Pós-Graduação em Ciência da Computação

"SISTEMAS INTELIGENTES HÍBRIDOS PARA CLASSIFICAÇÃO DE TEXTO"

Por

Joseane Pereira Rodrigues

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, MARCO 2009

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOSEANE PEREIRA RODRIGUES

SISTEMAS INTELIGENTES HÍBRIDOS PARA CLASSIFICAÇÃO DE TEXTO

ORIENTADORA: FLÁVIA DE ALMEIDA BARROS
CO-ORIENTADOR: RICARDO B. C. PRUDÊNCIO

RECIFE, MARÇO 2009

Agradecimentos

Cristo disse: "*Tenho-vos dito estas coisas, para que em mim tenhais paz. No mundo tereis tribulações; mas tende bom ânimo, eu venci o mundo."* João 16:33

Agradeço ao meu Deus, pelo Seu amor incondicional, por todas as bênçãos que Ele derramou e tem derramado em minha vida, pela proteção que sempre me concede e por nunca me deixar só! Te amo e confio em Ti, Senhor!

À minha linda família: mãe, pai e irmãos (Jeanne, José, e Manoel). Obrigada pelo amor, carinho e apoio. Sou muito feliz por tê-los em minha vida. Em especial, agradeço à minha mãe e à minha irmã Jeanne. Obrigada por tudo! Mãe, você é realmente a melhor mãe que uma pessoa poderia ter! Te amo infinitamente! Je, minha segunda mãe, meu amor por você supera a da uma irmã. Te quero muito bem!

À Marcelino, meu marido, presente de Deus! Pelo companheirismo, pelo incentivo, pelo carinho, pela paciência, enfim, por tudo! Obrigada por ser quem você é! Eu te amo em todos os sentidos! Meu primeiro, grande e único amor!

Aos meus amigos: do mestrado, do condomínio, e os da igreja. Em especial, à família de Snorem Holanda, vocês sempre estarão no meu coração!

Aos meus orientadores, Flávia Barros e Ricardo Prudêncio. Obrigada pelos conhecimentos passados e pela compreensão em alguns momentos!

Resumo

Grande parte da informação contida em repositórios digitais, como a *Web* e as Bibliotecas Digitais, está representada em formato de documentos de texto. Sistemas de Recuperação de Informação têm sido usados para prover acesso a documentos relevantes armazenados nesses repositórios. No entanto, esses sistemas ainda apresentam limitações a serem superadas. Muitos dos problemas desses sistemas têm sido tratados usando técnicas de classificação de texto oriundas da Inteligência Artificial (em especial os algoritmos de Aprendizado de Máquina). Cada técnica apresenta vantagens e limitações, considerando os conjuntos de textos em que são aplicadas.

Este trabalho investigou técnicas de combinação de classificadores de texto, em especial, técnicas baseadas em *boosting*. Essas técnicas tentam superaram as limitações dos classificadores sendo combinados, mantendo suas vantagens individuais, e assim apresentando um melhor desempenho nas tarefas em que são aplicados. Trabalhos anteriores apontam problemas em aberto em relação ao uso de métodos de combinação para classificadores de texto. Assim, esperamos neste projeto avançar o estado da arte sobre o tema.

No trabalho realizado, implementamos uma variação de Boosting proposta na literatura que usa informações de vizinhança, chamado LocalBoost. Essa variação tem alcançado bons resultados comparativos em dados benchmark. Propomos ainda variações do LocalBoost para tratamento de dados desbalanceados, além de outras modificações que foram avaliadas em experimentos tanto com bases de classificação de texto como para bases de dados não-textuais. Os resultados dos experimentos relevaram a viabilidade do uso das variações propostas.

Palavras-chave: Classificação de texto, combinação de classificadores de texto.

Abstract

A large part of the information available in digital repositories, such as Web and the Digital Libraries, is represented in textual documents. Information Retrieval systems have been used to provide access to documents stored in such repositories. However, these systems still have limitations to be overcome. Many of these problems have been treated by using techniques of text classification from the Artificial Intelligence (in particular the algorithms of Machine Learning). Each technique presents advantages and limitations, considering the corpus of texts being tackled.

This study investigated techniques that combine text classifiers, in particular, techniques based on Boosting. These techniques try to overcome the limitations of the text classifiers being combined, at same time maintaining its advantages, thus presenting a better performance in tasks in which they are applied. Previous studies suggested some open problems regarding the use of methods for combining text classifiers. Hence, we expect that this work advance the state of the art on the theme.

In the performed work, we implemented a variation of Boosting proposed in the literature which uses information of neighborhoods, called LocalBoost. This variation has achieved good comparative results in benchmark data. We also proposed changes in LocalBoost for processing unbalanced data and other modifications that were evaluated in experiments with both datasets of text classification and non-textual datasets. The results of the experiments showed the feasibility of using the proposed methods.

Keywords: Text Classification; Combination of Classifiers.

Sumário

CAPÍTULO I

1 INTRODUÇÃO.....1

CAPÍTULO II

2 CLASSIFICAÇÃO DE TEXTO.....4
2.1 Definição do Problema.....5
2.2 Critérios de análise.....8
2.3 Algoritmos de Classificação.....9
2.3.1 Naïve Bayes.....9
2.3.1.1 Análise do algoritmo.....13
2.3.2 kNN.....14
2.3.2.1 Análise do algoritmo.....16
2.3.3 Árvores de Decisão.....16
2.3.3.1 Análise do algoritmo.....19
2.3.4 SVM.....19
2.3.4.1 Análise do algoritmo.....22
2.4 Considerações Finais.....22

CAPÍTULO III

3 MÉTODOS DE COMBINAÇÃO DE CLASSIFICADORES.....23
3.1 Bagging 24
3.1.1 Descrição Básica do Método.....24
3.1.2 Variações.....26
3.2 Boosting.....28
3.2.1 Descrição Básica do Método.....28
3.2.2 Variações.....30
3.3 Stacking.....33

3.3.1 Descrição Básica do Método.....33
3.3.2 Variações.....36
3.4 MDT.....38
3.4.1 Descrição Básica do Método.....38
3.5 Considerações Finais.....41

CAPÍTULO IV
4 SISTEMAS INTELIGENTES HÍBRIDOS PARA CLASSIFICAÇÃO DE TEXTO.....44
4.1 Conjuntos de dados desbalanceados.....45
4.2 B-LocalBoost e outras variações.....47
4.3 Considerações Finais.....51

CAPÍTULO V
5 TESTES E RESULTADOS.....53
5.1 Conjuntos de Dados Utilizados.....53
5.1.1 Bases de dados de texto.....54
5.1.2 Bases de dados do UCI.....55
5.2 Metodologia do Experimento.....56
5.3 Resultados Obtidos.....57
5.4 Considerações Finais.....99

CAPÍTULO VI
6 CONCLUSÃO.....101
6.1 Principais Contribuições 101
6.2 Trabalhos Futuros.....102
REFERÊNCIAS BIBLIOGRÁFICAS.....104

Lista de Figuras

Figura 2.1 – Etapas da classificação de texto.....8
Figura 2.2 – Classificação pelo método KNN.....14
Figura 2.3 – Exemplo de uma Árvore de Decisão.....17
Figura 2.4 – Componentes de uma Árvore de Decisão e suas funções.....17
Figura 2.5 – Hiperplano ótimo.....20
Figura 3.1 – Ilustração do método *Bagging*.....25
Figura 3.2 – O método BEV para classificação de dados desbalanceados....27
Figura 3.3 – Funcionamento do *Stacking*.....34
Figura 3.4 – (a) Processo de validação cruzada para criar o conjunto de dados do nível-meta.....35
(b) Execução do algoritmo *Stacking*.....35
Figura 3.5 – Classificadores homogêneos combinados usando Bag-Stacking.....37
Figura 4.1 – Pseudocódigo do *B-LocalBoost*.....49
Figura 4.2 – Ilustração do processo de balanceamento dos dados.....50

Lista de Tabelas

Tabela 3.1 – Uma Meta Decision Tree.....40
Tabela 5.1 – Tabela geral de ganhos e perdas de cada algoritmo.....62
Tabela 5.2 – Tabela geral de ganhos e perdas de cada algoritmo.....67
Tabela 5.3 – Tabela geral de ganhos e perdas de cada algoritmo.....71
Tabela 5.4 – Tabela geral de ganhos e perdas de cada algoritmo.....76
Tabela 5.5 – Tabela geral de ganhos e perdas de cada algoritmo.....80
Tabela 5.6 – Tabela geral de ganhos e perdas de cada algoritmo.....84
Tabela 5.7 – Tabela geral de ganhos e perdas de cada algoritmo.....88
Tabela 5.8 – Tabela geral de ganhos e perdas de cada algoritmo.....93

1 Introdução

Grande parte da informação contida em repositórios digitais, como a Web e as Bibliotecas Digitais, está representada em formato de documentos de texto. Sistemas de Recuperação de Informação [Baeza-Yates and Ribeiro-Neto 1999], como os engenho de busca na Web, têm sido usados com relativo sucesso para prover acesso por parte dos usuários a documentos relevantes armazenados nesses repositórios. No entanto, esses sistemas ainda apresentam problemas graves e limitações a serem superadas.

Muitos dos problemas desses sistemas têm sido tratados usando-se técnicas de Classificação de Texto. Essas técnicas têm sido usadas, por exemplo, para auxiliar nas tarefas de indexação de documentos, filtragem de documentos e extração de informação [Sebastiani 2002]. Diferentes técnicas oriundas da Inteligência Artificial (em particular, o Aprendizado de Máquina) têm sido usadas em problemas de Classificação de Texto. Cada técnica apresenta vantagens e limitações inerentes, considerando os conjuntos de textos em que são aplicadas.

1.1 Trabalho Realizado

Neste trabalho de mestrado, tivemos como objetivo investigar técnicas e métodos para a construção de Sistemas Inteligentes Híbridos (SIHs), que combinam um ou mais algoritmos de Aprendizado de Máquina para problemas de Classificação de Texto. Os SIHs têm como propósito superar as limitações e manter as vantagens individuais dos classificadores de texto sendo combinados, e assim ter melhor desempenho nas tarefas em que são aplicados.

A criação do sistema proposto foi precedida por um estudo detalhado dos métodos de combinação de classificadores. Em particular, invetigamos os métodos Boosting [Iyer 2000] e Bagging [Breiman 1996], e suas variações. A partir desses estudos, escolhemos trabalhar o método LocalBoost [Zhang and Zhang

2008], uma variação do método Boosting, porque ele apresenta, em muitas situações, melhor precisão do que o algoritmo Boosting original.

Investigamos ainda nesse trabalho variações de métodos de Boosting para tratamento de conjuntos de treinamento desbalanceados. Conjuntos de dados desbalanceados geralmente fazem com que os algoritmos de aprendizado gerem classificadores tendenciosos, beneficiando as classes majoritárias, em detrimento da classificação das instâncias das classes minoritárias. Esse tipo de problema pode ter influência negativa no desempenho dos métodos de combinação, incluindo os métodos de Boosting.

Nesse contexto, propomos um novo método de combinação de classificadores que estende o método de LocalBoost para tratar dados desbalanceados, chamado de LocalBoost Balanceado (B-LocalBoost). Para isto, foi usada uma técnica similar à empregada no método BEV (Bagging Ensemble Variant) [Li 2007] para balanceamento de dados. Uma camada de balanceamento para o LocalBoost foi proposta e implementada. O principal objetivo desse novo método de combinação é melhorar a precisão do método LocalBoost quando lida com dados desbalanceados.

Além do B-LocalBoost, outras versões de Boosting também foram idealizadas e implementadas, e os testes foram realizados sobre diversos conjuntos de dados provenientes de fontes diferentes. Foram realizados experimentos com três conjuntos de dados referentes à classificação de texto, como os documentos do DMOZ[1] (ODP - Open Directory Project). Como base de comparação, foram realizados ainda experimentos em bases de dados não-textuais, extraídas do repositório UCI[2] (University of California). Os resultados obtidos demonstram a viabilidade das extensões aqui proposta sobre os métodos originais.

[1] http://www.dmoz.org
[2] http://archive.ics.uci.edu/ml

Obs. Os links nã podem ficar dentro do texto. Ou viram nota de rodapé, ou vão para a lisat de referencias bibliográficas.

1.2 Estrutura do documento

Esta dissertação está dividida em seis capítulos, sendo esta Introdução o primeiro deles. Os demais capítulos estão organizados da seguinte forma:

- Capítulo 2 – apresenta alguns algoritmos estado-da-arte de classificação, bem como a sua aplicação na tarefa de Classificação de Texto;

- Capítulo 3 – mostra algoritmos de combinação de classificadores, também do estado-da-arte, e suas variações aplicadas na resolução de problemas de Classificação de Texto em diversas áreas;

- Capítulo 4 – apresenta, em detalhes, o processo de desenvolvimento do método proposto, assim como das outras versões do LocalBoost aqui propostas. Além dos detalhes da implementação do protótipo, a arquitetura utilizada no desenvolvimento também é apresentada;

- Capítulo 5 – aqui são apresentados os resultados obtidos nos experimentos realizados com o algoritmo B-LocalBoost e outras versões, bem como a análise dos resultados em comparação com os de outros algoritmos;

- Capítulo 6 – são apresentadas as contribuições dadas por este trabalho de mestrado, além de apresentar a previsão de trabalhos futuros.

2 Classificação de Texto

Classificação de Texto consiste em associar documentos de texto a classes temáticas pré-definidas. Por exemplo, notícias jornalísticas podem ser associadas a categorias de política, esportes, entretenimento, dentre outras. Ou ainda, mensagens de e-mail podem ser classificadas como spam ou não-spam, ou um paciente como doente ou sadio, e assim por diante. Resolver algum desses problemas pode resultar em significativos avanços nas respectivas áreas e, se pudermos entender como programar computadores para resolvê-los, assim como para outros problemas de difícil solução e igual importância, poderemos obter resultados importantes na ciência da computação.

Uma das estratégias usadas para auxiliar na organização e recuperação de documentos digitais em formato de texto, tanto para acesso livre na Web como em outros repositórios, como as Bibliotecas Digitais, tem sido o uso de classificadores automáticos de texto [Hotho *et al*. 2005], com o objetivo de associar documentos digitais a classes pré-definidas, de acordo com o conteúdo e a estrutura dos textos.

De acordo com [Sebastiani 2002], duas abordagens distintas de Inteligência Artificial (IA) têm sido adotadas para desenvolver classificadores de texto:

(1) Engenharia de Conhecimento - regras são manualmente definidas por um engenheiro do conhecimento com o auxílio de um especialista no domínio da aplicação. As regras definem como classificar documentos a partir de características dos textos;

(2) Aprendizado de Máquina - um classificador é automaticamente construído por um algoritmo de aprendizado a partir de um conjunto de treinamento com documentos pré-classificados. O classificador construído deverá ser capaz de classificar novos documentos não vistos durante o aprendizado.

A abordagem de Aprendizado de Máquina apresenta algumas vantagens em relação à Engenharia do Conhecimento, uma vez que diminui a necessidade da interação entre engenheiros de conhecimento e especialistas no domínio (tarefa cara e demorada), e ainda assim alcança boa taxa de precisão [Sebastiani 2002]. Destacamos nesse contexto que diferentes famílias de algoritmos de Aprendizado de Máquina têm sido usadas em tarefas de Classificação de Texto, como algoritmos de Árvores de Decisão [Cohen and Singer 1999], algoritmos de Redes Neurais Artificiais [Li and Park 2006], aprendizado baseado em Instância [Tan 2006], aprendizado Bayesiano [Frank and Bouckaert 2006], e Support Vector Machine (SVM) [Kim *et al*. 2005].

A seguir, veremos conceitos básicos sobre o problema e o processo da classificação de texto, bem como a definição de uma nomenclatura para ser usada nesta dissertação. A seção 2.2 traz os critérios utilizamos neste trabalho para análise dos algoritmos de classificação apresentados na seção 2.3. Esses mesmos critérios nortearam as decisões adotadas durante a realização deste trabalho de pesquisa. A seção 2.3 apresenta quatro dos principais algoritmos de aprendizagem utilizados para classificação de texto: Naïve-Bayes, kNN, Árvore de Decisão e SVM. Por fim, temos algumas considerações finais que concluem este capítulo.

2.1 Definição do Problema

Classificação de texto pode ser definida como a associação de documentos de texto a classes predefinidas $V = (v_1,v_2,...,v_L)$. Quando um documento é associado a uma única categoria dentre as L classes disponíveis dizemos que se trata de um problema de classificação single-label (e.g., uma mensagem classificada exclusivamente como spam ou não-spam). Quando um documento pode ser associado a mais de uma categoria ao mesmo tempo, dizemos que como classificação multi-label (e.g., notícias classificadas como pertencentes a mais de um assunto). Neste trabalho de dissertação, focamos no problema de classificação single-label que é, de fato, o mais explorado na literatura. Além disso,

problemas de classificação multi-label podem ser transformados em muitos casos em um ou mais problemas single-label [Tsoumakas and Katakis 2007].

Dentro da abordagem de aprendizagem de máquina, classificadores são construídos via um processo indutivo, capaz de aprender, a partir de um conjunto de documentos pré-classificados, relacionamentos entre características dos documentos e as classes em questão [Sebastiani 2002]. Formalmente, durante o processo de aprendizagem, o algoritmo recebe um conjunto de documentos $D = (d_1, d_2...d_N)$, descritos através de um conjunto de atributos $(a_1, a_2...a_M)$. Assim, cada documento d_i é representado por um vetor $(a_{i1}, a_{i2},...,a_{iM})$, onde *aij* é o valor do atributo a_j para o documento d_i. A definição de um classificador de texto com algoritmos de aprendizado pode ser resumido nas seguintes etapas (ver Figura 2.1):

1. Aquisição de documentos pertencentes ao domínio em questão (ou seja, coleta dos documentos em *D*);

2. Criação de um vocabulário $(w_1,...,w_T)$ de *T* termos que será usado na representação dos documentos. Essa etapa envolve pré-processamento do texto, como a análise léxica (eliminação de dígitos, sinais de pontuação, etc), remoção de stopwords (artigos, preposições, etc), stemming das palavras (redução da palavra ao seu radical), dentre outros operadores de texto;

3. Criação da representação inicial dos documentos, ou seja, definição do conjunto de atributos que descrevem os documentos. Cada atributo pode ser simplesmente um valor booleano que indica se um dado termo do vocabulário está presente ou não no documento (i.e., representação booleana). Em outros trabalhos, cada atributo corresponde a um peso numérico associado a um dado termo, indicando a relevância do termo para o documento sendo descrito. O cálculo do peso de um termo para um documento é feito em muitos trabalhos usando o esquema TF-IDF (*Term Frequency – Inverse Document Frequency*) [Salton and Buckley 1988], como mostrado na equação 2.1.

$$\text{TF-IDF} = \frac{Freq_{wd}}{DocFreq_{w}} \qquad (2.1)$$

Onde, $Freq_{wd}$ corresponde ao número de vezes que o termo *w* aparece no documento *d* e $DocFreq_{w}$ corresponde ao número de documentos que o termo *w* aparece.

4. Redução da dimensionalidade, onde ocorre a seleção dos *M* atributos mais relevantes da representação inicial (com $M < T$). Essa etapa pode ser feita usando diferentes critérios de seleção de atributos como Ganho de Informação, Informação Mútua, X^2 statistic (Qui-quadrado), entre outros [Yang and Pedersen 1997].

5. Indução do classificador a partir de um conjunto de treinamento. Na seção 2.3, iremos abordar alguns algoritmos que podem ser usados para geração de classificadores;

6. Avaliação do desempenho a partir de um conjunto de teste. Dentre as métricas usadas para avaliação de classificadores podemos citar a precisão (número de documentos associados corretamente pelo classificador a uma dada classe dividido pelo número de documentos associados pelo classificador à classe), e cobertura (número de documentos associados corretamente pelo classificador a uma dada classe dividido pelo número de documentos existentes da classe). A avaliação de um algoritmo pode ser feita ainda através de experimentos de validação cruzada (*k-fold cross-validation*), onde diferentes conjuntos de treinamento e teste são usados para construção e avaliação dos classificadores.

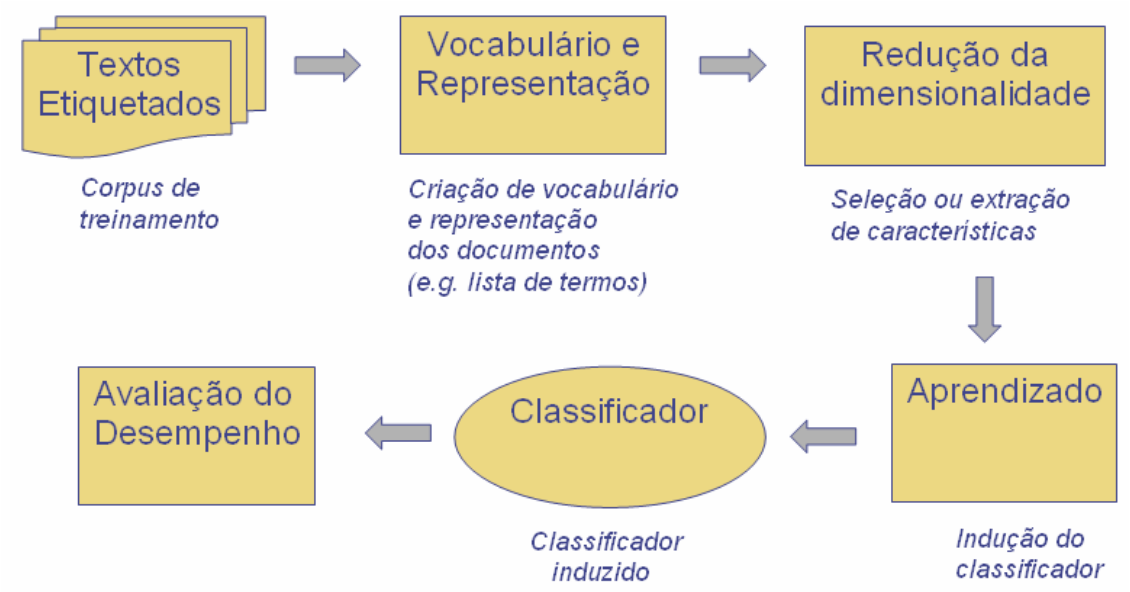


Figura 2.1. Etapas da classificação de texto

2.2 Critérios de análise

Existem, na literatura relacionada, alguns critérios para análise/avaliação de algoritmos de classificação. Esses critérios serão utilizados aqui para facilitar uma comparação entre esses algoritmos.

Critérios de análise [Mitchell 1997], [Rezende 2005], [Busemann *et al*. 2000]:

- *Eager* ou *lazy* - a classificação *eager* usa os dados de treinamento para construir um único conjunto de regras que é usado para classificar todas as instâncias de teste. Por outro lado, a classificação *lazy* usa os dados de treinamento para construir um conjunto de regras específico para cada instância de teste, e então, as decisões são baseadas sobre as propriedades particulares da instância de teste avaliada no momento [Veloso and Meira 2005];

- Custo computacional - recursos (tempo, memória, etc) utilizados pelo computador para o processo de classificação;

- Requer definição de parâmetros - necessidade de definição de parâmetros para a realização da classificação;

- Instável/estável aos dados - um algoritmo é instável quando pequenas mudanças nos conjuntos de treinamento podem facilmente provocar grandes mudanças no classificador gerado;

- Gera regras explicitas ou não - algoritmo de classificação que gera um classificador formado por regras explícitas ou implícitas [Pila 2006];

- Sensibilidade a atributos redundantes e/ou irrelevantes (dimensionalidade dos dados) - um algoritmo é dito sensível a atributos redundantes e/ou irrelevantes quando o classificador gerado é influenciado devido à presença ou ausência destes atributos; e

- Sensibilidade a classes desbalanceadas - quando algoritmos geram classificadores que tendem a valorizar classes predominantes e a ignorar classes de menor representação [Machado and Ladeira 2007].

2.3 Algoritmos de Classificação

Como citado anteriormente, existem diversos algoritmos que podem ser empregados na tarefa de classificação de texto. Descrevemos abaixo três dos algoritmos mais difundidos na literatura, e que foram utilizados nos nossos experimentos: Naive Bayes [Good 1965], kNN [Fix and Hodges 1951], e SVM (Support Vector Machine) [Vapnik 1995].

2.3.1 Naïve Bayes

Naive Bayes [Good 1965], [John and Langley 1995] é um algoritmo de classificação ingênuo probabilístico, baseado na aplicação do teorema de *Bayes* para determinar a classe de maior probabilidade para cada nova instância a ser classificada.

Para classificar uma nova instância, o algoritmo determina a classe mais provável, dados os atributos $(a_1,a_2,...,a_M)$ que descrevem a instância [Mitchell 1997]. A equação 2.2 mostra o cálculo da classe de maior probabilidade para o classificador *naive Bayes*, onde V_{NB} simboliza a resposta do classificador, $P(v_l)$ corresponde à freqüência estimada de instâncias de treinamento que pertencem a cada classe v_l, e $P(a_i|v_l)$ é a freqüência estimada dos valores do atributo a_i restrito aos exemplos de treinamento da classes v_l.

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j) \tag{2.2}$$

O algoritmo de *naive Bayes* considera que os atributos que descrevem as instâncias são completamente independentes, o que raramente é observado em problemas reais. Contudo, o algoritmo tem obtido bons resultados em situações complexas do mundo real, e em especial, em problemas de classificação de texto, e.g., [Frank and Bouckaert 2006].

Naive Bayes Multinomial [Kibriya *et al*. 2004] (*Multinomial naive Bayes*, *MNB*) é a versão de *naive Bayes* geralmente usada para a tarefa de classificação de texto. No classificador *MNB*, a probabilidade de se obter um valor de classe *v* dado um documento de teste *d* é computado como:

$$P(v|d) = \frac{P(v) \prod_{w \in d} P(w|v)^{TF_{wd}}}{P(d)} \tag{2.3}$$

onde TF_{wd} é o número de vezes que o termo *w* ocorre em um documento d, $P(w|v)$ é a probabilidade de se observar o termo *w* dada a classe *v*, $P(v)$ é a probabilidade prévia da classe *v*, e $P(d)$ é uma constante de normalização que faz com que a soma das probabilidades das diferentes classes resulte no valor 1 [Frank and Bouckaert 2006]. $P(v)$ é estimado pela proporção de documentos de treinamento pertencentes à classe, e , $P(w|v)$ é estimado como:

$$P(w|v) = \frac{1 + \sum_{d \in D_v} TF_{wd}}{T + \sum_{w'} \sum_{d \in D_v} TF_{w'd}} \quad (2.4)$$

onde D_v é a coleção de todos os documentos de treinamento na classe *v*, e *T* é o tamanho do vocabulário (i.e. o número de termos distintos em todos os documentos de treinamento). A adição do número 1 no numerador é chamado de correção *Laplace*, e corresponde à inicialização da quantidade de cada termo com o valor 1 em vez de 0. Ele requer a adição da variável *t* no denominador para obter uma distribuição de probabilidade que resulte na soma igual a 1. Este tipo de correção é necessária por causa do problema de freqüência-zero, onde um único termo em um documento de teste *d* que não ocorra em algum documento de treinamento pertencente a uma classe particular *v*, normalmente resultará em *P(v|d)* igual a 0 [Frank and Bouckaert 2006].

Todavia, muitos problemas de classificação de texto estão desbalanceados, e a prática de inicializar as freqüências dos termos de todas as classes com o mesmo valor, geralmente o valor 1, tende a influenciar a predição a favor das classes majoritárias. De acordo com [Frank and Bouckaert 2006], a quantidade de termos iniciais tem uma grande influência sobre a probabilidade predita quando há poucos dados, bem como nas classes menores no problema de classificação de texto. Por exemplo, em um problema binário, quando uma classe v_2 contém texto pequeno e é muito menor em relação à outra classe, a presença de um termo irrelevante, para a classificação, em um documento de teste, aumenta substancialmente a probabilidade estimada daquele documento de teste pertencer à classe v_1, devido à maior freqüência do termo *w* ser na classe v_1 $P(w|v_1)$. [Frank and Bouckaert 2006].

Por causa da correção de *Laplace* usada em (3), o desbalanceamento dos dados, isto é, classes que possuem tamanhos diferentes, pode causar problemas, já que a probabilidade de um termo irrelevante para uma classificação deveria ser 1, para que este termo não influenciasse a probabilidade da classe. Entretanto, somente se tem o valor 1 desejado para a probabilidade de um termo irrelevante

quando as classes contêm o mesmo número de termos [Frank and Bouckaert 2006].

Devido à queda de desempenho apresentada pelo *MNB* na classificação de dados desbalanceados, uma versão modificada deste algoritmo foi proposta por [Frank and Bouckaert 2006] na tentativa de resolver este problema.

Em [Frank and Bouckaert 2006], foi investigado o uso de quantidades diferentes de termos iniciais para tamanhos de classes distintas, e proposta uma heurística para escolher a quantidade inicial de termos para cada classe. Esta solução, que pode ser implementada como um passo de pré-processamento e pode ser vista como uma modificação da correção de *Laplace*, normaliza a quantidade de termos em cada classe, de forma que o tamanho total de cada uma das classes seja o mesmo após a normalização. Para fazer isto, o TF_{wd} do numerador da equação 2.4, onde d pertence a D_v, é substituído por

$$\alpha \times \frac{TF_{wd}}{\sum_{w'} \sum_{d \in D_v} TF_{w'd}} \qquad (2.5)$$

Com isto, o vetor de termos de cada classe é normalizado para ter tamanho α, quando medido com a equação 2.3. De acordo com [Frank and Bouckaert 2006], isso assegura que a probabilidade estimada para um termo irrelevante será igual a 1, independente do valor particular escolhido para a constante de normalização α, em (2.5).

Os experimentos, que utilizaram o classificador *MNB* com normalização por classe (MNB_{PCN}, *PCN* do inglês *Per-Class Normalization*) e sem normalização (*MNB*), foram realizados sobre conjuntos de dados do *Reuters*-21578, *Web KB*, *Industry Sector*, e 20 *Newsgroups*, com α igual a 1. Nos testes executados sobre os dados do *Reuters*, os resultados mostraram que o MNB_{PCN} melhorou o desempenho significantemente sobre três classes, do total de dez classes. Sobre o *Web KB*, MNB_{PCN} teve desempenho superior sobre todas as quatro classes.

Embora no *Industry Sector* ele tenha alcançado somente uma melhora significante, nenhuma perda significante de desempenho ocorreu. Porém, nos testes executados sobre os dados do 20 *Newsgroups*, *MNB* apresentou desempenho superior ao do MNB_{PCN}, tendo dezoito ganhos significantes e nenhuma perda significante. No entanto, verifica-se que o desempenho de MNB_{PCN} pode melhorar com a troca do valor de α (i.e. do tamanho do vetor de termos para cada classe) de 1 para o menor tamanho dos vetores das classes antes da normalização. Com isto, MNB_{PCN}, possuindo o menor valor para α, alcança o mesmo nível de desempenho que o *MNB* sobre as classes dos dados do 20 *Newsgroups*.

Apesar deste novo valor de α aumentar o desempenho de MNB_{PCN} sobre os dados do 20 *Newsgroups* e também sobre os dados do *Industry Sector*, ele diminui significantemente o desempenho deste classificador sobre os dados do *Reuters*-21578 e do *Web KB*. Portanto, segundo [Frank and Bouckaert 2006], a quantidade certa de nivelamento para *P(w|v)* depende do domínio. E, para melhorar o desempenho, um valor apropriado para α deve ser escolhido usando conjunto de validação ou validação cruzada.

2.3.1.1 Análise do algoritmo

O algoritmo *naive Bayes* tem as seguintes características: é um método eager, de baixo custo computacional, não gera regras explícitas, requer definição de parâmetros, instável em relação aos dados de treinamento, sensível a atributos redundantes, porém não é sensível a atributos irrelevantes, e é sensível a classes desbalanceadas. Este algoritmo, assim como outros métodos de aprendizado *Bayesiano*, tem ainda um diferencial importante pelo fato de calcular explicitamente probabilidades para os classificadores.

2.3.2 kNN

kNN (*k - Nearest Neighbor*) [Fix and Hodges 1951], [Cover and Hart 1967], [Aha and Kibler 1991] é o método mais básico dentre os algoritmos baseados em instâncias. Ele assume que todas as instâncias correspondem a pontos em um espaço *n*-dimensional. Na versão mais comum de *kNN*, para classificar uma nova instância *x*, o algoritmo atribui a *x* a classe mais freqüente entre as *k* instâncias de treinamento mais próximas, ou seja, que tenham menor distância a *x*. A vizinhança de uma instância é definida em termos de uma função de distância ou de uma função de similaridade (e.g., distância Euclidiana, medida de *Co-Seno*, etc). Para ilustrar, temos um exemplo na Figura 2.2, onde o exemplo "Desconhecido 1" é classificado como pertencente à Classe B, e o exemplo "Desconhecido 2" à Classe A, considerando $k = 7$.

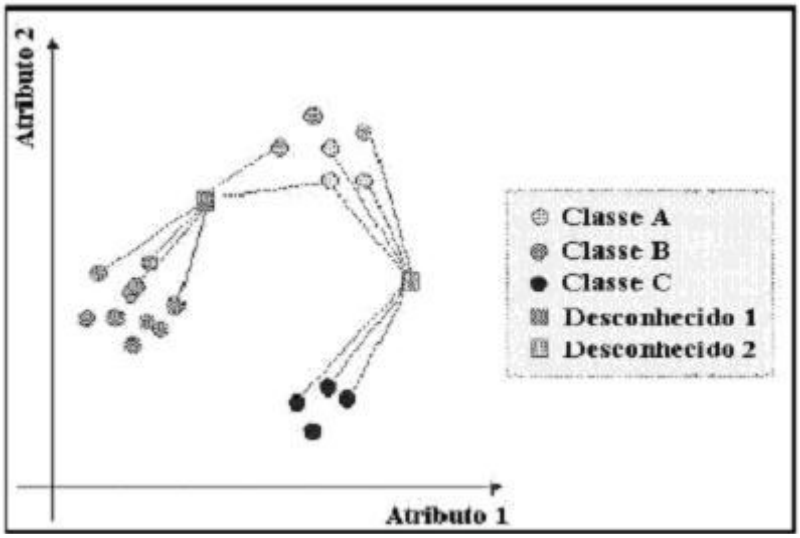


Figura 2.2. Classificação pelo método *kNN* [kNN-PUC 2007]

Aplicações de *kNN* para classificação de texto podem ser encontradas em [Tan 2006], [Yuan *et al*. 2005], dentre outros. Em [Yuan *et al*. 2005], uma versão modificada do algoritmo *kNN* foi aplicada na classificação de texto chinês. Antes, porém, algumas particularidades foram resolvidas: (1) a segmentação da palavra do texto chinês e o número de dimensões; (2) definição do valor apropriado de *k*

para assegurar alta precisão de classificação; (3) melhora da eficiência de classificação.

Para segmentar a palavra chinesa e reduzir o número de dimensões, foi utilizado um método para combinar técnicas baseadas em dicionário e baseadas em estatísticas. Este método compara uma string baseada no dicionário tema, e então segmenta palavras que estão no dicionário. Ao mesmo tempo, o método estatístico é usado para identificar as novas palavras que não existem no dicionário e adicioná-las a ele para a próxima segmentação de uma palavra do texto. Posteriormente, são descartadas as palavras que foram identificadas pelo dicionário, mas que tiverem uma freqüência (*TF-IDF* (*Term Frequency – Inverse Document Frequency*)) baixa de acordo com um limite estabelecido.

Para determinar o valor apropriado de *k*, no emprego do *kNN* para classificar texto chinês, foi utilizado um algoritmo genético [Holland 1975] para aprender o seu valor automaticamente.

Por fim, objetivando melhorar a eficiência da classificação, o modo de classificação gradual foi aplicado. Este método usa parte do texto em vez de analisá-lo todo para completar a classificação. Por exemplo, caso a classe do texto possa ser definida apenas pelo título, a classificação é finalizada. Caso contrário, o algoritmo continua analisando o abstract até um determinado ponto satisfatório ou até a análise do texto por completo, e uma classe será atribuída. Esse método melhora o desempenho do *kNN*, já que sua precisão é alta, mas sua eficiência é mais baixa.

Nos experimentos realizados com esta versão modificada do *kNN*, e executados sobre um conjunto de dados, ou *corpus*, composto por 520 artigos do *Computer World*, 30.9% dos textos puderam ser classificados utilizando, como informação, apenas o título e sua precisão foi de 89.9%; 31.8% puderam ser classificados utilizando o parágrafo chave, ou *abstract*, e sua precisão foi de 90.4%; e 37.0% dos textos puderam ser classificados utilizando o texto completo e sua precisão foi de 92.0%. Comparando o desempenho geral do método

tradicional de classificação de texto Chinês, que analisa o texto completo, com o desempenho do modo de classificação gradual, este último teve a precisão total de 90.57% e a média de tempo da classificação de 5 segundos, enquanto que o método de classificação tradicional teve 91.54% de precisão e um tempo médio de classificação de 27 segundos. Assim, o método de classificação gradual executou o processo de classificação em menos tempo e com praticamente a mesma precisão da classificação tradicional.

2.3.2.1 Análise do Algoritmo

O algoritmo *kNN* tem as seguintes características: é um método *lazy*, alto custo computacional para fornecer as classificações, não gera regras explícitas, requer definição de parâmetros (o valor de *k*), estável em relação aos dados de treinamento, sensível a atributos redundantes e irrelevantes, e é sensível a classes desbalanceadas.

2.3.3 Árvores de Decisão

Árvore de Decisão (AD) [Quilan 1986] [Quilan 1993] é um modelo estatístico que utiliza um treinamento supervisionado para classificação e previsão de dados, através do qual a função aprendida é representada por uma árvore de decisão. Entretanto, uma Árvore de Decisão também pode ser representada como um grupo de regras “se-então” para, assim, melhorar a compreensão humana [Mitchell 1997]. A Figura 2.3 mostra um exemplo de Árvore de Decisão.

Figura 2.3. Exemplo de uma Árvore de Decisão [Martins *et al*. 2007]

Árvores de decisão classificam instâncias escolhendo, do nó raiz para algum nó folha, o que fornece a classificação da instância. Cada nó na árvore especifica um teste de algum atributo da instância, e cada ramo descendente daquele nó corresponde a um dos possíveis valores para este atributo. A classificação de uma instância inicia no nó raiz da árvore, testando o atributo especificado por este nó, movendo, então, para o ramo da árvore correspondente para o valor do atributo no dado exemplo. Este processo é então repetido para a subárvore fixada ao novo nó [Mitchell 1997]. Os componentes de uma Árvore de Decisão e suas funções estão representados na Figura 2.4.

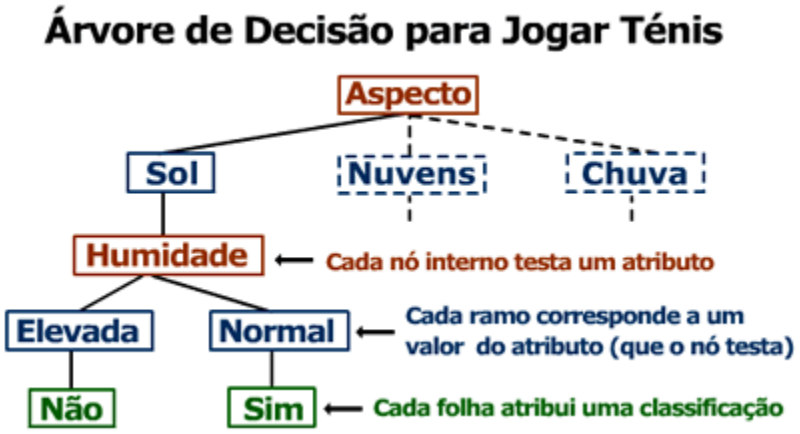


Figura 2.4. Componentes de uma Árvore de Decisão e suas funções [Martins *et al*. 2007]

Em [Stein *et al*. 2005], ADs são utilizadas com o objetivo de aumentar a taxa de detecção de intrusos em uma rede e diminuir a taxa de alarme falso. O sistema proposto em [Stein *et al*. 2005] possui um componente de pesquisa, que utiliza um algoritmo genético, e um componente de avaliação, que usa Árvores de Decisão.

No sistema, primeiramente, o módulo de pesquisa, que utiliza um algoritmo genético, é executado, e uma população inicial é randomicamente gerada. Todo indivíduo da população tem 41 genes, cada um dos quais representa uma característica do dado de entrada e pode ter atribuído o valor 1 ou 0. O valor 1 significa que a característica representada é usada durante a construção das Árvores de Decisão; e 0 significa que não é usada. Como resultado, cada indivíduo na população representa uma escolha das características disponíveis. Para cada indivíduo na população atual, uma Árvore de Decisão é construída utilizando o algoritmo C4.5 [Quilan 1993]. Esta Árvore de Decisão resultante é então testada sobre nove conjuntos de dados de validação, a qual gera nove taxas de erro de classificação. A aptidão deste indivíduo é a reunião destas taxas de erro de classificação. Quanto menor a taxa de erro de classificação, mais a apto é o indivíduo. Assim que os valores de aptidão de todos os indivíduos da população atual tiverem sido computados, o algoritmo genético inicia a criação da próxima geração da população.

O processo descrito é executado iterativamente até atingir o número máximo de gerações, que é 100. Finalmente, o melhor indivíduo da última geração é escolhido para construir o classificador de Árvore de Decisão final, que é testado sobre o conjunto de dados de teste.

Os experimentos, que tiveram como objetivo verificar se o sistema híbrido composto por Árvore de Decisão e algoritmo genético produzia melhor classificação que Árvore de Decisão sozinha, foram executados sobre o conjunto de dados KDDCUP 99[3], e mostraram que o sistema híbrido foi capaz de superar o

[3] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

desempenho da Árvore de Decisão sem a seleção de características. Além de demonstrar que a seleção que é feita no início do processo é capaz de melhorar as habilidades da Árvore de Decisão.

2.3.3.1 Análise do algoritmo

Árvores de Decisão têm as seguintes características: é um método *eager*, baixo custo computacional, gera regras explícitas, não requer definição de parâmetros, é instável em relação aos dados de treinamento, é sensível a atributos redundantes ou irrelevantes, e é sensível a classes desbalanceadas.

2.3.4 SVM

O algoritmo de *SVM* (*Support Vector Machine*) [Burges 1998] se baseia na idéia de encontrar um hiperplano ótimo que separe dois conjuntos de pontos linearmente separáveis no espaço. Ao se encontrar um hiperplano que separe esses dois conjuntos, a classificação de um novo ponto torna-se trivial, pois basta verificar em que região (a esquerda ou à direita do hiperplano) se encontra o novo ponto. No entanto, podem existir infinitos hiperplanos que separam dois conjuntos de pontos linearmente separáveis no espaço. É possível demonstrar que o hiperplano cuja margem para os pontos mais próximos (vetores suporte) é a maior, é o hiperplano que minimiza o risco de se classificar erroneamente um novo ponto. O desafio do algoritmo de *SVM* é, portanto, o de encontrar o hiperplano ótimo que tenha a maior margem para os pontos mais próximos a ele [Faria 2007] (ver Figura 2.5). Isso é feito através de técnicas de otimização com restrições (ver [Burges 1998]). Para encontrar hiperplanos em situações onde os conjuntos de pontos não são inteiramente separáveis (i.e. aproximadamente separáveis de forma linear), as *SVMs* permitem que pequenos erros sejam tolerados, atribuindo, no entanto, uma penalidade para as classificações incorretas.

No caso de pontos não linearmente separáveis, as *SVMs* realizam um mapeamento dos dados através de uma função de *Kernel*. A idéia é tentar mapear os pontos originais em um novo espaço onde eles sejam linearmente separáveis,

e em seguida encontrar um hiperplano ótimo. Diferentes funções de *Kernel* têm sido usadas em *SVMs*, como funções polinomiais (lineares e quadráticas), e funções *gaussianas*.

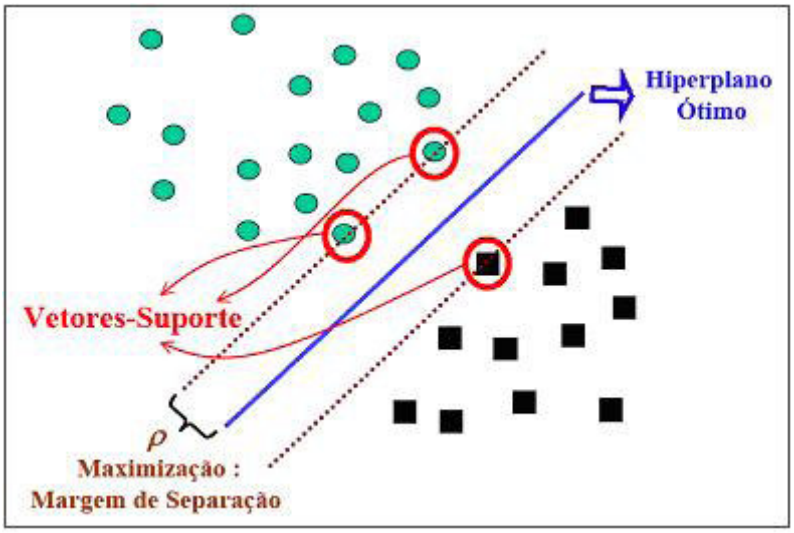


Figura 2.5. Hiperplano ótimo [Semolini 2007]

Em [Kim *et al*. 2005], [Du and Chen 2005], diversas aplicações de *SVM* para classificação de texto podem ser encontradas. As versões básicas do algoritmo SVM podem ser usadas diretamente na tarefa de classificação de texto. Entretanto, segundo [Du and Chen 2005], o uso de *SVMs* padrões para classificação de texto em conjuntos de treinamento com tamanhos de classe irregulares resultam em classificação tendenciosa em direção às classes com o maior freqüência no treinamento. A causa principal dessa observação é o fato de que a penalidade da classificação incorreta para cada exemplo de treinamento é considerada igualmente independente da distribuição das classes. Em [Du and Chen 2005], são propostos dois algoritmos *SVMs* ponderados, *C-SVM* e *V-SVM* ponderados, que determinam penalidades para os exemplos de treinamento pertencentes à mesma classe, inversamente proporcionais ao tamanho da classe. Isso compensa os efeitos indesejados causados por tamanhos de classes de treinamento irregulares, ou seja, de classes desbalanceadas.

O algoritmo *C-SVM* ponderado é similar ao *C-SVM* proposto por [Vapnik 1995], chamado de *C-SVM* padrão. No *C-SVM* padrão o parâmetro *C* (que consiste no parâmetro de penalidade) seja empiricamente selecionado, o mesmo valor de *C* é adotado, sem discriminação, para cada exemplo de treinamento. Enquanto que, no *C-SVM* ponderado, *C* terá diferentes valores de peso para cada amostra de treinamento, baseado na importância da amostra para a classificação, de modo que *C-SVM* ponderado possa separar corretamente estas amostras mais significativas, enquanto que algumas amostras de treinamento menos significativas podem tolerar classificação incorreta.

Já o V-*SVM* ponderado é similar ao algoritmo *V-SVM* criado por [Scholkopy *et al*. 2000], chamado de *V-SVM* padrão. Mas, diferente de *C-SVM* padrão, onde a escolha do parâmetro *C* não é intuitiva e depende altamente do problema em questão, o parâmetro *V* (que substitui o *C* do *C-SVM*), é selecionado de forma mais intuitiva, através de uma heurística. Entretanto, ele ainda possui o mesmo problema de tendência do *C-SVM* padrão, que aplica o parâmetro de penalidade de mesmo valor a todas as amostras de treinamento. De forma diferente, no *V-SVM* ponderado, cada amostra de treinamento possui o seu fator peso.

A aplicação do algoritmo *C-SVM* ponderado, assim como do *V-SVM* ponderado, melhora a precisão da classificação das classes minoritárias, mas possivelmente diminui a precisão da classificação das classes majoritárias, além da provável diminuição da precisão total da classificação. Testes executados sobre conjunto de dados do Repositório *UCI* de Aprendizado de Máquina utilizando tais algoritmos ponderados comprovaram este comportamento, mostrando que quando as classes minoritárias foram pesadas, a sua precisão aumentou, enquanto que a das classes majoritárias diminuiu.

2.3.4.1 Análise do algoritmo

O algoritmo *SVM* tem as seguintes características: é um método *eager*, alto custo computacional, não gera regras explícitas, requer definição de parâmetros, estável em relação aos dados de treinamento, pouco sensível a atributos redundantes ou irrelevantes, e é sensível a classes desbalanceadas.

2.4 Considerações Finais

Cada algoritmo de classificação possui particularidades, até mesmo os que pertencem à mesma família. E, da mesma forma que existem diversos classificadores diferentes, assim são as características dos dados que se deseja classificar. Desta maneira, é necessário conhecer ao menos algumas características dos dados que serão classificados, por exemplo, se estão desbalanceados ou não, se têm ruído ou não, para depois escolher o algoritmo para os classificar. De fato, segundo [Rezende 2005], não existe um único algoritmo que apresente o melhor desempenho para todos os problemas, sendo importante compreender o poder e a limitação dos diferentes algoritmos.

3 Métodos de Combinação de Classificadores

Combinação de classificadores tem como objetivo aumentar a precisão da classificação alcançada por apenas um classificador isoladamente. O conceito de combinação apareceu na literatura de classificação em 1965 [Nilsson 1965], e desde então tem sido tema de muitas pesquisas na área.

A idéia básica da combinação de classificadores é receber como entrada vários classificadores, gerando um novo classificador de saída, mais preciso do que os que foram combinados. Classificadores com comportamentos diferentes são combinados, de modo que o ponto fraco de um seja compensado pelo bom desempenho de outro.

Diferentes estratégias podem ser adotadas para combinar as respostas de diferentes classificadores. As mais freqüentemente usadas para a classificação de texto incluem:

(1) a Votação Majoritária (*Majority Voting*) [Li and Jain 1998], em que uma instância é classificada com a classe que recebe mais votos entre os classificadores sendo combinados; e

(2) a Combinação Linear Ponderada [Larkey and Croft 1996], em que os votos de cada classificador são ponderados por pesos que indicam a contribuição dos classificadores na resposta final.

Trabalhos em combinação de classificadores podem ser diferenciados ainda pela forma como os componentes da combinação são gerados. De uma forma geral, um comitê pode ser definido com: (1) classificadores homogêneos, quando foram gerados por um único algoritmo de Aprendizado de Máquina (AM), como em *Bagging* [Breiman 1996a] e Boosting [Iyer *et al*. 2000]; e (2) classificadores heterogêneos, em que os classificadores combinados foram gerados por mais de um algoritmo de AM, como ocorre comumente em *Stacking*

[Bennett *et al*. 2005] e *Meta Decision Trees* (*MDTs*) [Todorovski and Džeroski 2003].

Como visto, existem vários métodos de combinação de classificadores, como *Boosting*, *Bagging*, *Stacking*, *MDTs,* entre outros. Trabalhos na área de combinação de classificadores de texto têm apresentado bons resultados, como visto em [Dietterich 2000]. Entretanto, da mesma forma que não existe um algoritmo único que apresenta o melhor desempenho para todos os problemas de classificação [Rezende 2005], também não podemos apontar um método de combinação de classificadores que supere todos, sendo importante compreender o problema em mãos para escolher o método a ser utilizado.

Nesse capítulo, apresentamos alguns métodos de combinação de classificadores, em especial, os métodos usados mais comumente para classificação de textos, assim como algumas de suas variações. Na seção 3.1, apresentamos o método *Bagging*, na seção 3.2, *Boosting*, na seção 3.3, o método *Stacking*, e, na seção 3.4, apresentamos *MDT*. Finalmente, algumas considerações são feitas na seção 3.5.

3.1 *Bagging*

3.1.1 Descrição Básica do Método

Bagging [Breiman 1996a] é um método utilizado para combinar diversos classificadores de um mesmo algoritmo de aprendizado, ou seja, classificadores homogêneos. Esse método explora a instabilidade observada em alguns algoritmos de aprendizado, isto é, classificadores com comportamentos bastante distintos podem ser gerados a partir de pequenas variações do mesmo conjunto de treinamento. Classificadores gerados a partir de diferentes amostras de dados podem captar diferentes regularidades do problema de aprendizado sendo tratado. Neste caso, combinar tais classificadores poderia trazer um ganho na precisão na classificação.

Na Figura 3.1, é ilustrado o funcionamento geral do método de *Bagging*. Inicialmente, *T* amostras $(B_1, B_2, ..., B_T)$ são selecionadas aleatoriamente do conjunto de treinamento disponível. Em seguida, para cada amostra, o algoritmo em questão é usado para aprender um classificador diferente. Assim, *T* classificadores $(C_1, C_2, ..., C_T)$ são construídos a partir da aplicação do algoritmo. Finalmente, um classificador final C_f é construído através da combinação dos classificadores $C_1, C_2, ..., C_T$.

A predição final de C_f para uma nova instância *x* é comumente definida por Votação Majoritária, isto é, observando a classe predita com maior freqüência entre os classificadores sendo combinados. Geralmente, predições feitas por votação tornam-se mais confiáveis quanto mais votos recebem [Witten and Frank 1999]. *Bagging* pode também ser aplicado para predição numérica. A única diferença é que, sendo números reais as predições individuais, em vez de votação sobre o resultado, é calculada a média [Witten and Frank 1999].

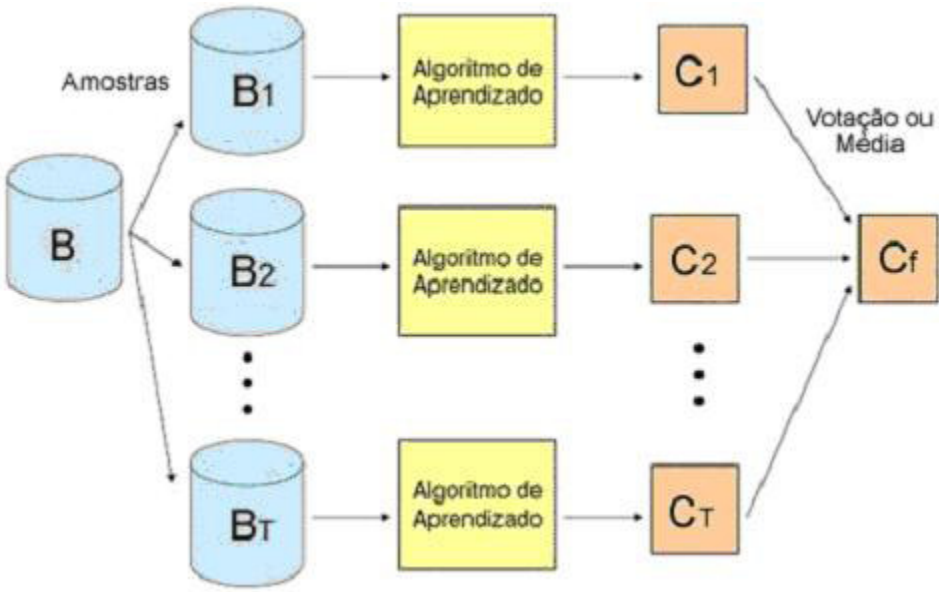


Figura 3.1. Ilustração do método *Bagging* [Zadrozny 2006]

3.1.2 Variações

O sistema *Bagging Ensemble Variant* (*BEV*) [Li 2007], uma variação do *Bagging* combinado às idéias de balanceamento de dados *under-sampling*, foi desenvolvido para classificar dados desbalanceados, ou seja, que possuem mais instâncias em algumas classes do que em outras. O desbalanceamento desfavorece especialmente os dados de classes minoritárias que, em muitos casos, são as classes prioritárias de serem classificadas corretamente.

A geração das amostras de dados que serão usadas para gerar os classificadores a serem combinados é apresentada como se segue. Suponha que existam duas classes no problema: uma classe minoritária com M_t instâncias, e outra majoritária com M_{nt} instâncias. O método *BEV* gera amostras de instâncias balanceadas, através da união do conjunto minoritário com um sub-conjunto de instâncias selecionadas da classe majoritária. Inicialmente, o conjunto de instâncias da classe majoritária é dividido em $T = [M_{nt}/M_t]$ subconjuntos de tamanho M_t. Em seguida, são geradas T amostras de treinamento, onde cada amostra é formada pela união de um subconjunto da classe majoritária e o conjunto inteiro da classe minoritária. Isto resulta em uma amostra de instâncias que contém o mesmo número de instâncias para as duas classes consideradas, isto é, resulta em um conjunto balanceado. Assim, o método usa todos os dados da classe minoritária, e, para originar conjuntos de treinamento diferentes e balanceados sem gerar dados artificiais, complementa com dados da classe majoritária.

Após a geração de T amostras de treinamento balanceadas, um classificador é induzido a partir de cada um deles, e os T classificadores gerados são combinados como no *Bagging* padrão. A Figura 3.2 ilustra os passos envolvidos nesta técnica.

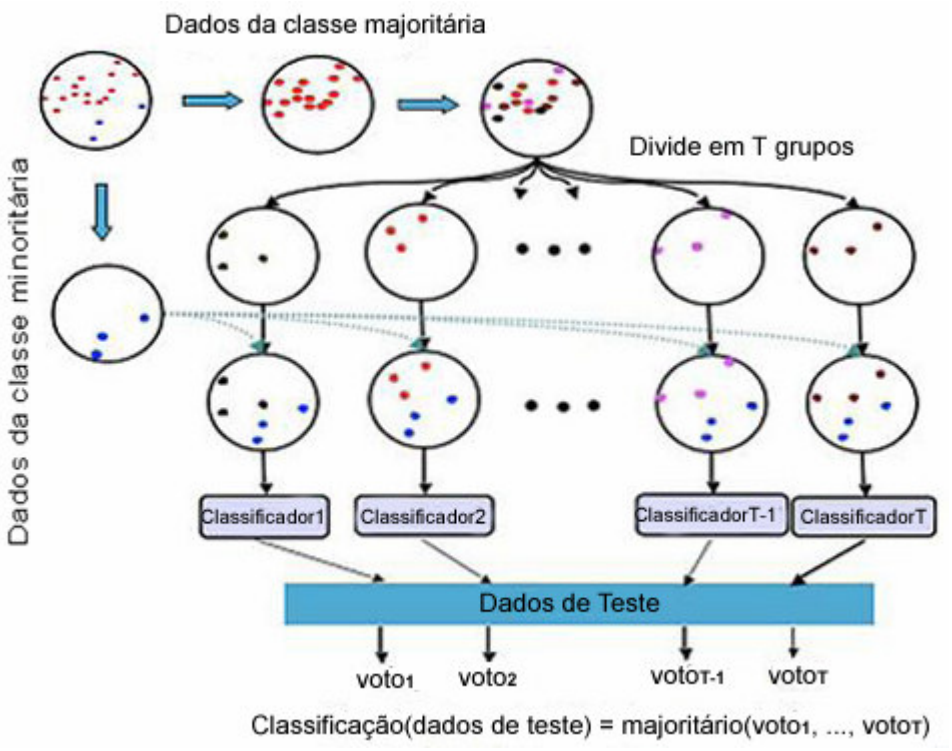


Figura 3.2. O método BEV para classificação de dados desbalanceados [Li 2007]

Um exemplo de aplicação do *BEV* pode ser visto em [Li 2007]. Nos experimentos, os algoritmos C4.5 [Quinlan 1993], da família das Árvores de Decisão, e o SVM_{light} [Joachims 1999], da família *Support Vector Machine*, foram aplicados sobre dados de inspeção de rodas de estrada de ferro para gerar os classificadores base. Uma pequena parte das rodas foi rotulada como ”falha”, e a outra como ”segura”. Tanto o algoritmo C4.5 quanto o SVM_{light} tiveram um desempenho ruim com os dados desbalanceados, classificando todos os dados que pertenciam à classe ”falha” como pertencentes à classe ”segura”.

Após o balanceamento dos dados com *BEV*, foram gerados $T = 11$ subconjuntos de treinamento, onde sobre cada subconjunto, um classificador foi treinado. Para comparação do desempenho dos classificadores base com o desempenho do sistema *BEV*, as seguintes medidas foram analisadas: precisão, sensibilidade (precisão na identificação de instâncias positivas, no caso, rodas

com falhas), especificidade (precisão na identificação de instâncias negativas, no caso, rodas sem falhas, ou seguras) e *gMeans* (estimativa geral, consistindo da raiz quadrada da sensibilidade vezes a especificidade).

Considerando o C4.5 para gerar os classificadores base, o sistema *BEV* teve precisão maior que a de 8 classificadores, sensibilidade maior que a de 9, especificidade maior que a de 7, e *gMeans*, com o valor 0.802, maior que a de todos os 11 classificadores obtidos a partir dos 11 conjuntos de treinamento. Considerando, agora, o *SVM* para gerar os classificadores base, o sistema *BEV* teve precisão maior que a de 9 classificadores, sensibilidade maior que a de todos, especificidade maior que a de 9, e *gMeans* também maior que a de todos, com o mesmo valor do experimento anterior, 0.802.

Diante dos resultados expostos, constata-se que o método *BEV* possui um desempenho muito satisfatório, por ter superado a maioria dos resultados alcançados pelos classificadores base, principalmente com *SVM*. Destacando, ainda, a obtenção do maior valor de *gMeans* nos dois experimentos, já que esta medida dá uma estimativa geral do desempenho na classificação.

3.2 *Boosting*

3.2.1 Descrição Básica do Método

O método *Boosting* [Schapire 2002] também é usado para combinar diversos classificadores homogêneos. Um ponto em comum entre esse método e *Bagging* é que ele também combina classificadores gerados a partir do mesmo algoritmo, usando partições diferentes dos dados de treinamento. Porém, *Boosting* é iterativo. Enquanto que em *Bagging* classificadores individuais são construídos a partir de amostras selecionadas de forma independente, em *Boosting* cada amostra de instâncias é selecionada considerando o desempenho dos classificadores construídos anteriormente [Witten and Frank 1999]. Cada instância é selecionada para uma amostra *B* com base em um peso (ou probabilidade) que varia conforme a precisão do classificador anterior para a instância.

Há diversos algoritmos que implementam o método *Boosting*, como exemplo o algoritmo *AdaBoost.M1* [Freund and Schapire 1995], que é aplicado nas tarefas de classificação. Neste algoritmo, inicialmente é atribuído o mesmo peso a todas as instâncias no conjunto de treinamento (isto é, todas as instâncias têm a mesma probabilidade de serem selecionadas na primeira iteração). A partir da amostra de dados gerada, um classificador é construído. Em seguida, o classificador é usado para classificar todas as instâncias de treinamento e, posteriormente, o peso das instâncias é modificado. As instâncias que foram classificadas erroneamente têm o seu peso aumentado, enquanto que as que foram classificadas corretamente têm o seu peso diminuído.

A equação 3.1 mostra o cálculo do erro do classificador, onde *I* é o indicador da função que recebe o valor 1 ou 0, dependendo se c_t classificou a *i*-ésima instância de forma errada ou não. A forma como os pesos são atualizados (ver equação 3.2, onde α é o peso da resposta de um classificador, c_t é o classificador, x_i é uma instância, v_i é uma classe, D_t é a distribuição da probabilidade, e Z_t é fator de normalização do peso) depende do desempenho global obtido pelo classificador no conjunto de instâncias. Assim, a cada iteração, as instâncias de peso maior terão maior probabilidade de serem selecionadas [Witten and Frank 1999].

$$\epsilon_t = \sum_{i=1}^{N} I(c_t(x_i) \neq v_i) D_t(i) \tag{3.1}$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha t}, & if\ c_t(x_i) = v_i \\ e^{\alpha t}, & if\ c_t(x_i) \neq v_i \end{cases} = \frac{D_t(i)\exp(-\alpha_t v_i c_t(x_i))}{Z_t} \tag{3.2}$$

Durante a classificação de uma nova instância, cada classificador recebe um peso para sua resposta, através do parâmetro α (equação 3.3). Assim, a resposta final será uma combinação linear ponderada das respostas individuais fornecidas pelos classificadores (equação 3.4).

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \tag{3.3}$$

$$H(x) = sign\left(\sum_{t=1}^{T}\alpha_t c_t(x)\right) \tag{3.4}$$

3.2.2 Variações

Em [Zhang and Zhang 2008], é apresentado um algoritmo de *Boosting* local, o *LocalBoost*. Este algoritmo é baseado na versão do *AdaBoost* [Freund and Schapire 1995], que, nesse trabalho, é referenciado como *Boosting* global, ou *GlobalBoost*.

Como o desempenho de um classificador sobre uma instância particular está geralmente muito relacionado ao seu desempenho sobre os vizinhos daquela instância, o algoritmo *LocalBoost* propõe modificar os pesos das instâncias, a cada iteração, considerando o desempenho obtido por um classificador apenas na vizinhança da instância. Como visto, no *Boosting* global o peso de cada instância é modificado considerando o desempenho do classificador em todas as instâncias de treinamento, independente da relação de vizinhança da instância.

No *LocalBoost*, em cada iteração do método, para toda instância de treinamento *i (i = 1, 2, ..., N)*, o erro do classificador (equação 3.5, onde *N(i)* representa a vizinhança da instância, e o *I* é o indicador da função) é calculado na vizinhança da instância e usado para atualizar a probabilidade da instância ser selecionada nas próximas iterações (equação 3.6, onde o parâmetro β controla a importância da informação da vizinhança). A vizinhança de uma instância pode ser definida, como no algoritmo de *kNN*, a partir da distância *Euclidiana*, ou alternativamente com outras funções de distância.

$$\epsilon_t(i) = \sum_{j\in N(i)} I(c_t(x_j) \neq v_j) D'_t(j), \quad onde \quad D'_t(j) = \frac{D_t(j)}{\sum_{l\in N(i)} D_t(l)'} \tag{3.5}$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t(i)/\beta}, & if\ c_t(x_i) = v_i \\ e^{\alpha_t(i)/\beta}, & if\ c_t(x_i) \neq v_i \end{cases} = \frac{D_t(i)\exp(-(\frac{\alpha_t(i)}{\beta})v_i c_t(x_i))}{Z_t} \quad (3.6)$$

Após o treinamento de um classificador, cada instância de treinamento passa a ter um coeficiente α próprio (equação 3.7), que depende da sua vizinhança. No *Boosting* global, por outro lado, um valor único de α era considerado para todas as instâncias. Na fórmula de atualização das probabilidades do *LocalBoost*, mostrado na equação 3.6, o parâmetro β controla o quanto a informação da vizinhança será considerada.

$$\alpha_t(i) = \frac{1}{2}\ln\left(\frac{1-\epsilon_t(i)}{\epsilon_t(i)}\right) \quad (3.7)$$

Durante a classificação de uma nova instância, a vizinhança também é levada em consideração. A importância do classificador considera a soma do valor α das instâncias da vizinhança ponderado pela distância para a instância *x* a ser classificada. Quanto maior o valor desta medida, mais influência a *i*-ésima instância de treinamento terá sobre a nova instância. A equação 3.8 apresenta a fórmula para o cálculo desta medida.

$$H(x) = sign\left(\sum_{t=1}^{T}\left(\sum_{i=1}^{N}\frac{\alpha_t(i)}{d^2(x,x_i)}\right)c_t(x)\right) \quad (3.8)$$

Nos experimentos realizados com *benchmarks* artificiais e utilizando um algoritmo da família das Árvores de Decisão para gerar os classificadores base, foram determinados diferentes valores para a quantidade de instâncias em cada vizinhança, *N(i)*, e para o parâmetro β. Verificou-se que a taxa de erro da classificação aumentava à medida que o valor de β aumentava. Assim, um valor menor para o parâmetro β resulta em taxas de erro menores. Por outro lado,

constatou-se que a quantidade de instâncias de treinamento em cada vizinhança teve pouca influência na taxa de erro. Desta forma, pode-se reduzir o custo computacional com a escolha de menos instâncias para as vizinhanças ao custo de uma perda limitada da precisão da classificação.

O desempenho do *LocalBoost* foi comparado ao desempenho de vários métodos de classificação, sendo eles: *SingleTree* [Breiman *et al*. 1984], *Bagging*, *RandomForest* [Breiman 2001] e *Boosting*, chamado aqui de *GlobalBoost*. Foram executados 100 testes, com cada algoritmo combinando 50 Árvores de Decisão em cada teste, e computado o erro em cada um deles. O *LocalBoost* apresentou desempenho superior em 81 dos 100 testes executados. Em média, a taxa de erro do *LocalBoost* foi 0.86% maior que o melhor desempenho alcançado pelos outros algoritmos nos testes, considerando que *GlobalBoost*, *RandomForest* e *Bagging* obtiveram, respectivamente, o melhor desempenho em 13, 2 e 4 dos testes, e suas taxas de erro foram, em média, respectivamente, 15.45%, 32.88% e 25.48% maiores que o melhor desempenho alcançado nos experimentos. O algoritmo *SingleTree* não obteve o melhor desempenho em nenhum dos testes, e, em média, ele foi 55.48% pior que o melhor resultado alcançado nos testes.

Alguns testes, utilizando os mesmos *benchmarks* artificiais e algoritmos, já citados, para gerar os classificadores base, também foram executados para analisar a robustez do *LocalBoost* na classificação de dados com diferentes níveis de ruído (dados imperfeitos, como instâncias com os mesmos valores de atributos mas com classes diferentes). Nestes testes, as maiores taxas de precisão também foram alcançadas pelo *LocalBoost*. Quando o nível de ruído nos dados é menor, ele tem melhor desempenho do que os outros métodos.

Embora o *LocalBoost* se comporte levemente pior que *Bagging* e *RandomForest* quando o nível de ruído é relativamente grande, a diferença entre eles não é significante. Portanto, estes resultados mostram que na prática o *LocalBoost* pode ser a melhor opção se nenhuma informação sobre o nível de ruído nos dados está disponível.

Para a execução de testes utilizando vários *benchmarks* reais disponíveis no repositório *UC*I, o *LocalBoost* foi estendido da versão de classificação binária para uma versão para resolver problemas multi-classes, usando uma idéia similar ao do *Adaboost.M1*. Nestes testes, o *LocalBoost* também apresentou ótimos resultados, superando, na maioria das vezes, o desempenho dos outros métodos de combinação citados anteriormente.

3.3 *Stacking*

3.3.1 Descrição Básica do Método

Stacked generalization ou *Stacking* [Wolpert 1992] é uma técnica de combinação de classificadores heterogêneos, e a sua arquitetura é baseada em camadas. Cada classificador base (ou de nível-0) é treinado sobre um conjunto de dados. Após o treinamento, o conjunto de dados é modificado, sendo estendido para incluir as predições destes classificadores. Camadas sucessivas recebem como entrada as predições da camada imediatamente anterior e a saída é passada para a próxima camada. Então, um único classificador no nível mais alto produz a predição final [Vilalta *et al*. 2005] (ver Figura 3.3).

A maioria das pesquisas nesta área trabalha com duas camadas [Wolpert 1992], [Breiman 1996b], [Chan and Stolfo, 1998], [Ting and Witten 1997]. Assim, um classificador de nível-meta (ou nível-1) é aprendido baseado na saída dos classificadores de nível-base (ou nível-0), estimada via validação cruzada [Sigletos *et al*. 2005].

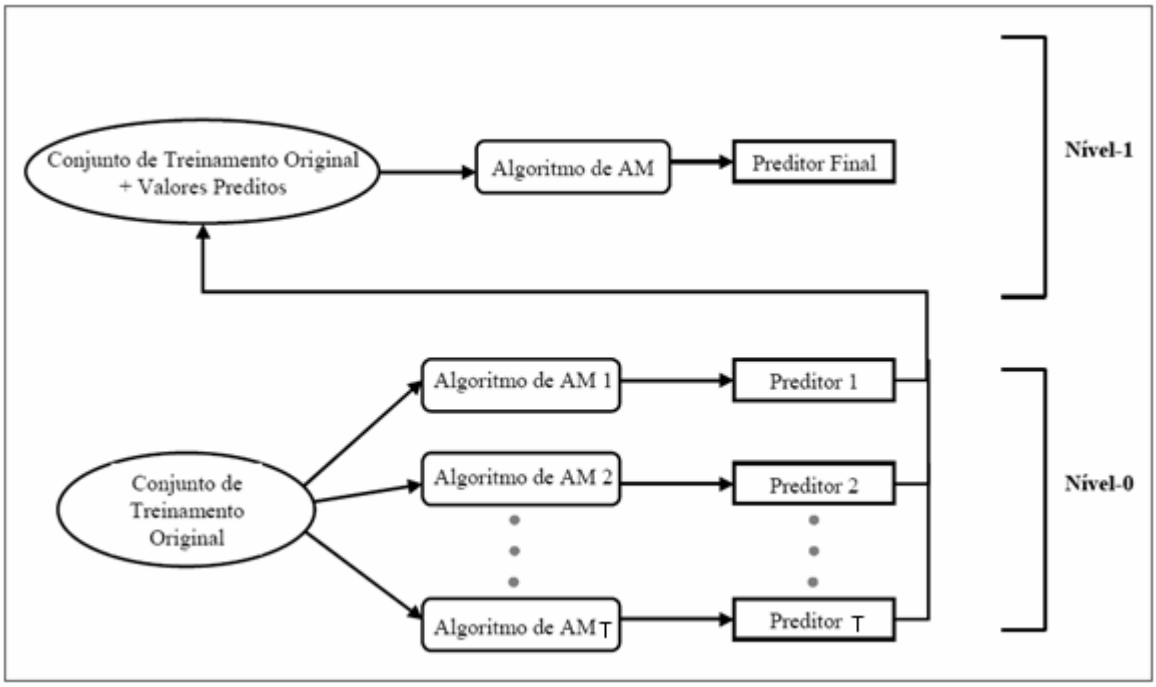


Figura 3.3. Funcionamento do *Stacking* [Sinoara *et al*. 2003]

Mais especificamente, inicialmente, um conjunto de dados *D* é definido, como dados do nível-0, e um conjunto de *T* algoritmos de aprendizado diferentes. Durante o processo de validação cruzada, o conjunto de dados *D* é randomicamente dividido em *J* partes (*folds*), $D_1, ..., D_J$. Em cada passo $j = 1, ..., J$, são separados *J*-1 folds para indução de T classificadores $C_1(j)... C_T(j)$, e o *fold* restante D_j é separado para teste. As predições concatenadas dos classificadores induzidos a partir de D_j, junto com o valor da classe original, formam um novo conjunto de dados do nível-meta. MD_j [Sigletos *et al*. 2005].

No final do processo de validação cruzada, a união dos dados de nível meta formados a partir de cada *fold* constitui o conjunto de dados completo do nível-meta, também referenciado como dados do nível-1, o qual é usado para aplicar um algoritmo de aprendizado L_M e induzir o classificador de nível-meta C_M. O algoritmo de aprendizado L_M que é empregado no nível-meta pode ser um dos algoritmos $AM_1, ..., AM_T$ ou um diferente [Sigletos *et al*. 2005].

Finalmente, após a formação do conjunto de dados do nível-meta no processo de validação cruzada, os algoritmos de aprendizado $AM_1, ..., AM_T$ são aplicados no conjunto de dados do nível-0 inteiro para induzir os classificadores de nível-base finais $C_1, ..., C_T$ que serão utilizados em tempo de execução [Sigletos *et al*. 2005].

Para fazer a classificação de uma nova instância, as predições concatenadas de todos os classificadores de nível-base $C_1, ..., C_T$ formam um vetor de nível-meta ao qual é determinado um valor de classe pelo classificador de nível-meta C_M [Sigletos *et al*. 2005]. A Figura 3.4(a) ilustra o processo de validação cruzada dos dados, enquanto que a Figura 3.4(b) representa a execução de *Stacking*.

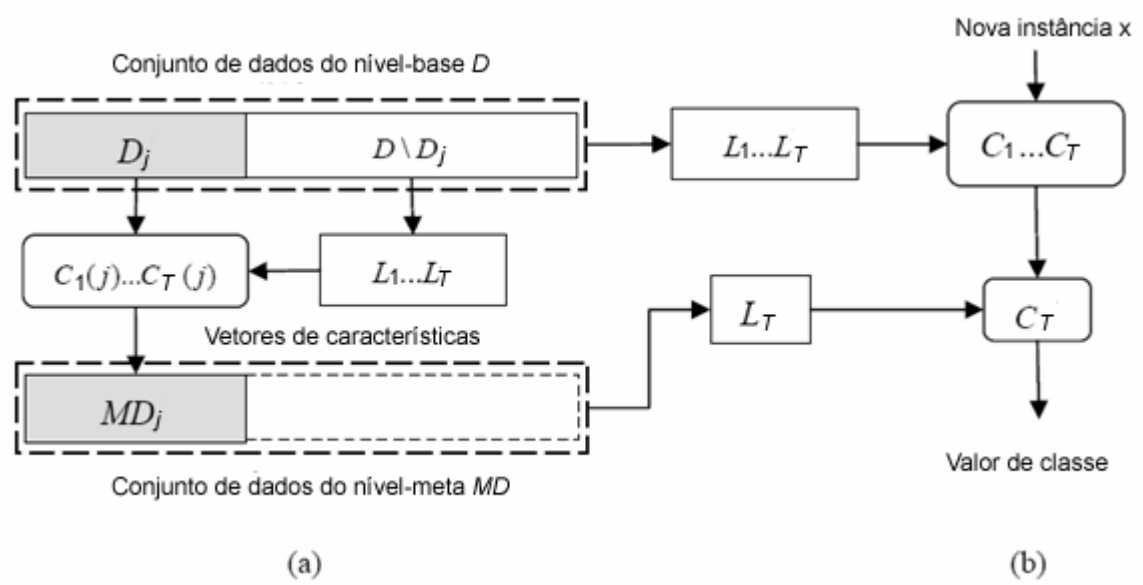


Figura 3.4. (a) Processo de validação cruzada para criar o conjunto de dados do nível-meta; (b) Execução do algoritmo *Stacking* [Sigletos *et al*. 2005]

3.3.2 Variações

[Wang and Zhao 2008] expõe algumas dificuldades enfrentadas na realização do reconhecimento de Entidades Nomeadas (*NE*, do inglês *Named Entity*) do domínio de Bioinformática. E, que por causa disto, esta tem se tornado a tarefa mais fundamental na descoberta de conhecimento biomédico [Wang and Zhao 2008].

Em [Wang and Zhao 2008], é proposto o método *Bag-Stacking*, para auxiliar no reconhecimento de *NEs*. *Bag-Stacking* é a variação homogênea de *Stacking*, e consiste da combinação dos conceitos de *Bagging* e de *Stacking*. Como acontece em *Bagging*, o conjunto de treinamento é dividido em *T* amostras, e sobre cada amostra é aplicado um algoritmo de classificação que irá gerar um classificador (classificador de nível-0 ou classificador de nível-base) referente a cada amostra. O método *Bagging* usa Voto Majoritário para combinar as saídas da classificação dos classificadores. O *Bag-Stacking*, por sua vez, usa um algoritmo de nível mais alto para fazer a combinação dos classificadores base.

Para a realização de testes dos classificadores base, o conjunto *D* inteiro é utilizado. No final do processo de testes, o conjunto de dados *D*, que será utilizado pelo classificador de nível-meta M_{Stack}, pode ser composto, dependendo da escolha, pelo valor original da classe, pelos valores das classes preditas pelos classificadores de nível-base $(C_t(x),\ t = 1, 2, ..., T)$, pelo grau de confiança das classes preditas *(confidencet(x))*, e pelos vetores de atributos das instâncias *(attrvec(x))*. Este novo conjunto é chamado de dados do nível-1 ou nível-meta, sobre o qual será aplicado o classificador de nível-meta, que fará a predição final. Este processo é ilustrado na Figura 3.5.

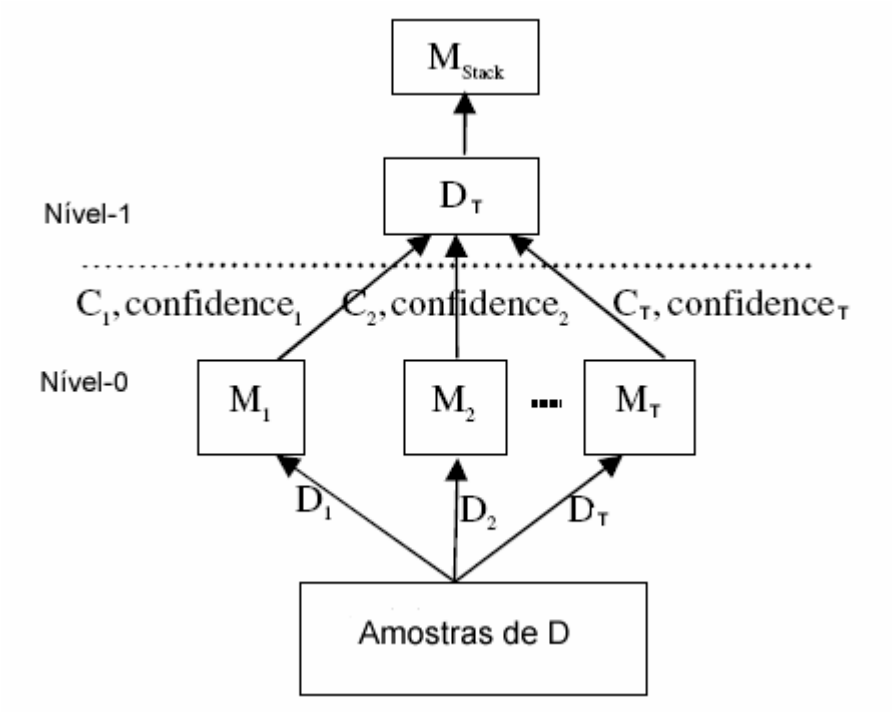


Figura 3.5. Classificadores homogêneos combinados usando Bag-Stacking [Wang and Zhao 2008]

O método de *Bag-Stacking* foi avaliado de três formas diferentes em [Wang and Zhao 2008]: (1) *Class-bag-stacking*, em que as instâncias do nível meta são formadas da classificação correta e das classes preditas pelos classificadores base; (2) *Class-attribute-bag-stacking*, em que as instâncias são formadas da classificação correta, das classes preditas pelos classificadores base, juntamente com os vetores de características originais; e (3) *Class-confidence-attribute-bag-stacking*, em que as instâncias são compostas da classificação correta, das classes preditas, do grau de confiança das classes preditas pelos classificadores base e dos vetores de características. Os autores ainda avaliaram *Stacking* padrão com as configurações: (1) *Class-stacking*, com instâncias formadas da classificação correta e das classes preditas pelos classificadores base; e (2) *Class-attribute-stacking*, instâncias formadas da classificação correta e classes preditas pelos classificadores base com a adição dos vetores de características.

Nos experimentos, foram utilizados conjuntos de dados do JNLPBA 2004[4]. Os dados foram rotulados com cinco classes: *Protein, DNA, RNA, Cell_line* e *Cell_type*. Precisão, Cobertura e *F-Score* foram as medidas utilizadas para a avaliação do desempenho dos sistemas de classificação. Os algoritmos de classificação individuais empregados foram o *Generalized Winnow* (GW) [Zhang *et al*. 2002], *ConditionalRandom Fields* (CRFs) [Lafferty *et al*. 2001], *SVM* e *Maximum Entropy* (ME) [Berger *et al*. 1996].

Segundo [Wang and Zhao 2008], os resultados dos experimentos mostraram que as técnicas desenvolvidas obtiveram ótima performance. A melhor estratégia de combinação, que foi a *Class-attribute-stacking*, alcançou *F-score* de 77,57%, superando os melhores resultados já publicados. Demonstrando, assim, que são adequadas para o reconhecimento de *NEs*.

3.4 MDT

3.4.1 Descrição Básica do Método

A estrutura de uma Árvore de Decisão Meta (*MDT*, do inglês *Meta Decision Tree*) [Todorovski and Dzeroski 2000] é idêntica à estrutura de uma Árvore de Decisão Comum (*ODT*, do inglês *Ordinary Decision Tree*) [Todorovski and Džeroski 2003], como a árvore de decisão C4.5. Mas a forma de classificação é diferente. Na *MDT*, um nó (interno) de decisão especifica um teste para ser feito sobre o valor de um único atributo e cada resultado do teste tem seu próprio ramo que conduz à subárvore apropriada. Em um nó-folha, uma *MDT* prediz qual classificador deve ser usado na classificação de uma instância, em vez de predizer diretamente o valor da classe, como ocorre em uma *ODT* [Todorovski and Džeroski 2003]. Esta é a característica principal da *MDT* que a diferencia de uma *ODT* [ZENKO *et al*. 2001].

Na indução de *MDTs*, dois tipos de atributos são usados: atributos comuns e atributos-classe. Os atributos comuns são usados nos nós (internos) de decisão.

[4] http://research.nii.ac.jp/~collier/workshops/JNLPBA04st.htm

O papel destes atributos é idêntico ao papel dos atributos usados para induzir *ODTs*. Atributos-classe são usados somente nos nós-folha. Cada algoritmo de aprendizado de nível-base tem seu atributo-classe: os valores do atributo-classe são iguais às predições obtidas pelo algoritmo de nível-base. Dessa forma, o atributo-classe determinado no nó-folha da *MDT* decide qual algoritmo de nível-base deve ser usado para a predição [Todorovski and Dzeroski 2000].

Atributos em *MDTs* são propriedades das distribuições da probabilidade da classe predita para uma dada instância, pelos algoritmos de nível-base. Suponha que o classificador de nível-base C_L gerado com o algoritmo de aprendizado *AM* retorne a distribuição da probabilidade p_L, quando aplicado à instância x: $p_L(x) = (p_L(1)(x), p_L(2)(x), ..., p_L(m)(x))$, onde m é o número de classes. O k-ésimo elemento neste vetor representa a probabilidade que a instância x tem de pertencer à respectiva classe quando estimada pelo classificador C_L. A classe com a maior probabilidade da classe $p_L(*)$ é predita pelo classificador de nível-base C_L [Todorovski and Dzeroski 2000].

Três propriedades das distribuições da probabilidade da classe são usadas como atributos em *MDTs*: (1) *maxprob* - o maior valor da distribuição de probabilidade; (2) *entropy* - a entropia da distribuição da probabilidade da classe; e (3) *weight* - a fração dos exemplos de treinamento usados para estimar a distribuição da classe para a instância *x*. O atributo weight não se aplica aos classificadores *naive Bayes* e *k-NN* [Todorovski and Dzeroski 2000].

A Tabela 3.1 mostra uma *MDT* onde o nó-folha representado por *(*)* especifica que o algoritmo *LTree* deve ser usado para classificar uma instância se a *entropy* da distribuição da probabilidade retornada por ele for menor que 0.38 e a *maxprob* na distribuição da probabilidade retornada pelo algoritmo *k-NN* for menor que 0.75. Os atributos *entropy* e *maxprob* de uma distribuição da probabilidade podem ser interpretados como estimativas da confiabilidade de um classificador em sua predição. Se a distribuição da probabilidade retornada é altamente dispersa, o valor de *maxprob* será baixo e o de *entropy* será alto,

indicando que a predição do classificador não é muito confiável. Por outro lado, se a distribuição da probabilidade retornada está altamente focada em uma dada classe, a *maxprob* é alta e a *entropy* baixa, indicando, assim, que a predição do classificador é confiável. E a propriedade *weight* quantifica quão confiável é a estimativa do modelo sobre sua própria confiabilidade [Todorovski and Dzeroski 2000].

Tabela 3.1 Uma *Meta Decision Tree* [Todorovski and Dzeroski 2000]

```
ltree_entropy <= 0.37699:
|       knn_maxprob <= 0.75079: LTREE (*)
|       knn_maxprob > 0.75079: KNN
ltree_entropy > 0.37699:
|       knn_entropy > 1.49841: KNN
|       knn_entropy <= 1.49841:
|       |       c45_weight <= 0.11388: LTREE
|       |       c45_weight > 0.11388:
|       |       |       c45_maxprob <= 0.95: LTREE
|       |       |       c45_maxprob > 0.95: C45
```

Os experimentos com *MDT* foram executados sobre vinte e um conjuntos de dados do *UCI Repository of Machine Learning Databases and Domain Theories*. Os resultados do desempenho das *MDTs* foram comparados ao desempenho de *Voting* e *Stacking* com *ODTs*, com o desempenho de *Boosting* e *Bagging* e com o de outros métodos.

Comparando com as ODTs, os classificadores combinados com as *MDTs* alcançaram, de modo geral, precisão significantemente melhor em 15 conjuntos de dados e significantemente pior em 2. As *MDTs* tiveram um desempenho levemente superior em termos de precisão média. Entretanto, há o fato importante de que as *MDTs* são muito menores que as *ODSTs*, sendo que as *ODTs* ainda são posteriormente podadas.

MDTs em comparação ao *Voting* é significantemente melhor em 10 domínios e significantemente pior em 6. Porém, a melhora é muito maior que a queda de precisão, dando uma melhora de precisão geral de 22%. De modo geral, os resultados mostraram que *MDTs* tiveram desempenho superior quando comparado com o método de combinação de classificadores *Voting*.

Finalmente, o desempenho de *MDTs* quando comparado com o de *Boosting* se mostrou significantemente melhor em 13 conjuntos de dados, e significantemente melhor em 10 em relação ao desempenho de *Bagging*. *MDTs* tiveram desempenho significantemente pior que *Boosting* em 6 e significantemente pior que *Bagging* em 4, somente. De maneira geral, a melhora relativa é de 9% sobre *Boosting* e 22% sobre *Bagging*.

Os resultados apresentados do desempenho de *MDTs* quando comparado com os desempenhos de algoritmos de combinação do estado-da-arte servem para mostrar que *MDTs* são competitivas com estas técnicas.

3.5 Considerações Finais

Diante da apresentação, neste capítulo, de alguns métodos de combinação e de algumas das suas variações, é possível observar pontos positivos e negativos inerentes a estes métodos.

As *MDTs* são muito menores que as *ODTs* [Todorovski and Džeroski 2003], mesmo sem serem posteriormente podadas como acontece com as *ODTs*, sendo, assim, mais fáceis de compreender. Geralmente alcançam desempenho superior quando comparadas com outros métodos de combinação, como *Bagging* e *Boosting*. Agregando, desta forma, fácil compreensão e bom desempenho, ao contrário de *Bagging*, *Boosting* e *Stacking* que, apesar de conseguirem precisão alta, são difíceis de se analisar [Witten and Frank 2005].

O método *Stacking*, assim como as *MDTs*, é um método de combinação heterogênea, e tem a característica de tentar utilizar o máximo de informações

possíveis dos dados e explorar da melhor maneira as características diversas dos seus classificadores heterogêneos. Entretanto, pelo fato da sua análise teórica ser mais difícil que a de *Bagging* e *Boosting* é uma técnica bem menos utilizada [Witten and Frank 2005].

O método *Bagging*, que gera os conjuntos de treinamento de forma independente, possibilita, desta forma, a criação dos classificadores em paralelo, fazendo com que o processo de classificação seja mais rápido. Por outro lado, as instâncias mal classificadas não recebem um tratamento especial, como ocorre no *Boosting*, que procura focar o aprendizado em instâncias que foram mal classificadas.

Boosting geralmente produz classificadores que são significantemente mais precisos sobre dados puros, sem ruídos, do que os classificadores gerados por *Bagging*. Porém, diferente de *Bagging*, *Boosting* algumas vezes falha em situações práticas, podendo gerar um classificador que é significantemente menos preciso que um único classificador gerado dos mesmos dados. Indicando, assim, que o classificador combinado sofreu *overfitting*, isto é, que o classificador superajustou-se ao conjunto de treinamento [Witten and Frank 1999].

Outra característica positiva de *Bagging* é a sua tentativa de neutralizar a instabilidade dos algoritmos de classificação, trabalhando bem com algoritmos instáveis, como Árvores de Decisão (principalmente as sem poda, pois a poda aumenta a estabilidade), mas sem conseguir dar muita contribuição para o aumento de precisão de algoritmos estáveis, como o *kNN* [Witten and Frank 1999].

Quando o desempenho é analisado na classificação de dados desbalanceados, os métodos de combinação, de modo geral, não apresentam resultados satisfatórios [Zhang and Zhang 2008]. O sistema *BEV*, variação de *Bagging*, tenta resolver este problema, balanceando os dados antes de gerar os classificadores que serão combinados.

O método *LocalBoost* é outra extensão do *Boosting* Padrão que tem apresentado resultados promissores. Entretanto, o LocalBoost não faz qualquer tratamento para contornar o problema de dados não-balanceados. O sistema *B-LocalBoost* (abordado detalhadamente no próximo capítulo), que foi proposto e implementado, é uma extensão do *LocalBoost* que tem como objetivo tratar a queda de desempenho do método perante dados desbalanceados, que, segundo [Zhang and Zhang 2008], também é um problema apresentado em *Boosting*.

4 Sistemas Inteligentes Híbridos para Classificação de Texto

No capítulo 2, foram expostos alguns algoritmos de classificação do estado-da-arte que pertencem a famílias diferentes e, que por isso, apresentam características e limitações particulares que, geralmente, determinam em que situações a sua utilização renderá desempenho satisfatório. Já no capítulo 3, algoritmos de combinação de classificadores foram apresentados, tanto de combinação homogênea, que combinam classificadores gerados por um único algoritmo de classificação, quanto de combinação heterogênea, que combinam classificadores gerados a partir de algoritmos de classificação diferentes.

Os desempenhos alcançados pelos algoritmos, tanto os de classificação quanto os de combinação, diminuem ao tratarem problemas de classificação que possuem tamanho de classes desproporcionais [Visa and Ralescu 2005], ou seja, conjuntos nos quais algumas classes possuem, consideravelmente, mais dados que outras. Neste contexto, este trabalho de mestrado teve como objetivo investigar o desempenho de alguns algoritmos de combinação de classificadores perante conjuntos de dados desbalanceados, bem como as soluções utilizadas para resolver, ou minimizar, este problema. Como resultado, criamos uma modificação do algoritmo *LocalBoost* (variação do *Boosting* apresentado no capítulo 3) para tratar dados desbalanceados.

A seção 4.1 apresenta uma visão geral sobre os problemas gerados pelo desbalanceamento dos dados e a seção 4.2 apresenta as características do algoritmo *B-LocalBoost*, assim como detalhes de sua implementação e de outras variações.

4.1 Conjuntos de dados desbalanceados

Devido o comprometimento do desempenho, de maneira geral, dos algoritmos de AM (Aprendizado de Máquina) na presença de classes de tamanhos desproporcionais, e por ter sido observado que muitos conjuntos de dados do mundo real estão desbalanceados [Visa and Ralescu 2005], estudos estão sendo realizados para solucionar este problema. Segundo [Visa and Ralescu 2005], o desempenho pobre dos classificadores produzidos pelos algoritmos de AM padrões sobre conjuntos de dados desbalanceados é principalmente devido aos seguintes fatores: (1) os algoritmos padrões são dirigidos pela precisão (minimização do erro geral), para a qual as classes minoritárias contribuem muito pouco; (2) os classificadores atuais assumem que os algoritmos serão executados sobre dados extraídos da mesma distribuição dos dados de treinamento; e (3) os classificadores atuais assumem que os erros vindos de diferentes classes têm o mesmo custo. Entretanto, quando os dados são analisados com maior atenção, é percebido que os conjuntos de dados reais não respeitam estes fatores.

Na literatura encontramos diferentes técnicas para balancear dados:

- *Under-sampling* – Reduz o tamanho dos dados das classes majoritárias, removendo randomicamente os dados destas classes até que elas fiquem com aproximadamente o mesmo tamanho das classes minoritárias [Kubat and Matwin 1997].

- *Over-sampling* – Aumenta o tamanho dos dados das classes minoritárias. Uma das formas de se realizar o *over-sampling* é repetidamente e randomicamente duplicar os dados destas classes [Ling and Li 1998].

- Combinação de *under-sampling* e *over-sampling* – No mesmo conjunto de dados é feito o *under-sampling* das classes majoritárias e o *over-sampling* dos dados das classes minoritárias [Estabrooks *et al*. 2004], e;

• Modificação dos métodos de classificação – Os métodos de classificação são modificados para diminuírem o efeito dos dados das classes majoritárias. Por exemplo, no algoritmo *SVM*, o hiperplano pode ser empurrado para mais perto da fronteira das classes majoritárias através da realização de ajustes especiais durante a computação da fronteira, ou por incluir vetores de suporte dos rótulos das classes minoritárias [Akbani *et al*. 2004].

Métodos de amostragem têm desvantagens conhecidas: *under-sampling* envolve perda de informação (por descartar potenciais dados úteis) e *over-sampling* aumenta o tamanho de treinamento sem qualquer ganho de informação [Provost 2000] ou, talvez até perda, por replicar os mesmos dados contribuindo para o acontecimento de um provável *overfitting*.

Pesquisas realizadas mostram que *under-sampling* produz melhores resultados que *over-sampling* em muitos casos. Acredita-se que isso se deve ao fato da influência não-natural do *over-sampling* em favor das classes minoritárias [Wang and Japkowicz 2008]. Acredita-se, também, que a utilização de *over-sampling* com dados sintéticos seja superior que *over-sampling* com reposição [Chawla *et al*. 2002]. Já a combinação de *under-sampling* com *over-sampling*, realizados puramente de forma randômica, segundo [Ling and Li 1998], não apresenta melhora significativa.

Diante do fato dos algoritmos de combinação de classificadores também apresentarem dificuldades perante dados desbalanceados [Zhu 2007] [Chawla *et al*. 2003], nos propormos a investigar técnicas de balanceamento de dados para métodos de *Boosting*, e mais especificamente para *LocalBoost*.

Como já mencionado, algoritmos treinados com dados desbalanceados geram classificação tendenciosa em direção às classes com maior quantidade de dados. Segundo [Witten and Frank 1999], *Boosting* é um dos métodos de combinação de classificadores de melhor compreensão teórica, além de geralmente conseguir resultados satisfatórios na tarefa de classificação. Por *LocalBoost* ser uma variação atual de *Boosting*, por também alcançar bom

desempenho na classificação de muitos conjuntos de dados, e por sua aplicação com balanceamento de dados ser uma lacuna aberta, dedicamos tempo e esforço no desenvolvimento do *LocalBoost* balanceado, o qual chamamos de *B-LocalBoost* (*Balanced LocalBoost*).

Uma visão geral do *B-LocalBoost*, dos módulos que o compõe e de alguns detalhes da implementação são apresentados nas próximas seções.

4.2 *B-LocalBoost* e outras variações

Baseado no *LocalBoost*, o algoritmo *B-LocalBoost*, fruto do presente trabalho de mestrado, tem por objetivo gerar os classificadores a partir de dados previamente balanceados, para que depois possam ser combinados. A principal diferença entre os dois algoritmos é encontrada na fase de seleção dos dados que irão compor o conjunto de treinamento, a partir do qual os classificadores serão gerados. O *B-LocalBoost* seleciona dados a partir de um conjunto de instâncias intermediário previamente balanceado. O balanceamento de dados no *B-LocalBoost* é baseado em *Over-sampling* de forma que o tamanho de cada classe será o mesmo tamanho da classe majoritária.

A Figura 4.1 mostra o pseudocódigo do algoritmo *B-LocalBoost*. No início de cada iteração, mostrado no passo 1 da Figura 4.1, o conjunto de treinamento é balanceado, de maneira que as classes menores têm o tamanho alterado para o mesmo tamanho da classe majoritária do conjunto. Após o balanceamento dos dados o *B-LocalBoost* opera de forma semelhante ao *LocalBoost*. A princípio, cada instância tem o valor da probabilidade igual à divisão do valor 1 pelo tamanho do conjunto de dados *(1/N)*. Com base no peso D_t de cada instância, é definido um conjunto de treinamento da iteração atual (passo 2).

Após a construção do conjunto de treinamento balanceado, o algoritmo de classificação escolhido é aplicado e o erro da classificação sobre cada instância, considerando a sua vizinhança *N(i)*, é computado (passo 3). Caso o valor do erro local (i.e. *epsilon*) seja maior que 0.5, o classificador gerado nesta iteração é

desconsiderado, e outra iteração é iniciada, se a quantidade limite de iterações não tiver sido atingida ainda (passo 4). E, se o valor do erro local for igual a 0 ou menor que 0.5, os passos 5 e 6 são executados para atualização da distribuição de probabilidades ou pesos das instâncias de treinamento. No passo 6, o parâmetro β controla a importância da informação da vizinhança na classificação, tendo a princípio o valor padrão 1. Para a classificação de uma nova instância, as respostas dos classificadores são combinadas. No cálculo das respostas, a vizinhança da instância também é considerada.

- **Dado:** um conjunto de dados $L = \{(x_i, y_i)\}_{i=1}^{N}$, onde $x_i \in X \subset R^p$ e $y_i \in Y = \{-1, +1\}$; um algoritmo fraco W, número de iterações T; constante positiva β; quantidade de vizinhanças V.
- **Inicialize:** determine a distribuição da probabilidade sobre L como D_1(i) $= 1/N$ (i = 1,2, ..., N).
- **Iteração:**

  Para $t = 1, ..., T$:

  Passo 1. Balanceie o cojunto de dados fazendo com que todas as classes do conjunto tenham tamanho igual ao tamanho da maior classe do mesmo.

  Passo 2. De acordo a distribuição D_t, extraia randomicamente N instâncias de treinamento de L com reposição, para compor um novo conjunto de treinamento $L_t = \{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^{N}$.

  Passo 3. Treine o algoritmo de aprendizado fraco W com L_t para obter o classificador h_t : X $\rightarrow \{-1, +1\}$ e compute o erro local de h_t sobre a $i$ésima instância como

$$\varepsilon_t(i) = \sum_{j \in N(i)} I(h_t(x_j) \neq y_j) D'_t(j), \text{ onde } D'_t(j) = \frac{D_t(j)}{\sum_{l \in N(i)} D_t(l)}, \qquad (1)$$

  no qual N(i) indica a vizinhança da $i$ésima instância de treinamento e $I(.)$ é o indicador da função.

  Passo 4. Se $\varepsilon_t(i) > 0.5$, então determine D_{1+1}(i) $= 1/N$ (i = 1,2, ..., N) e vá para o passo 1; se $\varepsilon_t(i) = 0$, então determine $\varepsilon_t(i) = 10^{-10}$ para continuar os próximos passos.

  Passo 5. Faça $\alpha_t(i) = \frac{\varepsilon_t(i)}{1-\varepsilon_t(i)}$.

Passo 6. Atualize a distribuição da probabilidade sobre L como

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \frac{\alpha_t(i)}{\beta}, \ if \ \mathrm{h}_t(x_i) = y_i \\ 1, \ if \ \mathrm{h}_t(x_i) \neq y_i \end{cases} \quad (2)$$

onde Z_t é um fator de normalização (ele deve ser escolhido de forma que faça que D_{t+1} seja uma distribuição sobre L).

- **Saída:** o classificador combinado como

$$H(x) = sign\left(\sum_{t=1}^{T}\left(\sum_{t=1}^{N}\frac{\log\frac{1}{\alpha_t(i)}}{d^2(x,x_i)}\right)h_t(x)\right), \quad (3)$$

onde $d^2(x, x_i)$ representa a distância entre a nova instância a ser predita e a $i$ésima instância de treinamento.

Figura 4.1. Pseudocódigo do *B-LocalBoost*

Um ponto importante do *B-LocalBoost* é o processo de balanceamento dos dados realizado no passo 1 da Figura 4.1. Para isto, inicialmente, as classes minoritárias são complementadas, de forma que tenham o mesmo tamanho que a maior classe do conjunto, através da seleção aleatória de instâncias da classe original em questão. Entretanto, como o balanceamento das classes de forma aleatória não apresenta bons resultados na classificação, conforme comentado anteriormente, as instâncias, que farão parte do novo conjunto intermediário balanceado, que será utilizado no treinamento, são selecionadas da seguinte forma: a princípio, uma probabilidade é calculada, aleatoriamente, para cada uma das instâncias, e é atribuída de maneira acumulada para a instância do momento; posteriormente, as probabilidades são normalizadas, sendo que cada uma é dividida pelo valor resultante da divisão do somatório das probabilidades pelo somatório dos pesos da classe. Finalmente, no momento da seleção, caso a probabilidade da instância seja menor que o somatório dos pesos das instâncias que já fazem parte da classe, em questão, no novo conjunto, ela será adicionada ao novo conjunto, caso contrário, não será adicionada. Este processo é feito para cada uma das classes, e até o tamanho da classe atual ser completado. Feito isto, o tamanho de cada classe será igual ao tamanho da maior classe do conjunto,

sem ser de forma aleatória. A Figura 4.2 ilustra o processo de balanceamento dos dados.

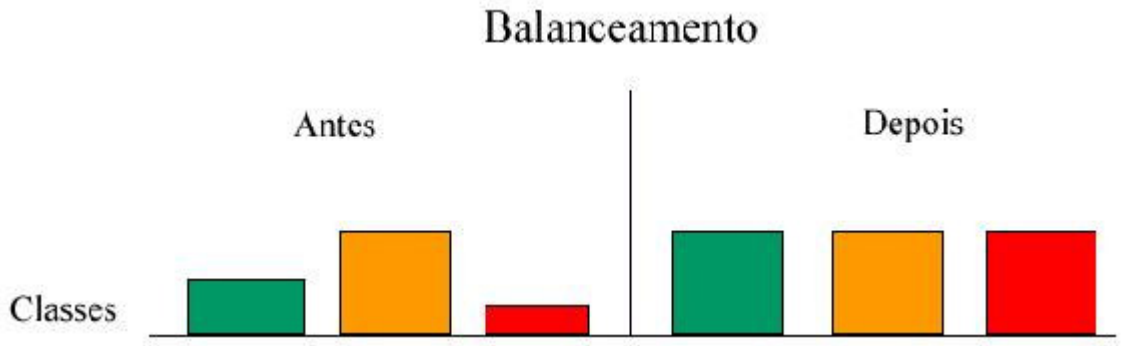


Figura 4.2. Ilustração do processo de balanceamento dos dados

Ressaltamos aqui que tanto o procedimento de balanceamento de dados como o uso de informação local (i.e. informação de vizinhança) são modificações do *Boosting* original que podem ser avaliadas de forma isolada. No nosso trabalho, além do *LocalBoost* e do *B-LocalBoost*, implementamos ainda três variação de *Boosting*:

- *B-Boosting*: Versão balanceada do *Boosting* (*Balanced Boosting*). O processo de balanceamento utilizado nesta versão é o mesmo processo utilizado no *B-LocalBoost*.

- *W-LocalBoost*: Esta é uma versão ponderada do *LocalBoost* (*Weighted LocalBoost*). Os valores da distância existente entre uma instância que está sendo classificada e as instâncias da vizinhança são utilizadas no cálculo do erro da instância. No *LocalBoost* original, essas distâncias não são utilizadas. A equação 4 mostra a equação do *W-LocalBoost* que utiliza o valor da distância. Ela é uma variação da equação 4.1 mostrada no pseudocódigo do *B-LocalBoost* na Figura 4.1, definida de forma que instâncias mais próximas (i.e. menores distâncias) tenham uma contribuição maior no cálculo do erro obtido na vizinhança.

$$\varepsilon_t(i) = \sum_{j \in N(i)} I(h_t(x_j) \neq y_j) D'_t(j) \times \frac{1}{d^2(x_i, x_j)} \quad (4.1)$$

- *BW-LocalBoost*: O *BW-LocalBoost* é, na realidade, uma outra versão do *LocalBoost*, que resulta da união das suas versões balanceada, o *B-LocalBoost*, e ponderada, o *W-LocalBoost*.

Os algoritmos avaliados nesse trabalho de mestrado foram implementados em Java de forma integrada com o *WEKA*[5] (*Waikato Environment for Knowledge Analysis*), que é uma coleção de algoritmos de aprendizado de máquina para tarefas de mineração de dados implementado em Java. O *LocalBoost*, o *B-LocalBoost*, e as outras versões aqui apresentadas foram implementados como novas classes do pacote de algoritmos meta do *WEKA*. O pacote meta incluía originalmente outros algoritmos de combinação de classificadores como o *AdaBoostM1*, que é a versão do *Boosting*. Através da integração, a implementação foi facilitada pela possibilidade de utilizar a interface e as diversas funcionalidades oferecidas pelo *WEKA*, como ferramentas para pré-processamento de dados, classificação e outras.

4.3 Considerações Finais

Neste capítulo, foi apresentado inicialmente um algoritmo proposto, o *B-LocalBoost*. Tal algoritmo tem o objetivo de combinar classificadores homogêneos, ou seja, classificadores gerados pelo mesmo algoritmo de classificação. Ele propõe o balanceamento dos dados que representam cada classe, de forma que o classificador gerado não seja tendencioso na classificação em favor das classes majoritárias e em detrimento da classificação das classes minoritárias. E realiza, ainda, após o balanceamento, uma classificação a nível local. Apresentamos ainda outras versões de *Boosting* que foram implementadas para explorar tanto o balanceamento de dados como o uso de vizinhança para melhorar o desempenho

[5] http://www.cs.waikato.ac.nz/ml/weka/

de *Boosting*. A avaliação experimental dos algoritmos propostos é apresentada no próximo capítulo.

5 Testes e Resultados

Neste capítulo, são descritos os testes realizados para verificar o comportamento do algoritmo *B-LocalBoost*, e de outras versões, na classificação de conjuntos de dados do mundo real provenientes de diferentes fontes de dados, de maneira que os testes foram executados tanto em bases de classificação de texto como em bases *benchmarks*.

Os resultados alcançados pelos métodos propostos são comparados aos resultados obtidos sobre os mesmos dados por outros algoritmos de combinação, como o *Boosting* e o *Bagging* padrão.

A seguir são descritos os conjuntos de dados utilizados nos testes, a metodologia aplicada nos experimentos, bem como os resultados obtidos.

5.1 Conjuntos de Dados Utilizados

Os experimentos foram realizados utilizando vários conjuntos de dados reais de diferentes domínios. Os conjuntos de dados foram obtidos do repositório da UCI *(University of California - Irvine)*, do DMOZ (*ODP - Open Directory Project*), do *Reuters*-21578 [6], e do *NILC* [7] (Núcleo Interinstitucional de Lingüística Computacional). A obtenção de dados de variadas fontes foi para garantir a diversidade de características dos mesmos.

Na próxima subseção é apresentada uma descrição geral de cada conjunto de dados utilizado nos experimentos.

[6] http://kdd.ics.uci.edu/databases/
[7] http://www.nilc.icmc.usp.br/nilc/

5.1.1 Bases de dados de texto

- **Dados do DMOZ**

O conjunto de dados utilizado do DMOZ contém 1061 instâncias provenientes de trechos de textos em inglês sobre Inteligência Artificial, com 100 atributos. As classes do conjunto, e a quantidade de instâncias pertencentes a cada uma delas, são: vision (70), neural (294), fuzzy (48), natural (125), agents (78), belief (60), philosophy (43), genetic (68), support (28) e machine (247). Tal conjunto de dados é um problema de classificação multiclasse e apresenta desbalanceamento entre as classes.

- **Dados da NILC**

O conjunto de dados da NILC é formado por textos jornalísticos, e os assuntos relacionados a eles foram extraídos da Folha de São Paulo do ano de 1994. Os dados deste conjunto foram processados por [Silva 2004], e contém 855 instâncias com 100 atributos. As instâncias têm as possíveis classes: esportes (171), imoveis (171), informatica (171), politica (171) e turismo (171). Este problema de classificação é multiclasse e balanceado.

- **Dados do Reuters-21578**

Este conjunto de dados se originou das reportagens de 1987 da agência de notícias Reuters. Ele contém 1228 instâncias e 847 atributos. Entretanto, para a utilização nos experimentos, foram selecionados alguns atributos destes dados através do avaliador de atributos *InfoGainAttributeEval* e do método de pesquisa *Ranker*, presentes no WEKA. Com isso, foram formados dois conjuntos, um com 100 atributos, nomeado aqui como Reuters_100, e outro com 500 atributos, nomeado como Reuters_500, para a análise do desempenho dos algoritmos quando possuem menos informações, com Reuters_100, e mais informações, com Reuters_500, no momento de classificar uma instância. As instâncias podem ser classificadas como pertencentes às classes com identificadores que vão de 1 à 65.

Este problema de classificação é caracterizado como multiclasse e apresenta desbalanceado de dados.

5.1.2 Bases de dados do UCI

- Glass – Este conjunto de dados é sobre tipos de vidros encontrados em locais sob investigação criminal, já que o tipo de vidro e suas características podem ser usados como uma evidência na investigação. O conjunto de dados tem 214 instâncias que são compostas por 10 atributos preditores, e um atributo classe com valores distribuídos da seguinte forma: b_w_f_processed (70), b_w_n_f_processed (76), v_w_f_processed (17), containers (13), tableware (9), e headlamps (29). Tratando, portanto, de um problema de classificação multiclasse e com desbalanceamento entre as classes.

- Hepatitis – Conjunto de dados que possui atributos que representam as características apresentadas por pessoas com a doença hepatite, e quais combinações das mesmas levam à possível morte ou vida de um paciente. O conjunto possui 155 instâncias, formadas por 19 atributos, e pertencem às possíveis classes: die (32) e live (123). Este problema de classificação é binário e desbalanceado.

- Ionosphere – Nesse conjunto de dados, o problema é predizer quando um radar encontra algum tipo de estrutura na ionosfera (“good”) ou não (“bad”). O conjunto é composto por 351 instâncias e cada uma contendo 34 atributos. As instâncias podem ser classificadas como: good (225) ou bad (126). Sendo, portanto, um problema de classificação binário e que apresenta desbalanceado de dados.

- Íris – Esse conjunto de dados contém informações referentes aos três tipos da planta Íris. Ele possui 150 instâncias compostas por 4 atributos contínuos. Seus atributos-classe são: Iris-setosa (50), Iris-versicolor (50) e Iris-virginica (50). E se trata de um problema de classificação multiclasse e balanceado.

5.2 Metodologia do Experimento

Os experimentos foram realizados com o objetivo de analisar a influência do módulo de balanceamento integrado ao algoritmo LocalBoost, na classificação de dados desbalanceados ou não. Apesar do B-LocalBoost ter sido desenvolvido para ajudar a resolver o problema de classificação de dados desbalanceados, era preciso analisar, também, o seu desempenho na classificação de dados balanceados.

Nos experimentos, foram avaliados inicialmente os métodos propostos e descritos no capítulo 4: (1) B-LocalBoost; (2) B-Boosting; (3) W-LocalBoost e (4) BW-LocalBoost. Como base de comparação, foram realizados ainda experimentos com os seguintes algoritmos já propostos na literatura: (1) Boosting; (2) LocalBoost; (3) Bagging; (4) J48 (versão do algoritmo C4.5 implementado no WEKA para árvores de decisão). Ressaltamos que o algoritmo J48 foi empregado como algoritmo base para os algoritmos de combinação, seguindo, por exemplo, o padrão de esperimentos realizados com o LocalBoost em [Zhang and Zhang 2008].

Os algoritmos, como já mencionado, foram aplicados sobre vários conjuntos de dados. Para cada conjunto de dados, executamos cada algoritmos de combinação variando a quantidade de iterações de boosting (5, 10 e 20 iterações), que corresponde ao número de classificadores a serem combinados.

Nos algoritmos LocalBoost, B-LocalBoost, W-LocalBoost e BW-LocalBoost variamos ainda o tamanho da vizinhança. Cada um desses algoritmos foi executado usando: (1) 100% dos dados do conjunto em uma única vizinhança, ou seja, o conjunto todo; (2) 50% dos dados em 2 vizinhanças; (3) 25% dos dados em 4 vizinhanças; (4) 10% dos dados em 10 vizinhanças; (5) 5% dos dados em 20 vizinhanças.

Em relação ao parâmetro β, ele foi utilizado com o valor 1 na maioria dos experimentos, não tendo assim, influência nos resultados. E, em alguns experimentos que serão abordados para discutir a influência do mesmo na

precisão da classificação, ele foi utilizado com o valor 5. Valor este indicado em [Zhang and Zhang 2008], pelos criadores do LocalBoost, por, segundo eles, causar efeito benéfico nos resultados dos experimentos.

O critério utilizado para medir o desempenho dos algoritmos foi a precisão alcançada nas classificações, ou seja, a porcentagem total de acerto que um algoritmo conseguiu na classificação dos dados do conjunto. Foi utilizada validação cruzada 10-fold para cálculo da precisão.

Os resultados obtidos nos experimentos são apresentados e analisados na próxima seção.

5.3 Resultados Obtidos

Como descrito anteriormente, os experimentos foram executados sobre diversos conjuntos de dados: Íris, Ionosphere, Hepatitis, Glass, DMOZ, NILC, Reuters_100 e Reuters_500. Nos gráficos e tabelas a seguir são mostrados os resultados obtidos pelos algoritmos sobre cada um destes conjuntos de dados.

- Conjunto de dados Íris

Esse conjunto de dados, conforme citado anteriormente, possui 150 instâncias e três classes, cada uma delas composta exatamente por 50 instâncias. Sendo, assim, um conjunto balanceado.

LocalBoost e B-LocalBoost

Os Gráficos 5.1, 5.2 e 5.3 mostram o desempenho dos algoritmos LocalBoost e B-LocalBoost com 5, 10 e 20 iterações, respectivamente, sobre este conjunto de dados. Como podemos observar no Gráfico 5.1, com 5 iterações, o LocalBoost consegue superar levemente o desempenho do B-LocalBoost com 1 e 2 vizinhanças. Já com 4 vizinhanças, o B-LocalBoost é quem ultrapassa o desempenho alcançado por LocalBoost. Entretanto, com 10 e 20 vizinhanças, a

sua precisão volta a cair e é superada com uma maior diferença pelas precisões do LocalBoost.

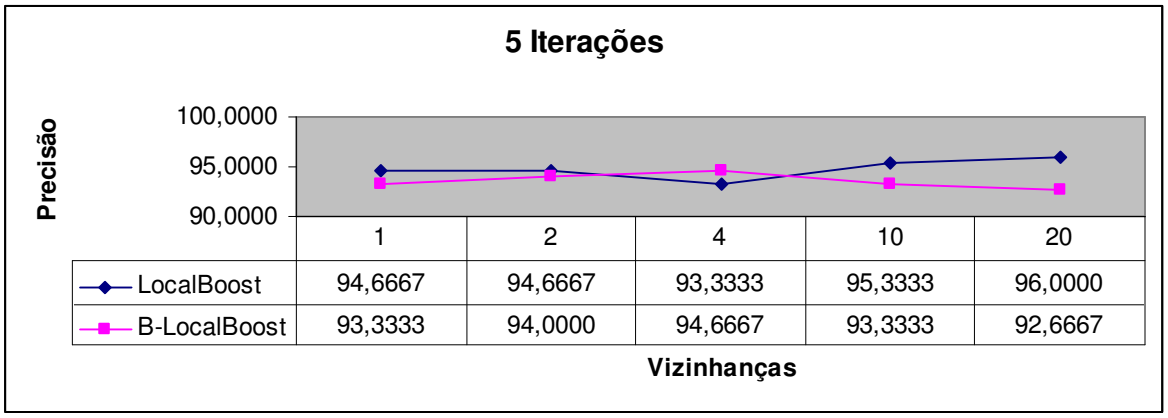


Gráfico 5.1. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Íris com 5 iterações

No Gráfico 5.2, com 10 iterações, o B-LocalBoost consegue obter, de maneira geral, resultados melhores. Com 1 vizinhança, o B-LocalBoost apresenta uma precisão um pouco menor que a do LocalBoost. Com 2 vizinhanças, a situação é inversa, com B-LocalBoost apresentando um desempenho um pouco melhor que o do LocalBoost. Já com 4 vizinhanças, eles obtêm a mesma precisão. Mas o desempenho do B-LocalBoost volta a cair com 10 e 20 iterações.

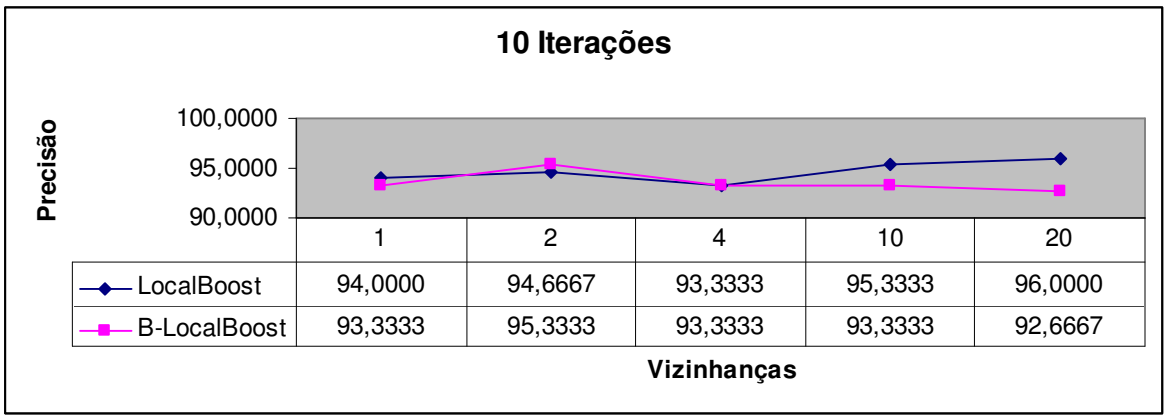


Gráfico 5.2. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Íris com 10 iterações

Finalmente, conforme mostrado no Gráfico 5.3, com 20 iterações, o desempenho de LocalBoost e B-LocalBoost segue igual com 1, 2 e 4 vizinhanças. Entretanto, o B-LocalBoost apresenta, novamente, queda de desempenho com 10 e 20 vizinhanças, sendo superado, mais uma vez, pelo LocalBoost.

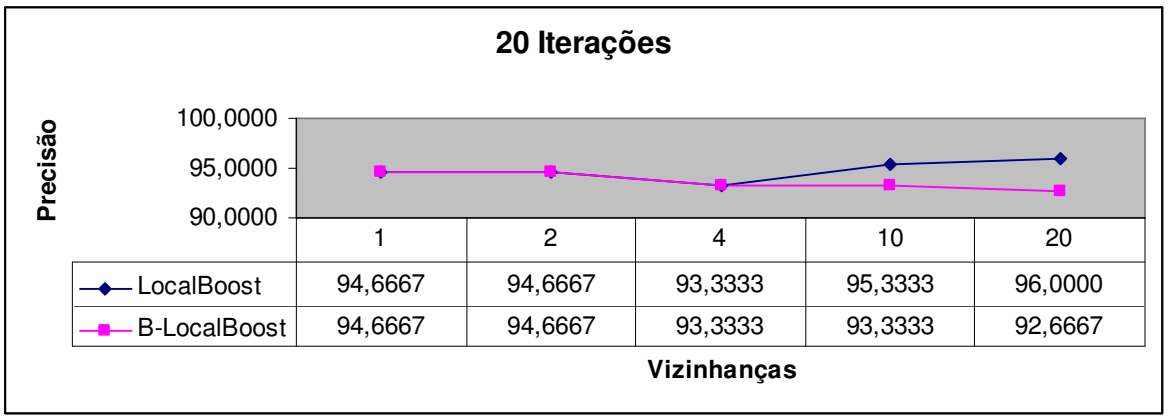


Gráfico 5.3. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Íris com 20 iterações

Boosting e B-Boosting

No Gráfico 5.4, é mostrada a comparação do desempenho do B-Boosting com o do Boosting. Nele é possível perceber que o aumento do número de iterações executadas ajudou a elevar o desempenho do B-Boosting. Tendo, o mesmo, precisões inferiores em relação ao Boosting com 5 e 10 iterações, sendo que com 10 iterações a diferença foi menor do que com 5 iterações, e alcançando a mesma precisão que o Boosting quando executado com 20 iterações.

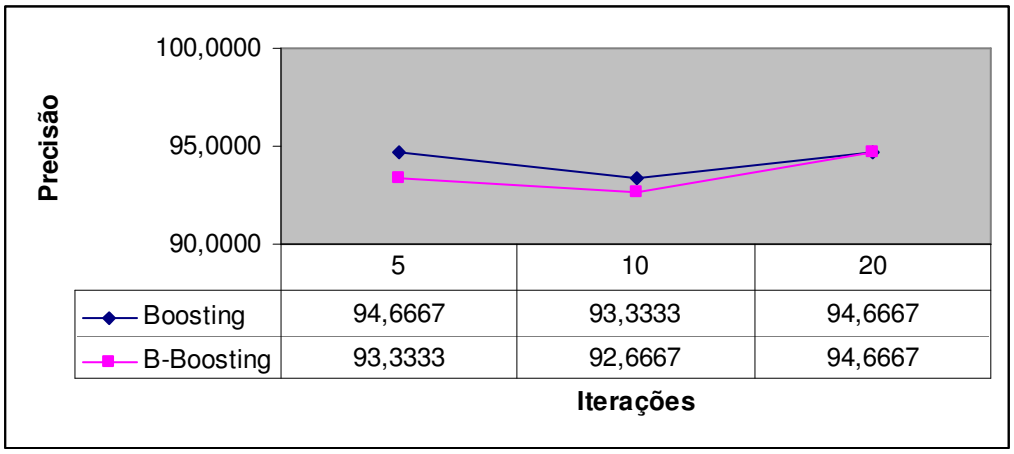


Gráfico 5.4. Desempenho dos algoritmos Boosting e B-Boosting sobre o conjunto de dados Íris

Algoritmos do Estado-da-Arte (Boosting, *Bagging* e J48)

O Gráfico 5.5 apresenta o desempenho alcançado por alguns algoritmos do estado-da-arte, aqui utilizados, sobre o conjunto de dados Iris. O objetivo é comparar o desempenho de Boosting aos de outros algoritmos que também pertencem ao estado-da-arte.

O gráfico mostra que, quando comparado ao Bagging e ao J48, a precisão obtida pelo Boosting, neste conjunto, é inferior em todas as iterações.

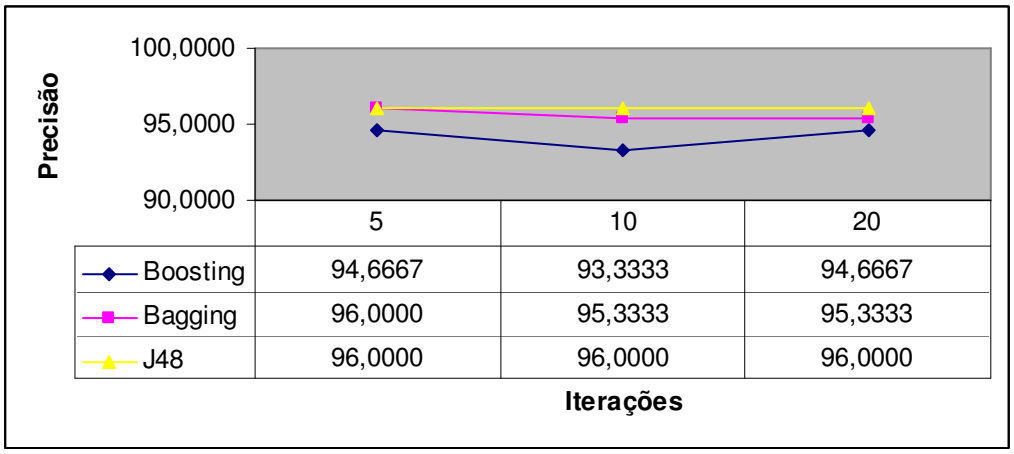


Gráfico 5.5. Desempenho dos algoritmos do estado-da-arte, Boosting, Bagging e J48, sobre o conjunto de dados Íris

Todos os algoritmos

O Gráfico 5.6 apresenta o desempenho de todos os algoritmos na base Íris. Nesse gráfico, a legenda LocalBoost(1) significa que o algoritmo foi executado com 1 vizinhança. Nesse gráfico, é possível obter uma melhor visualização do desempenho de cada algoritmo, em cada iteração, em relação aos outros. Como podemos observar, a maior precisão obtida com 5 iterações foi 96%, alcançada pelo LocalBoost(20), por Bagging e por J48. Com 10 e 20 iterações, a maior precisão também foi 96%, alcançada por LocalBoost(20).

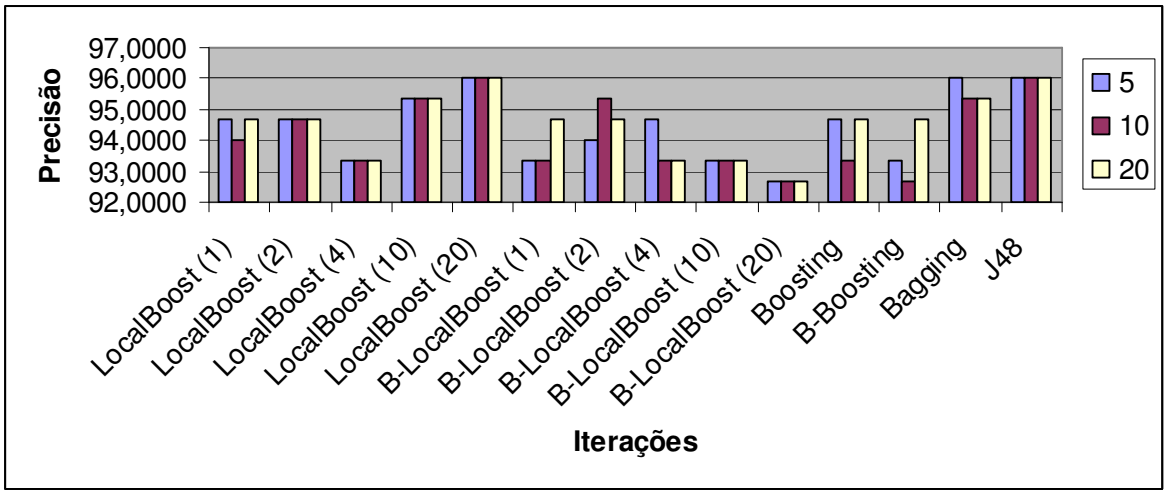


Gráfico 5.6. Comparação dos desempenhos obtidos pelos algoritmos sobre o conjunto de dados Íris

Tabela de ganhos e perdas

A Tabela 5.1 mostra a relação de ganhos e perdas de cada algoritmo, quando comparado a cada um dos algoritmos utilizados nos experimentos que foram executados sobre o conjunto de dados Iris. Por exemplo, nesta tabela, Boosting quando comparado a B-Boosting alcançou precisão maior que ele duas vezes e nenhuma vez menor. Ou seja, apresentou ganho igual a 2 e perda igual a 0, ilustrado na tabela como (2/0$^-$).

Tabela 5.1. Tabela geral de ganhos e perdas de cada algoritmo

+/-	Boost.	B-Boost.	Bag.	J48	LocalB.*	B-LocalB.*
Boost.		2/0$^-$	0/3$^-$	0/3$^-$	2/8$^-$	8/1$^-$
B-Boost.	0/2$^-$		0/3$^-$	0/3$^-$	1/11$^-$	4/6$^-$
Bag.	3/0$^-$	3/0$^-$		0/2$^-$	10/2$^-$	14/0$^-$
J48	3/0$^-$	3/0$^-$	2/0$^-$		12/0$^-$	15/0$^-$
LocalB.*	8/2-	11/1-	2/10-	0/12-		9/2-
B-LocalB.*	1/8$^-$	6/4$^-$	0/14$^-$	0/15$^-$	2/9$^-$	

*Considerando todas as vizinhanças (1, 2, 4, 10, e 20).

- Conjunto de dados Ionosphere

Conforme já mencionado, este conjunto de dados contém 351 instâncias, é um problema de classificação binário e é desbalanceado, contendo 126 instâncias em uma classe e 225 em outra.

LocalBoost e B-LocalBoost

Nos Gráficos 5.7, 5.8 e 5.9 é mostrado o desempenho dos algoritmos LocalBoost e B-LocalBoost com 5, 10 e 20 iterações, respectivamente, sobre este conjunto de dados. Como podemos observar no Gráfico 5.7, com 5 iterações, o LocalBoost consegue superar levemente o desempenho do B-LocalBoost com 1 e 2 vizinhanças. Já com 4 e 10 vizinhanças, o B-LocalBoost é quem ultrapassa um pouco o desempenho alcançado pelo LocalBoost. Entretanto, com 20 vizinhanças, a sua precisão volta a cair e é superada, com uma diferença maior, pela precisão obtida pelo LocalBoost.

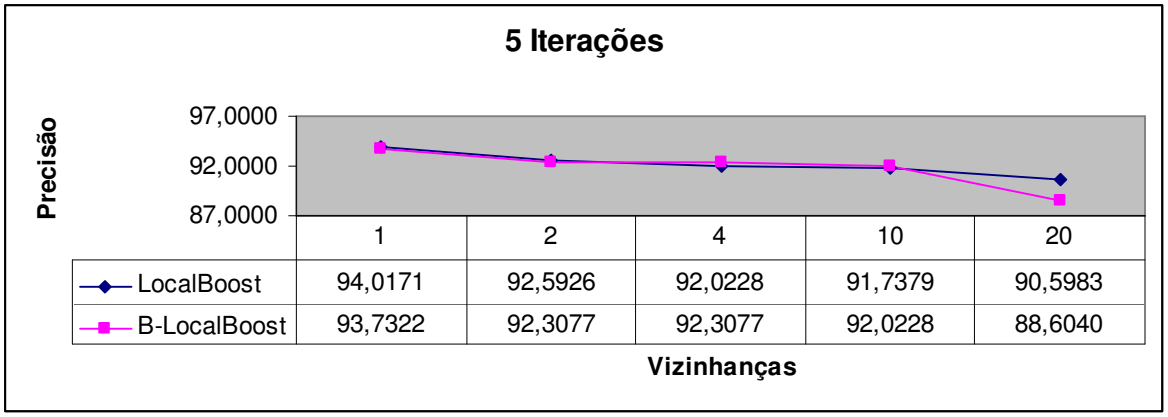


Gráfico 5.7. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Ionosphere com 5 iterações

No Gráfico 5.8, com 10 iterações, podemos ver que o B-LocalBoost consegue melhorar, um pouco mais, seu desempenho em relação a execução de LocalBoost. Com 1 e 2 vizinhanças, o B-LocalBoost apresenta uma precisão um

pouco melhor que a do LocalBoost. Com 4 e 10 vizinhanças, a precisão dos dois é igual. Porém, o desempenho do B-LocalBoost volta a cair com 20 iterações.

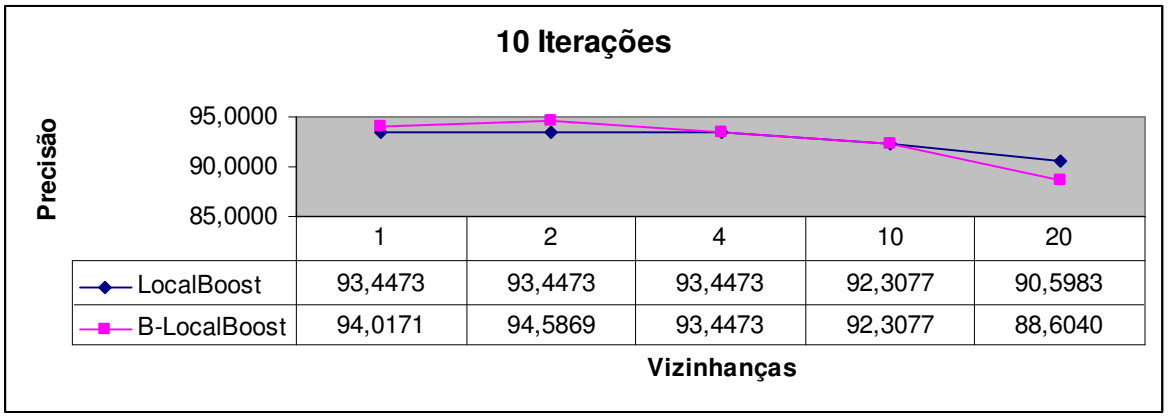


Gráfico 5.8. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Ionosphere com 10 iterações

E, com 20 iterações, conforme o Gráfico 5.9, LocalBoost tem um desempenho melhor que B-LocalBoost com 1 vizinhança. Já com 2 e 4 vizinhanças, B-LocalBoost consegue obter precisão maior que LocalBoost. Mas seu desempenho cai, novamente, com 10 e 20 vizinhanças, sendo menor que o de LocalBoost.

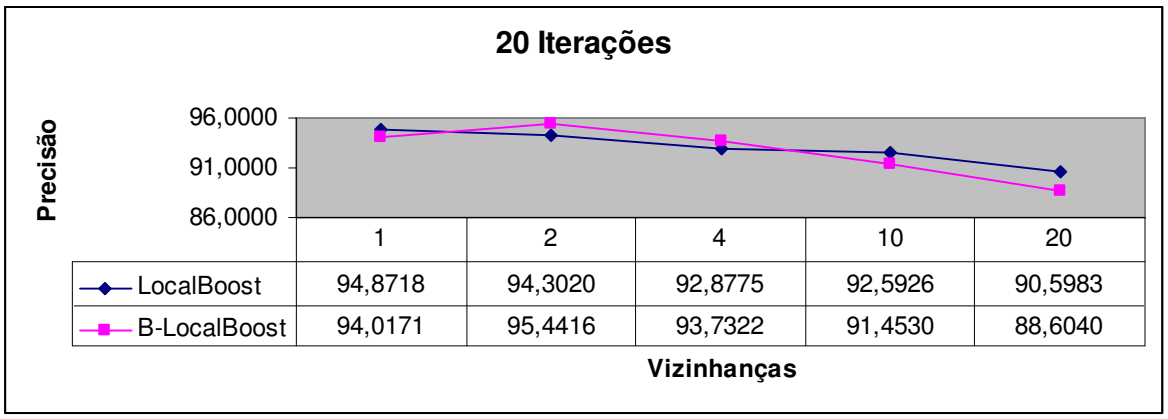


Gráfico 5.9. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Ionosphere com 20 iterações

Boosting e B-Boosting

Comparando B-Boosting com Boosting sobre o conjunto de dados Ionosphere, percebemos que a camada de balanceamento proporcionou resultados satisfatórios a B-Boosting. Inicialmente, apresentando uma precisão um pouco menor em relação a Boosting com 5 iterações, mas o superando com 10 e 20 iterações, como mostrado no Gráfico 5.10.

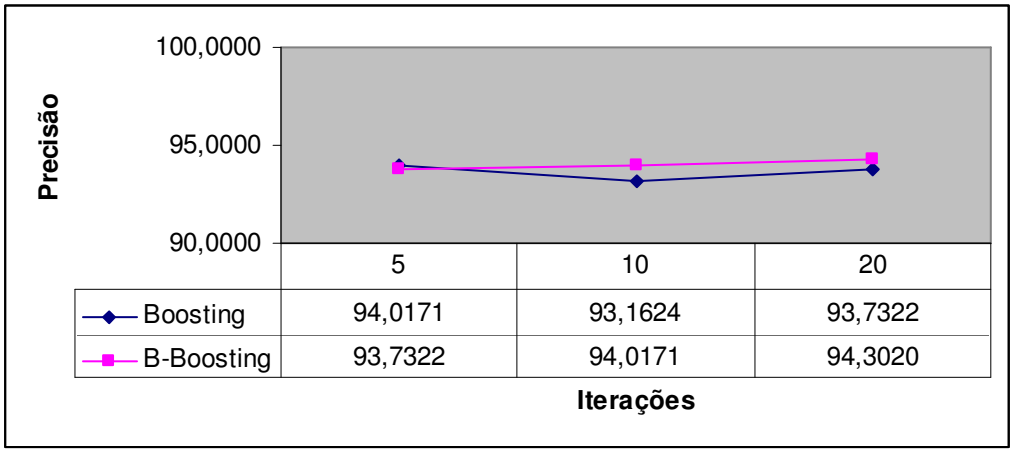


Gráfico 5.10. Desempenho dos algoritmos Boosting e B-Boosting sobre o conjunto de dados Ionosphere

Algoritmos do Estado-da-Arte (Boosting, *Bagging* e J48)

Em relação aos algoritmos Bagging e J48, Boosting tem um bom desempenho, obtendo a maior precisão com 5 e 20 iterações, e uma precisão um pouco menor que a de Bagging com 10 iterações (Gráfico 5.11).

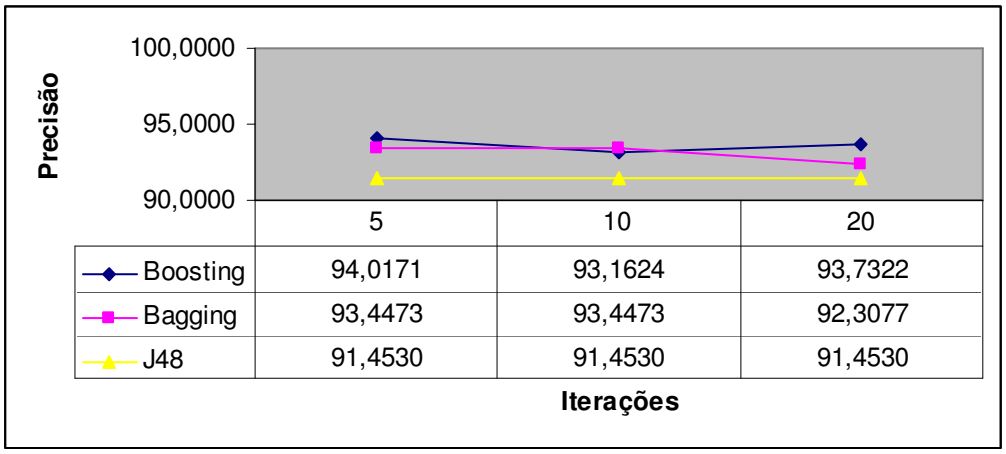


Gráfico 5.11. Desempenho dos algoritmos do estado-da-arte, Boosting, Bagging e J48, sobre o conjunto de dados Ionosphere

Todos os algoritmos

No gráfico contendo o desempenho de todos os algoritmos executados com 5, 10 e 20 iterações sobre o conjunto de dados Ionosphere (Gráfico 5.12), constatamos que, com 5 iterações, o melhor desempenho foi dos algoritmos LocalBoost(1) e Boosting, empatados com 94,0171% de precisão. E, com 10 e 20 iterações, as precisões mais altas, 94,5869% e 95,4416%, respectivamente, foram obtidas pelo B-LocalBoost(2).

No geral, a precisão mais alta atingida sobre o conjunto Ionosphere foi 95,4416% pelo algoritmo B-LocalBoost(2).

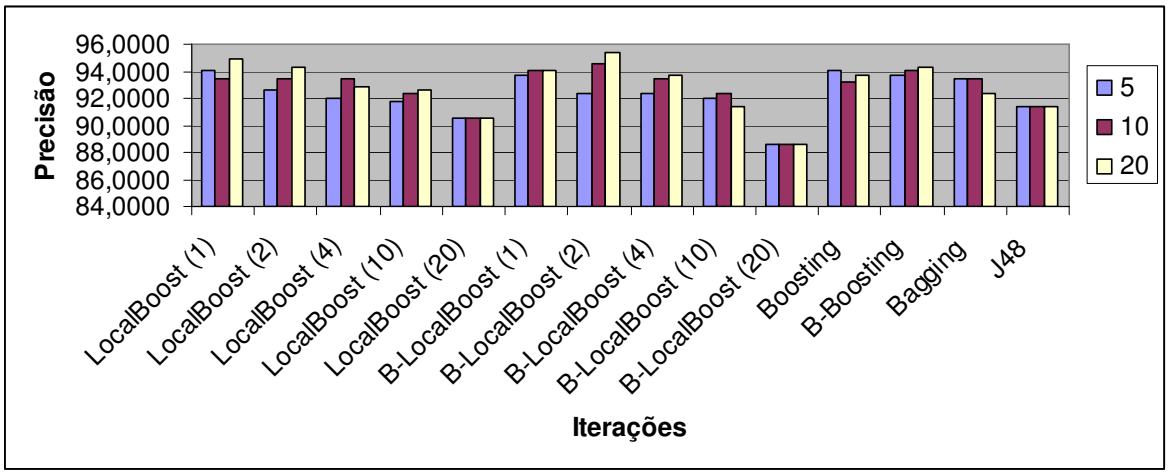


Gráfico 5.12. Comparação dos desempenhos obtidos pelos algoritmos sobre o conjunto de dados Ionosphere

Tabela de ganhos e perdas

Na Tabela 5.2 é possível visualizar os ganhos e perdas de cada algoritmo em relação aos outros sobre este conjunto.

Tabela 5.2. Tabela geral de ganhos e perdas de cada algoritmo

+/-	Boost.	B-Boost.	Bag.	J48	LocalB.*	B-LocalB.*
Boost.		1/2-	2/1-	3/0-	9/5-	9/5-
B-Boost.	2/1-		3/0-	3/0-	12/2-	11/2-
Bag.	1/2-	0/3-		3/0-	7/5-	8/6-
J48	0/3-	0/3-	0/3-		3/12-	3/11-
LocalB.*	5/9-	2/12-	5/7-	12/3-		7/6-
B-LocalB.*	5/9-	2/11-	6/8-	11/3-	6/7-	

*Considerando todas as vizinhanças (1, 2, 4, 10, e 20).

- Conjunto de dados Hepatitis

Este conjunto de dados possui 155 instâncias e duas classes, contendo 32 instâncias em uma classe e 123 na outra, caracterizando, assim, um conjunto desbalanceado. Um outro problema apresentado por ele é que muitas instâncias estão com alguns dos seus atributos sem valores.

LocalBoost e B-LocalBoost

Através da análise dos Gráficos 5.13, 5.14 e 5.15, podemos observar que o B-LocalBoost não conseguiu superar o desempenho do LocalBoost em nenhuma das execuções. A sua melhor performance obtida, neste conjunto, em relação ao LocalBoost, foi com 1 vizinhança, onde as precisões alcançadas eram mais próximas às de LocalBoost. Já com as outras quantidades de vizinhanças (2, 4, 10 e 20) o desempenho do B-LocalBoost piorava.

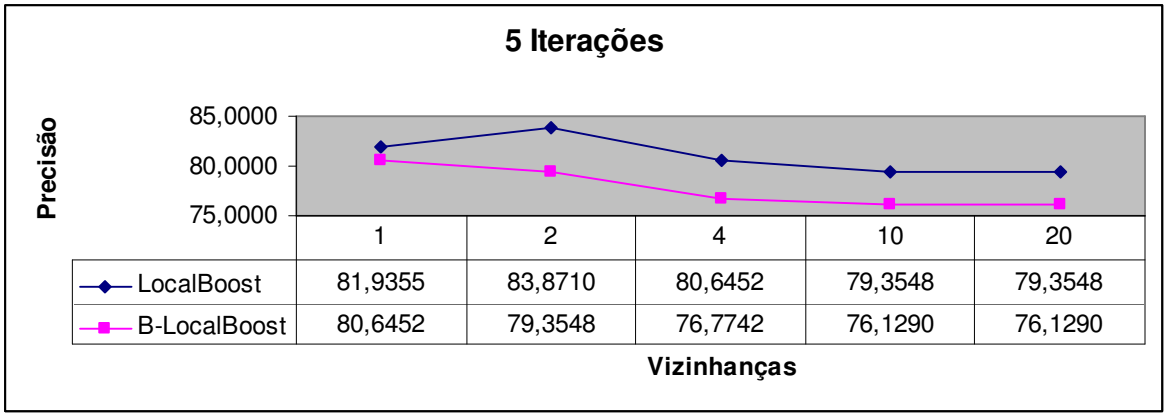


Gráfico 5.13. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Hepatitis com 5 iterações

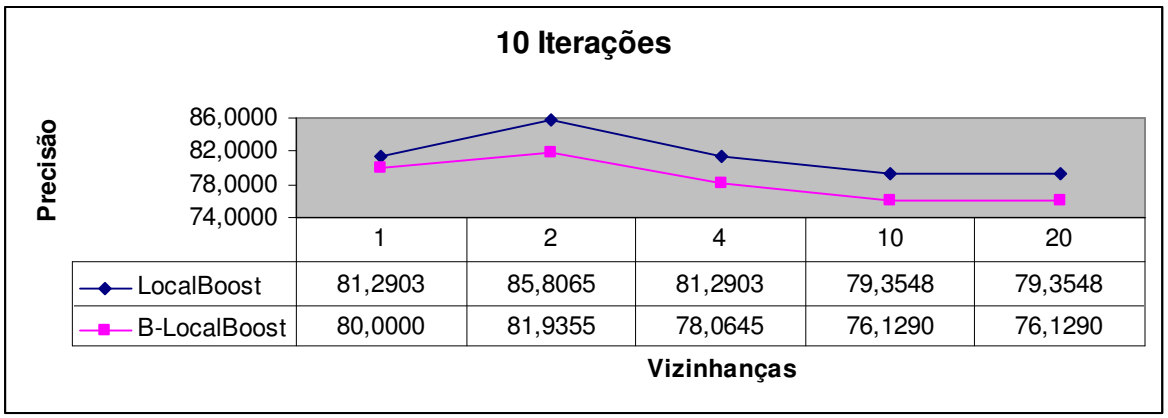


Gráfico 5.14. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Hepatitis com 10 iterações

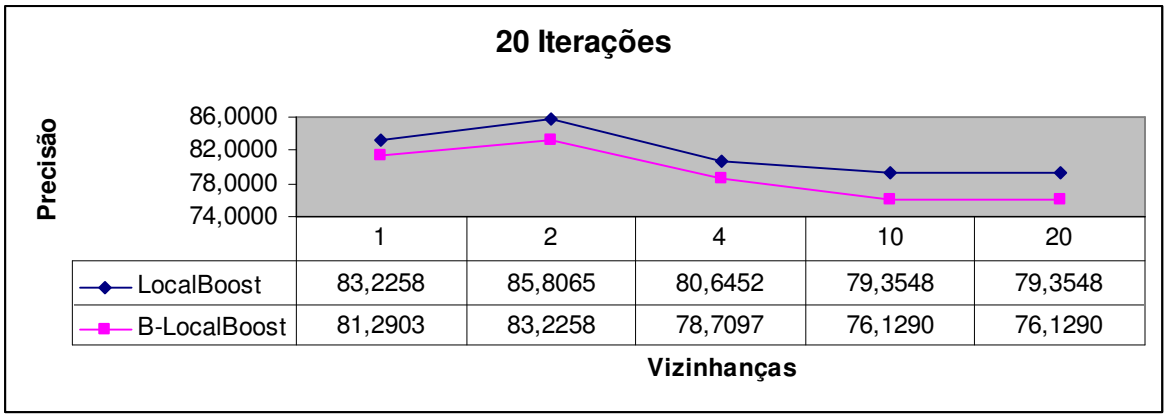


Gráfico 5.15. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Hepatitis com 20 iterações

Boosting e B-Boosting

O B-Boosting, assim como o B-LocalBoost em relação ao LocalBoost, também não obteve bons resultados quando comparado ao Boosting. O B-Boosting não conseguiu superar o Boosting em nenhuma das situações apresentadas no Gráfico 5.16. O melhor que conseguiu foi alcançar a mesma precisão que o Boosting com 10 iterações.

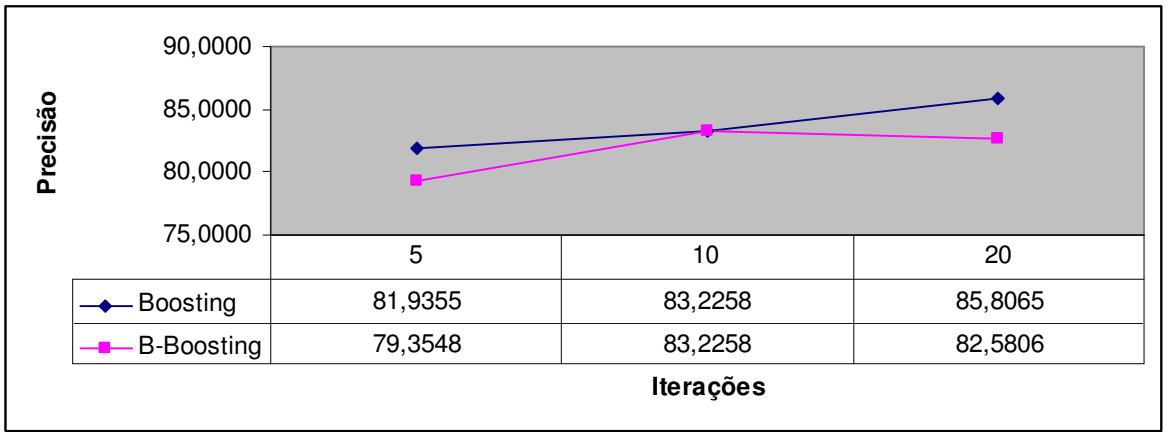


Gráfico 5.16. Desempenho dos algoritmos Boosting e B-Boosting sobre o conjunto de dados Hepatitis

Algoritmos do Estado-da-Arte (Boosting, *Bagging* e J48)

Como podemos ver no Gráfico 5.17, quando comparado com os algoritmos Bagging e J48, Boosting apresenta a precisão mais baixa com 5 iterações. Já com 10 iterações, ele consegue obter quase a mesma precisão que os outros dois algoritmos. Finalmente, com 20 iterações, Boosting alcança a maior precisão, e com uma boa diferença em relação a Bagging.

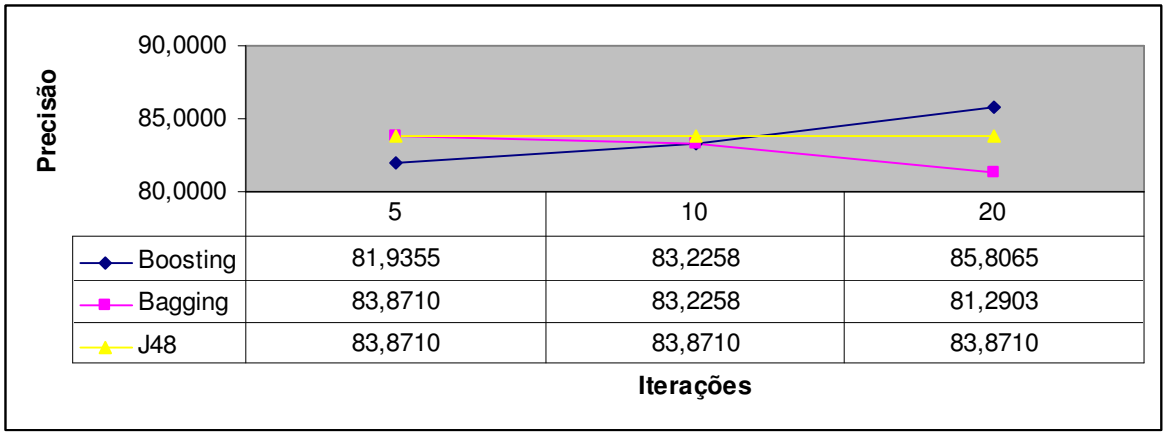


Gráfico 5.17. Desempenho dos algoritmos do estado-da-arte, Boosting, Bagging e J48, sobre o conjunto de dados Hepatitis

Todos os algoritmos

O Gráfico 5.18 apresenta o desempenho dos algoritmos, executados com 5, 10 e 20 iterações, sobre o conjunto de dados Hepatitis. Com 5 iterações, a maior precisão obtida foi 83,8710%, e foi alcançada pelo LocalBoost(2), Bagging e J48. Com 10 iterações, a maior precisão foi 85,8065%, sendo obtida pelo LocalBoost(2). E, com 20 iterações, a maior precisão também foi 85,8065%, obtida por LocalBoost(2) e por Boosting.

Portanto, a maior precisão atingida sobre o conjunto de dados Hepatitis foi 85,8065%, sendo alcançada por LocalBoost(2) e por Boosting.

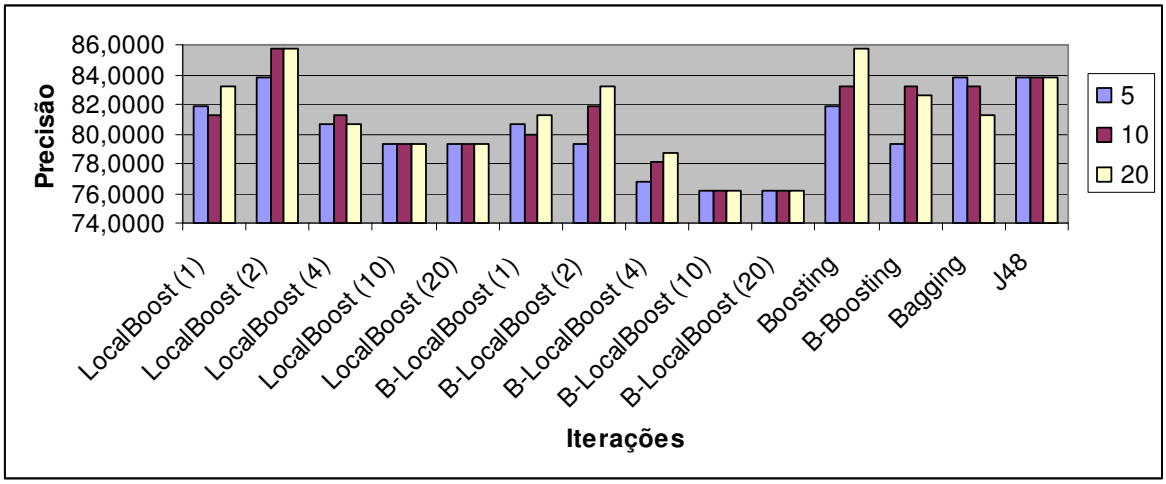


Gráfico 5.18. Comparação dos desempenhos obtidos pelos algoritmos sobre o conjunto de dados Hepatitis

Tabela de ganhos e perdas

A Tabela 5.3 mostra os ganhos e perdas de cada algoritmo em relação aos outros sobre o conjunto de dados Hepatitis.

Tabela 5.3. Tabela geral de ganhos e perdas de cada algoritmo

+/-	Boost.	B-Boost.	Bag.	J48	LocalB.*	B-LocalB.*
Boost.		2/0-	1/1-	1/2-	11/2-	15/0-
B-Boost.	0/2-		1/1-	0/3-	7/6-	12/2-
Bag.	1/1-	1/1-		0/2-	11/3-	13/1-
J48	2/1-	3/0-	2/0-		12/2-	15/0-
LocalB.*	2/11-	6/7-	3/11-	2/12-		15/0-
B-LocalB.*	0/15-	2/12-	1/13-	0/15-	0/15-	

*Considerando todas as vizinhanças (1, 2, 4, 10, e 20).

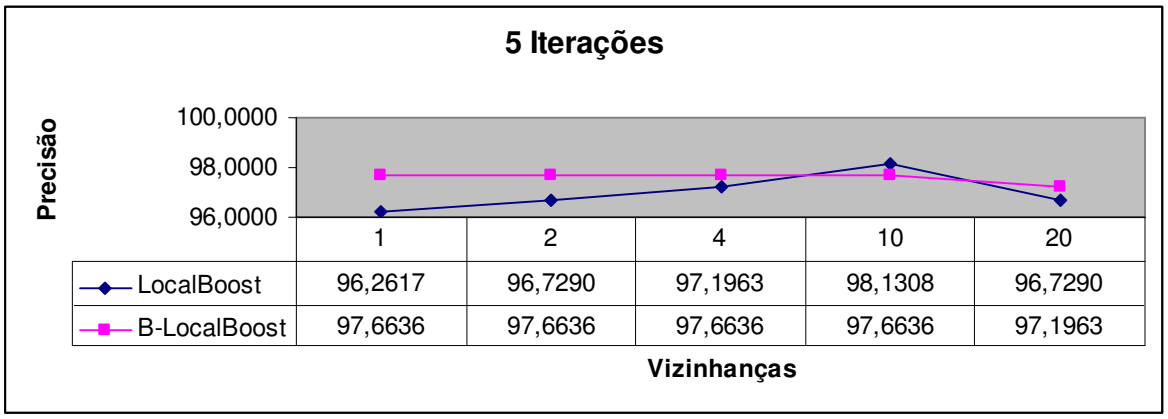
- Conjunto de dados Glass

Conforme já mencionado, o conjunto de dados Glass possui 214 instâncias e 7 classes, que possuem tamanhos diferentes, ou seja, é desbalanceado, sendo que uma das classes não possui nenhuma representação no conjunto.

LocalBoost e B-LocalBoost

O desempenho dos algoritmos LocalBoost e B-LocalBoost com 5, 10 e 20 iterações, sobre este conjunto de dados, é mostrado, respectivamente, nos Gráficos 5.19, 5.20 e 5.21.

Como podemos observar no Gráfico 5.19, com 5 iterações, o B-LocalBoost consegue superar o desempenho do LocalBoost com 1, 2, 4 e 20 vizinhanças, apresentando uma leve queda de precisão, em relação ao LocalBoost, apenas com 10 vizinhanças.



	1	2	4	10	20
LocalBoost	96,2617	96,7290	97,1963	98,1308	96,7290
B-LocalBoost	97,6636	97,6636	97,6636	97,6636	97,1963

Gráfico 5.19. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Glass com 5 iterações

Já com 10 e 20 iterações, o B-LocalBoost apresenta o mesmo comportamento, superando o desempenho do LocalBoost com 1 e 20 vizinhanças, obtendo a mesma precisão com 4 vizinhanças, e atingindo uma precisão menor que a do LocalBoost com 2 e 10 vizinhanças (Gráfico 5.20 e 5.21).

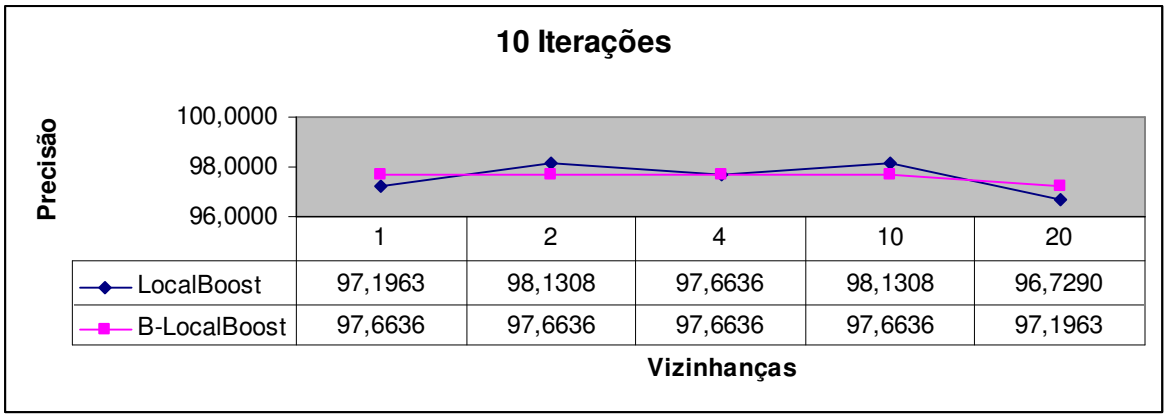


Gráfico 5.20. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Glass com 10 iterações

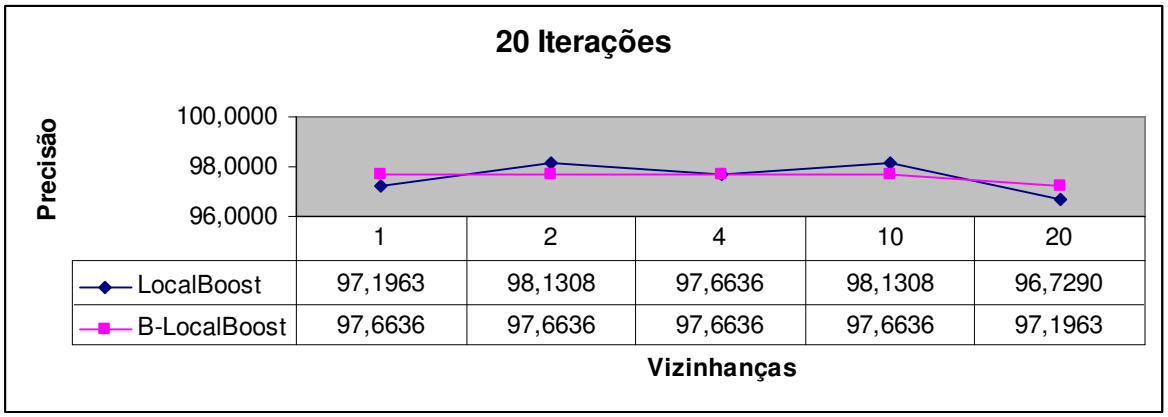


Gráfico 5.21. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Glass com 20 iterações

Boosting e B-Boosting

O B-Boosting também apresentou boa performance sobre o conjunto de dados Glass, pois conseguiu superar o Boosting com 5 iterações, e empatou com 10 e 20 iterações, conforme podemos ver no Gráfico 5.22.

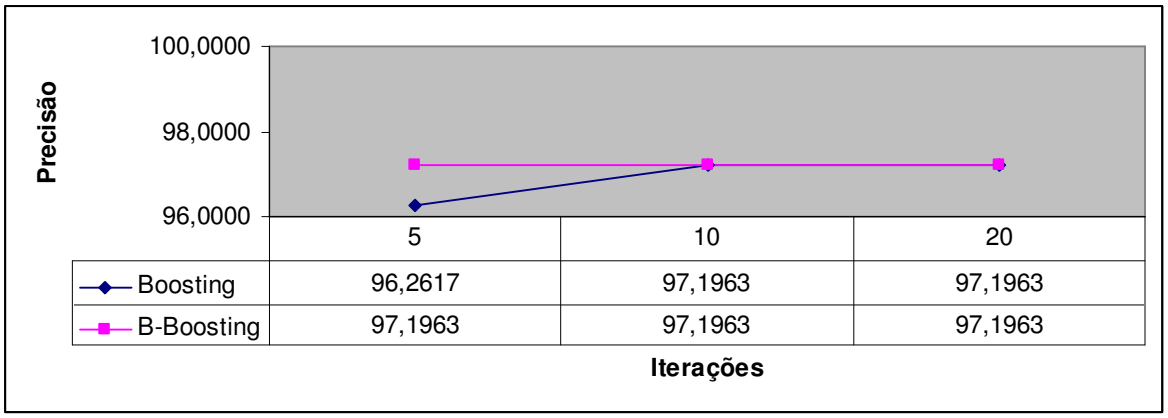


Gráfico 5.22. Desempenho dos algoritmos Boosting e B-Boosting sobre o conjunto de dados Glass

Algoritmos do Estado-da-Arte (Boosting, *Bagging* e J48)

Os algoritmos Boosting, Bagging e J48 apresentaram desempenho bem parecido sobre este conjunto. Com 5 iterações, Bagging e J48 empataram e superaram levemente a precisão alcançada por Boosting. E, com 10 e 20 iterações, foi Boosting que conseguiu uma leve superação em relação às precisões dos outros dois algoritmos (Gráfico 5.23).

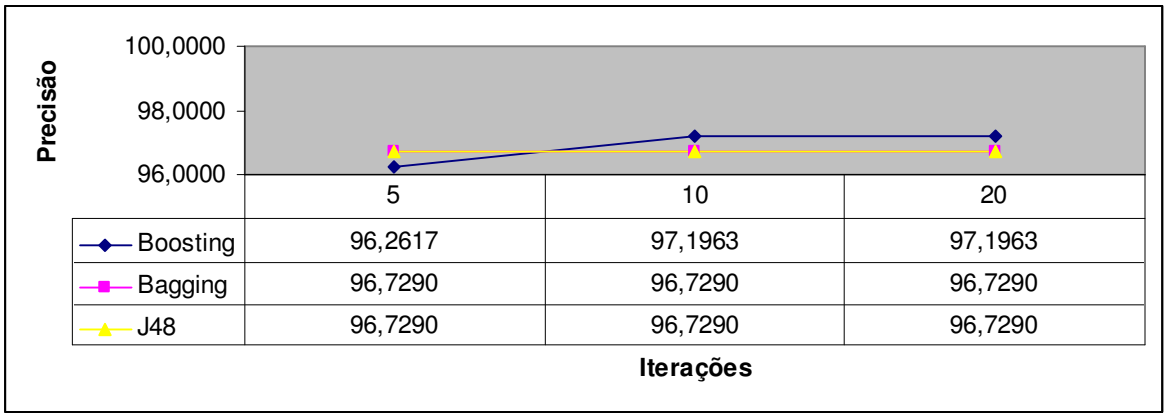


Gráfico 5.23. Desempenho dos algoritmos do estado-da-arte, Boosting, Bagging e J48, sobre o conjunto de dados Glass

Todos os algoritmos

Apesar do B-LocalBoost e B-Boosting terem alcançado bom desempenho quando comparados com o LocalBoost e com o Boosting, respectivamente, não atingiram as maiores precisões nas iterações.

Como podemos visualizar no Gráfico 5.24, LocalBoost(10) obteve a maior precisão, 98,1308%, com 5 iterações. E com 10 e 20 iterações, a maior precisão, 98,1308% também foi alcançada por LocalBoost(10) juntamente com LocalBoost(2).

A maior precisão obtida sobre este conjunto de dados foi de 98,1308%, atingida pelos algoritmos já citados anteriormente.

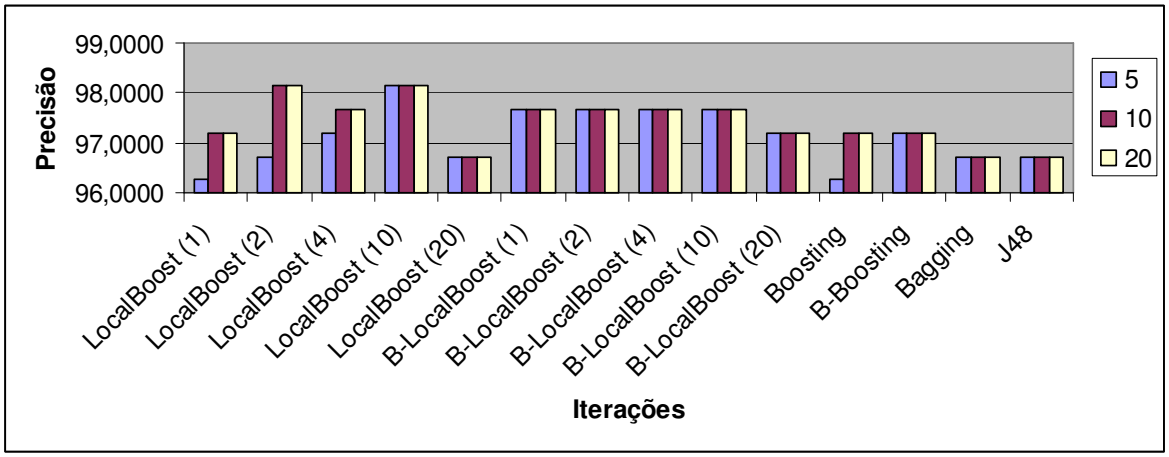


Gráfico 5.24. Comparação dos desempenhos obtidos pelos algoritmos sobre o conjunto de dados Glass

Tabela de ganhos e perdas

A Tabela 5.4 mostra os ganhos e perdas de cada algoritmo em relação aos outros sobre o conjunto de dados Glass.

Tabela 5.4. Tabela geral de ganhos e perdas de cada algoritmo

+/-	Boost.	B-Boost.	Bag.	J48	LocalB.*	B-LocalB.*
Boost.		0/1-	2/1-	2/1-	2/10-	0/13-
B-Boost.	1/0-		3/0-	3/0-	5/7-	0/12-
Bag.	1/2-	0/3-		**	1/10-	0/15-
J48	1/2-	0/3-	**		1/10-	0/15-
LocalB.*	10/2-	7/5-	10/1-	10/1-		5/8-
B-LocalB.*	13/0-	12/0-	15/0-	15/0-	8/5-	

*Considerando todas as vizinhanças (1, 2, 4, 10, e 20).
**Empate em todas as execuções

- Conjunto de dados DMOZ

Como dito anteriormente, o conjunto de dados do DMOZ contém 1061 instâncias e 10 classes desbalanceadas.

LocalBoost e B-LocalBoost

Através da análise dos Gráficos 5.25, 5.26 e 5.27, com 5, 10 e 20 iterações, respectivamente, podemos observar que o B-LocalBoost não conseguiu superar o desempenho do LocalBoost com 1 e 2 vizinhanças nas três situações apresentadas. Entretanto, nas três execuções, conseguiu a maior precisão com 4, 10 e 20 vizinhanças.

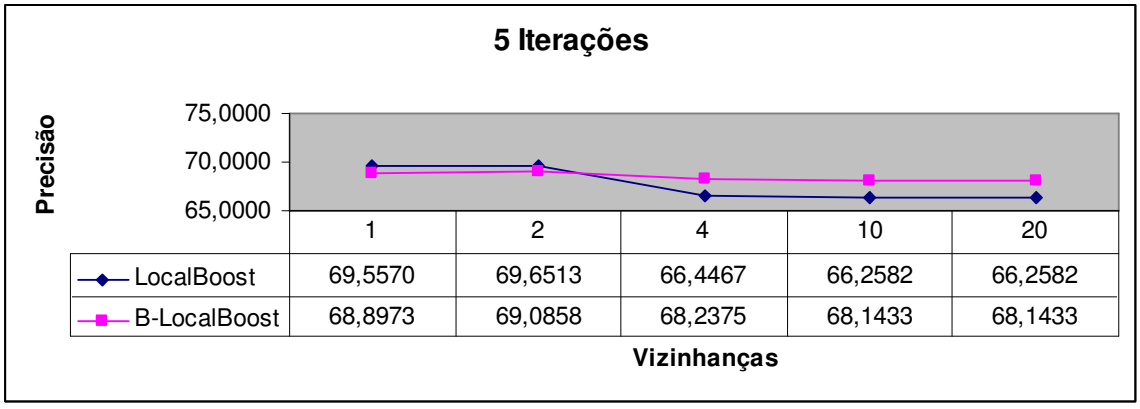


Gráfico 5.25. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados DMOZ com 5 iterações

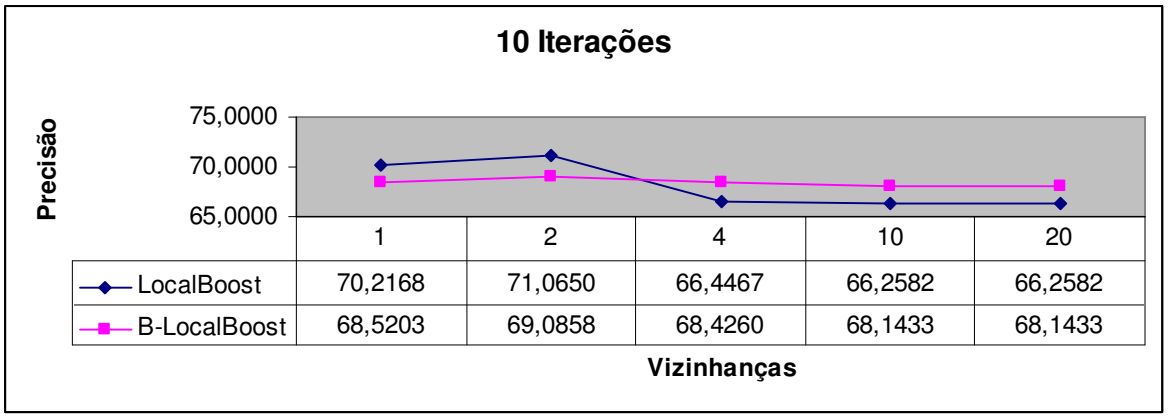


Gráfico 5.26. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados DMOZ com 10 iterações

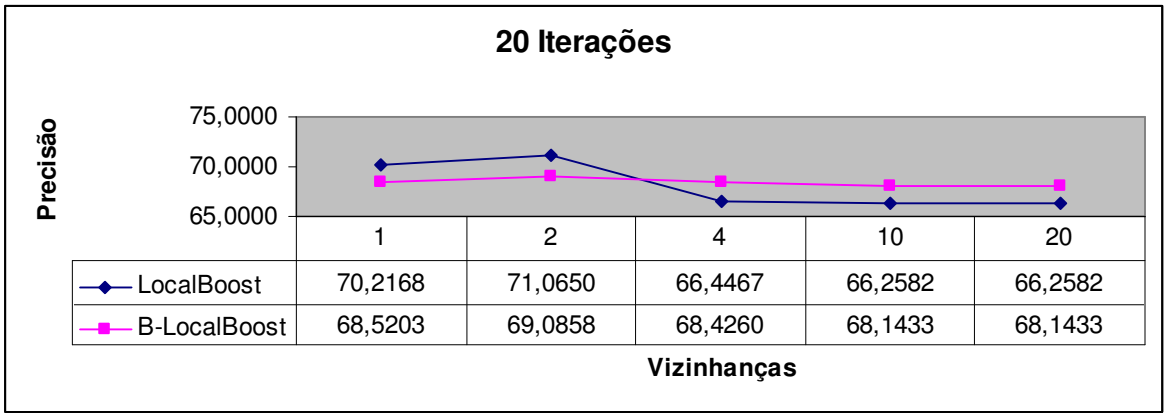


Gráfico 5.27. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados DMOZ com 20 iterações

Boosting e B-Boosting

O B-Boosting apresentou desempenho muito bom sobre o conjunto de dados DMOZ, pois conseguiu superar as precisões obtidas por Boosting em todas as situações, ou seja, com 5, 10 e 20 iterações, como podemos ver no Gráfico 5.28.

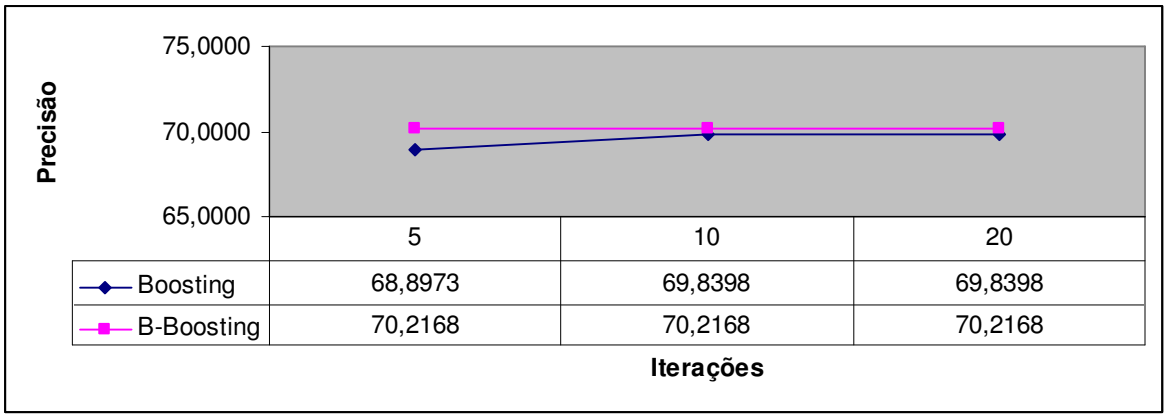


Gráfico 5.28. Desempenho dos algoritmos Boosting e B-Boosting sobre o conjunto de dados DMOZ

Algoritmos do Estado-da-Arte (Boosting, *Bagging* e J48)

O algoritmo Boosting, conforme visto no Gráfico 5.29, também não obteve bons resultados quando comparado com os algoritmos Bagging e J48. O seu desempenho foi inferior ao de Bagging em todas as situações apresentadas no gráfico, e obteve precisão menor que a do algorimto J48 com 5 iterações. Entretanto, superou o seu desempenho, mesmo que levemente, com 10 e 20 iterações.

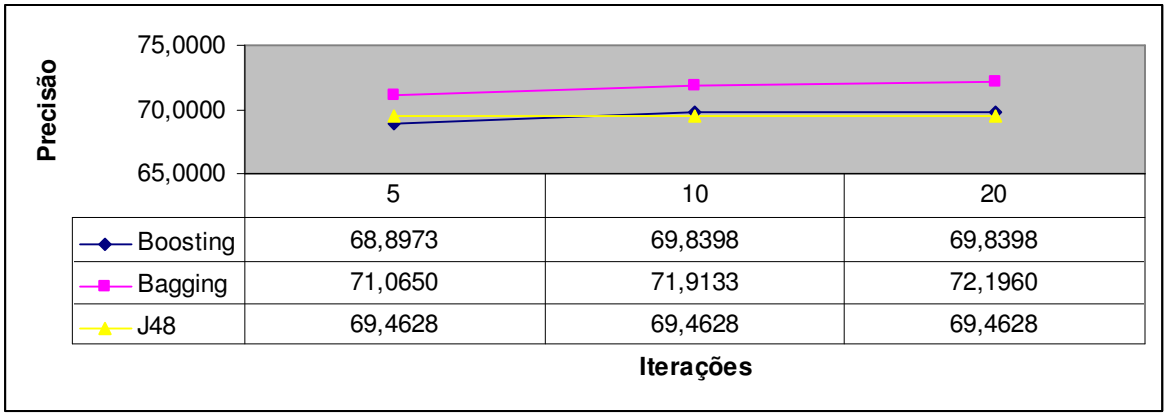


Gráfico 5.29. Desempenho dos algoritmos do estado-da-arte, Boosting, Bagging e J48, sobre o conjunto de dados DMOZ

Todos os algoritmos

O Gráfico 5.30 apresenta o desempenho dos algoritmos, executados com 5, 10 e 20 iterações, sobre o conjunto de dados DMOZ. Em todas as situações, ou seja, com 5, 10 e 20 iterações, a maior precisão foi alcançada pelo algoritmo Bagging. Com 5 iterações, a precisão obtida por ele foi de 71,0650%. Com 10 iterações, foi de 71,9133%. E, com 20 iterações, ele atingiu 72,1960% de precisão.

Portanto, a maior precisão atingida sobre o conjunto de dados DMOZ foi 72,1960%, sendo obtida pelo algoritmo Bagging.

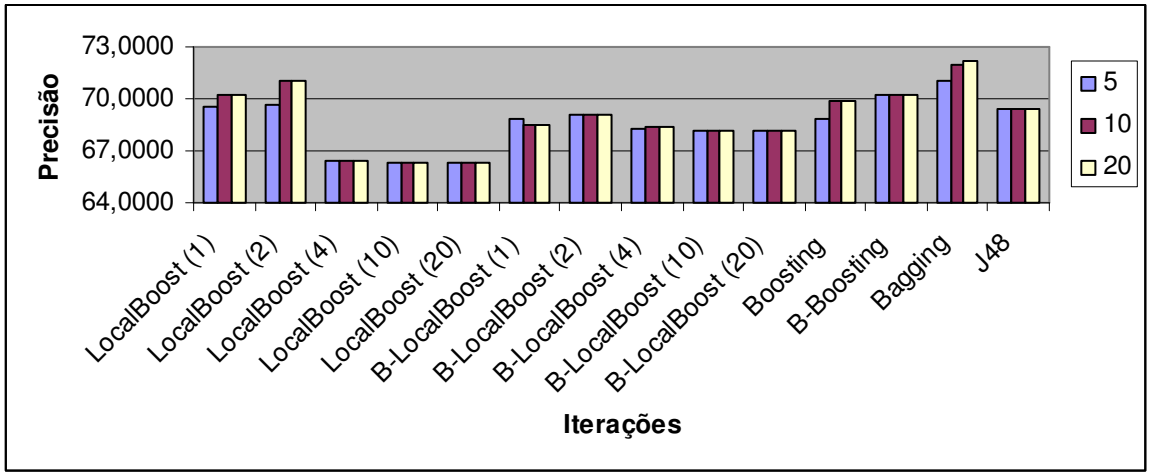


Gráfico 5.30. Comparação dos desempenhos obtidos pelos algoritmos sobre o conjunto de dados DMOZ

Tabela de ganhos e perdas

A Tabela 5.5 mostra os ganhos e perdas de cada algoritmo em relação aos outros sobre o conjunto de dados DMOZ.

Tabela 5.5. Tabela geral de ganhos e perdas de cada algoritmo

+/-	Boost.	B-Boost.	Bag.	J48	LocalB.*	B-LocalB.*
Boost.		0/3-	0/3-	2/1-	9/6-	13/1-
B-Boost.	3/0-		0/3-	3/0-	11/2-	15/0-
Bag.	3/0-	3/0-		3/0-	15/0-	15/0-
J48	1/2-	0/3-	0/3-		9/6-	15/0-
LocalB.*	6/9-	2/11-	0/15-	6/9-		6/9-
B-LocalB.*	1/13-	0/15-	0/15-	0/15-	9/6-	

*Considerando todas as vizinhanças (1, 2, 4, 10, e 20).

- Conjunto de dados NILC

O conjunto de dados NILC, conforme mencionado anteriormente, contém 855 instâncias e 5 classes, todas com a mesma quantidade de instâncias, ou seja, é um conjunto balanceado.

LocalBoost e B-LocalBoost

Como podemos ver nos Gráficos 5.31, 5.32 e 5.33, com 5, 10 e 20 iterações, respectivamente, o B-LocalBoost não apresentou boa performance sobre o conjunto de dados NILC. Com 1, 2, 10 e 20 vizinhanças, com 5 e 20 iterações, o B-LocalBoost obteve precisão menor que a de LocalBoost nestas mesmas situações, e conseguiu atingir a mesma precisão que LocalBoost com 4 vizinhanças. Entretanto, com 10 iterações, em todas as execuções, o B-LocalBoost teve precisão inferior à precisão de LocalBoost.

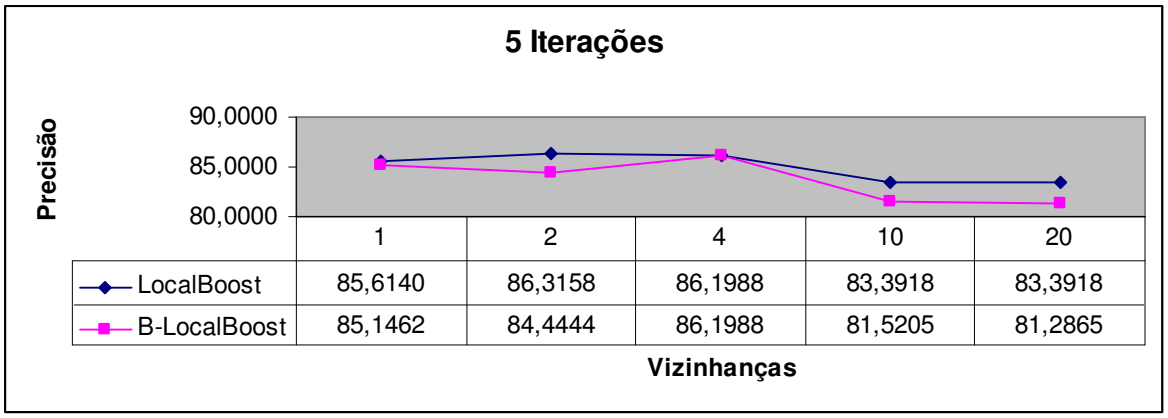


Gráfico 5.31. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados NILC com 5 iterações

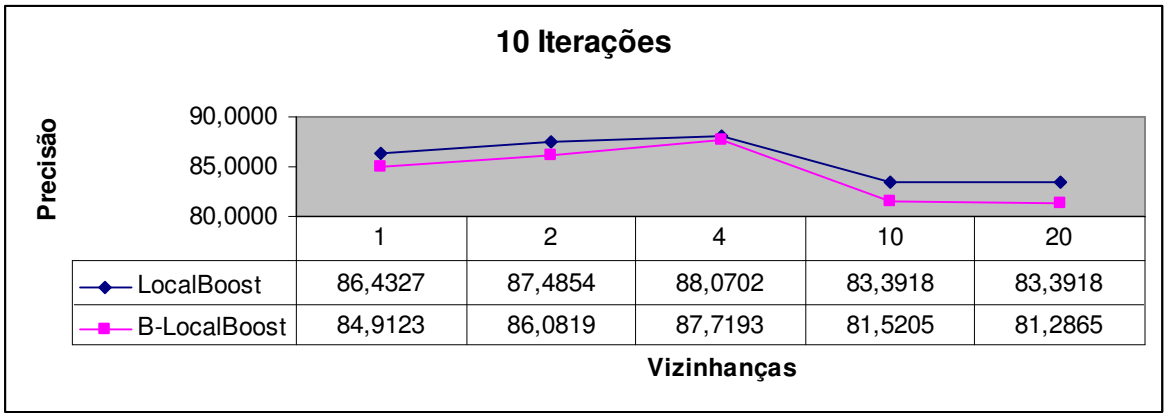


Gráfico 5.32. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados NILC com 10 iterações

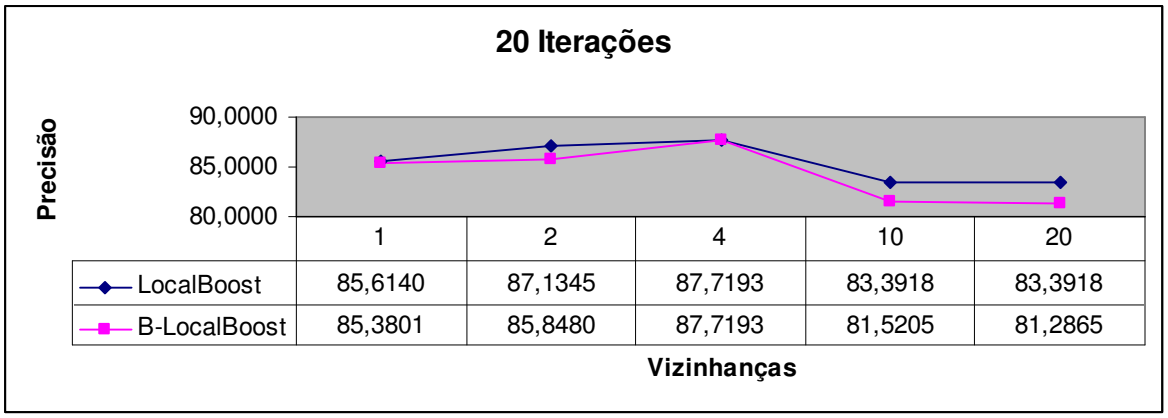


Gráfico 5.33. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados NILC com 20 iterações

Boosting e B-Boosting

O B-Boosting também não obteve bons resultados quando comparado ao Boosting. O B-Boosting não superou o Boosting em nenhuma das situações apresentadas no Gráfico 5.34. Ele apenas conseguiu alcançar a mesma precisão que Boosting com 5 iterações.

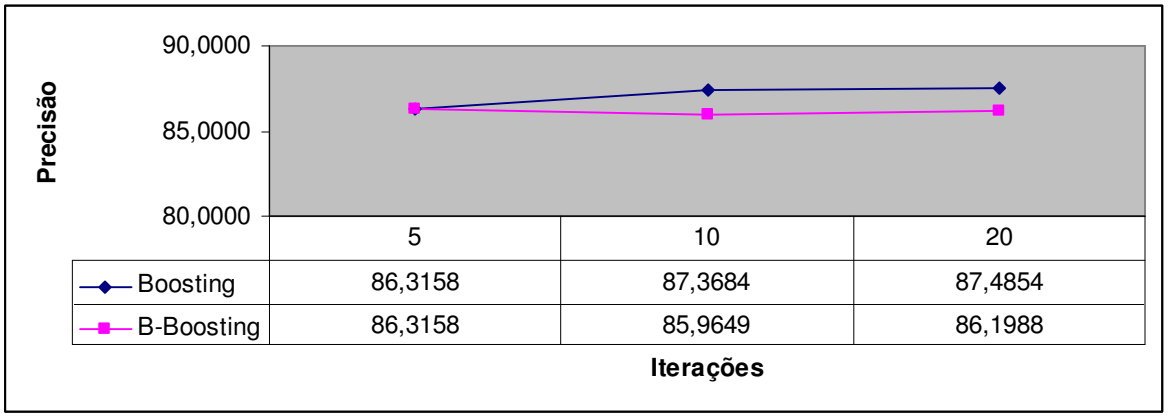


Gráfico 5.34. Desempenho dos algoritmos Boosting e B-Boosting sobre o conjunto de dados NILC

Algoritmos do Estado-da-Arte (Boosting, *Bagging* e J48)

Na comparação realizada do desempenho de Boosting com o de Bagging e com o de J48, Boosting demonstrou melhor performance que estes outros algoritmos sobre o conjunto de dados NILC. A comparação pode ser observada no Gráfico 5.35.

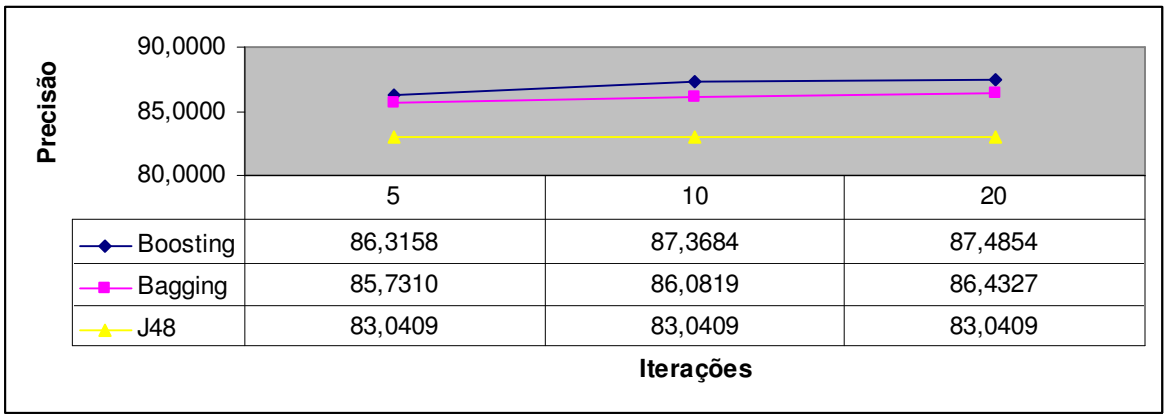


Gráfico 5.35. Desempenho dos algoritmos do estado-da-arte, Boosting, Bagging e J48, sobre o conjunto de dados NILC

Todos os algoritmos

O Gráfico 5.36 apresenta o desempenho dos algoritmos, executados com 5, 10 e 20 iterações, sobre o conjunto de dados NILC. Com 5 iterações, a maior precisão obtida foi 86,3158%, e foi alcançada pelo LocalBoost(2), Boosting e B-Boosting. Com 10 iterações, a maior precisão foi 88,0702%, sendo obtida pelo algoritmo LocalBoost(4). E, com 20 iterações, a maior precisão obtida foi 87,7193%, alcançada por LocalBoost(4) e por B-LocalBoost(4).

Portanto, a maior precisão atingida sobre o conjunto de dados NILC foi 88,0702%, sendo alcançada por LocalBoost(4).

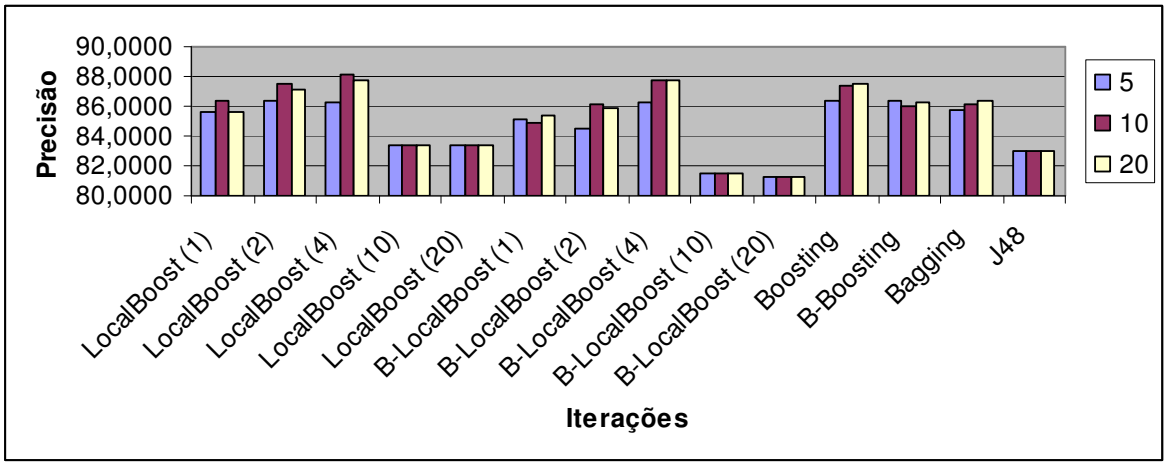


Gráfico 5.36. Comparação dos desempenhos obtidos pelos algoritmos sobre o conjunto de dados NILC

Tabela de ganhos e perdas

A Tabela 5.6 mostra os ganhos e perdas de cada algoritmo em relação aos outros sobre o conjunto de dados NILC.

Tabela 5.6. Tabela geral de ganhos e perdas de cada algoritmo

+/-	Boost.	B-Boost.	Bag.	J48	LocalB.*	B-LocalB.*
Boost.		2/0-	3/0-	3/0-	11/3-	13/2-
B-Boost.	0/2-		1/2-	3/0-	9/5-	12/3-
Bag.	0/3-	2/1-		3/0-	8/7-	11/3-
J48	0/3-	0/3-	0/3-		0/15-	6/9-
LocalB.*	3/11-	5/9-	7/8-	15/0-		13/0-
B-LocalB.*	2/13-	3/12-	3/11-	9/6-	0/13-	

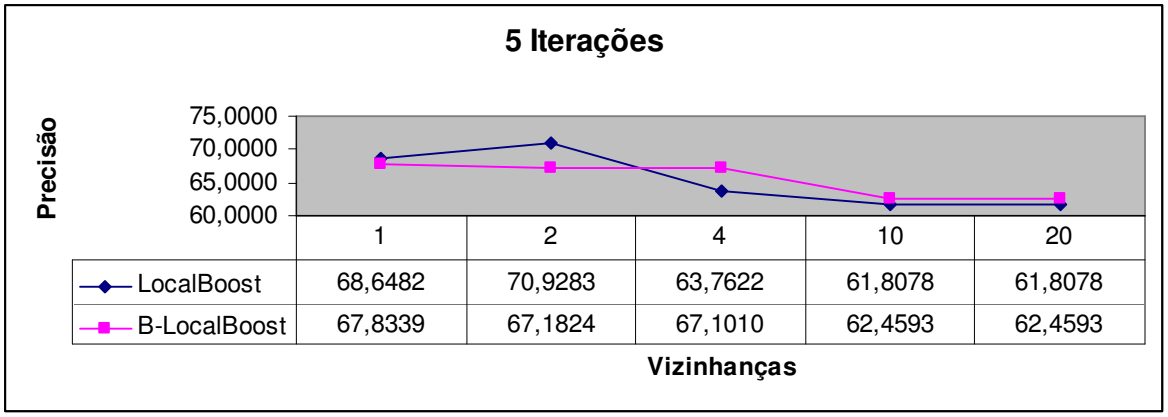
*Considerando todas as vizinhanças (1, 2, 4, 10, e 20).

- Conjunto de dados Reuters_100

Este conjunto de dados, conforme já mencionado, contém 1228 instâncias, 100 atributos e 65 classes, que possuem quantidade de instâncias diferentes, inclusive classes sem nenhuma representação no conjunto, sendo, assim, um conjunto desbalanceado.

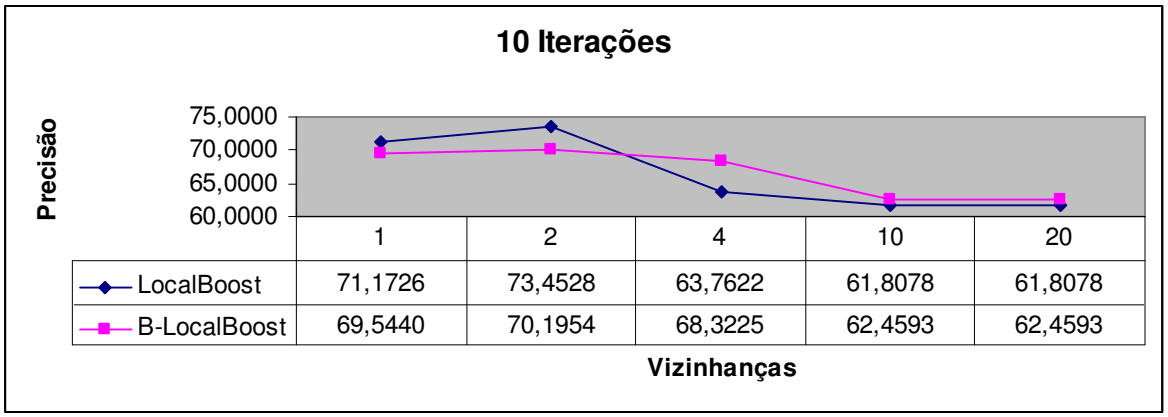
LocalBoost e B-LocalBoost

Através da análise dos Gráficos 5.37, 5.38 e 5.39, com 5, 10 e 20 iterações, respectivamente, podemos observar que o B-LocalBoost não conseguiu superar o desempenho do LocalBoost com 1 e 2 vizinhanças nas três situações apresentadas. Entretanto, nas três execuções, conseguiu a maior precisão com 4, 10 e 20 vizinhanças.



	1	2	4	10	20
LocalBoost	68,6482	70,9283	63,7622	61,8078	61,8078
B-LocalBoost	67,8339	67,1824	67,1010	62,4593	62,4593

Gráfico 5.37. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Reuters_100 com 5 iterações



	1	2	4	10	20
LocalBoost	71,1726	73,4528	63,7622	61,8078	61,8078
B-LocalBoost	69,5440	70,1954	68,3225	62,4593	62,4593

Gráfico 5.38. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Reuters_100 com 10 iterações

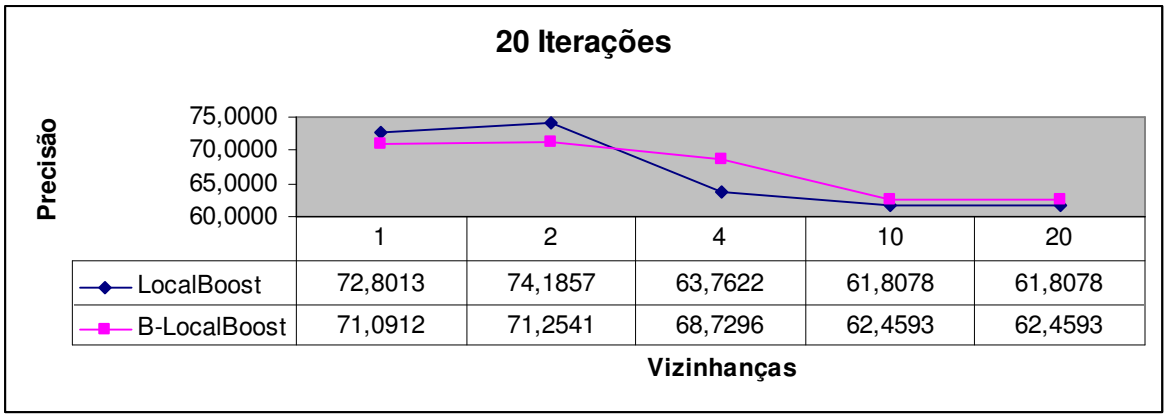


Gráfico 5.39. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Reuters_100 com 20 iterações

Boosting e B-Boosting

Diferente do B-LocalBoost, que conseguiu superar o desempenho do LocalBoost com 4, 10 e 20 vizinhanças, o B-Boosting não conseguiu obter precisão maior que o Boosting em nenhuma das três execuções, conforme podemos ver no Gráfico 5.40.

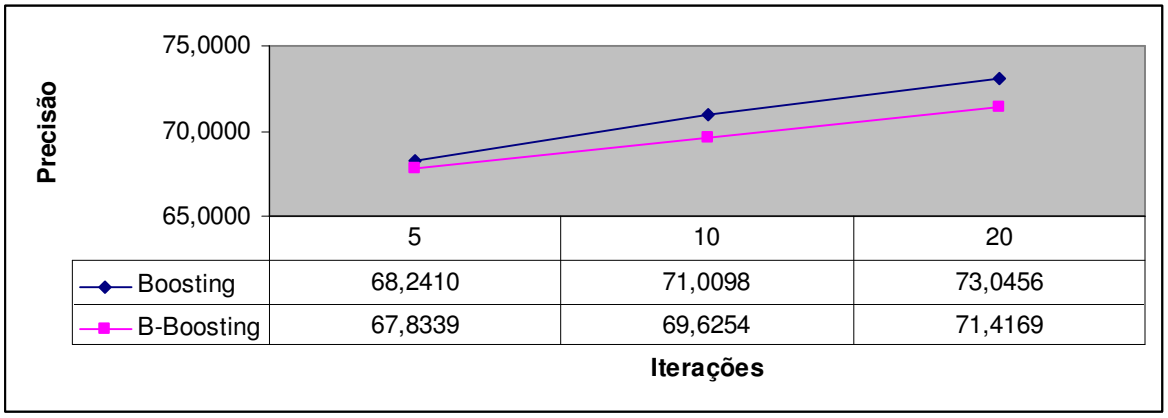


Gráfico 5.40. Desempenho dos algoritmos Boosting e B-Boosting sobre o conjunto de dados Reuters_100

Algoritmos do Estado-da-Arte (Boosting, *Bagging* e J48)

No Gráfico 5.41, que mostra os resultados dos algoritmos Boosting, Bagging e J48, é possível observar que, apesar de Boosting não ter alcançado a maior precisão em todas as situações exibidas no gráfico, ele conseguiu quase a mesma precisão que Bagging com 10 e 20 iterações, apresentando uma maior desvantagem apenas com 5 iterações em relação ao Bagging.

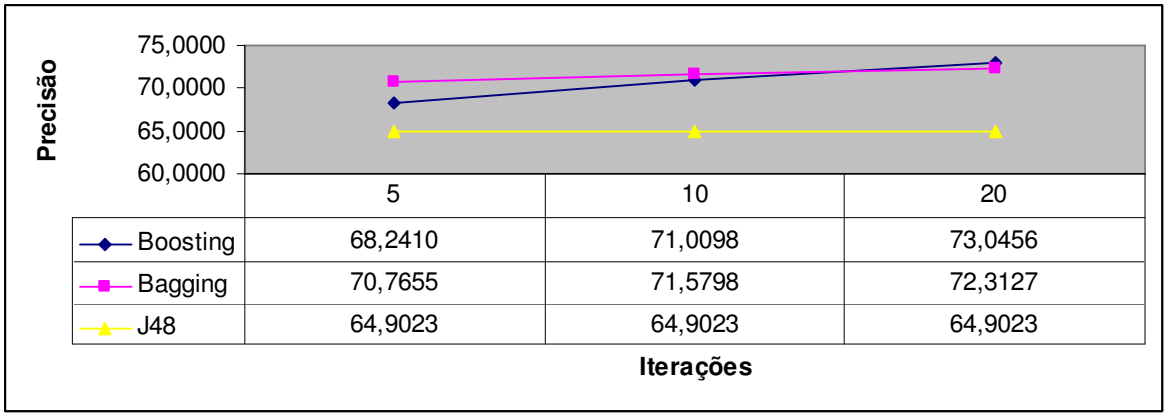


Gráfico 5.41. Desempenho dos algoritmos do estado-da-arte, Boosting, Bagging e J48, sobre o conjunto de dados Reuters_100

Todos os algoritmos

O Gráfico 5.42 apresenta o desempenho dos algoritmos, executados com 5, 10 e 20 iterações, sobre o conjunto de dados Reuters_100. Em todas as situações, ou seja, com 5, 10 e 20 iterações, a maior precisão foi alcançada pelo algoritmo LocalBoost(2). Com 5 iterações, a precisão obtida por ele foi de 70,9283%. Com 10 iterações, foi de 73,4528%. E, com 20 iterações, ele atingiu 74,1857% de precisão.

Portanto, a maior precisão atingida sobre o conjunto de dados Reuters_100 foi 74,1857%, sendo obtida pelo algoritmo LocalBoost(2).

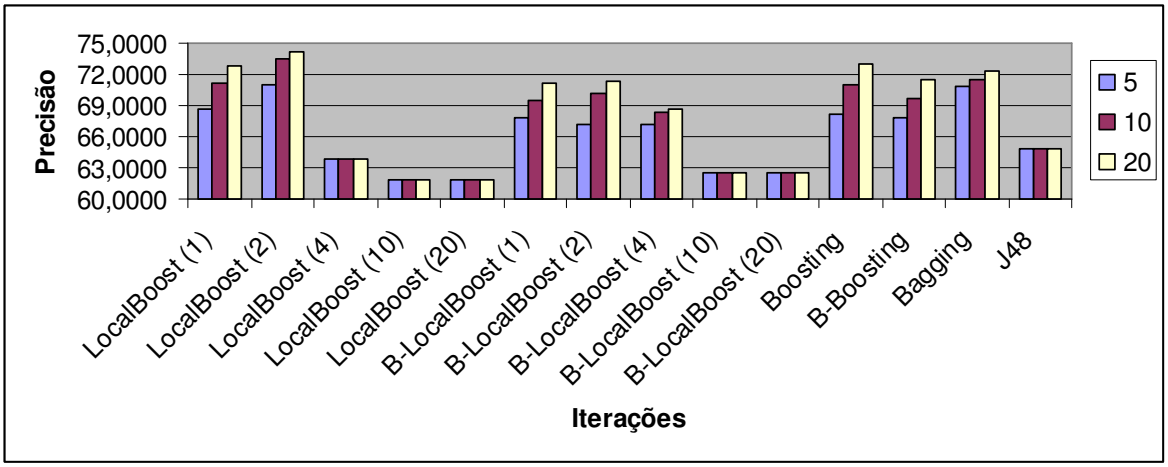


Gráfico 5.42. Comparação dos desempenhos obtidos pelos algoritmos sobre o conjunto de dados Reuters_100

Tabela de ganhos e perdas

A Tabela 5.7 mostra os ganhos e perdas de cada algoritmo em relação aos outros sobre o conjunto de dados Reuters_100

Tabela 5.7. Tabela geral de ganhos e perdas de cada algoritmo

+/-	Boost.	B-Boost.	Bag.	J48	LocalB.*	B-LocalB.*
Boost.		3/0-	1/2-	3/0-	10/5-	15/0-
B-Boost.	0/3-		0/3-	3/0-	9/6-	13/1-
Bag.	2/1-	3/0-		3/0-	11/4-	15/0-
J48	0/3-	0/3-	0/3-		9/6-	6/9-
LocalB.*	5/10-	6/9-	4/11-	6/9-		6/9-
B-LocalB.*	0/15-	1/13-	0/15-	9/6-	9/6-	

*Considerando todas as vizinhanças (1, 2, 4, 10, e 20).

- Conjunto de dados Reuters_500

O conjunto de dados Reuters_500, conforme já mencionado, contém 1228 instâncias, 500 atributos e 65 classes desbalanceadas. Possuindo, assim, quantidade de instâncias diferentes em cada uma delas, inclusive classes sem nenhuma representação no conjunto.

LocalBoost e B-LocalBoost

Analisando os Gráficos 5.43, 5.44 e 5.45 com os resultados, respectivamente, de 5, 10 e 20 iterações realizadas utilizando os algoritmos LocalBoost e B-LocalBoost sobre o conjunto Reuters_500, e os comparando com os Gráficos 5.37, 5.38 e 5.39 contendo o resultado do desempenho destes mesmos algoritmos sobre o conjunto Reuters_100, percebemos que o aumento da quantidade de atributos no conjunto proporcionou uma melhora no desempenho de B-LocalBoost. Com 5 iterações, sobre o conjunto Reuters_500, B-LocalBoost superou o desempenho de LocalBoost em quase todas as situações. Com 1 vizinhança, o B-LocalBoost teve uma precisão um pouco menor que a de LocalBoost. Entretanto, com 2, 4, 10 e 20 vizinhanças, ele superou o seu desempenho, apresentando uma grande diferença de precisão com 4 vizinhanças (Gráfico 5.43).

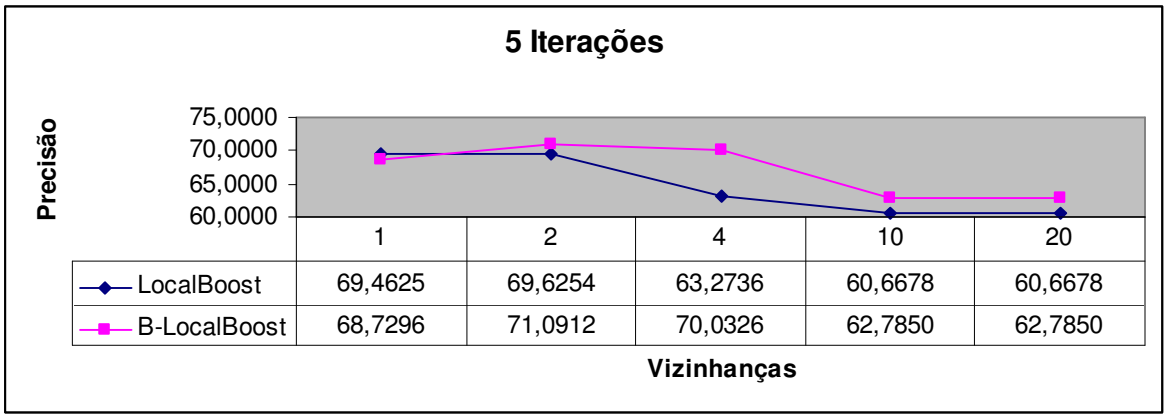


Gráfico 5.43. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Reuters_500 com 5 iterações

Já com 10 iterações, B-LocalBoost teve sua precisão menor que a de LocalBoost com 1 e 2 vizinhanças, mas voltou a superá-la com 4, 10 e 20 vizinhanças. E, desta vez, com 4 vizinhanças, a diferença de precisão foi maior ainda, chegando a ser de quase 8,5% (Gráfico 5.44).

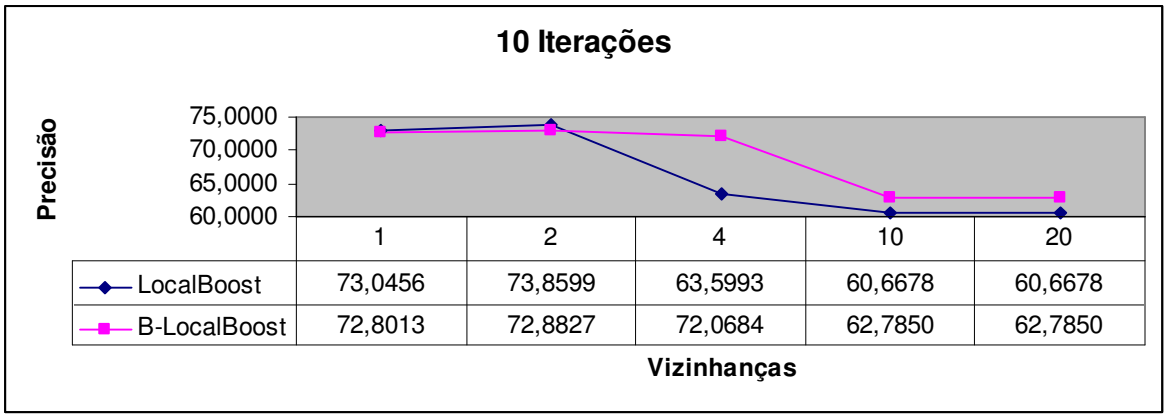


Gráfico 5.44. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Reuters_500 com 10 iterações

E, com 20 iterações, o desempenho de B-LocalBoost melhorou ainda mais. Com 1 e 2 vizinhanças, ele atingiu precisão um pouco menor que a de LocalBoost. Porém, com 4, 10 e 20 vizinhanças, superou novamente a precisão de LocalBoost. Sendo que, com 4 vizinhanças, a diferença de precisão aumentou mais ainda, chegando a quase 10%, como podemos verificar no Gráfico 5.45.

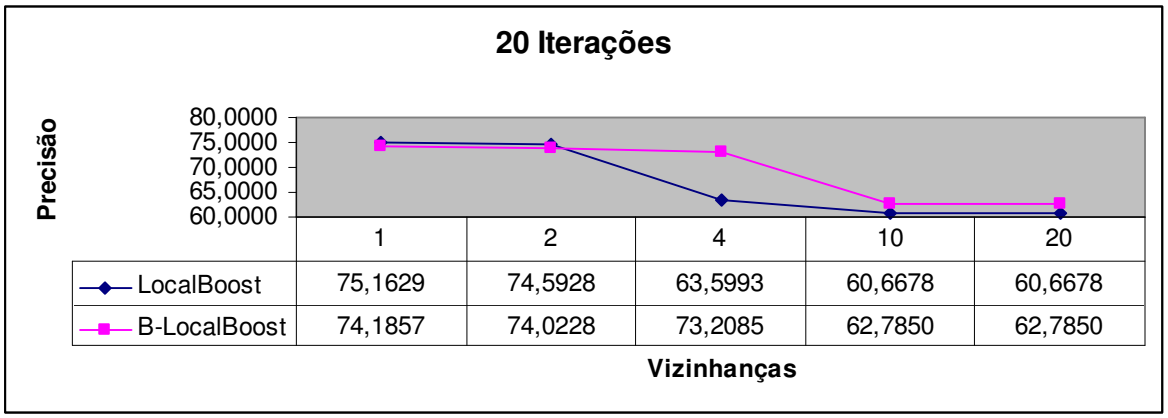


Gráfico 5.45. Desempenho dos algoritmos LocalBoost e B-LocalBoost sobre o conjunto de dados Reuters_500 com 20 iterações

Boosting e B-Boosting

Apesar do algoritmo B-LocalBoost ter apresentado uma performance muito boa perante LocalBoost, o algoritmo B-Boosting não conseguiu o mesmo diante do algoritmo Boosting. Em todas as execuções, com 5, 10 e 20 iterações, sobre o conjunto de dados Reuters_500, B-Boosting não conseguiu obter precisão maior que a de Boosting em nenhuma delas, conforme mostrado no Gráfico 5.46.

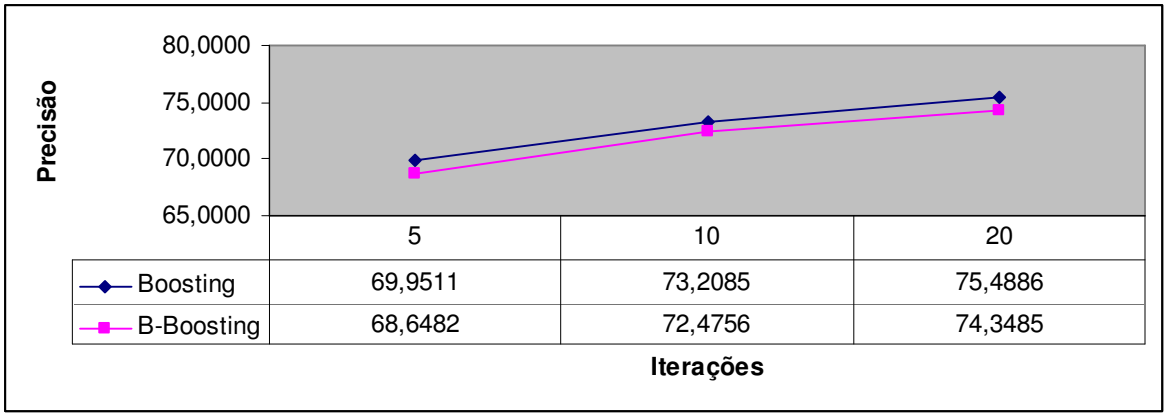


Gráfico 5.46. Desempenho dos algoritmos Boosting e B-Boosting sobre o conjunto de dados Reuters_500

Algoritmos do Estado-da-Arte (Boosting, *Bagging* e J48)

Como podemos ver no Gráfico 5.47, percebemos que o algoritmo Boosting, quando comparado aos algoritmos Bagging e J48 sobre o conjunto de dados Reuters_500, consegue melhorar seu desempenho, mesmo que pouco, ao longo das iterações.

Sobre este conjunto, o Boosting obteve precisão menor que a de Bagging com 5 iterações, mas conseguiu as precisões mais altas com 10 e 20 iterações, apesar de apresentar pequenas diferenças, nos dois casos.

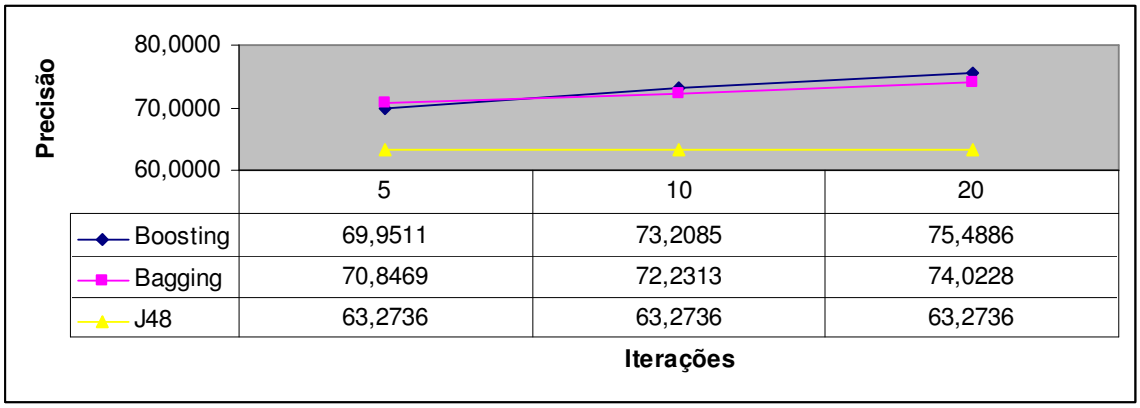


Gráfico 5.47. Desempenho dos algoritmos do estado-da-arte, Boosting, Bagging e J48, sobre o conjunto de dados Reuters_500

Todos os algoritmos

No Gráfico 5.48, que contém o desempenho de todos os algoritmos executados com 5, 10 e 20 iterações sobre o conjunto de dados Reuters_500, constatamos que, com 5 iterações, o melhor desempenho foi do algoritmo B-LocalBoost(2), com 71,0912% de precisão. Já com 10 iterações, a precisão mais alta, 73,8599%, foi obtida por LocalBoost(2). E, com 20 iterações, o algoritmo Boosting foi quem teve o melhor desempenho, atingindo 75,4886% de precisão.

No geral, a precisão mais alta, atingida sobre o conjunto de dados Reuters-_500, foi de 75,4886% de precisão, sendo alcançada pelo algoritmo Boosting.

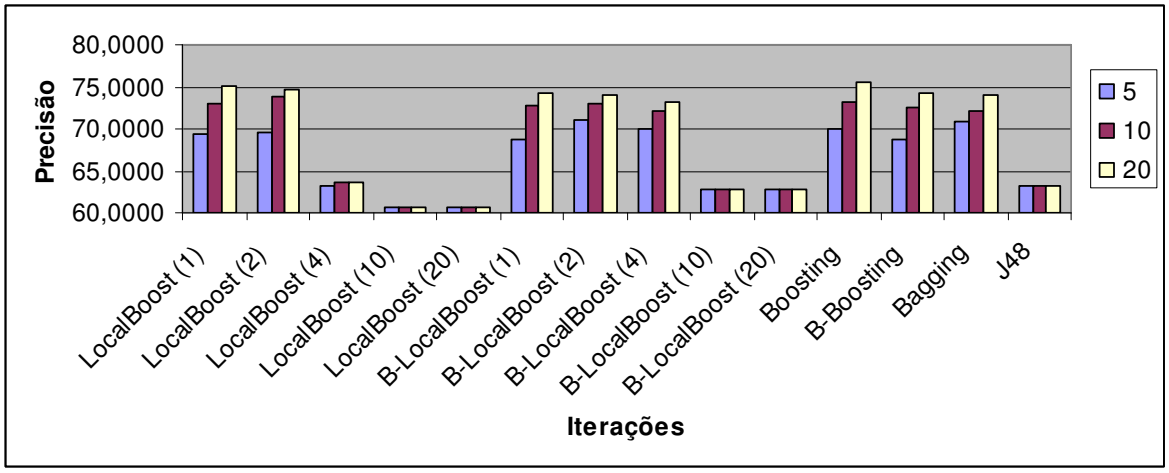


Gráfico 5.48. Comparação dos desempenhos obtidos pelos algoritmos sobre o conjunto de dados Reuters_500

Tabela de ganhos e perdas

A Tabela 5.8 mostra os ganhos e perdas de cada algoritmo em relação aos outros sobre o conjunto de dados Reuters_500

Tabela 5.8. Tabela geral de ganhos e perdas de cada algoritmo

+/-	Boost.	B-Boost.	Bag.	J48	LocalB.*	B-LocalB.*
Boost.		3/0-	2/1-	3/0-	14/1-	13/2-
B-Boost.	0/3-		2/1-	3/0-	9/6-	10/5-
Bag.	1/2-	1/2-		3/0-	11/4-	10/4-
J48	0/3-	0/3-	0/3-		6/8-	6/9-
LocalB.*	1/14-	6/9-	4/11-	8/6-		5/10-
B-LocalB.*	2/13-	5/10-	4/10-	9/6-	10/5-	

*Considerando todas as vizinhanças (1, 2, 4, 10, e 20).

LocalBoost, B-LocalBoost, W-LocalBoot, BW-LocalBoost e a influência do parâmetro β

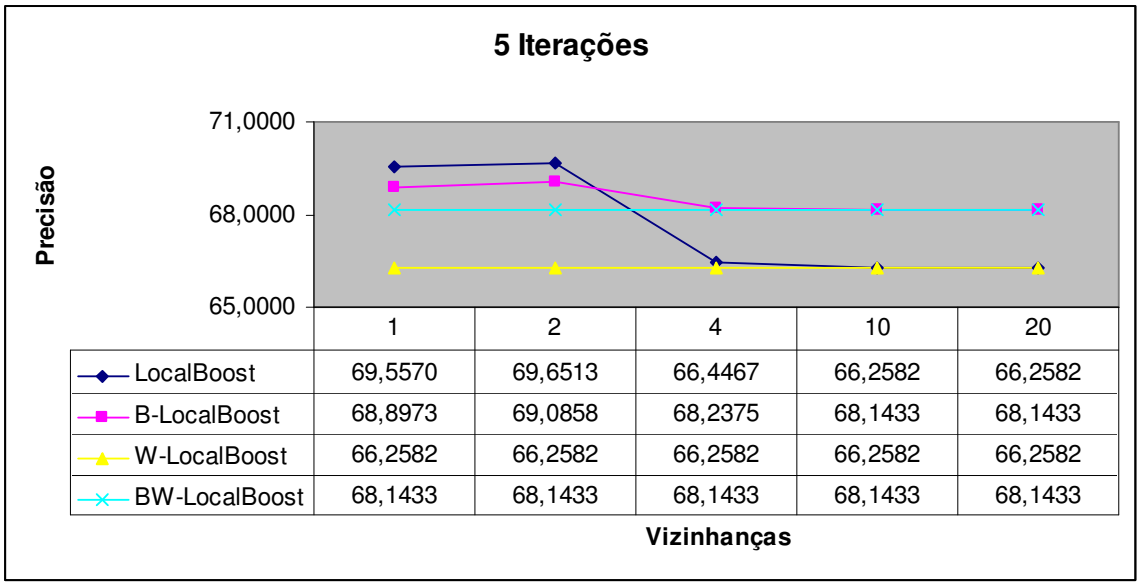
Outros experimentos foram executados com o intuito de analisar o desempenho dos algoritmos W-LocalBoost e BW-LocalBoost em relação ao desempenho de LocalBoost e B-LocalBoost. Os experimentos, utilizando estes

algoritmos, foram realizados sobre o conjunto de dados DMOZ, e com dois valores diferentes para o parâmetro β. Em um experimento, o valor de β foi 1, desta forma, o parâmetro não exercia influência na atualização dos pesos das instâncias. Mas, em outro experimento, o seu valor foi 5 (o valor 5 foi utilizado devido [Zhang and Zhang 2008] afirmar, após testes, que o parâmetro β deveria ter um valor pequeno, e utilizaram este valor em outros experimentos que realizaram).

Experimentos com β = 1

Como podemos ver nos Gráficos 5.49, 5.50 e 5.51, os algoritmos W-LocalBoost e BW-LocalBoost apresentaram comportamento estável em todos os experimentos realizados com valores diferentes de iterações (5, 10 e 20) e de vizinhanças (1, 2, 4, 10 e 20). Ao contrário de W-LocalBoost e BW-LocalBoost, os algoritmos LocalBoost e B-LocalBoost apresentaram oscilações no seu desempenho conforme algumas mudanças de iterações e vizinhanças.

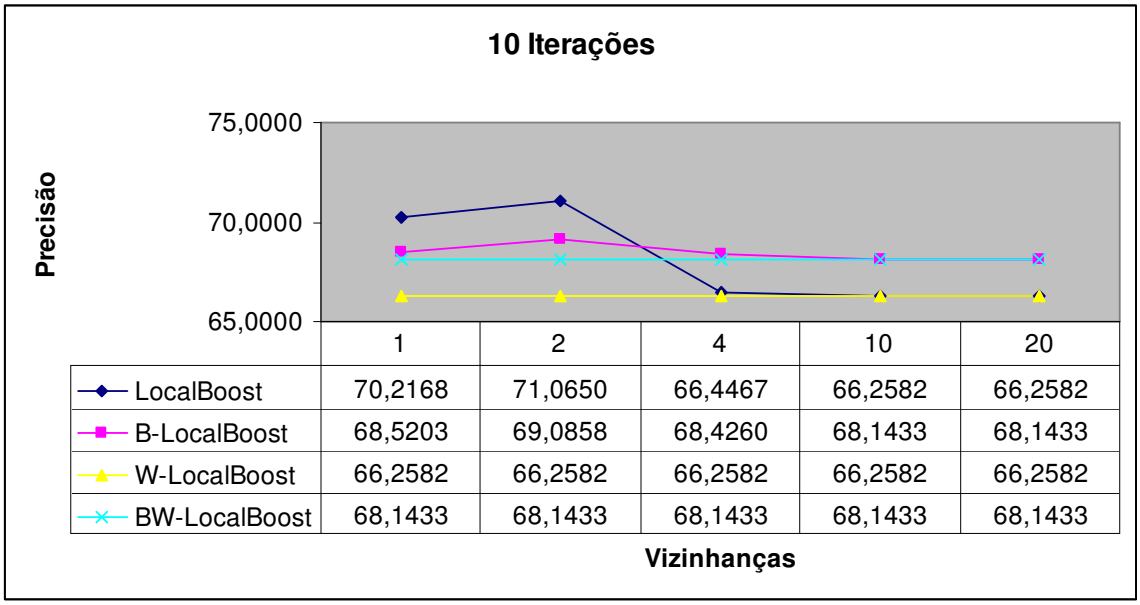
No Gráfico 5.49, a maior precisão, 69,6513%, foi alcançada pelo algoritmo LocalBoost com 2 vizinhanças. Entretanto, apesar de LocalBoost ter atingido as maiores precisões com 1 e 2 vizinhanças, a sua precisão cai consideravelmente com 4, 10 e 20 vizinhanças. Por outro lado, W-LocalBoost e BW-LocalBoost mantêm sempre o mesmo desempenho, mesmo com as mudanças de valores de parâmetros. Entretanto, a precisão obtida por eles está abaixo, principalmente a do W-LocalBoost, do desempenho apresentado pelo algoritmo B-LocalBoost, que, no geral, foi o algoritmo que demonstrou melhor desempenho com 5 iterações.



	1	2	4	10	20
LocalBoost	69,5570	69,6513	66,4467	66,2582	66,2582
B-LocalBoost	68,8973	69,0858	68,2375	68,1433	68,1433
W-LocalBoost	66,2582	66,2582	66,2582	66,2582	66,2582
BW-LocalBoost	68,1433	68,1433	68,1433	68,1433	68,1433

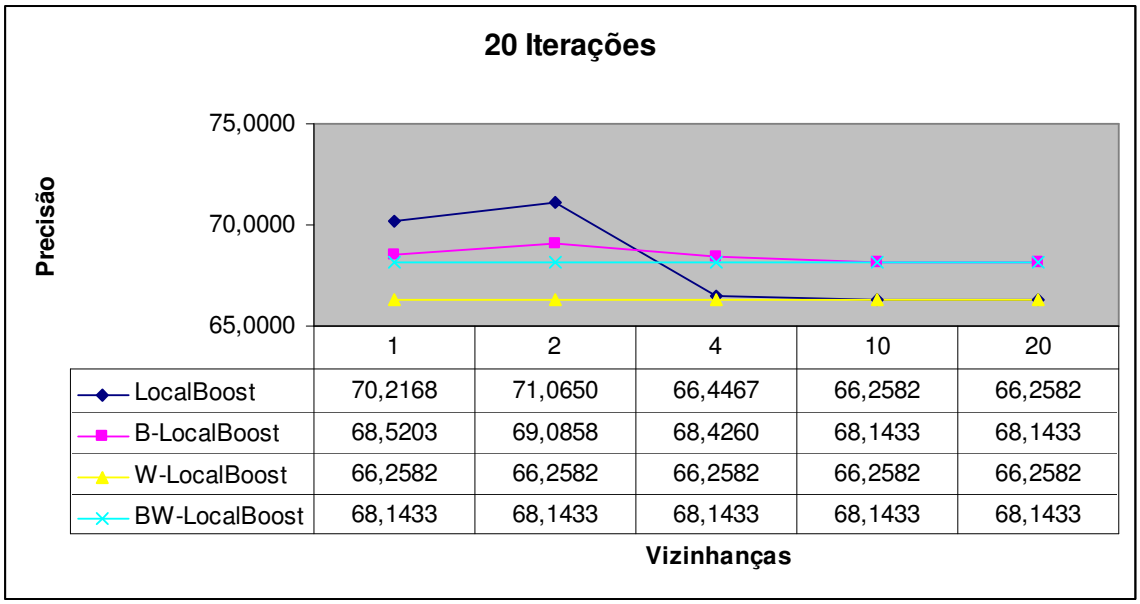
Gráfico 5.49. Resultado da aplicação de LocalBoost e variações, com 5 iterações e β = 1, sobre o conjunto de dados DMOZ

Com 10 e 20 iterações, o comportamento dos algoritmos foi o mesmo, como mostrado, respectivamente, nos Gráficos 5.50 e 5.51. Novamente, a maior precisão foi alcançada pelo algoritmo LocalBoost com 2 vizinhanças, em ambos os casos. Mas, da mesma forma, ele volta a cair com 4, 10 e 20 vizinhanças. Enquanto os algoritmos W-LocalBoost e BW-LocalBoost mantêm, no geral, o desempenho abaixo dos outros, e o B-LocalBoost, acima.



	1	2	4	10	20
LocalBoost	70,2168	71,0650	66,4467	66,2582	66,2582
B-LocalBoost	68,5203	69,0858	68,4260	68,1433	68,1433
W-LocalBoost	66,2582	66,2582	66,2582	66,2582	66,2582
BW-LocalBoost	68,1433	68,1433	68,1433	68,1433	68,1433

Gráfico 5.50. Resultado da aplicação de LocalBoost e variações, com 10 iterações e β = 1, sobre o conjunto de dados DMOZ



	1	2	4	10	20
LocalBoost	70,2168	71,0650	66,4467	66,2582	66,2582
B-LocalBoost	68,5203	69,0858	68,4260	68,1433	68,1433
W-LocalBoost	66,2582	66,2582	66,2582	66,2582	66,2582
BW-LocalBoost	68,1433	68,1433	68,1433	68,1433	68,1433

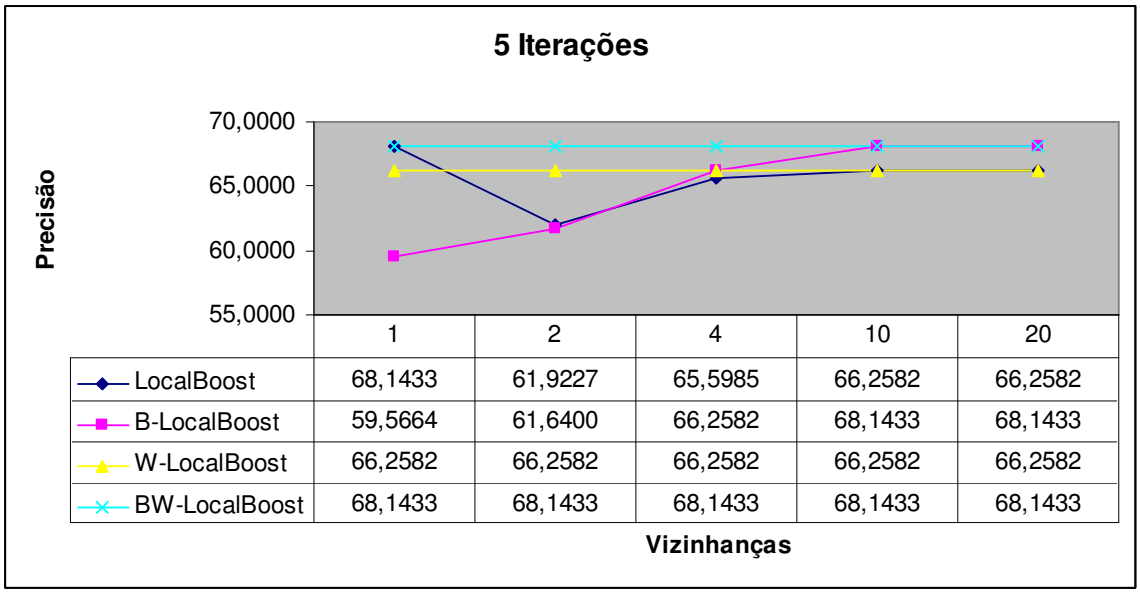
Gráfico 5.51. Resultado da aplicação de LocalBoost e variações, com 20 iterações e β = 1, sobre o conjunto de dados DMOZ

Experimentos com β = 5

Conforme já mencionado, outros testes foram executados com o algoritmo LocalBoost, e variações, sobre o conjunto de dados DMOZ, mas com o valor do parâmetro β igual a 5. Comparando os resultados destes experimentos com os realizados com β igual 1, percebemos que o desempenho de alguns algoritmos caiu.

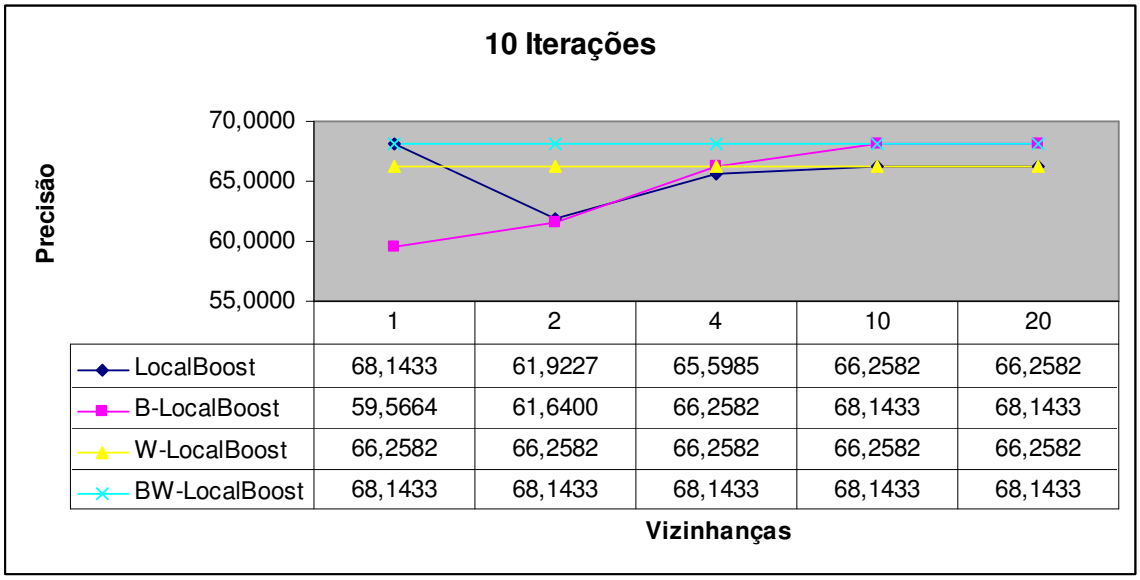
Os gráficos 5.52, 5.53 e 5.54 mostram os resultados obtidos com β igual a 5. E, através deles, percebemos que, independente da quantidade de vizinhanças e de iterações executadas, os valores não alteram. Outra constatação é que, os algoritmos que, anteriormente, haviam conseguido os melhores resultados, LocalBoost e B-LocalBoost, com β igual a 1, tiveram sua precisão diminuída, consideravelmente, com 1 e 2 vizinhanças, e nenhum aumento, da mesma, com 4, 10 e 20 vizinhanças. Nem os algoritmos W-LocalBoost e BW-LocalBoost tiveram benefício com este valor de β.

Assim, com a mudança do valor de β e as alterações de precisão causadas por ele, o algoritmo BW-LocalBoost acabou conquistando o melhor desempenho, conforme mostrado nos gráficos já citados.



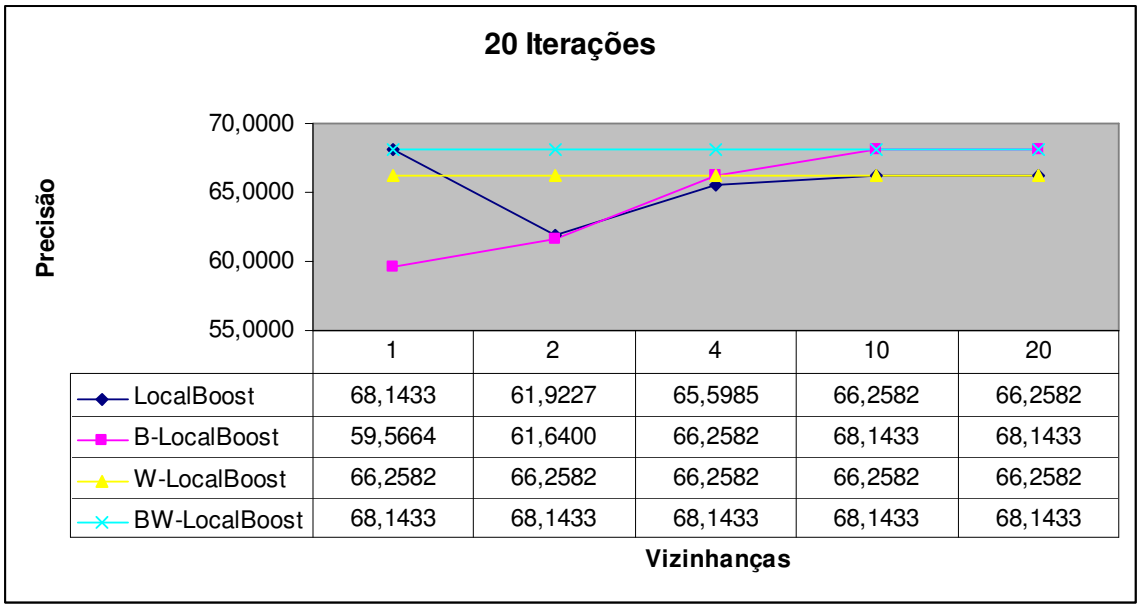
	1	2	4	10	20
LocalBoost	68,1433	61,9227	65,5985	66,2582	66,2582
B-LocalBoost	59,5664	61,6400	66,2582	68,1433	68,1433
W-LocalBoost	66,2582	66,2582	66,2582	66,2582	66,2582
BW-LocalBoost	68,1433	68,1433	68,1433	68,1433	68,1433

Gráfico 5.52 Resultado da aplicação de LocalBoost e variações, com 5 iterações e β = 5, sobre o conjunto de dados DMOZ



	1	2	4	10	20
LocalBoost	68,1433	61,9227	65,5985	66,2582	66,2582
B-LocalBoost	59,5664	61,6400	66,2582	68,1433	68,1433
W-LocalBoost	66,2582	66,2582	66,2582	66,2582	66,2582
BW-LocalBoost	68,1433	68,1433	68,1433	68,1433	68,1433

Gráfico 5.53 Resultado da aplicação de LocalBoost e variações, com 10 iterações e β = 5, sobre o conjunto de dados DMOZ



	1	2	4	10	20
LocalBoost	68,1433	61,9227	65,5985	66,2582	66,2582
B-LocalBoost	59,5664	61,6400	66,2582	68,1433	68,1433
W-LocalBoost	66,2582	66,2582	66,2582	66,2582	66,2582
BW-LocalBoost	68,1433	68,1433	68,1433	68,1433	68,1433

Gráfico 5.54 Resultado da aplicação de LocalBoost e variações, com 20 iterações e β = 5, sobre o conjunto de dados DMOZ

5.4 Considerações Finais

Neste capítulo foram mostrados os resultados experimentais obtidos com o algoritmo B-LocalBoost, e outras variações, utilizando conjuntos de dados variados do mundo real. Diante dos resultados apresentados, é possível perceber que LocalBoost apresenta, geralmente, resultados muito bons quando executado com 2 vizinhanças.

Em relação ao algoritmo B-LocalBoost, ele demonstrou conseguir melhorar, com sua camada de balanceamento, a precisão do LocalBoost perante dados desbalanceados, conforme pudemos verificar em várias situações, principalmente com 4, 10 e 20 vizinhanças, como nos conjuntos de dados Ionosphere, DMOZ, Reuters com 100 e 500 atributos, entre outros. Entretanto, quando o conjunto não apresentava nenhum nível de desbalanceamento (conjuntos de dados Íris e NILC) ou quando havia atributos sem valores (conjunto de dados Hepatitis), o seu desempenho apresentava queda considerável.

Quanto ao B-Boosting, que foi fruto da integração da camada de balanceamento, aqui proposta, ao Boosting, também conseguiu, em alguns casos, superar as precisões alcançadas por Boosting, por exemplo, no conjunto de dados Ionosphere e DMOZ.

Por fim, analisamos o comportamento dos algoritmos W-LocalBoost e BW-LocalBoost em relação ao comportamento de LocalBoost e B-LocalBoost, quando executados sobre o conjunto de dados DMOZ, além de verificar, também, a influência do parâmetro β nestes algoritmos. E pudemos concluir que, W-LocalBoost e BW-LocalBoost obtiveram, no geral, desempenho inferior ao de LocalBoost e o de B-LocalBoost, da mesma forma que a utilização do valor 5 para o parâmetro β não proporcionou, nos experimentos realizados, nenhum benefício à classificação.

6 Conclusões

Neste trabalho foi proposto um algoritmo, denominado *B-LocalBoost*, a ser utilizado no processo de indução de classificadores de textos mais precisos sobre conjuntos de dados que estejam desbalanceados.

Baseado no algoritmo *LocalBoost*, que é uma variação de classificação local do conhecido algoritmo *Boosting*, o algoritmo aqui proposto utiliza técnicas de balanceamento de dados semelhante ao do sistema BEV, para melhorar o processo de classificação, e está baseado em duas premissas. A primeira delas é que conjuntos de dados desbalanceados geralmente fazem com que os algoritmos de classificação gerem classificadores tendenciosos, no sentido de beneficiar as classes majoritárias, classificando corretamente uma quantidade maior de instâncias pertencentes a ela, em detrimento da classificação das instâncias das classes minoritárias. A segunda premissa diz que, em muitos problemas de classificação, a classificação correta de instâncias de classes com menor representação no conjunto é o objetivo principal buscado.

Diante das premissas acima, a criação e o uso do *B-LocalBoost* foram motivados pelo fato de que muitos conjuntos de dados do mundo real encontram-se desbalanceados, precisando, assim, de um algoritmo capaz de tratar este problema, que afeta diretamente a precisão do classificador gerado, principalmente em relação às classes minoritárias.

6.1 Principais Contribuições

As principais contribuições deste trabalho foram:

- o desenvolvimento de uma pesquisa abrangente sobre combinação de classificadores;
- a investigação aprofundada de métodos locais de *Boosting;*

- o projeto e a implementação de novos algoritmos de combinação, com o intuito de solucionar o problema ocasionado pelo desbalanceamento presente em muitos conjuntos de dados.

A dificuldade de se encontrar uma solução para o problema dos dados desbalanceados é uma realidade identificada no levantamento bibliográfico, e comprovada nos experimentos realizados com o algoritmo *B-LocalBoost*, e as outras versões aqui apresentadas, devido ao fato de cada conjunto de dados possuir uma particularidade.

Apesar das dificuldades encontradas no desenvolvimento deste projeto, pode-se verificar, através dos resultados apresentados, que *B-LocalBoost* conseguiu alcançar resultados muito bons em vários casos. Por outro lado, sabe-se que melhorias ainda têm que ser feitas.

6.2 Trabalhos Futuros

Como trabalhos futuros, indicamos as seguintes atividades:

- Realização de análise detalhada dos conjuntos de dados, para a descoberta das causas das oscilações de desempenho apresentadas pelo *B-LocalBoost* sobre alguns desses conjuntos;
- Execução de mais testes utilizando outros algoritmos para a classificação base, como por exemplo o uso do algoritmo SVM, e posterior análise dos resultados obtidos;
- Aplicação do método gráfico de análise ROC (*Receiver Operating Characteristic*) para a avaliação de modelos de classificação, pois é útil em domínios nos quais existe uma grande desproporção entre as classes, ou quando se deve levar em consideração diferentes custos/benefícios para os diferentes erros/acertos de classificação;
- Experimentos utilizando outros valores para o parâmetro β, com realização, também, de análise dos resultados; e

Desenvolvimento de outras técnicas que utilizem a distância existente entre as instâncias, de forma que consiga influenciar, positivamente, o desempenho do algoritmo na classificação dos dados.

Referências Bibliográficas

[Aha and Kibler 1991] AHA, D., KIBLER, D. 1991. Instance-based learning algorithms. *Machine Learning*. Vol.6 (3), pp. 37-66.

[Akbani *et al.* 2004] AKBANI, R., KWEK, S., AND JAPKOWICZ, N. 2004. Applying support vector machines to imbalanced datasets. In *Proceedings of the 15th European Conference on Machine Learning*, Pisa, IT, 39-50.

[Baeza-Yates and Ribeiro-Neto 1999] BAEZA-YATES, R. AND RIBEIRO-NETO, B. 1999. *Modern Information Retrieval*. Addison Wesley.

[Bennett *et al.* 2005] BENNETT, P. N., DUMAIS, S. T. AND HORVITZ, E. 2005. The Combination of Text Classifiers Using Reliability Indicators. *Information Retrieval* 8, 1, 67-100.

[Berger *et al.* 1996] BERGER, A. L., PIETRA, V. J. AND PIETRA, S. A. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics* 22, 1, 39-71.

[Breiman *et al.* 1984] BREIMAN, L., FRIEDMAN, J.H., OLSHEN, R.A. AND STONE, C.J.1984. *Classification and Regression Trees*. Wadsworth, Belmont, CA.

[Breiman 1996a] BREIMAN, L. 1996. Bagging Predictors. *Machine Learning* 24, 2, 123-140.

[Breiman 1996b] BREIMAN, L. 1996. Stacked Regressions. *Machine Learning* 24, 1, 49-64.

[Breiman 2001] BREIMAN, L. 2001. Random Forests. *Machine Learning* 45, 1, 5-32.

[Burges 1998] BURGES, C. J. 1998. A Tutorial on Support Vector Machines for Pattern Recognition. Data Min. Knowl. Discov. 2, 2, 121-167.

[Busemann *et al*. 2000] BUSEMANN, S., SCHMEIER, S. AND ARENS, R. 2000. Message Classification in the Call Center. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, Seattle, Washington, April 29 - May 04, 2000, Applied Natural Language Conferences. Association for Computational Linguistics, Morristown, NJ, 158-165.

[Chawla *et al*. 2002] CHAWLA, N., BOWYER, K., HALL, L. AND KEGELMEYER, W. P. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 1, 321-357.

[Chawla *et al*. 2003] CHAWLA, N.V., LAZAREVIC, A., HALL, L.O. AND BOWYER, K.W. 2003. SMOTEBoost: improving prediction of the minority class in boosting. In *Proceedings of the Seventh European Conference on Principles and Practice of Knowledge Discovery in Databases*, Dubrovnik, Croatia, 107-119.

[Cohen and Singer 1999] COHEN, W. AND SINGER, Y. 1999. Context sensitive learning methods for text categorization. *ACM Transactions on Information Systems* 17, 2, 141-173.

[Cover and Hart 1967] COVER, T., HART, P. 1967. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13, 21-27.

[Dietterich 2000] DIETTERICH, T. G. 2000. Ensemble Methods in Machine Learning. In *Proceedings of the First international Workshop on Multiple Classifier Systems*, J. KITTLER AND F. ROLI, Eds. Lecture Notes In Computer Science, 1857, Springer-Verlag, London, UK, 1-15.

[Du and Chen 2005] DU, S. AND CHEN, S. 2005. Weighted Support Vector Machine for Classification. *IEEE International Conference on Systems, Man and Cybernetics*, 4, 3866-3871.

[Estabrooks *et al*. 2004] ESTABROOKS, A., JO, T., AND JAPKOWICZ, N. 2004. A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence* 20, 1, 18-36.

[Faria 2007] FARIA, B. A. 2007. *Classificação Textual Via Máquinas De Vetores Suporte (SVM)*. Puc-Rio. Disponível em: http://www:puc-rio.br/ensinopesq/ccpg/pibic/relatorioresumo2006/Resumos2006/CTC/INF-OK/BrenoAlbertiFaria.pdf. Acessado em: 04/10/2007.

[Fix and Hodges 1951] FIX, E. AND HODGES, J. L. 1951. Discriminatory analysis: Non-parametric discrimination: Consistency properties. *USAF School of Aviation and Medicine, Technical Report* 4, 261-279.

[Frank and Bouckaert 2006] FRANK, E. AND BOUCKAERT, R. R. 2006. Naive Bayes for Text Classification with Unbalanced Classes. *Knowledge Discovery in Databases: PKDD 2006, Lecture Notes in Computer Science*, 4213, 503-510.

[Freund and Schapire 1995] FREUND, Y. AND SCHAPIRE, R. E. 1995. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *Proceedings of the Second European Conference on Computational Learning theory*, P. M. VITÁNYI, Eds. Lecture Notes In Computer Science, 904, Springer-Verlag, London, UK, 23-37.

[Good 1965] GOOD, I. J. 1965. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. M.I.T. Press.

[Holland 1975] HOLLAND, J. H. 1975. *Adaptation in natural and artificial systems*. University of Michigan Press (reprinted in 1992 by MIT Press, Cambridge, MA).

[Hotho *et al*. 2005] HOTHO, A., NÜRNBERGER, A. AND PAAß, G. 2005. A Brief Survey of Text Mining. *Journal for Computational Linguistics and Language Technology*, 20, 1, 19-62.

[Iyer *et al.* 2000] IYER, R. D., LEWIS, D. D., SCHAPIRE, R. E., SINGER, Y. AND SINGHAL, A. 2000. Boosting for Document Routing. In *Proceedings of the Ninth international Conference on information and Knowledge Management*, McLean, E.U.A., Novembro 2000, CIKM '00, ACM, New York, NY, 70-77.

[Joachims 1999] JOACHIMS, T. 1999. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Learning*, B. SCHÖLKOPF, C. J. BURGES, AND A. J. SMOLA, Eds. MIT Press, Cambridge, MA, 169 -184.

[John and Langley 1995] JOHN, G. H. AND LANGLEY, P. 1995. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, San Mateo, pp. 338-345.

[Kibriya *et al*. 2004] KIBRIYA, A.M., FRANK, E., PFAHRINGER, B., HOLMES, G. 2004. Multinomial naive Bayes for text categorization revisited. In *Proc. Australian Conf. on AI*, 488-499.

[Kim *et al*. 2005] KIM, H., HOWLAND, P. AND PARK, H. 2005. Dimension reduction in text classification with support vector machines. *Journal of Machine Learning Research*, 6, 37-53.

[kNN-PUC 2007] kNN-PUC *KNN (K - Nearest Neighbors)*. Puc-Rio - Certificação Digital. Disponível em: http://www:Maxwell.lambda.ele.puc-rio.br/cgi-bin/PRG_0599.EXE/7587_6.PDF?NrOcoSis=21790&CdLinPrg=pt. Acessado em: 25/10/2007.

[Kubat and Matwin 1997] KUBAT, M. AND MATWIN, S. 1997. Addressing the curse of imbalanced data set: One sided sampling. In *Proceedings of the Fourteenth International Conference on Machine Learning*, Eds. Morgan Kaufmann, San Francisco, CA, 179-186.

[Lafferty *et al.* 2001] LAFFERTY, J. D., MCCALLUM, A. AND PEREIRA, F. C. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth international Conference on Machine Learning*, Junho – July 2001, C. E. BRODLEY AND A. P. DANYLUK, Eds. Morgan Kaufmann, San Francisco, CA, 282-289.

[Larkey and Croft 1996] LARKEY, L. S. AND CROFT, W. B. 1996. Combining Classifiers in Text Categorization. In *Proceedings of the 19th Annual international ACM SIGIR Conference on Research and Development in information Retrieval*, Zurique, Suíça, Agosto 1996, SIGIR '96, ACM, New York, NY, 289-297.

[Li 2007] LI, C. 2007. Classifying Imbalanced Data Using a Bagging Ensemble Variation (BEV). 2007. In *Proceedings of the 45th Annual Southeast Regional Conference*, ACM, New York, NY, 203-208.

[Li and Jain 1998] LI, Y. H. AND JAIN, A. K. 1998. Classification of Text Documents. *The Computer Journal* 41, 8, 537-546.

[Li and Park 2006] LI, C. H. AND PARK, S. C. 2006. A novel algorithm for text categorization using improved backpropagation neural network. *Lecture Notes Computer Science*, 4223, 452-460.

[Ling and Li 1998] LING, C. AND LI, C. 1998. Data Mining for Marketing: Problems and Solutions. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, 73-79.

[Machado and Ladeira 2007] MACHADO, E. L. AND LADEIRA, M. 2007. Um Estudo de Limpeza em Base de Dados Desbalanceada com Sobreposição de Classes. *VI Encontro Nacional de Inteligência Artificial*.

[Martins *et al*. 2007] MARTINS, D., DIONÍSIO, G. AND DECKMANN, R. O. 2007. Árvores e Tabelas de Decisão. Universidade do Vale do Rio dos Sinos – UNISINOS. Disponível em: http://www.inf.unisinos.br/~cazella/dss/200601/ad_td.pdf. Acessado em: 12/12/2007.

[Mitchell 1997] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[Nilsson 1965] NILSSON, N. J. 1965 *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw Hill, New York, EUA.

[Pila 2006] PILA, A. D. 2006. *História e Terminologia a Respeito da Computação Evolutiva*. Revista de Ciências Exatas e Tecnologia, 1, 1, 1-25.

[Provost 2000] PROVOST, F. 2000. Machine learning from imbalanced data sets. *Proceedings* of the AAAI Workshop on Imbalanced Data Sets.

[Quilan 1986] QUINLAN, J. R. 1986. Induction of decision trees. *Machine Learning*, 1, 1, 81-106.

[Quinlan 1993] QUINLAN, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA.

[Rezende 2005] REZENDE, S. O. 2005. *Sistemas Inteligentes: Fundamentos e Aplicações*. Manole, Barueri, SP.

[Salton and Buckley 1988] SALTON, G. AND BUCKLEY, C. 1988. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24, 5, 513-523.

[Schapire 2002] SCHAPIRE, R.E. 2002. The Boosting Approach to Machine Learning an Overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, Berkeley, CA, 1-23.

[Scholkopy *et al*. 2000] SCHÖLKOPY, B., SMOLA, A. J. AND BARTLETT, P. L. 2000. New support vector algorithms. *Neural Computation*, 12, 5, 1207-1245.

[Sebastiani 2002] SEBASTIANI, F. 2002. Machine learning in automated text categorization. ACM Computing Surveys 34, 1, 1-47.

[Semolini 2007] SEMOLINI, R. 2007 *Support Vector Machines, Inferência Transdutiva e o Problema de Classificação*. Unicamp. Disponível em: ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/theses/semolini_mest/semolini_tese_mest.pdf. Acessada em: 04/10/2007.

[Sigletos *et al.* 2005] SIGLETOS, G., PALIOURAS, G., SPYROPOULOS, C. D., AND HATZOPOULOS, M. 2005. Combining Information Extraction Systems Using Voting and Stacked Generalization. *The Journal of Machine Learning Research* 6, 1751-178.

[Silva 2004] SILVA, C. F. 2004. Uso de Informações Lingüísticas na Etapa de Pré-Processamento em Mineração de Textos. *Dissertação de Mestrado*, 1-109.

[Sinoara *et al.* 2003] SINOARA, R.A., PUGLIESI, J.B. AND REZENDE, S.O. 2003. Combinação de Classificadores Homogêneos e Heterogêneos. *Relatório Técnico do ICMC-USP* 184, 1-26.

[Stein *et al*. 2005] STEIN, G., CHEN, B., WU, A. S., AND HUA, K. A. 2005. Decision tree classifier for network intrusion detection with GA-based feature selection. In *Proceedings of the 43rd Annual Southeast Regional Conference* - 2. ACM-SE 43. ACM, New York, NY, 136-141.

[Tan 2006] TAN, S. 2006. An effective refinement strategy for KNN text classifier. *Expert Systems with Applications*, 30, 2, 290-298.

[Ting and Witten 1997] TING, K. M. AND WITTEN I. H. 1997. Stacked Generalization: When Does it Work?. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Nagóia, Japão, Agosto 1997, Eds. Morgan Kaufmann, 866-873.

[Todorovski and Dzeroski 2000] TODOROVSKI, L. AND DZEROSKI, S. 2000. Combining Multiple Models with Meta Decision Trees. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, Lyon, France, September 2000, D. A. ZIGHED, H. J. KOMOROWSKI AND J. M. ZYTKOW, Eds. Lecture Notes In Computer Science, 1910, Springer-Verlag, London, UK, 54-64.

[Todorovski and Džeroski 2003] TODOROVSKI, L. AND DŽEROSKI, S. 2003. Combining Classifiers with Meta Decision Trees. *Machine Learning* 50, 3, 223-249.

[Tsoumakas and Katakis 2007] TSOUMAKAS, G., KATAKIS, I.: Multi-label classification: An overview. International Journal of Data Warehousing and Mining 3(3) (2007) 1-13.

[Vapnik 1995] VAPNIK, V. N. 1995. *The nature of statistical learning theory*. Springer.

[Veloso and Meira 2005] VELOSO, A. AND MEIRA, W. 2005. Eager, lazy and hybrid algorithms for multi-criteria associative classification. *Anais do Workshop sobre Algoritmos de Mineração de Dados (WAMD)*, 17-25.

[Vilalta *et al.* 2005] VILALTA R., GIRAUD-CARRIER C., AND BRAZDIL P. 2005. Meta-Learning: Concepts and Techniques. In *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers*, O. Maimon and L. Rokach, Eds. Springer, 731 - 748.

[Visa and Ralescu 2005] VISA, S. AND RALESCU, A. 2005. Issues in Mining Imbalanced Data Sets - A Review Paper. In *Proceedings of the Sixteen Midwest Artificial Intelligence and Cognitive Science Conference*, MAICS-2005, Dayton, OH, 67-73.

[Wang and Japkowicz 2008] WANG, B. X. AND JAPKOWICZ, N. 2008. Boosting Support Vector Machines for Imbalanced Data Sets. *Foundations of Intelligent Systems* 4994, 38-47.

[Wang and Zhao 2008] WANG, H. AND ZHAO, T. 2008. Identifying Named Entities in Biomedical Text Based on Stacked Generalization. In *Proceedings of the 7th World Congress on Intelligent Control and Automation*, Chongqing, China, Junho 2008, IEEE, 1-5.

[Witten and Frank 1999] WITTEN, I. AND FRANK, E. 1999. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, CA.

[Witten and Frank 2005] WITTEN, I. H. AND FRANK, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, CA.

[Wolpert 1992] WOLPERT, D. H. 1992. Stacked Generalization. *Neural Networks* 5, 2, 241-259.

[Yang and Pedersen 1997] YANG, Y. AND PEDERSEN, J. O. 1997. A Comparative Study on Feature Selection in Text Categorization. In Proceedings of the Fourteenth international Conference on Machine Learning. D. H. Fisher, Ed. Morgan Kaufmann Publishers, San Francisco, CA, 412-420.

[Yuan *et al*. 2005] YUAN, F., YANG, L., YU, G. 2005. Improving the k-NN and Applying it to Chinese Text Classification. In *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, 3, 1547-1553.

[Zadrozny 2006] ZADROZNY, B. 2006. Tópicos Avançados em Otimização e IA II - Aprendizado Automático – Aula 1. *Universidade Federal Fluminense*. Disponível em: http://www.ic.uff.br/bianca/topicos/. Acessado em: 06/11/2007.

[ZENKO *et al.* 2001] ZENKO, B., TODOROVSKI, L. AND DZEROSKI, S. 2001. A Comparison of Stacking with Meta Decision Trees to Bagging, Boosting, and Stacking with other Methods. In *Proceedings of the 2001 IEEE international Conference on Data Mining*, San Jose, E.U.A., Novembro - Dezembro 2001, N. CERCONE, T. Y. LIN, AND X. WU, Eds. ICDM. IEEE Computer Society, Washington, DC, 669-670.

[Zhang and Zhang 2008] ZHANG, C. AND ZHANG, J. 2008. A Local Boosting Algorithm for Solving Classification Problems. *Computational Statistics & Data Analysis* 52, 4, 1928-1941.

[Zhang *et al.* 2002] ZHANG, T., DAMERAU, F. AND JOHNSON, D. 2002. Text Chunking Based on a Generalization of Winnow. *The Journal of Machine Learning Research* 2, 615-637.

[Zhu 2007] ZHU, X. 2007. Lazy Bagging for Classifying Imbalanced Data. In *Proceedings of the 2007 Seventh IEEE international Conference on Data Mining*. ICDM. IEEE Computer Society, Washington, DC.