UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Igor Giusti da Silva

Implantação e Avaliação de um Método de Ranking Baseado em Classificação

Niterói
2012

Igor Giusti da Silva

Implantação e Avaliação de um Método de Ranking Baseado em Classificação

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação.

Orientadora: Bianca Zadrozny

Aprovado em Fevereiro de 2012

BANCA EXAMINADORA

Bianca Zadrozny

Rafael Barros Pereira

Simone de Lima Martins

Niterói

2012

RESUMO

Esta monografia tem por objetivo implantar e avaliar a técnica de *ranking reduzido a classificação binária* descrita no artigo (1). Tal técnica pertence à área de estudo chamada *Learning to Rank* inserida na macroárea de Aprendizado de Máquina no campo de Inteligência Artificial. O problema emerge de situações reais como: ordenar documentos de acordo com uma busca e escolha de produtos a serem recomendados em um *e-commerce*.

Algumas otimizações, como amostragem e votação, foram sugeridas e também implantadas e avaliadas. Todos os algoritmos foram programados sobre o *workbench WEKA* (*Waikato Environment for Knowledge Analysis*) em português, ambiente para análise de conhecimento de Waikato. Essa plataforma oferece diversos recursos para aprendizado de máquina.

Palavras Chave:

IA, Aprendizado de máquina, Learning to Ranking, Ranking, Classificação

ABSTRACT

That monograph has for goal to implement and evaluate a reduction from ranking to classification as proposed in (1). Such technique belongs to an area named Learning to Rank, inserted in Machine Learning study branch of Artificial Inteligence field. The problem arises from real situations as: document ordering according to some query and selection of products to recommend in an e-commerce.

Some optimizations, as sampling and voting, were suggested and also implemented and evaluated. All algorithms were programmed on top of WEKA (Waikato Environment for Knowledge Analysis) workbench. This platform provides a large set of resources for machine learning.

Keywords:

AI, Machine Learning, Learning to Rank, Ranking, Classification

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Segundo o dicionário Cambridge, a palavra *rank* significa uma posição particular, mais alta ou mais baixa que outras. Por exemplo, a seleção brasileira de futebol masculino ocupa uma posição na classificação das seleções da FIFA.

A posição ocupada pela seleção brasileira somada às posições ocupadas pelas outras seleções constituem o *ranking* das seleções de futebol masculino da FIFA. Pode-se citar como outros exemplos de *ranking*: o *ranking* do IDH (Índice de Desenvolvimento Humano) dos países, o *ranking* da competição da Yahoo! [1] sobre *Learning to rank*, entre outros.

Todos os *rankings* citados são ordenados de acordo com um critério. Por exemplo, para a definição do *ranking* de seleções da FIFA, calcula-se o total de pontos para cada seleção vinculada à FIFA e ordenam-se as seleções com base nas pontuações. As seleções com maior pontuação ocupam o topo do *ranking*, enquanto as seleções com menor pontuação ocupam a base.

A pontuação total de uma seleção em um período de quatro anos é definida por meio da soma da média de pontos ganhos em partidas disputadas nos 12 últimos meses e da média de pontos ganhos em partidas que ocorreram há mais de 12 meses, a qual se deprecia anualmente. Há uma fórmula matemática para definir a pontuação de uma equipe em uma partida.

Podemos afirmar que as fórmulas utilizadas pela FIFA para definir as pontuações das seleções constituem um modelo que determina uma ordem parcial do conjunto de seleções FIFA. Dizer que as fórmulas utilizadas pela FIFA constituem um modelo caracteriza a tarefa de ordenação das seleções FIFA como computável, ou seja, o modelo pode ser escrito em forma de algoritmo e avaliado, junto aos dados necessários, por um computador.

Outra característica do modelo FIFA é que ele foi definido por funcionários da instituição, ou seja, o modelo é fruto do intelecto humano. Apesar disso, seria possível o aprendizado automático de um modelo que ordenasse o conjunto de seleções vinculadas à FIFA.

[1] http://learningtorankchallenge.yahoo.com/leaderboard.php

As características de computabilidade e possibilidade de aprendizado automático, observadas no exemplo do *ranking* da FIFA, podem ser encontradas em diversos outros tipos de ordenações.

Talvez não haja muita valia em automatizar o aprendizado de um modelo para ordenar as seleções vinculadas à FIFA; o modelo definido pela FIFA já é bastante adequado. Porém, para uma grande quantidade de aplicações, aprender modelos que ordenem uma base de dados pode ser útil, ajudando a lidar com problemas como quantidade de dados e variáveis a serem consideradas.

Um problema beneficiado pelo aprendizado de um modelo de ordenação seria: Dado um conjunto de documentos em que cada documento pode receber um rótulo, atribuir uma posicao a cada documento do conjunto, tendo como insumo pares documento-rótulo ou uma relação de ordem total ou parcial entre os documentos.

O problema de ordenação de documentos é um dos problemas de interesse de uma área conhecida como *Learning to Rank* que tem atraído a atenção de muitos pesquisadores nos últimos anos. Já conta com sites especializados no assunto[2], competições[3] além do apoio das gigantes de TI.

Essa área de pesquisa está inserida no contexto de *inteligência artificial* e compartilha conhecimentos com áreas como *Aprendizado de máquina*, *Recuperação de informação* e *processamento de linguagem natural*.

Estudaremos a implantação e avaliação de uma técnica de aprendizado para ordenação de um conjunto no qual os elementos são passíveis de rotulação. Tal técnica reduz o problema de ordenação a uma problema de classificação de acordo com o descrito em (1).

A principal contribuição desse estudo é a implantação, avaliação e comparação da técnica, uma vez que a técnica havia sido especificada apenas teoricamente. Para a implantação do algoritmo de *ranking* foi escolhido o *workbench WEKA*[4], descrito no livro (2). *WEKA* fornece vários recursos para *aprendizado de máquina*.

O restante desta monografia está organizado da seguinte forma: O capítulo 2 faz uma introdução aos conceitos necessários para o entendimento do algoritmo *Ranking*; o capítulo 3 mostra os problemas e soluções que derivaram da implantação do *Ranking*; o capítulo 4 descreve a estratégia usada para avaliar o algoritmo e os resultados obtidos; e o capítulo 5 apresenta as principais descobertas obtidas nesse estudo.

[2] http://research.microsoft.com/en-us/um/beijing/projects/letor/
[3] http://learningtorankchallenge.yahoo.com/
[4] http://www.cs.waikato.ac.nz/ml/weka/

2 *RANKING*: NOÇÕES PRELIMINARES

Na área de *aprendizado de máquina*, classificação é tarefa de atribuir rótulos a cada elemento de um dado conjunto tendo como entrada pares elemento-rótulo. Comparativamente, *ranking* é a tarefa de atribuir posições a cada elemento de um dado conjunto tendo como entrada pares elemento-rótulo, uma relação de ordem parcial entre os elementos, ou uma relação de ordem total entre os elementos (3).

Embora desempenhem tarefas diferentes com saídas diferentes, algoritmos de classificação e de *ranking* podem compartilhar o mesmo tipo de entrada: pares elemento-rótulo. Isso é uma evidência de que se pode usar um algoritmo de classificação para compor um algoritmo de *ranking*.

De fato, a técnica de *ranking* descrita em (1) propõe envolver um algoritmo de classificação com etapas de pré-processamento e pós-processamento de forma que o produto final seja uma ordenação. Nesse estudo, essa técnica é chamada de *ranking reduzido a classificação*.

2.1 INTRODUÇÃO AO PROBLEMA

Dado um conjunto composto por elementos aos quais é possível atribuir um rótulo de valor 0 ou 1, deseja-se encontrar uma permutação dos elementos de maneira que os elementos que apresentem maior chance de receber o rótulo 0 devem preceder os com maior chance de receber o rótulo 1. Cada elemento é composto por um conjunto de características e a chance de um elemento receber o rótulo 0 ou o rótulo 1 deve ser calculada com base nessas características.

Substituindo-se a palavra rótulo pela palavra classe no enunciado acima, percebe-se que o problema de ordenação pode ser tratado como um problema de classificação. Mais especificamente, um problema de classificação binária, pois os valores de classe possíveis são 0 e 1.

Um algoritmo de aprendizado para classificação recebe como entrada um conjunto em que cada elemento possui uma classe assinalada e, partindo dessa entrada, deve aprender um classi-

ficador capaz de atribuir uma classe a qualquer elemento. A etapa em que o aprendizado ocorre é chamada de treinamento.

O treinamento de um classificador é um aprendizado supervisionado, pois cada elemento submetido à etapa de treinamento está vinculado à sua classe verdadeira. Dessa forma, pode-se medir a qualidade do classificador aprendido usando elementos com classes conhecidas, porém suprimindo-as quando os elementos forem submetidos a classificação.

O produto da classificação de um elemento é uma previsão do valor da classe à qual aquele elemento pertence. Sobre a previsão é possível obter uma quantidade de certeza, ou probabilidade, do elemento pertencer à classe prevista. Nesso caso, por se tratar de um problema de classificação binária, se a probabilidade de um elemento pertencer à classe 0 é p, então a probabilidade dele pertencer à classe 1 é $1 - p$.

Uma solução trivial para o problema de ordenação é utilizar a probabilidade das previsões de um classificador como uma relação de ordem no conjunto a ser ordenado. De posse da previsão das classes dos elementos de um dado conjunto, basta ordená-los de forma crescente em função da probabilidade de cada um pertencer à classe 0; os mais prováveis ocupam o topo, os menos prováveis ocupam o fundo.

Segundo (1), esse método para ordenação dos elementos pode resultar em um erro teórico máximo muito alto, dependendo da performance do classificador. A técnica de *ranking reduzido a classificação*, proposta no mesmo artigo, reduz o erro teórico máximo envolvendo o treinamento do classificador com uma etapa de pré-processamento e a avaliação com uma etapa de pós-processamento.

Na técnica de *ranking reduzido a classificação*, o treinamento do classificador ocorre pelo algoritmo *AUC-Train* após uma etapa inicial em que se aplica uma transformação nos exemplos da base de treinamento. Após o treinamento, a definição da ordem das instâncias na base de avaliação é feita pelo algoritmo *tournament* — torneio em português — que usa as previsões do classificador treinado para tal.

2.2 DEFINIÇÕES

Como dito anteriormente, o treinamento e avaliação de um classificador é um processo que manipula conjuntos compostos por elementos. Definições mais rigorosas são necessárias para a descrição da implantação da técnica de *ranking reduzido a classificação* no capítulo 3. As definições abaixo foram feitas levando-se em consideração o funcionamento da ferramenta

WEKA e os requisitos necessários para se modelar a solução de *ranking reduzido a classificação* exposta em (1).

- Uma base B é um conjunto de instâncias com cardinalidade n.

$$B = \{i_1, ..., i_n\} \qquad |B| = n$$

- Uma instância I é uma tupla $\langle A, C \rangle$ na qual A é um vetor de atributos com cardinalidade m e C é a classe da instância e pode valer 0 ou 1.

$$I = (A, C) \qquad A = \{a_1, ..., a_m\} \qquad C = x \mid x \in \{0, 1\}$$

Pode-se acessar os atributos de uma instância I através de $I(A)$ e a classe através de $I(C)$. Nos algortimos do próximo capítulo, quando duas instâncias forem combinadas, os vetores de atributos serão concatenados, isso é feito através do operador binário de concatenação $||$.

Não será necessária a definição do conceito de atributo. Uma vez que os atributos são manipulados pelo classificador e o funcionamento dos classificadores nas soluções aqui propostas é uma caixa preta, os atributos são transparentes para os propósitos desse trabalho.

A tabela 1 aprensenta um exemplo de base extraido do livro (2). Essa base indica se há condições para a prática de um esporte de acordo com medições meteorológicas. Um exemplo dessa base apresenta um conjunto de medições meteorológicas e, de acordo com essas medições, a classe do exemplo pode ser *sim* ou *não*.

panorama	temperatura	humidade	ventoso	adequado para jogo
ensolarado	quente	alta	falso	não
ensolarado	quente	alta	verdadeiro	não
nublado	quente	alta	falso	sim
chuvoso	branda	alta	falso	sim
chuvoso	frio	normal	falso	sim
chuvoso	frio	normal	verdadeiro	não
nublado	frio	normal	verdadeiro	sim
ensolarado	branda	alta	falso	não
ensolarado	frio	normal	falso	sim
chuvoso	branda	normal	falso	sim
ensolarado	branda	normal	verdadeiro	sim
nublado	branda	alta	verdadeiro	sim
nublado	quente	normal	falso	sim
chuvoso	branda	alta	verdadeiro	não

Tabela 1: Base de dados de tempo.

A primeira linha nomeia os quatro atributos da base e a classe (adequado para jogo), as linhas seguintes representam as instâncias da base. Nota-se que a classe recebe valores *sim* e *não*, apesar de serem valores diferentes de 0 e 1, como dito na definição, ainda se trata de uma base com classe binária.

Escrevendo a primeira instância da base na notação definida acima, tem-se: ((ensolarado, quente, alta, falso), não). O primeiro elemento da tupla é o vetor de atributos e o segundo elemento é a classe da instância.

2.3 MEDIDAS DE DESEMPENHO

Geralmente, a medida de eficiência mais utilizada para classificação é a acurácia: uma razão entre o número de instâncias corretamente classificadas sobre o número total de instâncias no conjunto de avaliação.

O erro decorrente de uma classificação afeta a acurácia de maneira linear. Como, para ordenações, a quantidade de acertos não é tão relevante quanto a posição das instâncias ordenadas, propõe-se outro tipo de medida para avaliação do *ranking* aprendido.

Uma medida comum de avaliação para algoritmos de *ranking* é a área sobre a curva *ROC (Receiver Operating Characteristic)*, comumente chamada de *AUC (Area Under the Curve)*.

A perda, $1 - AUC$, associada a essa medida é calculada pelo número de instâncias, normalizado pela quantidade de 0s vezes a quantidade de 1s, que necessitam ser trocadas para um *ranking* perfeito.

Uma ordenação é perfeita quando todas as instâncias com classe 0 precedem as com classe 1, nesse caso a perda na *AUC* é 0. No pior caso, em que todos os 1s precedem os 0s, a perda na *AUC* é 1.

Comparativamente, um erro de classificação pode ter maior influência na medida *AUC* que na acurácia afetando consideravelmente um *ranking*, mas não a classificação. A causa disso é a *AUC* considerar a relação entre as instâncias ordenadas, enquanto a acurácia considera apenas erros e acertos pontualmente. Abaixo ilustramos, através de um exemplo, uma relação entre essas medidas que comprova o intuído sobre erros na classificação.

Na tabela 2, temos quatorze instâncias ordenadas em um *ranking* com os atributos, as classes e as previsões dadas por um classificador. Podemos perceber que o classificador errou apenas a classe da primeira instância. Calculando a acurácia, temos treze acertos em quatorze possíveis, o que equivale a aproximadamente 93% de acerto.

panorama	temperatura	humidade	ventoso	classe	previsão
ensolarado	quente	alta	falso	não	sim
nublado	quente	alta	falso	sim	sim
chuvoso	branda	alta	falso	sim	sim
chuvoso	frio	normal	falso	sim	sim
nublado	frio	normal	verdadeiro	sim	sim
ensolarado	frio	normal	falso	sim	sim
chuvoso	branda	normal	falso	sim	sim
ensolarado	branda	normal	verdadeiro	sim	sim
nublado	branda	alta	verdadeiro	sim	sim
nublado	quente	normal	falso	sim	sim
ensolarado	quente	alta	verdadeiro	não	não
chuvoso	frio	normal	verdadeiro	não	não
ensolarado	branda	alta	falso	não	não
chuvoso	branda	alta	verdadeiro	não	não

Tabela 2: Exemplo de *ranking* e classificação na base weather.

Calculando a perda da AUC considerando como base a classe *sim*, a primeira instância precisa retroceder nove posições para uma ordenação perfeita, normalizando pelo número de *não*s vezes o número de *sim*s, temos $(1-AUC) = 9 \div (5*9) = 0,2$, logo a AUC vale 80%. Esse exemplo comprova que a AUC pode sofrer um impacto maior devido a erros de classificação se comparada à acurácia.

De acordo com (1), se um classificador gera um erro de ordem α na acurácia, ao ordenar uma base a partir das probabilidades das previsões desse classificador, o erro teórico máximo na AUC será de $\alpha \cdot n$, onde n é a cardinalidade da base avaliada. Enquanto para o mesmo classificador, a técnica de *ranking reduzido a classificação* apresenta um erro teórico máximo de $\alpha \cdot 2$.

O erro na AUC se intensifica a medida que o desbalanceamento de classes do conjunto usado no treinamento aumenta, pois quanto mais desbalanceadas as classes, mais provável que o classificador resultante seja tendencioso para a classe majoritária.

3 *RANKING*: IMPLANTAÇÃO

Como dito anteriormente, a técnica de *ranking reduzido a classificação* aplica uma transformação à base original antes da etapa de treinamento do classificador e modifica a etapa de avaliação a fim de ordenar as instâncias. A transformação da base e o treinamento do classificador compõem um algoritmo de nome *AUC-Train*, transcrito a partir de (1) no algoritmo 1.

Algoritmo 1: AUC-Train

Let $S' = \{\langle (x_1, x_2), 1(y_1 < y_2)\rangle : (x_1, y_1), (x_2, y_2) \in S \text{ and } y_1 \neq y_2\}$;
return $c = A(S')$

A transformação consiste em formar novas instâncias a partir de pares entre instâncias de classes diferentes. A ordem em que as instâncias aparecem no par é relevante; o par $\langle i_\alpha, i_\beta \rangle$ e o par $\langle i_\beta, i_\alpha \rangle$ representam informações diferentes para o aprendizado do classificador. Além disso, as classes das duas instâncias que formam um par são combinadas a fim de gerar uma nova classe para a nova instância.

Essa transformação foi implantada particionando a base de treinamento em dois subconjuntos, um subconjunto S_0 com as instâncias de classe 0 e outro, S_1, com as instâncias de classe 1. Com as duas partições definidas, basta combiná-las em um conjunto de elementos com formato final $\langle (i_\alpha(A), i_\beta(A)), 1(i_\alpha(C) < i_\beta(C))\rangle$.

A *função* 1, aplicada para obter a classe da nova instância, retorna 1, se o resultado da expressão avaliada é verdadeiro, ou 0, se o resultado da expressão é falsa. Como o argumento para a função é $i_\alpha(C) < i_\beta(C)$, se $i_\alpha(C)$ valer 1, a classe da nova instância será 0, se valer 0, a classe da nova instância será 1.

Função 1(*expr*)

avaliacao $\leftarrow 1$;
se $expr = False$ **então**
  avaliacao $\leftarrow 0$;
fim
retorna *avaliacao*

A modificação da etapa de avaliação tem como objetivo usar as previsões do classificador treinado pelo *algoritmo* AUC-Train para gerar o *ranking* da base de avaliação. O algoritmo que efetua a ordenação recebe o nome de *Tournament*, torneio em português. O porquê do nome é devido ao algoritmo se assemelhar muito a um torneio em que muitos competidores se enfrentam, formando um *ranking* de acordo com suas performances.

Continuando a analogia com uma competição, a performance de cada instância na ordenação é medida como a quantidade de ”vitórias” que essa instância obteve diante das outras instâncias na base a ser ordenada. Para gerar o *ranking*, basta promover o embate entre todas as instâncias na base e ordenar pelo número de ”vitórias”.

Como desejamos ordenar as instâncias de forma que as que tenham maior chance de pertencer à classe 0 estejam no topo, a semântica de ”vitoria” é: em um embate entre duas instâncias, a instância vencedora é a que possui a maior chance de pertencer à classe 0. Quem define a instância vitoriosa em um embate é a previsão do classificador binário.

Tecnicamente, um embate entre as instâncias i_α e i_β consiste em criar uma nova instância com formato $\langle (i_\alpha(A), i_\beta(A)) \rangle$ e submetê-la à classificação; se a classe prevista da nova instância for 1, i_α vence, se for 0, i_β vence. Nota-se que o formato da instância a ser classificada é o mesmo que o de uma instância usada no treinamento do classificador, exceto pela ausência de classe. O algoritmo 2 demonstra o processo de ordenação.

Algoritmo 2: Tournament

For $x \in U$, let $deg(x) = |\{x' : c(x,x') = 1, x' \in U\}|$;
Sort U in descending order of deg(x), breaking ties arbitrarily

Uma das vantagens da técnica de *ranking reduzido a classificação* é que o uso de um algoritmo de aprendizado para classificação é transparente. Essa característica possibilitou o uso dos algoritmos de classificação existentes na ferramenta *WEKA*.

O principal ponto negativo é a performance da técnica no que diz respeito a tempo computacional. Os algoritmos 1 *(AUC-Train)* e 2 *(Tournament)* que constituem o cerne da técnica de *ranking reduzido a classificação* possuem elevada complexidade.

Os algoritmos expostos nesse capítulo estão descritos em pseudo-código e em sua maioria se baseiam em conceitos de teoria de conjuntos e vetores; uma vez que as principais estruturas manipuladas, base e instâncias, foram definidas sobre esses conceitos.

Assim como a maioria das técnicas de aprendizado supervisionada, o *ranking reduzido a classificação* pode ser dividido em duas grandes partes distintas: a primeira é o treinamento de um ordenador e a segunda é a ordenação de uma base a partir do ordenador treinado.

Como os algoritmos nas etapas de treinamento e ordenação para a técnica de *ranking reduzido a classificação* são, respectivamente, *AUC-Train* e *Tournament*; a complexidade da técnica será estudada a partir das complexidades desses dois algoritmos separadamente.

3.1 TREINAMENTO: ESTUDO DE COMPLEXIDADE DO ALGORITMO AUC-TRAIN

Considera-se como entrada para o algoritmo *AUC-Train* uma base S com n instâncias das quais k possuem a classe 0 e $n-k$ possuem a classe 1. O algoritmo executa em três grandes etapas:

1. particionar a base de treinamento em duas bases S_0 e S_1; S_0 possui as instâncias de classe 0 e S_1 possui as instâncias de classe 1;
2. combinar as partições S_0 e S_1 de forma que a base de treinamento S' possua todos os pares entre instâncias de classe 1 e de classe 0;
3. aplicar um algoritmo de aprendizado A à base S' obtendo um classificador c como saída.

Em questão de custo computacional, o tempo de execução do algoritmo *AUC-Train* é composto pela soma dos tempos de particionamento da base de treinamento, da aplicação de duas combinações entre as partições e da aplicação do algoritmo de aprendizado.

Por se tratar de uma técnica baseada em metaclassificação, não é possível determinar um custo computacional para a 3^a etapa, pois algoritmos de aprendizado para classificação distintos executam em tempos distintos. Portanto, o estudo da complexidade da técnica *AUC-Train* será baseado na computação extra que as etapas 1 e 2 acrescentam ao aprendizado.

Para o particionamento da base, basta iterar uma vez pela base original incluindo as instâncias de classe 0 no conjunto S_0 e as de classe 1 no conjunto S_1, como mostrado na *função particionar*. Essa etapa executa sempre em tempo $O(n)$.

Após o particionamento da base, gera-se o conjunto S' a partir das combinações entre as partições S_0 e S_1. A *função combinar* é a encarregada de formar novas instâncias a partir da mesclagem das instâncias nas partições S_0 e S_1. Essa função deve ser executada duas vezes a fim de formar todos os pares.

A mesclagem das instâncias consiste em gerar uma nova instância a partir de duas outras instâncias, de acordo com o algoritmo original do *AUC-Train (1)*. A *função mesclar* desempenha esse papel. Essa função executa sempre em tempo $O(1)$.

Função particionar(S)

Entrada: Conjuntos de instâncias S
Saída: Partições de instâncias S_0, S_1
$S_0, S_1 \leftarrow \emptyset$;
para todo $i \in S$ **faça**
  se $i(C) = 0$ **então**
    $S_0 \leftarrow S_0 \cup \{i\}$;
  senão
    $S_1 \leftarrow S_1 \cup \{i\}$;
  fim
fim
retorna S_0, S_1

Função combinar(S_α, S_β)

Entrada: Conjuntos de instâncias S_α e S_β
Saída: Combinacao de S_α com S_β
$S_C \leftarrow \emptyset$;
para todo $\alpha \in S_\alpha$ **faça**
  para todo $\beta \in S_\beta$ **faça**
    $S_C \leftarrow S_C \cup \{mesclar(\alpha, \beta)\}$;
  fim
fim
retorna S_C

Função mesclar(α, β)

Entrada: Instâncias α e β
Saída: Nova instância, mescla de α com β
retorna $\langle \alpha(A) || \beta(A), 1 \cdot (\alpha(C) < \beta(C)) \rangle$

Os casos extremos da *função combinar* ocorrem quando $k = 0$ ou $k = n$. Nesses casos, todas as instâncias pertencem a mesma classe e o algoritmo não conseguiria formar pares para treinar um classificador; logo, esses casos não estarão incluídos em nossos estudos.

O melhor caso ocorre, com tempo computacional de $O(n)$, quando $k = 1$ ou $k = n - 1$. Nesses casos, todas as instâncias pertencem à mesma classe, exceto por uma, dessa forma um dos loops executa apenas uma vez, enquanto o outro executa n vezes.

No caso médio, parte-se da hipótese que todas as entradas são uniformemente prováveis. A probabilidade de uma entrada para um dado k é $\frac{1}{n-1}$ e a quantidade de iterações geradas por essa entrada é $k \cdot (n-k)$. Partindo desses dados, a complexidade do caso médio pode ser calculada pela fórmula:

$$f_{avg}(n) = \sum_{k=1}^{n-1} \frac{k \cdot (n-k)}{n-1}$$

Desenvolvendo o somatório:

$$\begin{aligned}
f_{avg}(n) &= \frac{1}{n-1} \cdot \sum_{k=1}^{n-1} k(n-k) \\
&= \frac{1}{n-1} \cdot \sum_{k=1}^{n-1} kn - k^2 \\
&= \frac{1}{n-1} \cdot \left(\sum_{k=1}^{n-1} kn - \sum_{k=1}^{n-1} k^2 \right) \\
&= \frac{1}{n-1} \cdot \left(n \cdot \sum_{k=1}^{n-1} k - \sum_{k=1}^{n-1} k^2 \right) \\
&= \frac{1}{n-1} \cdot \left(n \cdot \frac{n((n-1)+1)}{2} - \frac{(n-1)((n-1)+1)(2(n-1)+1)}{6} \right) \\
&= \frac{1}{n-1} \cdot \frac{n^3-n}{6} \\
&= \frac{n^2+n}{6}
\end{aligned}$$

Logo, a complexidade do caso médio é igual a $O(f_{avg}) = O(\frac{n^2+n}{6}) = O(n^2)$.

O pior caso acontece quando $k = \frac{n}{2}$, nesse caso a base de treinamento possui metade das instâncias com classe 0 e a outra metade com classe 1. A combinação ocorrerá entre dois conjuntos com $\frac{n}{2}$ instâncias, resultando na complexidade assintótica $O(\frac{n}{2} \cdot \frac{n}{2}) = O(\frac{n^2}{4}) = O(n^2)$.

Caso	Particionar	Combinar	AUC-Train
Melhor	$O(n)$	$O(n)$	$O(n)+O(n) = O(n)$
Médio	$O(n)$	$O(n^2)$	$O(n)+O(n^2) = O(n^2)$
Pior	$O(n)$	$O(n^2)$	$O(n)+O(n^2) = O(n^2)$

Tabela 3: Complexidade do algoritmo AUC-Train

Analisando a tabela 3, percebe-se que o gargalo do algoritmo *AUC-Train* reside na combinação das instâncias. Na prática, uma otimização foi aplicada ao algoritmo final escrito para o *workbench WEKA*: a *função combinar* foi levemente alterada para combinar tanto o par (α, β) quanto o par (β, α); isso elimina a necessidade da aplicação de duas combinações. Apesar da aplicação dessa otimização, a complexidade do algoritmo se mantem inalterada.

3.2 TREINAMENTO: OTIMIZAÇÕES E IMPLANTAÇÃO DO ALGORITMO AUC-TRAIN

Recapitulando, o algoritmo 1 *(AUC-Train)* é composto de três etapas principais: particionamento, com a *função particionar*; combinação das partições, com a *função combinar*; e treinamento do classificador.

De acordo com o estudo de complexidade , verificou-se que o custo computacional adicionado pelas duas primeiras etapas do algoritmo original, principalmente a de combinação, é alto. Duas estratégias, familiares no campo de aprendizado de máquina, são propostas para mitigar esse custo: Amostragem e Votação.

Amostragem consiste em utilizar apenas um subconjunto da base completa a ser usada no treinamento do classificador. O objetivo da estratégia não é reduzir a complexidade intríseca das etapas de particionamento e combinação, mas limitar a quantidade de instâncias a serem combinadas amortizando o que seria o tempo total de geração das novas instâncias. A *função amostragem* ilustra o processo de amostragem utilizado nesse estudo.

Função amostragem(S_α, S_β, p)

Entrada: Partições de instâncias S_α, S_β;
Pares por instância p
Saída: Amostra com pares S
$S \leftarrow \emptyset$;
para todo $\alpha \in S_\alpha$ **faça**
  para todo $\beta \in Ss_\beta \; tal \; que \; Ss_\beta \subseteq S_\beta \wedge |Ss_\beta| = p$ **faça**
    $S \leftarrow S \cup \{mesclar(\alpha, \beta)\}$;
  fim
fim
retorna S

Votação consiste em treinar vários classificadores em vez de apenas um. Para avaliar uma instância, basta submetê-la aos classificadores e coletar as previsões. A classe que mais figurar entre as previsões é então a classe da instância.

Não há sentido em executar uma votação em que os classificadores foram aprendidos a partir da mesma base de treinamento. Isso resultaria em classificadores iguais e, portanto, seus votos também seriam iguais. Esse panorama só deixaria de ocorrer se o próprio algoritmo de aprendizado inserisse alguma incerteza durante o treinamento.

Apesar de o uso da amostragem beneficiar o tempo de execução, o classificador resultante não será treinado com todas as informações disponíveis, o que pode causar um efeito

de intensificação dos erros de classificação. Com a votação, embora o treinamento de diversos classificadores resulte em acréscimo de tempo para o algoritmo, geralmente o resultado da classificação possuirá menos erros.

Há, assim, uma complementaridade entre as duas estratégias escolhidas para otimização do algoritmo 1 *(AUC-Train)*. Enquanto uma preza por melhorar o tempo de execução, o outra preza por dar maior segurança na classificação. Basta combiná-las para balancear o compromisso entre tempo de execução e redução dos erros de classificação.

De posse das informações necessárias para a implantação do algoritmo 1 *(AUC-Train)* e de suas otimizações, a etapa de treinamento do *Ranking* está exposta no algoritmo 3.

Algoritmo 3: Treinamento

Entrada: Base de treinamento S;
Algoritmo de aprendizado para classificação A;
Quantidade de classificadores a serem treinados n;
Pares por instância p
Saída: Conjunto de classificadores C
$C \leftarrow \emptyset$;
$S_0, S_1 \leftarrow particionar(S)$;
se $i = 1$ *e* $p = todas$ **então**
    $S' \leftarrow combinar(S_0, S_1)$;
    $S' \leftarrow combinar(S_1, S_0)$;
    $C \leftarrow \{A(S')\}$;
fim
senão se $i > 1$ *e* $0 < p \leq min(|S_0|, |S_1|)$ **então**
    para $i \leftarrow 1; i \rightarrow n$ **faça**
        $S' \leftarrow amostragem(S_0, S_1, p)$;
        $S' \leftarrow S' \cup amostragem(S_1, S_0, p)$;
        $C \leftarrow C \cup \{A(S')\}$;
    fim
fim
retorna C

O parâmetro de pares por instância indica a quantidade de pares que uma instância i deve formar como primeiro item da combinação entre i e β onde β é qualquer instância de classe antagônica à classe de i. O parâmetro n constitui o número de classificadores a serem treinados para votação. O valor de retorno é sempre um conjunto de classificadores.

O algoritmo 3 atende, através da manipulação dos parâmetros n e p, tanto ao propósito do algoritmo 1 *(AUC-Train)* quanto às otimizações propostas para esse. Para executar o algoritmo original, basta definir n como 1 e p como $todas$. Para executar o algoritmo com amostragens e votação, basta definir n com um valor maior que 1 e p com um valor entre 1 e o número de

instâncias da classe minoritária.

3.3 *ORDENAÇÃO: ESTUDO DE COMPLEXIDADE DO ALGORITMO* TOURNAMENT

Após a etapa de treinamento, a técnica de *ranking reduzido a classificação* procede à etapa de ordenação. Nessa etapa, o classificador aprendido na etapa de treinamento é utilizado para ordenar um conjunto de instâncias como descrito no algoritmo 2 *(Tournament)*.

O algoritmo 2 *(Tournament)* recebe como entrada um classificador c aprendido durante a etapa de treinamento e uma base B composta de n instâncias sem classes assinaladas. Esse algoritmo executa em duas grandes etapas:

1. Obter a pontuação para todas as instâncias na base B;
2. Ordenar as instâncias na base B.

Partindo dessa divisão, o tempo computacional do algoritmo é a soma dos tempos das etapas de obtenção das pontuações e de ordenação. A etapa de obtenção das pontuações é bem definida no algoritmo 2 *(Tournament)*, enquanto, na etapa de ordenação, qualquer algoritmo de ordenação pode ser usado: *mergesort* e *quicksort*, para citar alguns.

A parte do algoritmo que calcula as pontuações envolve uma chamada à *função votacao*, que invoca o classificador base, com dois *loops*. Portanto, o tempo de execução da etepa de obtenção da pontuação é composto pela quantidade de execuções dos loops multiplicada pelo tempo de execução da chamada à *votacao*.

Não é possível dizer exatamente qual é o limite superior de tempo de execução para o cálculo de todas as pontuações, pois classificadores distintos executam a classificação de um elemento em tempos distintos. Uma alternativa é, em vez de medir o tempo computacional, medir o limite superior de chamadas à *função votacao*, já que nela reside a incerteza da medida.

O pior caso, o caso médio e o melhor caso apresentam o mesmo limite superior de chamadas à *função votacao*, pois é sempre necessário promover o embate de todas as instâncias. Como a chamada à função está englobada por dois loops aninhados, isso resulta em um limite de chamadas de ordem $O(n^2)$. Dessa forma, a única afirmação que pode ser feita sobre o tempo de execução da etapa de pontuação é que ele pode ser maior ou igual a $O(n^2)$.

Como o pior tempo de execução observado para algoritmos de ordenação é da ordem de $O(n^2)$, o gargalo da etapa de ordenação reside na necessidade de se gerar todos os pares a partir

das instâncias no conjunto a ser ordenado. Qualquer tentativa de otimização desse processo deve levar em consideração essa observação.

3.4 ORDENAÇÃO: OTIMIZAÇÕES E IMPLANTAÇÃO DO ALGORITMO TOURNAMENT

Vale destacar que a otimização de votação, sugerida para a etapa de treinamento, tem extensões na etapa de ordenação. Para que a estratégia de votação funcione, o algoritmo de ordenação precisa utilizar a *função votacao*. Essa iterará sobre o conjunto de classificadores produzidos na etapa de treinamento, determinando a classe majoritária na votação.

Função votacao(C, i)

Entrada: Conjunto de classificadores C;
Instância a ser classificada i;
Saída: Previsão da classe de i
$classe \leftarrow 0$;
$zero \leftarrow 0$;
para todo $c \in C$ **faça**
  se $c(i) = 0$ **então**
    $zero \leftarrow zero + 1$;
  senão
    $zero \leftarrow zero - 1$;
  fim
fim
se $zero < 0$ **então**
  $classe \leftarrow 1$;
fim
retorna $classe$

De acordo com o exposto até agora sobre a etapa de ordenação, essa pode ser descrita como no algoritmo 4. Devido às especificações expostas, o algoritmo de ordenação é mais monolítico que o algoritmo de treinamento, que pode ser dividido em funções menores.

Como visto na seção anterior, o custo computacional da etapa de ordenação é composto pelas somas dos custos da obtenção das pontuações e da ordenação. A maior parte do custo é devida ao processo de obtenção das pontuações, de complexidade maior ou igual a $O(n^2)$. E com as pontuações obtidas, ainda é executado um algoritmo de ordenação.

Uma otimização possível, segundo (4), é utilizar o classificador treinado como uma relação de ordem a fim de estabelecer uma precedência entre as instâncias. Assim, seria possível utilizar o classificador diretamente no algoritmo de ordenação, em vez de coletar a pontuação de todas as instâncias.

Algoritmo 4: Ordenação

Entrada: Conjunto de classificadores C;
Base com instâncias de classe desconhecida S;
Mapeamento das instâncias de B com suas pontuações *pontuacao*
Saída: *Ranking* da base B
para todo $\alpha \in B$ **faça**
  para todo $\beta \in B \wedge \beta \neq \alpha$ **faça**
    $i \leftarrow \langle \alpha(A) || \beta(A) \rangle$;
    $classe \leftarrow votacao(C, i)$
    se $classe = 1$ **então**
      $pontuacao[\alpha] \leftarrow pontuacao[\alpha] + 1$
    senão
      $pontuacao[\beta] \leftarrow pontuacao[\beta] + 1$
    fim
  fim
fim
ordenar B com base em *pontuacao*

Essa otimização eliminaria a necessidade da etapa de pontuação. O artigo sugere que o classificador seja usado, em conjunto com o algoritmo de *quicksort*, como uma operação de comparação entre duas instâncias. A comparação ocorre no momento do particionamento do conjunto em que ocorre o reposicionamento do elemento pivot.

Combinado diretamente com o classificador, o algoritmo de ordenação quicksort ainda implica um limite superior de chamadas à *função votacao* de $O(n^2)$ no pior caso, porém reduz o limite para $O(n \cdot log(n))$ no caso médio e no melhor caso.

Ainda segundo (4), uma consequência dessa otimização é tornar viável o uso da técnica de *ranking reduzido a classificação* em sistemas comerciais, tais como: motores de busca e sistemas de recomendações.

Embora a ordenação por quicksort tenha sido implantada na solução para o *workbench WEKA*, não houve nenhuma avaliação empírica dessa estratégia, como será visto no próximo capítulo.

4 *RANKING*: RESULTADOS E AVALIAÇÃO

De acordo com o artigo (1) e como dito no capítulo 2, um classificador que apresente um erro α na acurácia, pode apresentar um erro teórico máximo de $\alpha \cdot n$, onde n é a cardinalidade da base a ser ordenada. A técnica de *ranking reduzido a classificação* diminui esse limite para $\alpha \cdot 2$ usando o mesmo classificador com erro α na acurácia.

Quanto maior o desbalanceamento entre as classes da base a ser ordenada, maior a chance de intensificação do erro na ordenação. Partindo desse princípio, foi montada uma estratégia de testes com cinco bases de dados com diferentes níveis de desbalanceamento. Tais bases podem ser encontradas no repositório da University of California, Irvine (UCI)[1].

4.1 CARACTERÍSTICAS DAS BASES

As bases usadas foram as seguintes: Breast Cancer; Statlog (Vehicle Silhouettes), chamada de Vehicle; Hepatitis; Glass Identification, chamada de Glass e Yeast. Como o algoritmo proposto em langford tem um custo computacional alto, $O(n^2)$ para treinamento e $O(n^2)$ para gerar o ranking; isso aliado ao baixo controle sobre o ambiente de execução culminou na escolha de bases pequenas para avaliação do experimento.

As bases Vehicle, Glass e Yeast tratam, originalmente, de problemas multiclasse. A essas bases foram aplicadas transformações, descritas em (5), a fim de torná-las problemas de classe binária. Para as bases Vehicle e Glass, foram unidas todas as classes, exceto a minoritária, em uma nova classe. Para a base Yeast, apenas as classes de valor 'CYT' ou 'POX' foram consideradas.

Cada base utilizada apresenta diferenças em: desbalanceamento entre as classes, número de atributos, número de instâncias, entre outros. A tabela 4 mostra um resumo sobre as características das bases.

[1] http://archive.ics.uci.edu/ml/

Bases	Atributos		Instâncias	Classe		
	Contínuos	Discretos		Minoritária	Majoritária	Distribuição
breast-cancer	0	9	286	85	201	30% - 70%
vehicle	18	0	846	199	647	23% - 77%
hepatitis	6	13	155	32	123	20% - 80%
glass	9	0	214	29	185	13% - 87%
yeast	8	0	483	20	463	4% - 96%

Tabela 4: Dados sobre as bases usadas para *ranking*.

4.2 EXECUÇÃO E AVALIAÇÃO DO ALGORITMO

Foram escolhidos quatro classificadores base para avaliar o algoritmo: J48, Naïve Bayes, Logistic e SMO. O motivo da escolha desses classificadores é o reconhecimento de tais como padrões em suas famílias, J48 para árvores de decisão (implanta a árvore de decisão C4.5), Naïve Bayes para estatísticos, Logistic (implanta curva logística) e SMO (implanta Support Vector Machine) para classificadores baseados em funções.

Para cada classificador base houve quatro baterias de execução com diferentes configurações:

1. Somente o classificador;
2. O classificador como base para a técnica de *ranking reduzido a classificação* original;
3. O classificador como base para o algoritmo de ranking com configurações de 1 par por instância e variando o número de classificadores na votação entre 1 e 20;
4. O classificador como base para o algoritmo de ranking com configurações de 1 classificador na votação e variando o número de pares por instância entre 1 e 20.

Nas três baterias que envolveram o algoritmo de *ranking*, o método de ordenação usado para gerar o resultado final foi o *torneio*, como explicado no artigo langford. Em todas as baterias foi executada uma validação cruzada com 10 partições e os resultados apresentados nas tabelas é a média das *AUCs* obtidas para cada partição.

Para as tabelas de número 5, 6, 7 e 8; o símbolo * significa ranking aplicado com 1 par por instância e 10 classificadores na votação. O símbolo ** significa ranking aplicado com 10 pares por instância e 1 classificador na votação.

4.2.1 DESEMPENHO PARA C4.5 *(TREES.J48)*

Bases	J48	J48 acrescido de		
		Ranking Original	Ranking*	Ranking**
breast-cancer	**0,62806 (0,01005)**	0,46784 (0,03049)	0,51289 (0,01950)	0,45055 (0,02984)
vehicle	0,93725 (0,00056)	0,91389 (0,00624)	**0.98072 (0.00017)**	0,95670 (0,00071)
hepatitis	0,69655 (0,04084)	0,67112 (0,03127)	**0,74322 (0,03925)**	0,72179 (0,04571)
glass	**0,90322 (0,01578)**	0,83772 (0,03524)	0,89016 (0,02934)	0,88860 (0,02668)
yeast	0,48918 (0,00021)	**0,95009 (0,01453)**	0,78550 (0,03630)	0.84806 (0.01924)

Tabela 5: Desempenho para árvore de decisão C4.5

(a) Breast cancer

(b) Glass

(c) Hepatitis

(d) Vehicle

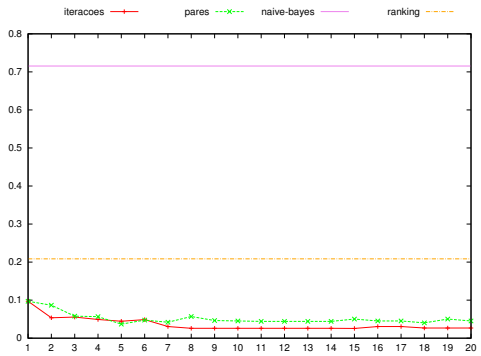
(e) Yeast

Figura 1: Gráficos de desempenho para árvore de decisão C4.5

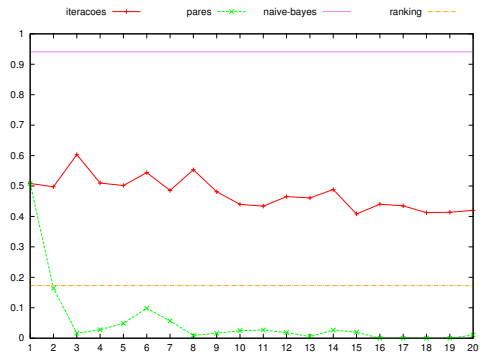
4.2.2 DESEMPENHO PARA NAÏVE BAYES *(BAYES.NAIVEBAYES)*

Bases	Naïve Bayes	Naïve Bayes acrescido de		
		Ranking Original	Ranking*	Ranking**
breast-cancer	**0,71543 (0,01899)**	0,20857 (0,00620)	0,04976 (0,00445)	0,04532 (0,00426)
vehicle	**0,80898 (0,00468)**	0,24323 (0,01559)	0,00310 (0,00004)	0,12514 (0,01546)
hepatitis	**0,85919 (0,01190)**	0,22489 (0,05188)	0,06052 (0,00265)	0.06608 (0,00101)
glass	**0,94084 (0,01099)**	0,17271 (0,02416)	0,02222 (0,00151)	0,02476 (0,00132)
yeast	0,82863 (0.06355)	0,84269 (0,03505)	**1,00000 (0,00000)**	**1,00000 (0,00000)**

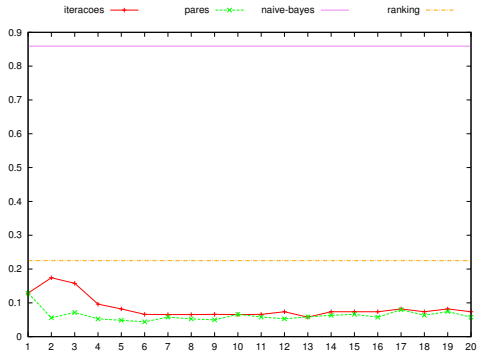
Tabela 6: Desempenho para Naïve Bayes



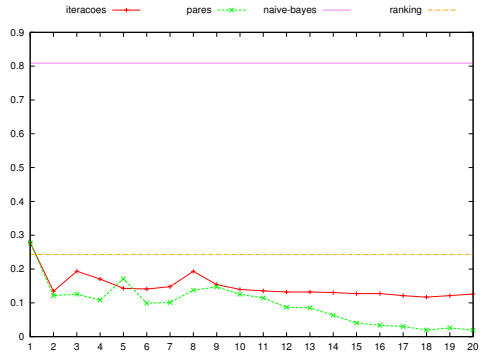
(a) Breast cancer



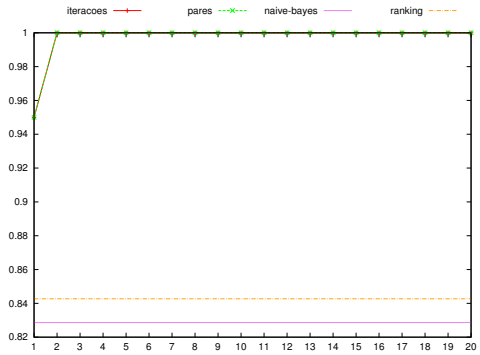
(b) Glass



(c) Hepatitis



(d) Vehicle



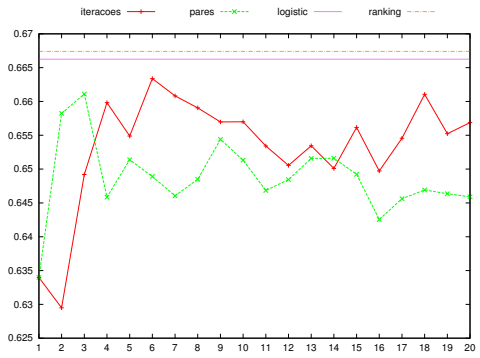
(e) Yeast

Figura 2: Gráficos de desempenho para Naïve Bayes

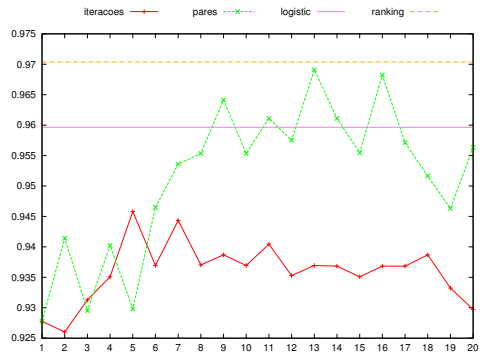
4.2.3 *DESEMPENHO PARA A* CURVA LOGÍSTICA *(FUNCTIONS.LOGISTIC)*

Bases	Logistic	Logistic acrescido de		
		Ranking Original	Ranking*	Ranking**
breast-cancer	0,66625 (0,02322)	**0,66740 (0,02364)**	0,65784 (0,02143)	0,65132 (0,02219)
vehicle	0,99358 (0,00003)	**0,99420 (0,00003)**	0,99358 (0,00003)	0,99234 (0,00002)
hepatitis	**0,80924 (0,02598)**	0,79882 (0,02432)	0,75064 (0,04278)	0,74557 (0,05075)
glass	0,95965 (0,00308)	**0,97037 (0,00192)**	0,96101 (0,00484)	0,95536 (0,00384)
yeast	0,85805 (0,04115)	0,83453 (0,04931)	**0,87414 (0,03463)**	0,85691 (0,03997)

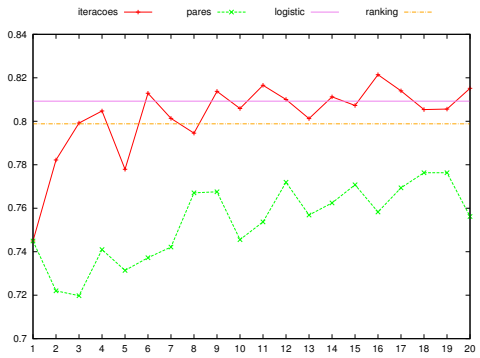
Tabela 7: Desempenho para Logistic



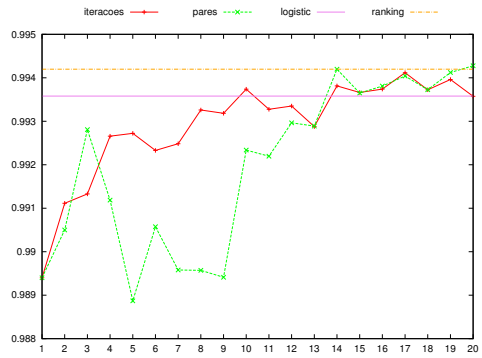
(a) Breast cancer



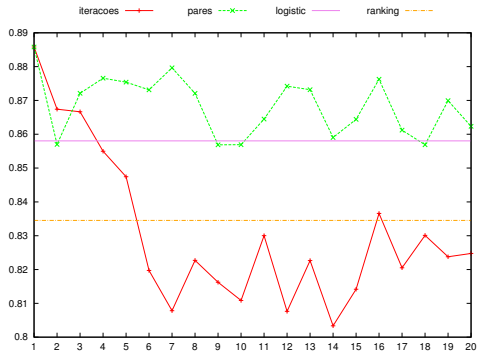
(b) Glass



(c) Hepatitis



(d) Vehicle



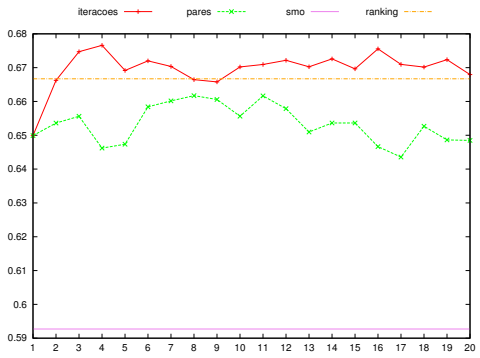
(e) Yeast

Figura 3: Gráficos de desempenho para Curva Logística

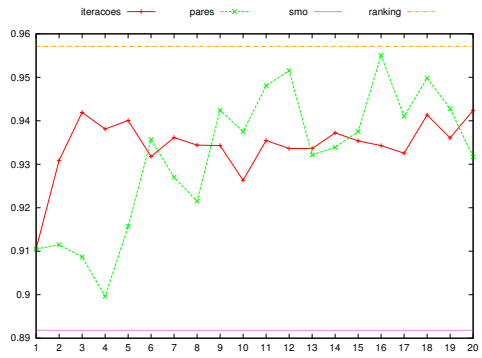
4.2.4 *DESEMPENHO PARA* SUPPORT VECTOR MACHINE *(FUNCTIONS.SMO)*

Bases	SMO	SMO acrescido de		
		Ranking Original	Ranking*	Ranking**
breast-cancer	0,59272 (0,00696)	**0,66670 (0,02294)**	0,65832 (0,02119)	0,65563 (0,02003)
vehicle	0,94434 (0,00169)	**0,99651 (0,00001)**	0,99396 (0,00002)	0,99380 (0,00002)
hepatitis	0,75128 (0,01618)	**0,81522 (0,01914)**	0,80759 (0,01899)	0,78189 (0,02640)
glass	0,89181 (0,00679)	**0,95712 (0,00175)**	0,93402 (0,00610)	0,93752 (0,00537)
yeast	0,77391 (0,04799)	0,83555 (0,04831)	**0,99891 (0,00001)**	**0,99891 (0,00001)**

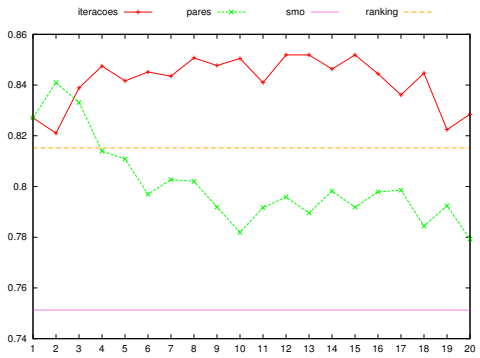
Tabela 8: Desempenho para Support Vector Machine



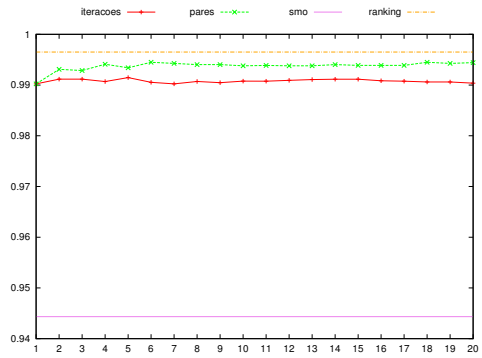
(a) Breast cancer



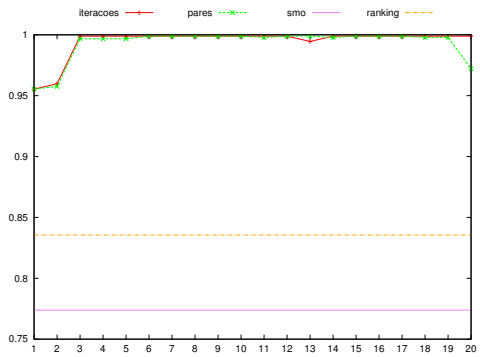
(b) Glass



(c) Hepatitis



(d) Vehicle



(e) Yeast

Figura 4: Gráficos de desempenho para Support Vector Machine

5 CONCLUSÃO

A primeira conclusão que pode ser tirada é que o algoritmo testado aqui possui tempo computacional elevado tanto para treinamento, quanto para avaliação. As estratégias para melhoria no tempo de treinamento, como uso de votação e de um número reduzido de pares por instância, ajudaram nesse aspecto e criaram resultados por vezes superiores ao algoritmo original e ao classificador base.

Os esforços para otimizar o tempo na etapa de avaliação consistiram em implantar um algoritmo baseado no algoritmo *Quicksort*, que teria um limite superior de chamadas à *função votacao* de $O(n \cdot \log(n))$, um avanço comparado ao limite superior do torneio que é $O(n^2)$. Embora esta estratégia tenha sido completamente implantada, não houve nenhum experimento que a utilizasse.

O classificador Naïve Bayes teve um desempenho incomum quando combinado com o algortimo de *Ranking*. Esse classificador gerou resultados absurdamente abaixo do esperado para todas as bases testadas, exceto para a base *yeast*.

Analisando os gráficos da seçao 4.2, repara-se que, para a base *glass*, os resultados começam a melhorar em relação aos obtidos para as bases *breast-cancer*, *vehicle* e *hepatitis*. Já para a base *yeast*, o Naïve Bayes aliado ao algoritmo de *Ranking* teve uma performance perfeita, acertando quase todas as ordenações. Esse comportamento bipolar não foi elucidado nesse estudo.

Comparando as estratégias implantadas para acelerar a etapa de treinamento de forma isolada, pode-se chegar a seguinte conclusão: o aumento do número de classificadores na votação cria resultados com menor variação na AUC que o aumento de pares por instância no treinamento.

Olhando as curvas geradas para cada uma dessas estratégias nos gráficos do capítulo 4, percebe-se que a tendência para a estratégia de aumento de classificadores na votação é, na maioria dos casos, crescente. Enquanto a tendência para a estratégia de aumento do número de pares por instância não pode ser definida com clareza em alguns casos.

O artigo (1) mostra que um classificador que produza um erro α na acurácia pode produzir um erro teórico máximo de $n \cdot \alpha$ na AUC, onde n é o número de exemplos a serem ordenados. Mostra também que a técnica de *ranking reduzido a classificação* produz um erro teórico máximo de $2 \cdot \alpha$.

Com a exceção do classificador naïve-bayes, que apresentou um comportamento anômalo, a técnica avaliada resultou em menor perda na AUC em comparação com a ordenação do classificador solo na maioria dos casos.

Com o uso da árvore de decisão C4.5 ordenando a base Yeast — a mais desbalanceada de todas as bases testadas — a técnica apresentou um incremento de aproximadamente 100% comparada à ordenação da árvore de decisão solo. Esse é um caso que tende a ratificar o ponto exposto em (1).

Em quase todos os casos, a otimização de votação elevou a performance da técnica original de forma que os resultados obtidos fossem melhores que os resultados dos classificadores solo, e muitas vezes superiores à técnica de *ranking reduzido a classificação*.

REFERÊNCIAS BIBLIOGRÁFICAS

1 BALCAN, M.-F. et al. Robust reductions from ranking to classification. In: *Proceedings of 20th annual conference on Learning theory*. [S.l.: s.n.], 2007.

2 WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd edition. ed. [S.l.]: Morgan Kaufmann, 2005.

3 LIU, T.-Y. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, v. 3, n. 3, 2009.

4 AILON, N.; MOHRI, M. An efficient reduction of ranking to classification. In: *In Conference on Learning Theory (COLT)*. [S.l.: s.n.], 2008.

5 GUO, H. Learning from imbalanced data sets with boosting and data generation: The databoost-im approach. *SIGKDD Explorations*, v. 6, p. 2004, 2004.