjunto de dados.

2. A análise ROC pode ser utilizada apenas para problemas com duas classes. Versões e variações de curvas ROC para mais de duas classes têm sido propostas na literatura (Fawcett, 2006). Entretanto, isso continua sendo uma das limitações de análise ROC.

   Quanto a ferramenta ProgROC desenvolvida, ela somente aceita como entrada conjuntos de dados com atributos binários. Entretanto, não é muito difícil estendê-la para conjuntos de dados com atributos numéricos já que, para esse tipo de atributos, a partição divide a árvore em apenas dois ramos.

3. Assim como em análise ROC, os métodos de calibração somente trabalham com problemas de duas classes. Variações para problemas com mais de duas classes utilizam a abordagem um *versus* todos, como mostrado em (Zadrozny and Elkan, 2002), para reduzir o problema para duas classes.

4. O algoritmo Co-training normalmente limita-se a problemas com duas descrições dos dados. Porém, como mostrado por Goldman and Zhou (2000), quando uma descrição dos dados estiver disponível, pode-se utilizar dois algoritmos indutores diferentes ao invés de duas descrições. Entretanto, essa abordagem, apesar de ser bastante válida, perde um pouco os conceitos utilizados em Co-training.

7.3 Trabalhos Futuros

Algumas sugestões de possíveis refinamentos e extensões dos algoritmos e métodos apresentados neste trabalho, além de novas idéias que surgiram durante o desenvolvimento desta tese, são apresentadas a seguir.

1. Como mencionado, LexRank possui limitações quanto ao conjunto de exemplos de entrada, o qual deve ter somente atributos binários. Desse

modo, algumas extensões podem ser melhor estudadas para melhorar o algoritmo.

2. LexRank requer apenas a ordenação dos exemplos e a definição de um valor de preferência. Assim, diversos métodos de seleção de atributos podem ser utilizados para ordenar os atributos e gerar versões diferentes de LexRank. Originalmente, LexRank utiliza razão de chances (*odds ratio*) para ordenar atributos. Apesar de já terem sido testadas outras medidas, como ganho de informação, uma exploração mais ampla com diversas outras medidas ainda é necessária.

3. Utilizando a mesma idéia do item anterior, mas agora utilizando métodos semi-supervisionados de seleção de atributos para ordenar atributos, pode-se obter uma versão semi-supervisionada de LexRank.

4. ProgROC pode ser estendido para problemas de atributos numéricos e nominais. Também, espera-se implementar num futuro próximo versões para outros algoritmos de aprendizado de máquina, tais como aprendizado por regras (Prati and Flach, 2005).

5. A decomposição encontrada do *Brier score* pode ser mais explorada e analisada. Acreditamos que essa decomposição também tenha uma relação com erro de bias e variância, os quais também podem ser melhor explorados.

6. Co-training requer duas descrições dos dados. Uma pergunta pertinente é: vale a pena investigar métodos para construir descrições a partir de uma só e fornecer para Co-training? Quais métodos de construção de duas visões podem ser interessantes?

Referências Bibliográficas

Asuncion, A. e Newman, D. (2007). UCI machine learning repository. Citado nas páginas 35, 97, e 105.

Ayer, M., Brunk, H. D., EWing, G. M., Reid, W. T., e Silverman, E. (1955). An empirical distribution function for sampling with incomplete information. *Annals of Mathematical Statistics*, 26(4):641–647. Citado na página 77.

Batista, G. E. A. P. A. e Monard, M. C. (2003). An analysis of four missing data treatment methods for supervised learning. *Applied Art. Intell.*, 17(5-6):519–533. Citado na página 106.

Blum, A. e Mitchell, T. (1998). Combining labeled and unlabeled data with CO-TRAINING. In *COLT' 98: Proceedings of the eleventh Annual Conference on Computational Learning Theory*, páginas 92–100, New York, NY, USA. ACM. Citado nas páginas 91, 92, e 97.

Breiman, L., Friedman, J. H., Olshen, R. A., e Stone, C. J. (1984). *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A. Citado na página 19.

Brier, G. (1950). Verification of forecasts expressed in terms of probabilities. *Monthly Weather Review*, 78:1–3. Citado nas páginas 75 e 84.

Chapelle, O., Schölkopf, B., e Zien, A., editors (2006). *Semi-Supervised Learning*. MIT Press, Cambridge, MA. Citado na página 11.

Cohen, I. e Goldszmidt, M. (2004). Properties and benefits of calibrated classifiers. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, páginas 125–136, New York, NY, USA. Springer Verlag. Citado na página 84.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30. Citado na página 82.

Dietterich, T. G. (1990). Machine learning. *Annual Review of Computer Science*, 4:255–306. Citado na página 7.

Fan, W., Davidson, I., Zadrozny, B., e Yu, P. S. (2005). An improved categorization of classifier's sensitivity on sample selection bias. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, páginas 605–608, Washington, DC, USA. IEEE Computer Society. Citado na página 95.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874. Citado nas páginas 1, 37, 41, 55, e 115.

Fawcett, T. e Niculescu-Mizil, A. (2007). PAV and the ROC convex hull. *Machine Learning*, 68(1):97–106. Citado nas páginas 72 e 113.

Ferri, C., Flach, P. A., e Hernández-Orallo, J. (2002). Learning decision trees using the area under the ROC curve. In Sammut, C. e Hoffmann, A. G., editors, *ICML '02: Proceedings of the Nineteenth International Conference*, páginas 139–146. Morgan Kaufmann. Citado nas páginas 19, 20, e 26.

Flach, P. A. (2003). The geometry of ROC space: Understanding machine learning metrics through roc isometrics. In Fawcett, T. e Mishra, N., editors, *ICML '03: Proceedings of the Twentieth International Conference on Machine Learning*, páginas 194–201. AAAI Press. Citado nas páginas 19, 54, e 55.

Flach, P. A. (2007). Putting things in order: On the fundamental role of ranking in classification and probability estimation. In Kok, J. N., Koronacki, J., de Mántaras, R. L., Matwin, S., Mladenic, D., e Skowron, A., editors, *ECML/PKDD '07: 18th European Conference on Machine Learning and the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 4702 of *Lecture Notes in Computer Science*, páginas 2–3. Springer Verlag. Citado nas páginas 4, 58, 112, e 113.

Flach, P. A. e Matsubara, E. T. (2007a). On classification, ranking, and probability estimation. In Raedt, L. D., Dietterich, T. G., Getoor, L., Kersting, K., e Muggleton, S., editors, *Probabilistic, Logical and Relational Learning - A Further Synthesis*, volume 07161 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany. Citado nas páginas 4 e 112.

Flach, P. A. e Matsubara, E. T. (2007b). A simple lexicographic ranker and probability estimator. In Kok, J. N., Koronacki, J., de Mántaras, R. L., Matwin, S., Mladenic, D., e Skowron, A., editors, *ECML '07: Proceedings of 18th European Conference on Machine Learning*, volume 4701 of *Lecture Notes in*

Computer Science, páginas 575–582. Springer Verlag. Citado nas páginas 4, 12, 29, 84, 112, e 113.

Flach, P. A. e Wu, S. (2005). Repairing concavities in ROC curves. In Kaelbling, L. P. e Saffiotti, A., editors, *IJCAI '05: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, páginas 702–707. Professional Book Center. Citado na página 45.

Fürnkranz, J. e Flach, P. A. (2005). ROC 'n' rule learning: towards a better understanding of covering algorithms. *Machine Learning*, 58(1):39–77. Citado na página 46.

Goldman, S. e Zhou, Y. (2000). Enhancing supervised learning with unlabeled data. In *ICML '00: Proceedings of the 17th International Conference on Machine Learning*, páginas 327–334. Morgan Kaufmann. Citado na página 115.

Grätzer, G. (1971). *Lattice Theory. First Concepts and Distributive Lattices*. Freeman and Co., San Francisco. Citado na página 25.

Hanley, J. A. e McNeil, B. J. (1982). The meaning and the use of the area unde a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36. Citado na página 57.

Heer, J., Card, S. K., e Landay, J. A. (2005). PREFUSE: a toolkit for interactive information visualization. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, páginas 421–430, New York, NY, USA. ACM. Citado na página 64.

Kearns, M. e Mansour, Y. (1996). On the boosting ability of top-down decision tree learning algorithms. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, páginas 459–468, New York, NY, USA. ACM. Citado na página 19.

Kuncheva, L. I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience. Citado na página 9.

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2(2):159–165. Citado na página 96.

Maletzke, A. G., Lee, H. D., Chung, W. F., Matsubara, E. T., Coy, C. S. R., Fagundes, J. S., e Góes, J. (2006). Uma metodologia para auxiliar no processo de mapeamento de formulários médicos para bases de dados estruturadas. In *CBIS '06: Anais do X Congresso Brasileiro de Informática em Saúde*, páginas 1–7, Florianópolis, Santa Catarina, Brasil. cdrom. Citado na página 114.

Maletzke, A. G., Lee, H. D., Zalewski, W., Edson T. Matsubara, R. F. V., Coy, C. S. R., ao José Fagundes, J., Góes, J. R. N., e Chung, W. F. (2007a). Mapeamento de informações médicas descritas em formulários para bases de dados estruturadas. In *WIM '07: VII Workshop de Informática Médica*, páginas 49–58, Porto de Galinhas, PE, Brasil. Citado na página 114.

Maletzke, A. G., Lee, H. D., Zalewski, W., Matsubara, E. T., Voltolini, R. F., Coy, C. S. R., ao José Fagundes, J., Góes, J. R. N., e Wu, F. C. (2007b). Transferência de formulários de informações médicas para bases de dados estruturadas: Estudo de caso. In *SIS '07: Anais do Simpósio de Informática y Salud das 36 Jornadas Argentinas de Informática*, páginas 11–20, Mar del Plata, Argentina. Citado na página 114.

Maletzke, A. G., Lee, H. D., Zalewski, W., Matsubara, E. T., Voltolini, R. F., Coy, C. S. R., Góes, J. R. N., ao José Fagundes, J., e Wu, F. C. (2007c). Construção de bases de dados apoiada por métodos computacionais a partir de formulários médicos: Estudo de caso em doença de crohn. In *Coloprocto '07: Anais do 56 Congresso Brasileiro de Coloproctologia*, páginas 39–40, Curitiba, Paraná. Citado na página 114.

Markman, A. B. (1998). *Knowledge Representation*. Lawrence Erlbaum Associates. Citado na página 8.

Martins, C. A., Monard, M. C., e Matsubara, E. T. (2003a). Reducing the dimensionality of bag-of-words text representation used by learning algorithms. In *AIA '03: Proceedings of 3rd IASTED International Conference on Artificial Intelligence and Applications*, páginas 228–233, Benalmádena, Espanha. Acta Press. Citado na página 114.

Martins, C. A., Monard, M. C., e Matsubara, E. T. (2003b). Uma ferramenta computacional para auxiliar no pré-processamento de textos. In de Oliveira Anido, R. e Masiero, P. C., editors, *ENIA '03 :Anais do IV Encontro Nacional de Inteligência Artificial*, volume VII, páginas 1–6, Campinas, SP. SBC - Campinas. Citado na página 114.

Martins, C. A., Monard, M. C., e Matsubara, E. T. (2004). Uma metodologia para auxiliar na seleção de atributos relevantes usados por algoritmos de aprendizado no processo de classificação de textos. In Solar, M., Fernández-Baca, D., e Cuadros-Vargas, E., editors, *CLEI '04: Proceedings of 30th Conferencia Latinoamericana de Informática*, páginas 21–32. Sociedad Peruana de Computación. ISBN 9972-9876-2-0. Citado na página 114.

Matsubara, E. T. (2004). O algoritmo de aprendizado semi-supervisionado CO-TRAINING e sua aplicação na rotulação de documentos. Dissertação de

Mestrado, ICMC-USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-19082004-092311/. Citado na página 114.

Matsubara, E. T., Martins, C., e Monard, M. C. (2003). PreText: uma ferramenta para pré-processamentos de textos utilizando a abordagem bag-of-words. Relatório Técnico 209, ICMC-USP. Citado nas páginas 96 e 114.

Matsubara, E. T. e Monard, M. C. (2004a). Análise experimental do algoritmo de aprendizado de máquina semi-supervisionado CO-TRAINING. Relatório Técnico 235, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_235.pdf. Citado na página 114.

Matsubara, E. T. e Monard, M. C. (2004b). Projeto e implementação do algoritmo de aprendizado de máquina semi-supervisionado CO-TRAINING. Relatório Técnico 229, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_229.pdf. Citado nas páginas 99 e 114.

Matsubara, E. T. e Monard, M. C. (2006). O algoritmo de aprendizado semi-supervisionado CO-TRAINING e sua aplicação na rotulação de documentos. V Concurso de Teses e Dissertações de IA (CTDIA 2006)(3ro Colocado). Citado na página 114.

Matsubara, E. T., Monard, M. C., e Batista, G. E. A. P. A. (2004). Aprendizado semi-supervisionado multi-visão para a classificação de bases de texto. In *WAI '04: Workshop on Artificial Intelligence*, páginas 1–9, Arica-Chile. Jornadas Chilenas de Computacion. Citado na página 114.

Matsubara, E. T., Monard, M. C., e Batista, G. E. A. P. A. (2005a). Multi-view semi-supervised learning: An approach to obtain different views from text datasets. In *Advances in Logic Based Intelligent Systems*, volume 132, páginas 97–104. IOS Press. Citado na página 114.

Matsubara, E. T., Monard, M. C., e Batista, G. E. A. P. A. (2005b). Utilizando algoritmos de aprendizado semi-supervisionados multi-visão como rotuladores de texto. In *TIL '05: Anais do Workshop em Tecnologia da Informação de da Linguagem Humana*, páginas 2108–2117. Citado na página 114.

Matsubara, E. T., Monard, M. C., e Prati, R. C. (2006). On the class-distribution labelling-step sensitivity of CO-TRAINING. In *IFIPAI '06: World Computer Congress - TC12 Artificial Intelligence in Theory and Practice*, páginas 199–208. Springer Verlag. Citado nas páginas 4, 96, e 114.

Matsubara, E. T., Monard, M. C., e Prati, R. C. (2007). Exploring unclassified texts using multi-view semi-supervised learning. In do Prado, H. A. e

Ferneda, E., editors, *Emerging Technologies of Text Mining: Techniques and Applications*, páginas 139–161. IDEA group. Citado na página 114.

Matsubara, E. T., Prati, R. C., Batista, G. E. A. P. A., e Monard, M. C. (2008). Missing value imputation using a semi-supervised rank aggregation approach. In *SBIA '08: Proceedings of Brazilian Symposium on Artificial Intelligence*, Lecture Notes in Computer Science. Springer Verlag. (aceito para publicação). Citado nas páginas 4 e 114.

Melo, V., Secato, M., e Lopes, A. A. (2003). Automatic extraction and identification of bibliographical information from scientific articles (in Portuguese). In *IV Workshop on Advances and Trend in AI*, páginas 1–10, Chile. Citado na página 97.

Michalski, R. S., Carbonell, J. G., e Mitchell, T. M. (1986). *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann. Citado na página 8.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York. Citado nas páginas 11, 12, e 20.

Niculescu-Mizil, A. e Caruana, R. (2005). Predicting good probabilities with supervised learning. In *ICML '05: Proceedings of the 22nd international Conference on Machine learning*, páginas 625–632, New York, NY, USA. ACM. Citado na página 77.

Platt, J. C. (1999). *Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods*, chapter Probabilistic SV Machines, páginas 61–74. MIT Press, Cambridge, MA, USA. Citado na página 76.

Prati, R. C. e Flach, P. A. (2005). Roccer: An algorithm for rule learning based on roc analysis. In *IJCAI '05:Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, páginas 823–828. Citado na página 116.

Provost, F. e Domingos, P. (2003). Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215. Citado nas páginas 20, 26, 37, e 84.

Provost, F. J. e Fawcett, T. (1997). Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *KDD '97: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, páginas 43–48. Citado na página 2.

Provost, F. J. e Fawcett, T. (1998). Robust classification systems for imprecise environments. In *AAAI/IAAI '98 : Proceedings of Tenth Innovative Applications of Artificial Intelligence Conference*, páginas 706–713. Citado na página 2.

Provost, F. J. e Fawcett, T. (2001). Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231. Citado nas páginas 39, 42, 54, 55, e 56.

Russell, S. J. e Norvig, P. (2002). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall. Citado nas páginas 12 e 28.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423. Citado na página 15.

Spackman, K. A. (1989). Signal detection theory: valuable tools for evaluating inductive learning. In *Proceedings of the Sixth International Workshop on Machine learning*, páginas 160–163, San Francisco, CA, USA. Morgan Kaufmann. Citado na página 41.

Vapnik, V. (1998). *Statistical Learning theory*. Wiley. Citado na página 11.

Weiss, G. M. e Provost, F. J. (2003). Learning when training data are costly: The effect of classdistribution on tree induction. *J. Artif. Intell. Res. (JAIR)*, 19:315–354. Citado na página 95.

Witten, I. H. e Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann. Citado nas páginas 12, 36, 64, e 105.

Zadrozny, B. (2004). Learning and evaluating classifiers under sample selection bias. In Brodley, C. E., editor, *ICML '04: Proceedings of the Twenty First International Conference on Machine Learning*, páginas 114–121. ACM. Citado na página 95.

Zadrozny, B. e Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *KDD '02: Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, páginas 694–699, New York, NY, USA. ACM. Citado nas páginas 4, 76, 77, e 115.

Zhu, X. (2005). Semi-supervised learning literature survey. Relatório Técnico 1530, Computer Sciences, University of Wisconsin-Madison. http://www.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf. Citado na página 92.

Zou, K. H. (2007). Receiver operating characteristic (ROC) literature research. http://www.spl.harvard.edu/archive/spl-pre2007/pages/ppl/zou/roc.html, acessado em 06/2008. Citado na página 41.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo