Instituto de Ciências Matemáticas e de Computação

Combinação de Classificadores Homogêneos e Heterogêneos

Roberta Akemi Sinoara
Jaqueline Brigladori Pugliesi
Solange Oliveira Rezende

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Janeiro/2003

Sumário

1 Introdução **1**

2 Métodos para Combinação de Preditores **3**
2.1 Métodos *Voting* versus Não-*Voting* 3
2.2 Métodos Lineares versus Não-Linear 4

3 Combinação de Classificadores Homogêneos **4**
3.1 *Bagging* 5
3.2 *Boosting* 6

4 Combinação de Classificadores Heterogênos **8**
4.1 Stacking 8

5 Descrição dos Experimentos **11**
5.1 Ferramentas e Recursos Utilizados 11
5.2 Procedimentos Utilizados 15

6 Bases de Dados Utilizadas e Resultados Obtidos **17**
6.1 Base de Dados Chess 17
6.2 Base de Dados Letter 18
6.3 Base de Dados Talassemia 20

7 Considerações Finais **22**

Referências **23**

1 Introdução

Uma das etapas do Processo de Extração de Conhecimento de Bases de Dados (DM – *Data Mining*) é a extração de padrões. Essa etapa normalmente é realizada com o uso de algoritmos de Aprendizado de Máquina (AM). Dado um conjunto de exemplos de treinamento, é possível aplicar diversos algoritmos de AM Supervisionado, obtendo para cada um desses algoritmos, uma descrição do conceito (ou classificador) que descreva esses exemplos de uma forma mais compacta. Um classificador pode ser visto como uma caixa preta que, para um dado problema, fornece uma decisão. Normalmente, diferentes classificadores possuem diferentes precisões.

A abordagem tradicional para se determinar qual é o melhor classificador para uma aplicação é selecionar aquele que fornece a maior precisão (ou menor erro) para essa aplicação. Porém, essa abordagem pode não ser a mais adequada. Informações valiosas descobertas pelos demais classificadores podem estar sendo ignoradas quando se opta por um único classificador. Assim, uma abordagem mais apropriada pode ser a combinação das saídas de diversos classificadores. O classificador resultante dessa combinação é conhecido por *ensemble* (LeBlanc & Tibshirani, 1996; Breiman, 2000).

A combinação de classificadores tem sido recomendada principalmente para problemas que não foram resolvidos satisfatoriamente por um único classificador e/ou no qual o desempenho apresentado pelo método de predição deve ser estável.

Pesquisas com a utilização de *ensembles* têm mostrado que é possível obter, através da combinação de classificadores, resultados com maior precisão comparados com o resultado obtido com um único classificador. Assim, vários algoritmos para realizar combinações vêm sendo pesquisados, porém, os resultados obtidos são essencialmente empíricos com pouca teoria para explicar o porque e como o erro de generalização é menor na maioria dos conjuntos de dados (Breiman, 2000).

Segundo Dietterich (Dietterich, 2000), no geral é possível construir bons *ensembles*, que possuam maior precisão do que um classificador simples. Isso se deve fundamentalmente a três razões.

A primeira delas é estatística. Um algoritmo de aprendizado faz uma busca no espaço H de hipóteses para identificar a melhor. O problema estatístico aparece quando o conjunto de treinamento é muito pequeno comparado ao espaço de hipóteses. Sem uma quantidade suficiente de dados, o algoritmo de aprendizado pode encontrar diferentes hipóteses em H com a mesma precisão no conjunto de treinamento. Construindo um *ensemble* com todos esses classificadores, o algoritmo pode fazer uma média dos resultados dos mesmos e reduzir o risco de escolher um classificador errado.

Na Figura 1(a), a situação descrita, envolvendo o problema estatístico, pode ser visualizada. A curva externa representa o espaço de hipóteses H enquanto o conjunto de hipóteses que oferecem uma boa precisão no conjunto de treinamento é representado pela curva interna. O ponto f é a hipótese verdadeira. Pode-se notar que fazendo a média das hipóteses que possuem maior precisão, obtém-se uma boa aproximação para f.

A segunda razão é computacional. Muitos algoritmos de aprendizado utilizam alguma forma de busca restrita que pode parar em um ótimo local. Em casos em que o conjunto de treinamento é suficiente (o problema estatístico é ausente), ainda pode ser muito difícil para o algoritmo de aprendizado encontrar a melhor hipótese. Um *ensemble* construído com a busca local partindo de diferentes pontos pode obter uma melhor aproximação da verdadeira função desconhecida que qualquer um dos classificadores individuais, como pode ser visto

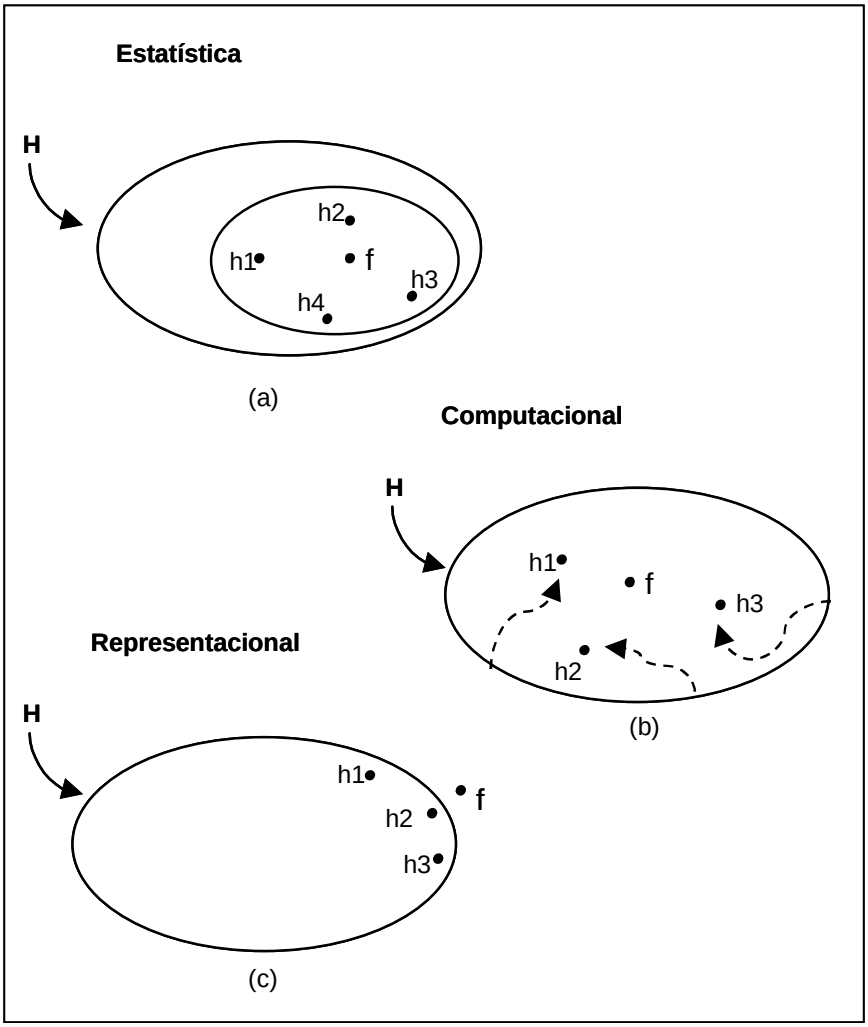

Figura 1: Três razões do *ensemble* poder trabalhar melhor que um classificador simples (Dietterich, 2000).

na Figura 1(b).

A terceira razão é representacional. Na maioria das aplicações de Aprendizado de Máquina, a hipótese verdadeira f pode não ser representada por nenhuma das hipóteses no espaço H. Desenvolvendo uma soma ponderada das hipóteses retiradas de H, pode ser possível ampliar o espaço das hipóteses representadas. A Figura 1(c) ilustra essa situação.

O tópico representacional é um pouco sutil, pois há muitos algoritmos de aprendizado para os quais H é, em princípio, o espaço de todos os possíveis classificadores. Dado um conjunto de treinamento suficiente, eles explorarão esse espaço. Por outro lado, com uma amostra de treinamento finita, esses algoritmos explorarão apenas um conjunto de hipóteses finito e finalizarão a busca quando encontrarem uma hipótese que satisfaça o conjunto de treinamento.

Por essa razão, na Figura 1, deve-se considerar o espaço H como o espaço das hipóteses efetivamente pesquisado pelo algoritmo de aprendizado para um dado conjunto de treinamento.

Esses três tópicos fundamentais são as principais falhas de um algoritmo de aprendi-

zado. O uso de *ensembles* tendem a reduzir (e talvez até eliminar) esses três tipos de falhas dos algoritmos de aprendizado de padrões.

O classificador combinado (ou *ensemble*) é caracterizado pela seleção de k classificadores e pela função de combinação. Como a combinação de classificadores depende desses dois fatores, vários caminhos podem ser utilizados visando um bom resultado. Os classificadores a serem combinados podem ser extraídos a partir de várias amostras de uma Base de Dados, utilizando um único algoritmo de AM (classificadores homogêneos) ou diferentes algoritmos (classificadores heterogêneos).

Neste relatório técnico é apresentada uma implementação de um algoritmo de combinação de classificadores homogêneos, denominado *bagging*. Essa implementação foi realizada utilizando a liguagem Perl e foi desenvolvida uma interface em Java.

Esse programa foi utilizado na realização de experimentos com combinação de classificadores. Também foram realizados experimentos com os algoritmos *boosting* e *stacking*. O primeiro é utilizado na combinação de classificadores homogêneos, enquanto o segundo na de classificadores heterogêneos. O objetivo da realização desses experimentos é a comparação das precisões obtidas com o uso desses três algoritmos de combinação com a obtida com apenas um classificador. Para atingir esse objetivo, utilizou-se três Bases de Dados e as ferramentas See5 (RuleQuestResearch, 2001) e Weka (Witten & Frank, 2000). Este relatório técnico apresenta a descrição dos procedimentos utilizados e os resultados obtidos.

Na Seção 2 são apresentados alguns métodos que podem ser utilizados para realizar a combinação de classificadores. A Seção 3 apresenta o método para combinação de classificadores homogêneos, bem como dois algoritmos que o utilizam. A combinação de classificadores heterogêneos é abordada na Seção 4. Na Seção 5 é feita uma descrição dos experimentos realizados, incluindo as ferramentas e os procedimentos utilizados. As Bases de Dados utilizadas, assim como os resultados obtidos com cada uma, são apresentados na Seção 6. Por fim, na Seção 7 encontram-se as considerações finais referentes aos experimentos descritos neste relatório técnico.

2 Métodos para Combinação de Preditores

No processo de combinação de preditores a função de combinação é baseada em diferentes métodos: os *voting* e não-*voting* (para problemas de classificação), e os lineares e não-lineares (para problemas de regressão). Neste relatório técnico foi abordada apenas a tarefa de classificação.

2.1 Métodos *Voting* versus Não-*Voting*

O *voting* é um dos métodos mais comum para combinar classificadores (Bauer & Kohavi, 1999). Os algoritmos *voting* têm como entrada um algoritmo de Aprendizado de Máquina e um conjunto de treinamento. Eles geram classificadores com diferentes versões desse conjunto, que são combinados para criar um classificador final e este será utilizado para predizer o conjunto de teste.

Na decisão, o voto de cada hipótese deve ser ponderado por uma probabilidade posterior àquela hipótese considerando os dados de treinamento. Algumas variantes dos métodos de *voting* podem ser encontradas na literatura de Aprendizado de Máquina. Uma delas é o *voting* uniforme no qual a opinião de todos os classificadores bases contribuem com a mesma força para a predição final. Uma outra variação comum é o *voting* com peso, no qual cada

classificador base possui um peso associado, que pode alterar com o tempo, e dessa forma fortalecer o atributo de predição para um classificador "ótimo".

Uma outra forma de combinar classificadores é através dos métodos não-*voting*. Uma melhora no *voting* uniforme é obtida quando cada classificador pode produzir estimativas de classe-probabilidade ao invés de uma simples decisão de predição. Uma estimativa da probabilidade para cada uma das classes de um dado exemplo é a probabilidade que a classe verdadeira seja k. As probabilidades da classe de todos os modelos podem ser combinadas, e então a probabilidade do *ensemble* pode ser considerada como: $\frac{1}{L}\sum_{i=1}^{L} p_i$ (Gama, 1999).

2.2 Métodos Lineares versus Não-Linear

Quando a saída de um preditor é um valor real, uma função de combinação linear é comumente utilizada (Breiman, 1996b). A função linear pode ser simples, média sem peso ou uma combinação de preditores com peso. Algoritmos simples de regressão linear são geralmente utilizados para calcular pesos no componente de predição dos preditores.

As vantagens dos métodos de combinação linear são que eles fazem análise formal mais facilmente; geralmente trabalham bem; e podem diminuir a variância da predição composta.

Por outro lado, uma combinação linear simples pode não refletir a especialidade local de um preditor componente para um exemplo em uma região particular do espaço de atributo. Diante disso, a contribuição de cada componente é a mesma, sem levar em consideração a entrada. Funções lineares falham no suporte de especialidade local, que é um aspecto importante em combinação de preditores.

Apesar da existência de trabalhos em combinar componentes de preditores não-lineares, algoritmos de combinação não-lineares têm recebido bem menos atenção. Funções de combinação não-lineares são normalmente implementadas como redes neurais, tais como perceptrons multicamada ou redes de funções *radial basis*. Não existe *a priori* razão alguma para crer que métodos lineares são adequados para modelar a combinação de regressores arbitrários, existindo até evidências do contrário. Devido a superioridade seletiva dos preditores em diferentes regiões, é esperado que os componentes terão contribuições relativas distintas em diferentes partes do espaço. Uma função linear dos resultados dos preditores sozinha pode não traçar as contribuições seletivas dos componentes (Skalak, 1997).

3 Combinação de Classificadores Homogêneos

Nessa Seção é apresentada a combinação de classificadores homogêneos, ou seja, a combinação de classificadores gerados por um único algoritmo de AM, ilustrada na Figura 2. Como diversidade é um fator importante para a uso de *ensembles*, a maioria das estratégias utilizadas para combinar esse tipo de classificadores manipulam o conjunto de treinamento a fim de obter hipóteses diferentes.

A seguir são descritos dois algoritmos que realizam a combinação de classificadores homogêneos, os algoritmo *bagging* e *boosting*. Uma das características que diferem esses dois algoritmos é que o algoritmo *boosting* altera a distribuição do conjunto de treinamento baseado na desempenho dos classificadores anteriores, enquanto o mesmo não acontece com o algoritmo *bagging*.

A seguir são descritos os dois algoritmos, *bagging* e *boosting*, que implementam o método *voting* para combinação de classificadores. As taxas de erro em três diferentes Bases de Dados são apresentadas e comparados na Seção 6.

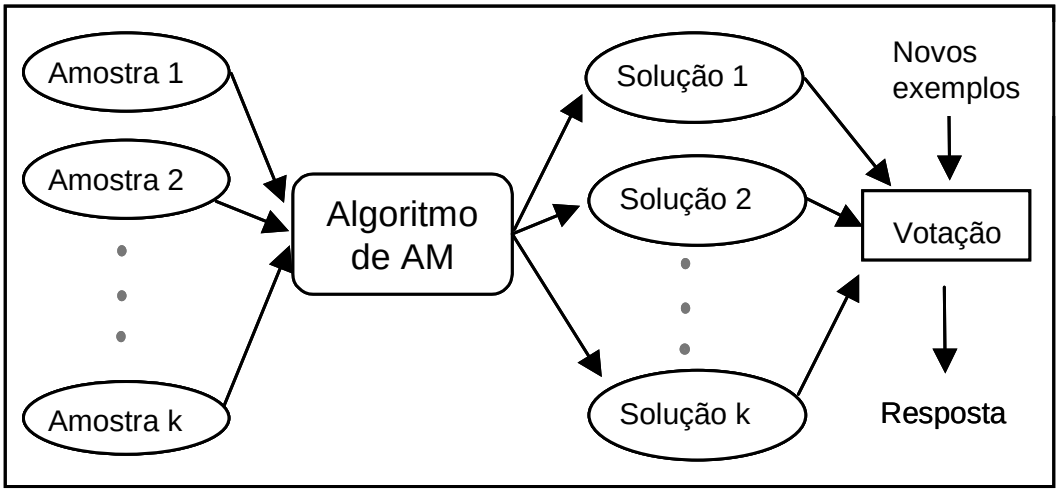

Figura 2: Combinação de Classificadores Homogêneos.

3.1 *Bagging*

O algoritmo descrito por Breiman em 1996 (Breiman, 1996a), que recebeu o nome de ***b**ootstrap* ***agg**regat**ing*** (ou simplesmente *bagging*), realiza a combinação de classificadores gerados pelo mesmo algoritmo de AM. De acordo com esse algoritmo, descrito no Algoritmo 1, dado um conjunto de treinamento T, com m exemplos, são geradas várias amostras desse conjunto, chamadas *bootstrap*. Cada *bootstrap* é formado tomando aleatoriamente m elementos de T, com substituição. Assim, eles possuem o mesmo número de exemplos de T, sendo que alguns podem aparecer mais de uma vez e outros podem não aparecer. A probabilidade de um exemplo ser selecionado, pelo menos uma vez, dentre os m exemplos de T é dada por $1-(1-1/m)^m$). Para um m grande, essa probabilidade é cerca de $1-1/e$. Portanto, cada *bootstrap* possui, em média, 63,2% dos elementos do conjunto original.

Algoritmo 1 Algoritmo *Bagging*.

Entradas:

conjunto de treinamento T
algoritmo de aprendizado L
constante inteira N {número de *bootstrap*}

1: **para** $n := 1$ a N **faça**
2: $T' :=$ *bootstrap* de T
3: $C_n := L(T')$
4: **fim-para**
5: $C^*(x) := \arg\max_{y \in Y} \sum_{n=1}^{N} C_n(x) = y$

Saída: Classificador C^*

Os *bootstraps* formados são usados como novos conjuntos de treinamento, dando origem a vários classificadores, como mostra a Figura 3. A fim de gerar o classificador final, que é utilizado para classificar o conjunto de teste, estes classificadores são combinados através

de uma votação não ponderada.

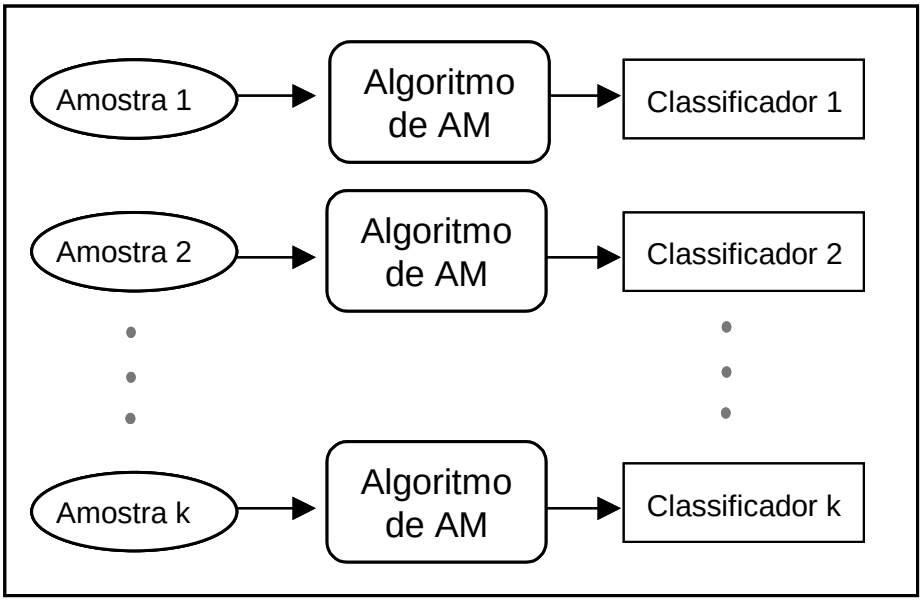

Figura 3: Geração Paralela de Classificadores.

Geralmente, o algoritmo *bagging* aumenta o desempenho de algoritmos de Aprendizado de Máquina instáveis, ou seja, algoritmos cuja saída seja um classificador que obtenha grandes mudanças em resposta a pequenas alterações ocorridas no conjunto de treinamento. São exemplos de algoritmos instáveis as árvores de decisão e as redes neurais. Vale ressaltar que, com o uso de *bagging*, ao mesmo tempo em que se ganha precisão obtém-se, comparado às árvores de decisão, a perda de uma estrutura simples e interpretável.

O motivo pelo qual *bagging* reduz a taxa de erro de algoritmos instáveis tem sido tema de alguns trabalhos. Domingos (Domingos, 1997) realizou testes empíricos com o objetivo de conhecer o motivo pelo qual a taxa de erro de árvores de decisão e outros algoritmos é reduzida com o uso deste algoritmo. Para isso, duas alternativas baseadas na teoria de Aprendizado Bayesiano foram testadas. A primeira hipótese é que *bagging* funciona por ser uma aproximação do ótimo procedimento do modelo Bayseano médio, com uma prévia aproximação implícita. A segunda é que *bagging* funciona, pois troca efetivamente o modelo espacial do algoritmo de Aprendizado de Máquina e/ou a prévia distribuição para uma que melhor se encaixe no domínio. Os experimentos realizados por esse pesquisador contradizem a primeira, confirmando a segunda hipótese.

Assim, *bagging* pode ser uma maneira simples de melhorar um método existente em problemas de Data Mining (Gama, 1999). É preciso apenas adicionar um *loop* antes de selecionar o *bootstrap* e mandá-lo para o algoritmo de aprendizado e, por fim, fazer a combinação dos classificadores gerados.

3.2 *Boosting*

Boosting (Freund & Shapire, 1996) engloba uma família de técnicas utilizadas para aumentar o desempenho de qualquer algoritmo de aprendizado. Na teoria, *boosting* pode ser usado para reduzir o erro de qualquer algoritmo "fraco", cujos classificadores devem ser apenas um pouco melhor que um classificador que prediz de forma aleatória.

Boosting combina classificadores gerados pelo mesmo algoritmo de aprendizado utilizando diferentes distribuições do conjunto de treinamento. A distribuição do conjunto de treinamento é alterada de acordo com os erros cometidos pelo classificador anterior. Portanto, *boosting* é uma tentativa de produzir novos classificadores, que são melhores principalmente por classificar exemplos corretamente para os quais a eficiência dos classificadores comuns é ruim.

Um dos algoritmos *boosting* mais utilizado é o *AdaBoost* (*Adaptative Boosting*), que foi apresentado por Freund e Schapire (Freund & Schapire, 1997) e é descrito no Algoritmo 2. Dado um conjunto de treinamento T, o algoritmo *AdaBoost* atribui um peso $D_1(m)$ a cada um de seus m exemplos, gerando um novo conjunto de treinamento T'_1. Em cada iteração n, os pesos $D_n(m)$ são alterados e um novo conjunto T'_n é gerado, com base em T'_{n-1}. A partir de cada um desses conjuntos um classificador é gerado. Note que *AdaBoost* gera os classificadores em série, como é ilustrado na Figura 4, de forma diferente de bagging, que os gera paralelamente.

Algoritmo 2 Algoritmo *AdaBoost*.

Entradas:

conjunto de treinamento T
"fraco" algoritmo L
constante inteira N {número de iterações}

1: **para** $i := 1$ a m **faça**
2: $D_1(i) = 1/m$ {atribui pesos aos exemplos de T}
3: **fim-para**
4: **para** $n := 1$ a N **faça**
5: $C_n := L(T)$ {utilizando a distribuição D_n}
6: $E_n := \sum_{i:h_n(x_i)\neq y_i} D_n(i)$
7: $\alpha_n := \frac{1}{2} \ln \frac{1-E_n}{E_n}$
8: **para** $i := 1$ a m **faça**
9: $D_{n+1}(i) := \frac{D_n(i)\exp(-\alpha_n y_i C_n(x_i))}{Z_n}$ {Z_n = fator normalizador}
10: **fim-para**
11: **fim-para**
12: $C^* := sign\Big(\sum_{n=1}^{N} \alpha_n h_n(x)\Big)$

Saída: Classificador C^*

AdaBoost pode usar os pesos para construir um novo conjunto de treinamento T'_n de duas maneiras que têm se mostrado muito eficazes (Quinlan, 1996). Na primeira, denominada *boosting* por amostragem, os exemplos são tomados de T'_{n-1} com substituição e probabilidade proporcional aos seus pesos. O segundo método, *boosting* por peso, pode ser usado com um algoritmo de AM que aceite diretamente um conjunto de treinamento ponderado.

Em qualquer um dos casos, em cada iteração os pesos são ajustados de acordo com o desempenho do classificador anterior. Os exemplos que foram classificados de forma errada têm o peso aumentado e os que foram classificados corretamente têm o peso diminuído. Com isso, os classificadores focalizam os exemplos mais difíceis de serem classificados, ou seja, aqueles que foram classificados de forma errônea pelo classificador anterior. Desta

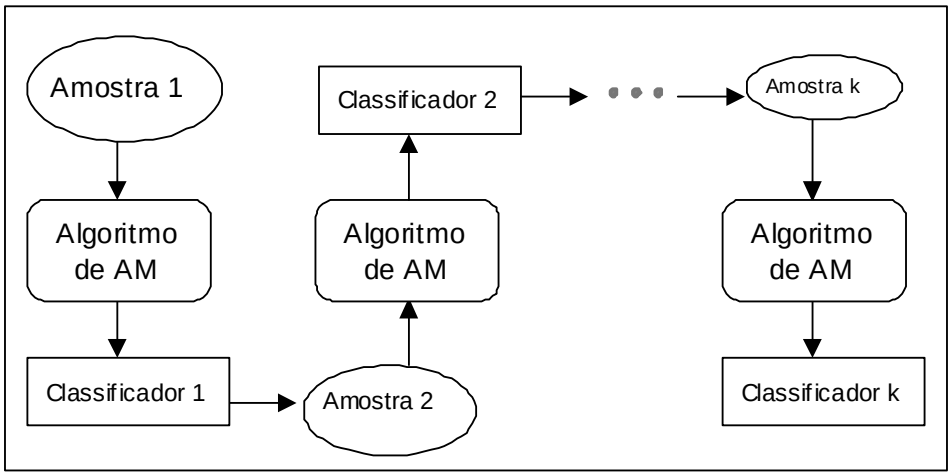

Figura 4: Geração de Classificadores em Série.

forma, o algoritmo *AdaBoost* torna os classificadores cada vez mais "fortes".

Após todas as iterações serem realizadas, a resposta do classificador final é formada através de uma votação ponderada. Nessa votação, o peso do voto de cada classificador depende do seu desempenho no conjunto de treinamento usado para construí-lo.

Há duas razões para o aumento no desempenho verificado por *boosting* (Gama, 1999). A primeira é que ele gera, pela combinação de muitos classificadores cuja taxa de erro pode ser grande, um classificador com uma taxa de erro menor. A segunda é a redução da variância.

O uso do algoritmo *boosting* normalmente é útil em problemas que possuem as seguintes propriedades: o algoritmo de aprendizado é sensível a mudanças no conjunto de treinamento e os exemplos observados possuem vários níveis de dificuldade, propriedade que pode ser facilmente encontrada nos problemas reais.

4 Combinação de Classificadores Heterogênos

A combinação de classificadores heterogêneos refere-se à combinação de classificadores gerados por diferentes algoritmos de aprendizado, como é ilustrado na Figura 5. Já que a diversidade dos classificadores é garantida pelo uso de diferentes algoritmos de AM, não é necessário manipular o conjunto de treinamento, podendo ser utilizado o mesmo conjunto para todos os algoritmos de aprendizado.

Um algoritmo normalmente utilizado para realizar esse tipo de combinação é o algoritmo *stacking*. Vários experimentos estão sendo realizados com o uso desse algoritmo (Fan et al., 1996; Ting & Witten, 1997). A seguir é apresentada uma descrição desse algoritmo, que foi utilizado nos experimentos apresentados neste relatório técnico.

4.1 Stacking

No *stacking*, também chamado de *stacked generalization*, diferentes algoritmos de AM geram classificadores com o conjunto de treinamento original. Logo após, as descrições dos exemplos de treinamento são estendidas para incluir os resultados da predição desses classificadores,

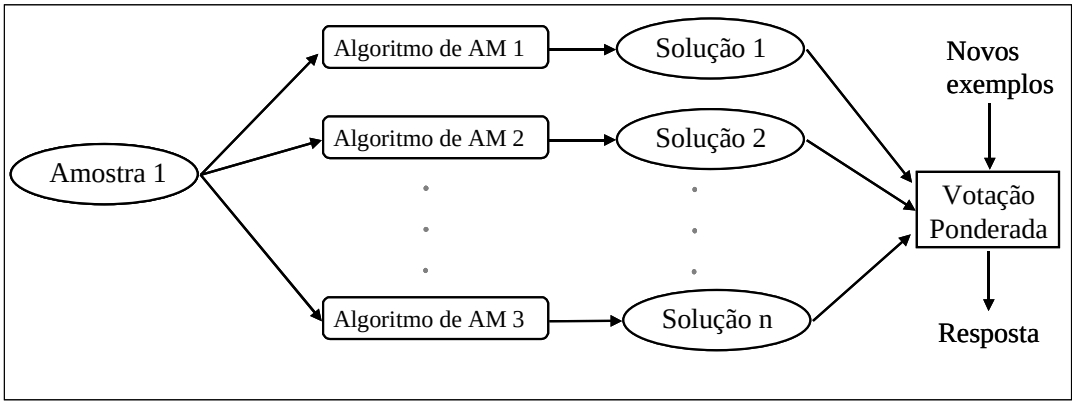

Figura 5: Combinação de Classificadores Heterogêneos.

formando um novo conjunto de treinamento. Esse novo conjunto é então analisado por outros algoritmos de AM a fim de produzir futuros classificadores, e assim por diante.

Wolpert (Wolpert, 1992) propôs o framework *stacked generalization*, que é uma arquitetura em camadas. Classificadores no nível-0 recebem como entrada o conjunto de dados original e cada um fornece uma predição. Camadas sucessivas recebem como entrada as predições das camadas imediatamente anteriores e fornecem a saída para a próxima camada. Um único classificador no ponto mais alto fornece a predição final.

Stacking é uma tentativa de minimizar o erro de generalização através do uso de classificadores em camadas mais altas para aprender o tipo de erros cometidos pelos classificadores imediatamente abaixo. Por essa perspectiva, ele pode ser visto como uma extensão para modelo de métodos de seleção, *cross-validation*. Esse método utiliza a estratégia "ganhador fica com tudo", com isso, apenas um único classificador com o menor erro é selecionado. A idéia do *stacking* é que pode existir uma maneira mais inteligente de usar um conjunto de classificadores. O papel dos classificadores posteriores é aprender como os anteriores erram, em quais classes eles concordam ou discordam e usar esse conhecimento ao fazer novas predições (Ting & Witten, 1997).

Trabalhos em arquitetura *stacking*, por exemplo (Gama, 1999; Merz, 1998), concentram-se em arquiteturas de duas camadas, como é ilustrada na Figura 6.

Segundo Gama (Gama, 1999), experimentos com esse algoritmo podem ser realizados em duas fases: a fase de treinamento e a fase de aplicação.

A partir de um conjunto de treinamento T, com n exemplos, a fase de treinamento começa com a geração de k pares de conjuntos, $T' = (x_1, y_1), (x_2, y_2), ..., (x_k, y_k)$. Tais pares podem ser gerados de duas maneiras diferentes. A primeira maneira, proposta por (Fan et al., 1996), são gerados n pares, ou seja, utiliza-se $k = n$. O primeiro conjunto de cada par (x_i) é formado por um único elemento de T e o segundo (y_i) é formado pelos $n - 1$ elementos restantes. A segunda maneira, utilizada em (Ting & Witten, 1997), gera um número definido de pares k, sendo $k < n$. Divide-se o conjunto T em k partes, sendo cada parte o primeiro conjunto de cada par (x_i). O segundo conjunto (y_i) será formado pelos exemplos de T retirando-se os elementos contidos em x_i.

Os algoritmos de nível-0 são utilizados para gerar classificadores tendo como conjuntos de treinamento cada conjunto y_i. Os classificadores gerados fornecem uma resposta para o(s) exemplo(s) contido(s) no respectivo conjunto x_i.

Para todos os elementos contidos em cada conjunto x_i forma-se uma instância para o

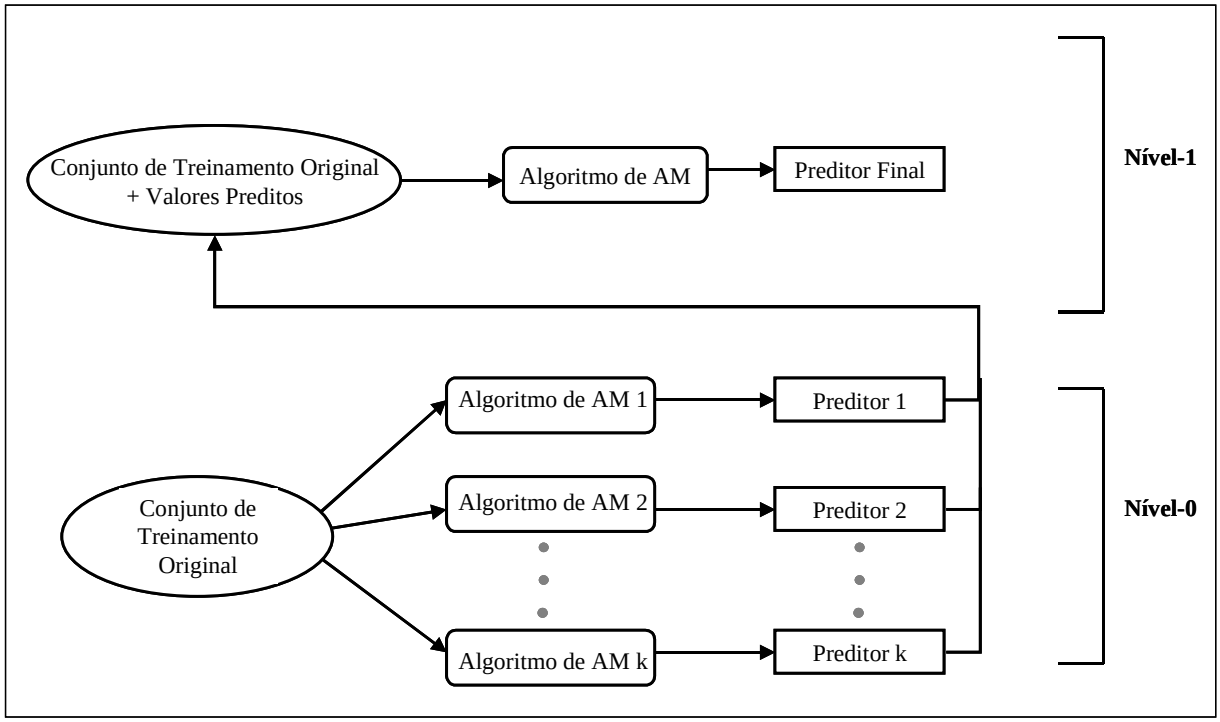

Figura 6: Arquitetura *Stacking*.

conjunto de treinamento do algoritmo de nível-1. As instâncias devem conter, além da classe correta, os valores preditos pelos classificadores de nível-0 para aquele exemplo. Assim, o número de exemplos do conjunto de treinamento do algoritmo de nível-1 será o mesmo do conjunto de treinamento original. O conjunto é então utilizado para formar um classificador de nível-1, com a utilização de um algoritmo de nível-1. A resposta desse classificador será a resposta do classificador combinado.

Para explorar completamente o conjunto de treinamento original, todos os classificadores de nível-0 são retreinados usando todo o conjunto de treinamento. Os modelos gerais são utilizados para predizer os exemplos do conjunto de teste.

Na fase de aplicação, como é ilustrado na Figura 7, quando um novo exemplo é apresentado, ele é calculado por todos os classificadores de nível-0. Um vetor de predições é formado e utilizado no nível-1, que fornece a predição final do exemplo.

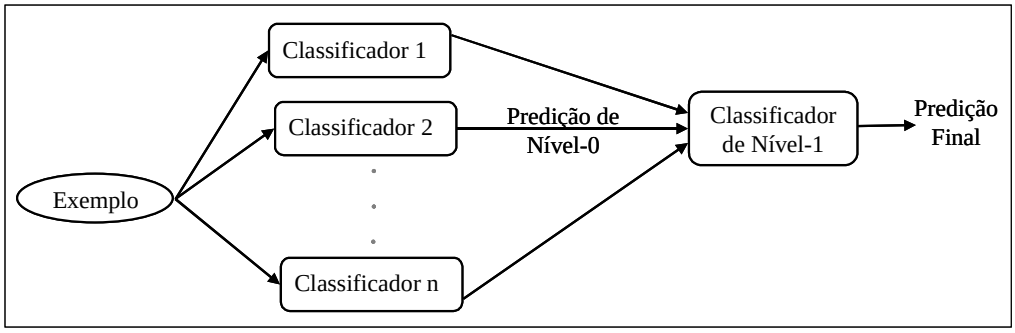

Figura 7: *Stacking* – Fase de Aplicação.

5 Descrição dos Experimentos

Nessa seção é apresentada uma descrição detalhada das ferramentas, recursos e procedimentos utilizados na execução dos experimentos apresentados neste relatório técnico, bem como as implementações realizadas.

5.1 Ferramentas e Recursos Utilizados

Para a realização dos experimentos apresentados neste relatório técnico, foram utilizadas ferramentas para a geração e combinação de classificadores. As combinações foram realizadas com o uso de dois algoritmos para combinação de classificadores homogêneos, *bagging* e *boosting*, e um para a combinação de classificadores heterogêneos, *stacking*.

Os classificadores combinados pelos algoritmos *bagging* e *boosting* foram gerados pela ferramenta See5, (RuleQuestResearch, 2001). Essa ferramenta, bastante utilizada em Data Mining e compatível com várias versões do sistema operacional Windows (95/98/NT/2000), gera árvores de decisão e conjuntos de regras, ambos para problemas de classificação. A ferramenta See5 é um aperfeiçoamento do algoritmo *C4.5*, incorporando diversas vantagens e inovações, tais como, maior precisão nos conjuntos de regras, menor tempo de execução, e implementação do algoritmo *boosting*, além de uma interface gráfica, tornando-a amigável para o usuário.

A realização dos experimentos com o algoritmo *bagging* contou com o auxílio de dois programas, sendo um deles, o programa de integração da ferramenta See5, que é disponibilizado pelo fabricante dessa ferramenta, *RuleQuest Research Data Mining Tools*. Esse programa de integração permite que os classificadores gerados pelo See5 possam ser utilizados por outros sistemas. Dado um classificador e um conjunto de exemplos, a integração é feita através de um arquivo contendo a classe real e a predita pelo classificador para cada exemplo dado.

O outro programa utilizado nos experimentos com o algoritmo *bagging* foi desenvolvido em Perl, com interface em Java, especialmente para a realização desses experimentos. Esse programa implementa o algoritmo *bagging* e é dividido em dois módulos: geração de amostras e combinação de classificadores. Seguindo o Algoritmo 3, a geração de amostras é apresentada nas linhas de 1 a 11 e a combinação de classificadores, nas linhas de 14 a 29.

Na janela principal do programa, ilustrada na Figura 8, o usuário fornece o diretório onde o Perl se encontra instalado em sua máquina e pode acessar os dois módulos do programa.

Figura 8: Implementação do Algoritmo *Bagging* – Janela Principal.

Como foi apresentado na Subseção 3.1, o primeiro passo do algoritmo *bagging* é a geração das amostras (ou *bootstraps*), que são utilizadas pelo algoritmo de Aprendizado

Algoritmo 3 Algoritmo da Implementação do *Bagging*.

Entradas:

nome do arquivo de entrada
constante inteira *numAmostras* {número de amostras}
nomes dos arquivos de saída

1: Verifica o tamanho do conjunto de treinamento e atribui a variável *numExemplos*
{Início da Geração de Amostras}
2: **para** $i := 1$ a *numAmostras* **faça**
3: Cria os novos arquivos .data, .names e .test
4: **fim-para**
5: **para** $i := 1$ a *numAmostras* **faça**
6: Copia o conteúdo do arquivo .names para o arquivo correspondente da amostra
7: Copia o conteúdo do arquivo .test para o arquivo correspondente da amostra
8: **para** $j := 1$ a *numExemplos* **faça**
9: Copia um exemplo (aleatório) para o arquivo .data da amostra
10: **fim-para**
11: **fim-para**
{Fim da Geração de Amostras}
12: Gera um classificador com cada amostra
13: Gera o arquivo contendo a classe real e a predita para cada classificador com o conjunto de teste
{Início da Combinação de Classificadores}
14: Lê os arquivos gerados
15: **para** $i := 1$ a *numExemplos* **faça**
16: **para** $i := 1$ a *numAmostras* **faça**
17: **se** classe real = predita **então**
18: $certo := certo + 1$
19: **senão**
20: $errado := errado + 1$
21: **fim-se**
22: **fim-para**
23: **se** $certo > errado$ **então**
24: $totalCertos := totalCertos + 1$
25: **senão**
26: $totalErrados := totalErrados + 1$
27: **fim-se**
28: **fim-para**
29: $taxaErro := (\frac{totalErrados}{numExemplos}) * 100$
{Fim da Combinação de Classificadores}

Saída: Taxa de Erro do Classificador Combinado – *taxaErro*

de Máquina a fim de gerar os classificadores. Assim, a geração de amostras é realizada no primeiro módulo do programa, cuja janela, que pode ser vista na Figura 9, é acessada através do botão *"Gerar Amostras"* da janela principal.

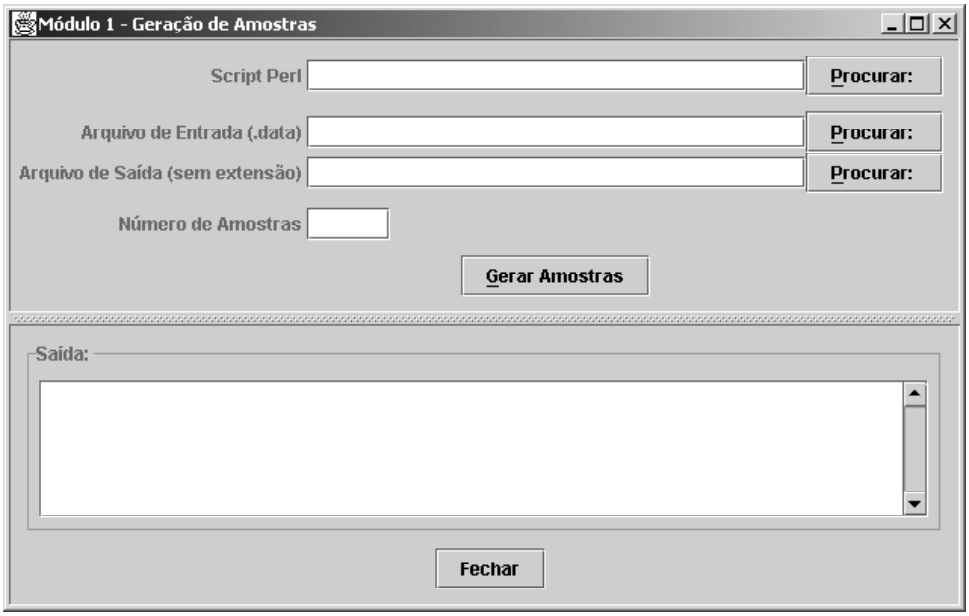

Figura 9: Janela do Módulo de Geração de Amostras.

Através da janela do módulo de geração de amostras, o usuário fornece os parâmetros necessários para que o script, implementado em Perl, possa ser executado. Para isso, o usuário deve fornecer o diretório onde se encontra o script que gera as amostras, o nome dos arquivos de entrada e de saída, e o número de amostras a serem geradas.

O arquivo de entrada é o arquivo que contém o conjunto de treinamento, cujo formato e extensão (.data) são iguais ao arquivo utilizado pela ferramenta See5. O arquivo contendo o conjunto de teste (.test) e o arquivo contendo os nomes e tipos dos atributos da base (.names), também utilizados pela ferramenta See5, devem estar no mesmo diretório do arquivo de entrada, pois estes também são gerados para cada amostra gerada nesse módulo.

O nome do arquivo de saída deve ser fornecido sem extensão, pois o script Perl gera vários arquivos, três para cada amostra (conjunto de treinemento, de teste e atributos), nomeando-os com o nome fornecido pelo usuário seguido pelo número da amostra e a extensão. As extensões dos arquivos do conjunto de treinamento e de atributos são iguais às dos arquivos de entrada, .data e .names, respectivamente. Já a extensão do conjunto de teste, que era .test, passa a ser .cases, pois essa é a extensão utilizada pelo programa de integração da ferramenta See5.

Fornecidos todos os parâmetros necessários, o script Perl é acionado pelo usuário através do botão *"Gerar Amostras"*, encontrado na janela do módulo de geração de amostras, sendo que a saída fornecida pelo script também é mostrada ao usuário nesta mesma janela.

Em posse das amostras geradas, o usuário deve utilizar a ferramenta See5, para gerar os classificadores, e o programa de integração, para gerar, para cada classificador, um arquivo contendo a classe real e a predita para cada exemplo do conjunto de teste. Esses arquivos são utilizados como entrada no segundo módulo do programa, o módulo de combinação de classificadores, cuja janela pode ser vista na Figura 10.

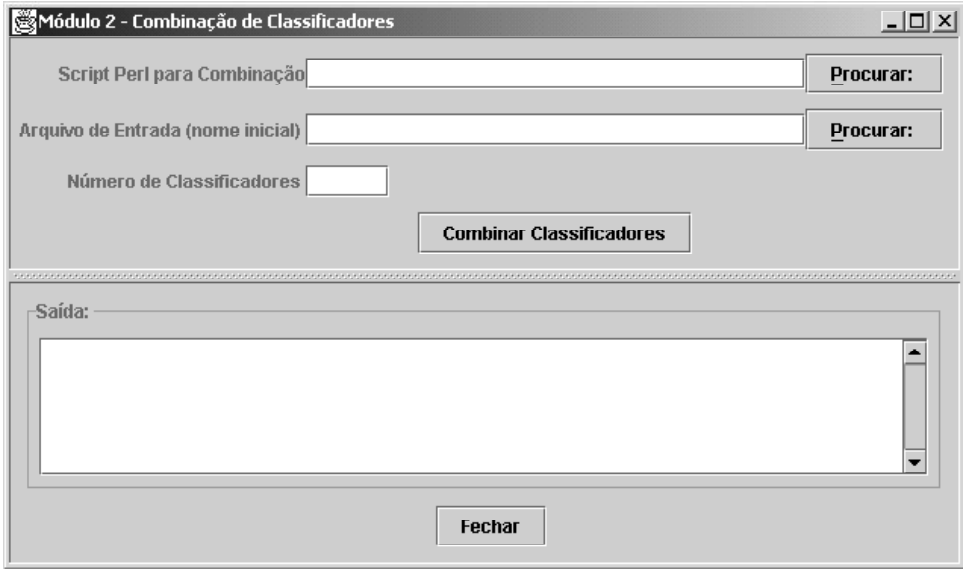

Figura 10: Janela do Módulo de Combinação de Classificadores.

No módulo de combinação de classificadores, o usuário fornece os seguintes parâmetros: diretório onde se encontra o script Perl que executa a combinação, o nome "inicial" dos arquivos de entrada e o número de classificadores que serão combinados.

O nome "inicial" dos arquivos de entrada deve ser o nome dos arquivos que o programa de integração da ferramenta See5 gerou sem o número de identificação da amostra, ou seja, esse nome deve ser comum a todos os arquivos que foram gerados e que serão combinados.

Ao acionar o botão *"Combinar Classificadores"* o usuário dá início ao script Perl que realiza a combinação. Todas as mensagens que o script possa enviar ao usuário são mostradas nesta mesma janela. Nesta janela também é informado o número de exemplos do conjunto de teste, o número de exemplos que a combinação de classificadores classificou corretamente e o número que esta classificou de forma errônea, bem como a taxa de erro do classificador combinado.

Quanto ao algoritmo *boosting* optou-se pela utilização do algoritmo presente na ferramenta See5, já que esta ferramenta contém o *AdaBoost* implementado e fornece grande facilidade para a utilização do mesmo. O algoritmo *AdaBoost* é baseado nas pesquisas de Rob Schapire e Yoav Freund (Freund & Schapire, 1997), descrito na Subseção 3.2. A janela principal do See5 e a caixa de diálogo para geração de classificadores são apresentadas na Figura 11.

Já os experimentos com o algoritmo *stacking* foram realizados com o auxílio da ferramenta Weka (*Waikato Environment for Knowledge Analysis*) (Witten & Frank, 2000). Essa ferramenta foi desenvolvida na Universidade de Waikato, na Nova Zelândia, implementada em Java e testada nos sistemas operacionais Linux, Windows e Macintosh. Ela pode ser vista como uma coleção de algoritmos de Aprendizado de Máquina que podem ser utilizados para resolver problemas de Data Mining do mundo real. Além de algoritmos para classificação, regressão, associação e clustering, a ferramenta Weka também fornece certo suporte às fases de pré-processamento e de visualização dos dados. A janela de classificação dessa ferramenta pode ser vista na Figura 12.

Os experimentos apresentados neste relatório técnico foram realizados com três Bases

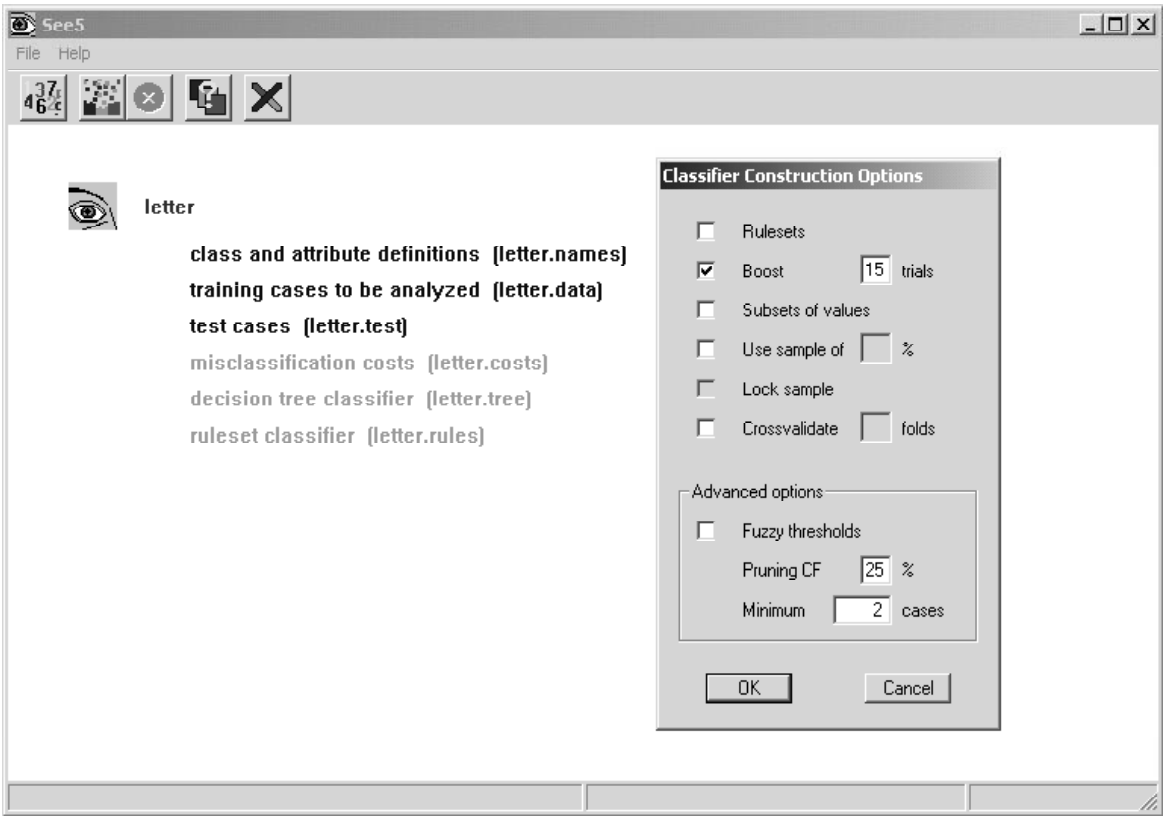

Figura 11: Ferramenta See5 – Caixa de Diálogo para Geração de Classificadores.

de Dados, sendo uma delas uma Base de Dados real, a Talassemia, e as outras duas, Chess e Letter, pertencentes ao Repositório da UCI (Blake & Merz, 1998). Para as três foram feitas análises com os algoritmos *bagging*, *boosting* e *stacking*.

Na Seção 6 são apresentadas as descrições das Base de Dados seguidas pelos resultados obtidos com as mesmas.

5.2 Procedimentos Utilizados

Os experimentos com o algoritmo *bagging* foram realizados em três fases. Na primeira, realizada no módulo de geração de amostras do programa descrito na subseção anterior, foram gerados os novos conjuntos de treinamento (ou *bootstraps*), através da retirada aleatória dos exemplos do conjunto de treinamento original, com substituição (linhas de 1 a 11 do Algoritmo 3).

Na segunda fase, cada novo conjunto de treinamento foi apresentado ao See5, que gerou os classificadores (linha 12 do Algoritmo 3). Com os classificadores gerados, o programa de integração da ferramenta See5, também descrito na subseção anterior, gerou os arquivos contento a classe real e a predita por cada classificador para os exemplos do conjunto de teste (linha 13 do Algoritmo 3).

Esses arquivos serviram como entrada para o módulo de combinação de classificadores do programa que implementa o algoritmo *bagging*, onde ocorreu a última fase do experimento com esse algoritmo, ou seja, a votação. Nessa votação, para cada exemplo do conjunto de teste, foi verificada a classe predita por cada classificador, ganhando aquela que mais

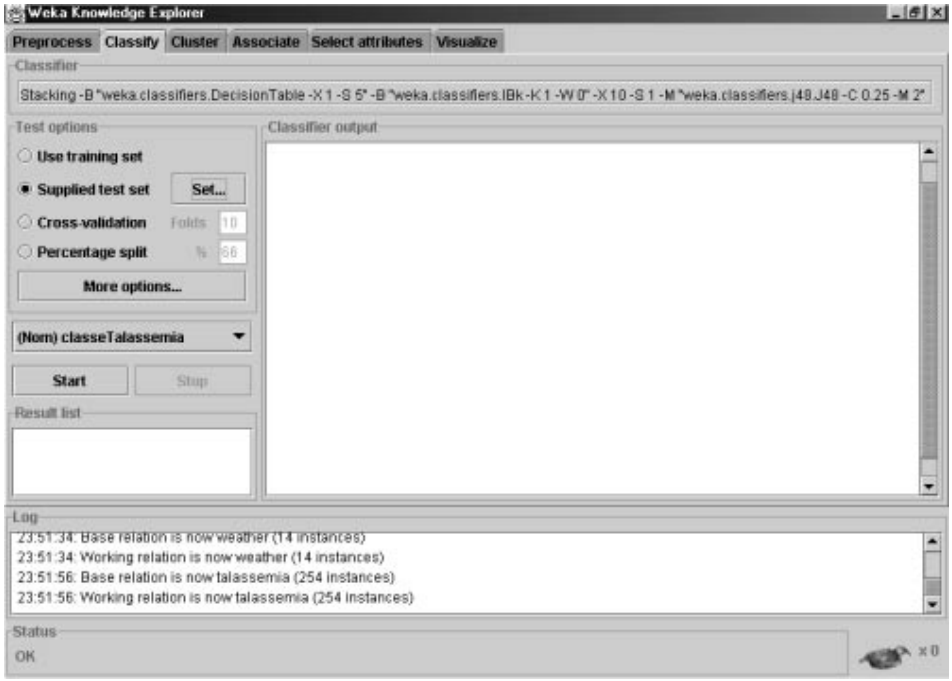

Figura 12: Ferramenta Weka – Janela de Classificação.

apareceu, ou seja, a classe que foi predita por mais classificadores. A classe vencedora é a classe predita pelo classificador final e, a partir das respostas deste, calculou-se a taxa de erro do classificador gerado pelo algoritmo *bagging* (linhas de 14 a 29 do Algoritmo 3).

Como mencionado na subseção anterior, os experimentos com o algoritmo *boosting* foram realizados na ferramenta See5. Quando selecionada a opção *boosting* nessa ferramenta, o usuário deve escolher a quantidade de classificadores a serem gerados e combinados.

Selecionada a opção *boosting* e fornecida a quantidade de classificadores, a ferramenta See5 forneceu a taxa de erro gerada pela combinação dos classificadores através da votação, além da taxa de erro de cada classificador intermediário, tanto no conjunto de treinamento como no de teste.

Através da ferramenta Weka, os experimentos com o algoritmo *stacking* foram realizados tendo *Decision Table* e *Instance-based Learner* como algoritmos de nível-0 e *C4.5* como o algoritmo de nível-1. Foram utilizados esses algoritmos por serem os mais comuns entre os algoritmos para classificação existentes na ferramenta Weka. Selecionados esses algoritmos e a quantidade de pares a ser utilizada pelo algoritmo *stacking*, a ferramenta Weka fornece as taxas de erro obtidas nos conjuntos de treinamento e de teste.

Tanto nos experimentos com o algoritmo *bagging*, quanto nos experimentos com o algoritmo *boosting* (*AdaBoost*), foram combinados classificadores gerados por 3, 5, 7, 10, 15 e 20 amostras do conjunto de treinamento original. Nos experimentos com o algoritmo *stacking* foram utilizadas essas mesmas quantidades de pares de conjuntos, ou seja, essas foram as quantidades de classificadores gerados para cada algoritmo de nível-0. Na próxima seção são apresentadas as três Bases de Dados utilizadas nos experimentos, bem como os resultados obtidos.

6 Bases de Dados Utilizadas e Resultados Obtidos

Nessa seção é feita uma descrição de cada Base de Dados utilizada nos experimentos apresentados neste relatório técnico, juntamente com os resultados obtidos com e sem a combinação de classificadores para as mesmas.

6.1 Base de Dados Chess

A Base de Dados Chess (*king-rook-vs-king-pawn*), retirada do Repositório da UCI (Blake & Merz, 1998), trata do tema jogos de xadrez. Acredita-se que o *White* (branco) não seja capaz de vencer se o peão *Black* (preto) puder avançar com segurança. A descrição dos dados desta base encontra-se na Tabela 1.

Nº de Exemplos	3.196
Nº de Atributos Contínuos	0
Nº de Atributos Nominais	36
Classes	*"won"* e *"nowin"*

Tabela 1: Descrição da Base de Dados Chess.

A Base de Dados Chess possui 3196 exemplos, 36 atributos e a classe com dois possíveis valores: *White-can-win* (*"won"*) e *White-cannot-win* (*"nowin"*). A distribuição entre as duas classes (*"won"* e *"nowin"*) é razoavelmente balanceada, ou seja, possuem quantidade semelhante de exemplos.

O formato dos exemplos nesta Base de Dados é uma seqüência de 37 atributos, sendo que cada exemplo existente é uma descrição do tabuleiro para o final de um jogo de xadrez. Os 36 primeiros atributos descrevem o tabuleiro, e o último atributo é a classificação *"won"* ou *"nowin"*. Esta base não possui atributos ausentes.

Como a Base de Dados Chess foi retirada do Repositório da UCI, não se fez necessária nenhuma limpeza prévia dos dados. Apenas foi feita uma divisão do conjunto de dados original em dois conjuntos disjuntos, sendo um denominado conjunto de treinamento com 66,6% dos exemplos e o outro conjunto de teste com 33,3% dos exemplos. Essa divisão foi realizada devido a outros experimentos que estão sendo realizados no LABIC[1].

A Tabela 2 e o gráfico ilustrado na Figura 13 apresentam as taxas de erro obtidas no conjunto de teste desta Base de Dados com o uso de um único classificador, e as taxas obtidas com a combinação de classificadores gerados a partir de 3, 5, 7, 10, 15 e 20 amostras do conjunto de dados. No caso da combinação de classificadores heterogêneos com o algoritmo *stacking*, esses números representam o número de pares de conjuntos utilizados por cada algoritmo de nível-0.

Os resultados apresentados mostram que os dois algoritmos de combinação de classificadores homogêneos testados alcançaram, em certas combinações de classificadores, taxas de erros menores que a taxa apresentada por um único classificador para a Base de Dados em questão.

Pode-se notar que, para a combinação de 5 classificadores, o uso do algoritmo *boosting* forneceu a menor taxa de erro, 0,2%, encontrada para a Base de Dados Chess nos experimen-

[1]Laboratório de Inteligência Computacional – http://labic.icmc.usp.br

classificador único	0,5%		
	Bagging	*Boosting*	*Stacking*
3 classificadores	0,9%	0,4%	1,9%
5 classificadores	**0,4%**	**0,2%**	1,8%
7 classificadores	0,7%	1,1%	2,0%
10 classificadores	0,8%	0,4%	1,9%
15 classificadores	**0,4%**	0,7%	1,9%
20 classificadores	**0,4%**	0,3%	1,9%

Tabela 2: Resultados obtidos com a Base de Dados Chess.

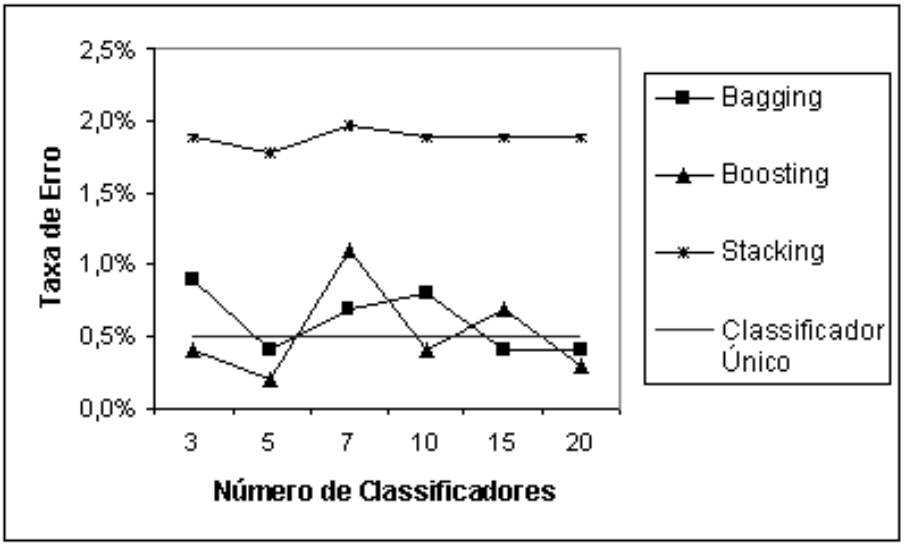

Figura 13: Taxas de Erro para a Base de Dados Chess.

tos realizados. Com o uso do algoritmo *bagging* a menor taxa de erro encontrada foi 0,4%, nas combinações de 5, 15 ou 20 classificadores.

É importante ressaltar que, mesmo tendo um desempenho inferior ao do algoritmo *boosting*, nas combinações citadas, o algoritmo *bagging* superou, em alguns casos, o classificador único, cuja a taxa de erro foi 0,5%.

O mesmo não aconteceu com a combinação de classificadores heterogêneos através do algoritmo *stacking*, que para todas as variações realizadas com a Base de Dados Chess forneceu uma taxa de erro maior do que a obtida com o uso de um único classificador.

6.2 Base de Dados Letter

A segunda Base de Dados, também retirada do Repositório da UCI (Blake & Merz, 1998), é a Letter, cuja descrição dos dados é apresentada na Tabela 3. Essa é uma base utilizada para reconhecimento de letras e seu objetivo é identificar cada conjunto de inúmeros *pixels* "preto-e-branco" retangulares mostrados como uma das 26 letras maiúsculas do alfabeto inglês.

As imagens dos caracteres foram baseadas em 20 fontes diferentes e cada letra com

essas 20 fontes foi aleatoriamente distorcida para produzir um arquivo com 20.000 estímulos únicos. Cada estínulo foi convertido em 16 atributos numéricos primitivos, que foram depois escalados para ajustar-se em um intervalo de valores inteiros de 0 a 15.

Nº de Exemplos	20.000
Nº de Atributos Contínuos	16
Nº de Atributos Nominais	0
Classes	*"A"*, *"B"*,..., *"Z"*

Tabela 3: Descrição da Base de Dados Letter.

Potanto, a Base de Dados Letter possui 20.000 exemplos e 17 atributos, sendo 16 numéricos (inteiros) e 1 classe categórica (letra maiúscula, ou seja, 26 valores de A até Z). Essa base não possui atributos ausentes, e todas as classes possuem praticamente o mesmo número de exemplos.

Assim, como nos experimentos realizados com a Base de Dados Chess, para a realização dos experimentos com a Base de Dados Letter não foi necessária nenhuma limpeza dos dados originais. De acordo com outros experimentos que estão sendo realizados no LABIC, a divisão do conjunto de dados original foi realizada da seguinte forma: os conjuntos de treinamento e de teste receberam, respectivamente, 75% e 25% dos exemplos do conjunto original.

Os resultados dos experimentos realizados com esta base são apresentados na Tabela 4 e ilustrados na Figura 14. Como pode ser visto, para a Base de Dados Letter, as três formas de combinação de classificadores testadas forneceram, na maioria dos casos, taxas de erros menores que a encontrada com o uso de um único classificador, ou seja, menores que 12,2%.

classificador único	12,2%		
	Bagging	*Boosting*	*Stacking*
3 classificadores	13,0%	9,7%	4,7%
5 classificadores	12,0%	6,9%	4,7%
7 classificadores	12,0%	6,1%	4,7%
10 classificadores	13,2%	5,1%	4,8%
15 classificadores	**11,2%**	**4,7%**	**4,6%**
20 classificadores	11,7%	**4,7%**	4,7%

Tabela 4: Resultados obtidos com a Base de Dados Letter.

O uso do algoritmo *bagging* apresentou taxas de erro menores que a do classificador único nas combinações de 5, 7, 15 e 20 classificadores, atingindo a maior precisão na combinação de 15, cuja taxa de erro foi 11,2%. Quanto ao algoritmo *boosting*, este mais uma vez mostrou-se superior ao algoritmo *bagging* nos experimentos realizados, sendo todas as taxas de erro menores que de um único classificador, e a menor delas igual a 4,7% na combinação de 15 e 20 classificadores.

Nos experimentos realizados com a Base de Dados Letter, o algoritmo *stacking* apresentou bons resultados. Esse algoritmo forneceu a menor taxa de erro encontrada para esta base, igual a 4,6%, sendo ligeiramente melhor que o algoritmo *boosting*. Essa taxa de erro foi encontrada utilizando-se 15 classificadores distintos para cada algoritmo de nível-0.

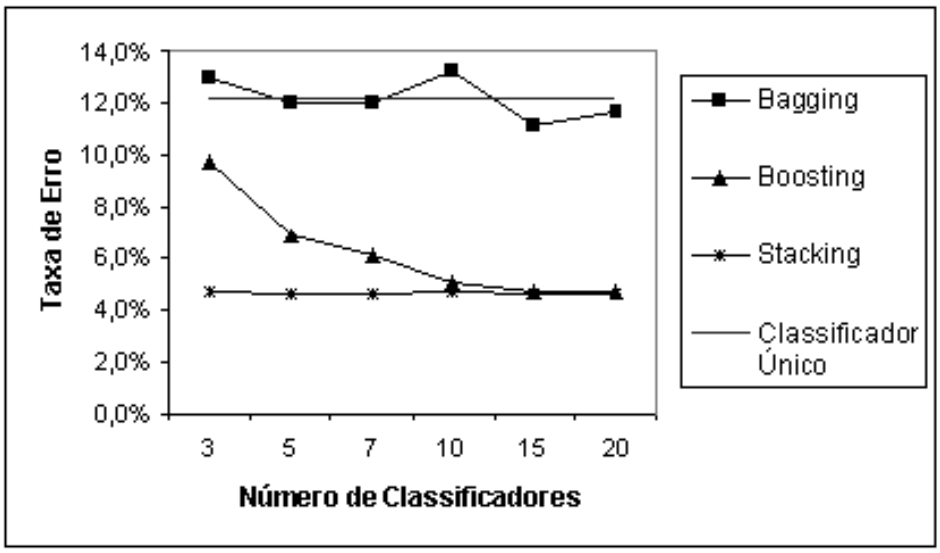

Figura 14: Taxas de Erro para a Base de Dados Letter.

Deve ser observado que, com o uso do algoritmo *boosting* e do algoritmo *stacking*, todas as combinações utilizadas nos experimentos realizados com esta base apresentaram maior precisão que o classificador único. Além disso, as maiores taxas de erro encontradas com esses algoritmos, 9,7% e 4,8%, ainda são menores que a taxa apresentada pela melhor combinação com o algoritmo *bagging* que, como já citado, foi de 11,2%.

6.3 Base de Dados Talassemia

As síndromes talassêmicas α representam uma alteração genética freqüente, de distribuição mundial, ocasionada pela deficiência de síntese das cadeias α da hemoglobina (Hb). A redução de cadeias promove uma menor formação de Hb nos eritrócitos, causando microcitose e hipocromia, e levando ao acúmulo das outras cadeias cuja síntese está normal.

O diagnóstico da talassemia α apresenta dificuldades. Os heterozigotos das talassemias α^0 e α^+ e os homozigotos da talassemia α^+ só podem ser diagnosticados na vida adulta por análise de DNA, pois a Hb Bart's desaparece após o período neonatal. Além disso, a pouca quantidade produzida nos heterozigotos da talassemia α^+ (1-3%) pode muitas vezes não ser detectada pelos métodos rotineiramente empregados.

A microcitose e a hipocromia, sem o concomitante aumento da HbA_2, podem ser resultantes da presença de talassemia α, de anemia por deficiência de ferro ou, ocasionalmente, de anemia por doenças crônicas. Não raramente, indivíduos com microcitose e hipocromia, sem anemia e sem a elevação da HbA_2 são detectados em exames hematológicos de rotina (Borges, 2000). Com o objetivo de investigar a contribuição da talassemia α nesses casos, foram analisados 339 indivíduos adultos, atendidos nos ambulatórios do Hospital das Clínicas da Universidade Estadual de Campinas – UNICAMP. Os dados obtidos são de relevância clínica, uma vez que a microcitose e a hipocromia são comumente interpretadas como sinais de deficiência de ferro.

Assim, o conjunto de dados original continha 339 exemplos e 23 atributos para cada paciente. A aquisição dos valores de cada atributo de decisão é custosa, pois os valores

de alguns deles são determinados pelo exame de DNA. Levando-se em consideração essa informação e as necessidades da especialista, foi possível a exclusão de alguns atributos que não deveriam fazer parte do processo de extração de conhecimento, uma vez que os valores desses atributos foram determinados posteriormente através do exame de DNA, ou mesmo, segundo a especialista, não são diretamente relevantes para o problema em questão. Após essa análise foram excluídos 13 atributos, considerando os seguintes fatores:

- atributos que eram apenas o identificador do exemplo;
- atributos cujos valores foram determinados posteriormente utilizando os resultados do DNA;
- atributos que eram outros resultados da técnica da Reação em Cadeia da Polimerase não ligados diretamente à determinação da presença de talassemia α.

Após a exclusão desses atributos, o conjunto de dados passou a possuir apenas atributos relacionados a informações extraídas do hemograma.

Os valores do atributo classe são encontrados através do exame de DNA, contendo os valores relativos à presença de talassemia α com 100% de confiança nessa determinação. Inicialmente, os possíveis valores para esse atributo eram *hetero, homo e normal*. Ambos valores *hetero e homo* indicam a presença de talassemia α, porém com o diferencial de saber se a doença era proveniente do pai ou da mãe, enquanto que o valor *normal* indica que o paciente não possui talassemia α.

Como o interesse da especialista está relacionado somente à determinação da talassemia α sem a necessidade de saber sua origem, e ainda devido ao fato da classe *homo* possuir poucos exemplos (apenas 5,6% do total), foi decidido substituir o atributo original por um novo atributo de decisão com apenas dois valores: normal e anormal. O valor *anormal* representa os exemplos de ambas classes *hetero e homo* do antigo atributo.

A descrição dos dados utilizados nos experimentos com a Base de Dados Talassemia pode ser vista na Tabela 5.

Nº de Exemplos	339
Nº de Atributos Contínuos	7
Nº de Atributos Nominais	2
Classes	*"Anormal"* e *"Normal"*

Tabela 5: Descrição da Base de Dados Talassemia.

Após a limpeza realizada, o conjunto de dados passou a ter 339 exemplos (51,9% da classe normal e 48,1% da anormal), 9 atributos (7 contínuos e 2 nominais), a classe com dois possíveis valores (*"anormal" e "normal"*) e nenhum valor ausente. Para a realização dos experimentos, este conjunto de dados foi dividido em conjunto de treinamento e conjunto de teste. O conjunto de treinamento recebeu 75% do conjunto de dados e o conjunto de teste recebeu 25%.

Podem ser vistos na Tabela 6 e no gráfico da Figura 15 os resultados obtidos com os experimentos realizados com a Base de Dados Talassemia. Nesses experimentos, o uso do algoritmo *bagging* mostrou-se um pouco mais eficiente que o uso do algoritmo *boosting*,

já que a menor taxa de erro apresentada por uma combinação realizada com o algoritmo *boosting* foi igual à taxa obtida com um único classificador (28,2%).

O mesmo pode ser dito ao comparar os resultados do algoritmo *bagging* com os do algoritmo *stacking*, pois em todas as variações realizadas com esta base, o uso do *stacking* forneceu uma única taxa de erro, e esta também é igual à taxa obtida com o uso de um único classificador.

classificador único	28,2%		
	Bagging	*Boosting*	*Stacking*
3 classificadores	32,9%	34,1%	28,2%
5 classificadores	**25,9%**	34,1%	28,2%
7 classificadores	34,1%	35,3%	28,2%
10 classificadores	42,3%	31,8%	28,2%
15 classificadores	34,1%	28,2%	28,2%
20 classificadores	34,1%	31,8%	28,2%

Tabela 6: Resultados obtidos com a Base de Dados Talassemia.

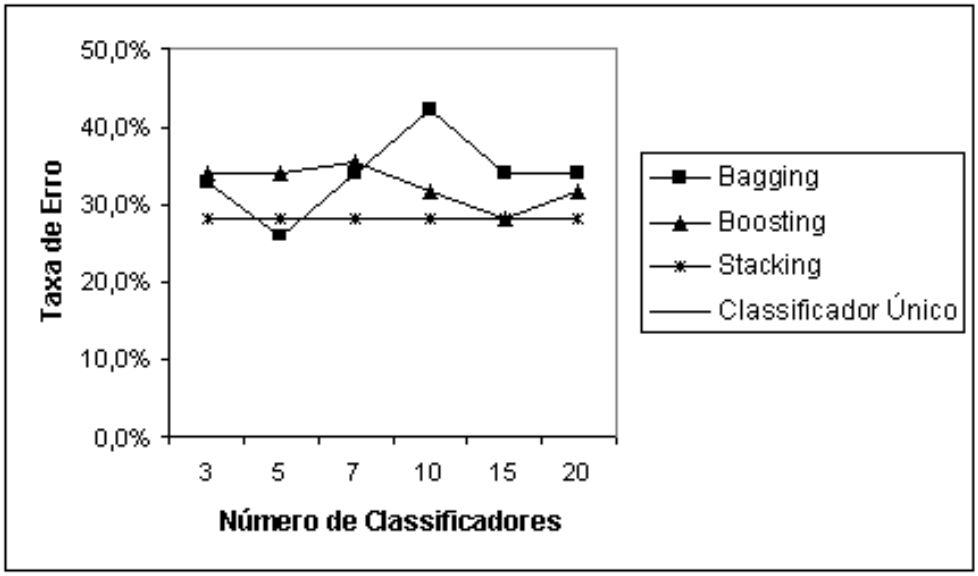

Figura 15: Taxas de Erro para a Base de Dados Talassemia.

Na combinação de 5 classificadores utilizando o algoritmo *bagging* foi alcançada uma taxa de erro de 25,9%. Esse foi o melhor resultado encontrado nos experimentos realizados com esta base.

7 Considerações Finais

A combinação de classificadores tem sido recomendada principalmente para problemas que não foram resolvidos satisfatóriamente por um único classificador, visto que esta pode ser

uma abordagem mais apropriada do que a simples seleção do classificador que apresentou menor erro.

Os experimentos apresentados neste relatório técnico foram realizados com o objetivo de comparar a precisão dos classificadores combinados com a precisão de um único classificador, verificando assim, através de resultados empíricos, que com o uso da combinação de classificadores é possível conseguir uma melhor precisão.

Esses experimentos foram realizados em três Bases de Dados diferentes e os algoritmos utilizados para realizar a combinação de classificadores foram os algoritmos *bagging*, *boosting* e *stacking*. Os dois primeiros realizam a combinação de classificadores homogêneos e o último, a de classificadores heterogêneos. Foi utilizada a ferramenta See5 para gerar os classificadores e combiná-los através do algoritmo *boosting*, já que esse algoritmo está presente nessa ferramenta. Para realizar a combinação através do algoritmo *bagging* foi necessário desenvolver um programa que implementasse esse algoritmo. Esse programa foi implementado em Perl e sua interface foi desenvolvida em Java. Os experimentos com o algoritmo *stacking* foram realizados com o uso da ferramenta Weka.

Os resultados obtidos mostram que a combinação de classificadores, realizadas com os algoritmo *bagging* e *boosting*, apresentou, na maioria dos casos, uma precisão ligeiramente maior do que a obtida com um único classificador. O uso do algoritmo *stacking* mostrou ser viável em apenas uma das Bases de Dados utilizadas.

Pode-se notar que cada algoritmo apresentou o melhor resultado em determinada Base de Dados. Para a Base de Dados Chess, o algoritmo que apresentou a menor taxa de erro foi o *boosting* ; para a Letter, foi o *stacking*; e para a Base de Dados Talassemia, foi o algoritmo *bagging*. É importante ressaltar que, não só o algoritmo de combinação mais apropriado variou de acordo com a Base de Dados utilizada, mas também o número de classificadores a serem combinados.

Referências

Bauer, E. & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning*, 36:105–142.

Blake, C. L. & Merz, C. J. (1998). UCI Repository of machine learning databases. Disponível em: `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Borges, E. (2000). Contribuição da talassemia α como causa de microcitose e hipocromia em uma população brasileira. Dissertação de Mestrado, Universidade Estadual de Campinas (UNICAMP).

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1996b). Stacked regressions. *Machine Learning*, 24:49–64.

Breiman, L. (2000). Some infinity theory for predictor ensemble. Technical Report 577, University of California, Berkeley. Disponível em: `http://citeseer.nj.nec.com/breiman00some.html`.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *Multiple Classifier Systems*, pages 1–15.

Domingos, P. (1997). Why does bagging works? A bayesian account and its implications. *In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 155–158.

Fan, D., Chan, P., & Stolfo, S. (1996). A comparative evaluation of combiner and stacked generalization. In *Proceedings of AAAI-96 workshop on Integrating Multiple Learned Models*, pages 40–46.

Freund, Y. & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Freund, Y. & Shapire, R. E. (1996). Experiments with a new boosting algorithm. In *13 th. International Conference on Machine Learning*, pages 148–156.

Gama, J. M. P. (1999). *Combining Classification Algorithms*. PhD thesis, Faculdade de Ciências da Universidade do Porto, Departamento de Ciências de Computadores.

LeBlanc, M. & Tibshirani, R. (1996). Combining estimates in regression and classification. *Journal of the American Statistical Association*, 91(436):1641–1650.

Merz, C. J. (1998). *Classification and Regression by Combining Models*. PhD thesis, University of California, Irvine. Disponível em: `http://www.ics.uci.edu/∼cmerz`.

Quinlan, J. (1996). Bagging, boosting and C4.5. *Proceedings of the Thirteenth National Conference on Artifiicial Intelligence*, pages 725–730.

RuleQuestResearch (2001). Data mining tools See5 and C5.0. Disponível em: `http://www.rulequest.com/see5-info.html`.

Skalak, D. B. (1997). *Prototype Selection for Composite Nearest Neighbor Classifiers*. PhD thesis, University of Massachusetts Amherst.

Ting, K. M. & Witten, I. H. (1997). Stacked generalization: when does it work? In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 866–871, San Francisco. Morgan Kaufmann Publishers.

Witten, I. H. & Frank, E. (2000). *Data Mining: practical machine learning tools and techiniques with Java implementations*. Morgan Kaufmann.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5:241–259.