UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

ISSN 0103-2569

Combinação de Regressores Homogêneos e Heterogêneos: Precisão e Compreensibilidade

Jaqueline Brigladori Pugliesi

Roberta Akemi Sinoara

Solange Oliveira Rezende

Nº 203

RELATÓRIOS TÉCNICOS

São Carlos - SP
Julho/2003

Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Combinação de Regressores Homogêneos e Heterogêneos: Precisão e Compreensibilidade

Jaqueline Brigladori Pugliesi
Roberta Akemi Sinoara
Solange Oliveira Rezende

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Julho/2003

Combinação de Regressores Homogêneos e Heterogêneos: Precisão e Compreensibilidade*

Jaqueline Brigladori Pugliesi
Roberta Akemi Sinoara
Solange Oliveira Rezende

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística
Laboratório de Inteligência Computacional
Caixa Postal 668, 13560-970 - São Carlos, SP, Brasil
e-mail: {jbpuglie, roberta, solange}@icmc.usp.br

Resumo

A combinação de regressores pode ser realizada com o objetivo de aumentar a precisão do conhecimento extraído de grandes Bases de Dados. Os métodos de *bagging*, *boosting* e *stacking* realizam essa combinação. Neste relatório é apresentada uma implementação dos métodos *bagging* e *boosting*, bem como uma análise comparativa entre métodos de combinação de regressores homogêneos e heterogêneos utilizando as ferramentas Cubist e Weka. Para a implementação do método *boosting*, é utilizada uma estratégia baseada em um limiar que leva em consideração o cálculo do desvio padrão do erro da regra encontrado no conjunto de dados, a fim de determinar quais exemplos foram corretamente preditos. Foram realizados experimentos com três Bases de Dados para verificar o erro entre o conhecimento obtido a partir da combinação dos regressores.

Palavras-Chave: Combinação de Regressores, *Boosting*, *Bagging*, *Stacking*.

Julho 2003

*Trabalho realizado com auxílio da FAPESP e CNPq.

Sumário

1 Introdução **1**

2 Técnicas para Combinação de Preditores **3**
2.1 *Bagging* 3
2.2 *Boosting* 4
2.3 *Stacking* 5

3 Algoritmos para Combinação de Regressores **7**

4 Experimentos Realizados **12**
4.1 Ferramenta Cubist e Weka 12
4.2 Bases de Dados Utilizadas 13
4.2.1 Abalone 13
4.2.2 CPU Performance 14
4.2.3 Housing 14
4.3 Análise dos Resultados 14

5 Considerações Finais **17**

Referências **20**

Lista de Figuras

1 *Bagging* – Geração Paralela de Preditores. 4
2 Geração de Preditores em Série. 5
3 Arquitetura *Stacking* – Treinamento. 6
4 Arquitetura *Stacking* – Aplicação. 7

Lista de Tabelas

1 Descrição da Base de Dados Abalone. 13
2 Descrição da Base de Dados CPU Performance. 14
3 Descrição da Base de Dados Housing. 14
4 Resultados dos Experimentos com Cubist para a Base de Dados Abalone. . 15
5 Resultados dos Experimentos com Weka para a Base de Dados Abalone. . . 15
6 Resultados dos Experimentos com Cubist para a Base de Dados CPU Performance. 16
7 Resultados dos Experimentos com Weka para a Base de Dados CPU Performance. 16
8 Resultados dos Experimentos com Cubist para a Base de Dados Housing. . 16
9 Resultados dos Experimentos com Weka para a Base de Dados Housing. . . 17

Lista de Algoritmos

1 Algoritmo da Implementação do *Bagging* – Geração de Amostras 8
2 Algoritmo da Implementação do *Bagging* – Combinação de Regressores . . 9
3 Algoritmo da Implementação do *Boosting* – Geração de Amostras 11

[1]Este documento foi elaborado com o LaTeXe sua bibliografia gerada automaticamente pelo BibTeX.

1 Introdução

A extração de conhecimento a partir de grande quantidade de dados é vista como um processo interativo e iterativo, e não como um sistema de análise automática, sendo centrado na interação entre usuários, especialistas do domínio e responsáveis pela aplicação. Dessa maneira, não se pode esperar a extração de conhecimento útil simplesmente submetendo um conjunto de dados a uma "caixa preta" (Mannila 1997).

Assim, o processo de Extração de Conhecimento de Bases de Dados, ou Mineração de Dados, inicia-se com o entendimento do domínio da aplicação, considerando aspectos como os objetivos dessa aplicação e as fontes de dados (Base de Dados da qual se pretende extrair conhecimento). Em seguida, é realizada uma seleção de dados a partir dessas fontes, de acordo com os objetivos do processo. Os conjuntos de dados resultantes dessa seleção são, então, pré-processados, ou seja, recebem um tratamento para poderem ser submetidos aos métodos e ferramentas na etapa de extração de padrões.

A etapa de extração de padrões tem o objetivo de encontrar modelos (conhecimento) a partir de dados. Normalmente, como resultado dessa etapa, tem-se preditores que, para um dado problema, fornecem uma decisão. Após a etapa de extração de padrões, vem a de pós-processamento, na qual o conhecimento é avaliado quanto a sua qualidade e/ou utilidade para que, em caso positivo, seja utilizado para apoio a algum processo de tomada de decisão.

É importante notar que, por ser um processo eminentemente iterativo, as etapas da Mineração de Dados não são estanques, ou seja, a correlação entre as técnicas e métodos utilizados nas várias etapas é considerável, a ponto da ocorrência de pequenas mudanças em uma delas afetar substancialmente o sucesso de todo o processo. Portanto, os resultados de uma determinada etapa podem acarretar mudanças a quaisquer das etapas posteriores ou, ainda, o recomeço de todo o processo (Fayyad, Haussler, & Stolorz 1996).

Durante a etapa de pós-processamento, normalmente, é selecionado o preditor que fornece a maior precisão (ou menor erro) para a aplicação. Porém, se for interessante aumentar a precisão do conhecimento extraído, pode ser utilizada a combinação da saída de diversos preditores. Quando essa estratégia é utilizada, o preditor resultante dessa combinação é chamado de *ensemble* (LeBlanc & Tibshirani 1996; Breiman 2000).

A combinação de preditores tem sido recomendada principalmente para problemas que não foram resolvidos satisfatoriamente por um único preditor 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, por meio da combinação de preditores, resultados com maior precisão quando comparados com o resultado obtido com um único preditor. Vários métodos 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).

O preditor combinado é caracterizado pela seleção de k preditores e pela função de combinação. Como a combinação de preditores depende desses dois fatores, vários caminhos podem ser utilizados visando um bom resultado. Os preditores a serem combinados podem ser extraídos a partir do mesmo conjunto de treinamento ou de várias amostras de

uma Base de Dados. Além disso, pode-se gerar preditores utilizando o mesmo algoritmo de Aprendizado de Máquina (AM), chamados de preditores homogêneos, ou fazendo uso de diferentes algoritmos, sendo chamados de preditores heterogêneos. Freqüentemente, são utilizados para combinar preditores homogêneos os métodos *bagging* e *boosting*, e para preditores heterogêneos, o método *stacking*.

Neste relatório técnico são apresentados experimentos realizados com a combinação de regressores[1] homogêneos e heterogêneos utilizando os três métodos citados (*bagging*, *boosting* e *stacking*), bem como as ferramentas Cubist (RuleQuestResearch 2003) e Weka (Witten & Frank 2000). Detalhes a respeito da teoria e experimentos de combinação de classificadores estão detalhados em (Sinoara, Pugliesi, & Rezende 2003).

Vale ressaltar que o objetivo principal deste trabalho é investigar o comportamento dos métodos para combinação de modelos, tanto homogêneos quanto heterogêneos, em problemas de regressão, ou seja, em problemas em que o atributo-meta é contínuo. Para isso foram realizadas implementações visando essa combinação. Ressalta-se que a principal dificuldade em regressão refere-se ao método *boosting*, uma vez que regras de regressão não predizem um valor categórico definido e sim, um valor numérico, envolvendo um grau de incerteza quanto à classe predita.

Assim, por ser o atributo meta contínuo, o cálculo da cobertura da regra, ou seja, a verificação da classe estar correta ou não em problemas de regressão se torna mais complicado do que em classificação. Essa verificação é necessária no método *boosting*, uma vez que a cada nova iteração deve-se verificar quais exemplos foram cobertos corretamente pelo algoritmo. Para resolver esse problema foi utilizada uma estratégia baseada em um limiar, utilizando desvio-padrão, para verificar quais exemplos são cobertos corretamente pelo regressor (Pugliesi & Rezende 2003). Além disso, como o método *boosting* utiliza pesos nos exemplos, foi implementado o método da roleta para a amostragem aleatória.

Paralelamente, em nosso laboratório de pesquisa — LABIC (Laboratório de Inteligência Computacional)[2] — está sendo desenvolvido um ambiente denominado Discover, que visa fornecer um ambiente para apoiar as etapas do processo de descoberta do conhecimento, oferecendo funcionalidades voltadas para Aprendizado de Máquina, Mineração de Dados e Textos. Os programas para esse ambiente estão sendo implementados em forma de *scripts*[3] na linguagem PERL (Till 1996) — *Practical Extraction and Report Language*. Até o momento, as principais atividades desenvolvidas nesse projeto estão relacionadas com classificação e todo este trabalho, que está relacionado com a melhora da precisão de regressores faz agora parte do Discover.

Além da precisão, avaliar a compreensibilidade do conhecimento extraído é importante especialmente para viabilizar um melhor entendimento do modelo. Porém, quando se utiliza combinação de regressores normalmente ganha-se em precisão e perde-se em compreensibilidade.

Quanto à organização, este relatório está dividido da seguinte maneira: na Seção 2 são apresentados os três métodos utilizados para combinar preditores: *bagging*, *boosting*

[1] Os modelos de regressão são também conhecidos como regressores.
[2] http://labic.icmc.usp.br/
[3] Scripts são pequenos programas que realizam tarefas atômicas.

e *stacking*. Como este relatório trata de problemas de regressão, na Seção 3 são apresentados os algoritmos que foram implementados para a combinação de regressores. As descrições das três Bases de Dados utilizadas, das ferramentas Cubist e Weka, bem como dos resultados obtidos nos experimentos são apresentadas na Seção 4. Por fim, na Seção 5 encontram-se as considerações finais deste relatório.

2 Técnicas para Combinação de Preditores

Nesta seção são apresentados três métodos que são normalmente utilizados para realizar a combinação de preditores, *bagging*, *boosting* e *stacking*. Enquanto os dois primeiros métodos são para combinação de preditores homogêneos, o terceiro realiza a combinação de preditores heterogêneos.

Uma das características que diferem os dois métodos utilizados para combinar preditores homogêneos é que o método *boosting* altera a distribuição do conjunto de treinamento baseado no desempenho dos preditores anteriores, enquanto o mesmo não acontece com o método *bagging*. Assim, enquanto o *boosting* gera as amostras (ou novos conjuntos de treinamento) em série, o *bagging* as gera em paralelo.

2.1 *Bagging*

O método descrito por Breiman em 1996 (Breiman 1996), que recebeu o nome de ***b**ootstrap **agg**regat**ing*** (ou simplesmente *bagging*), realiza a combinação de preditores gerados pelo mesmo algoritmo de Aprendizado de Máquina. De acordo com esse método, 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 exemplos podem aparecer mais de uma vez e outros podem não aparecer.

Os *bootstraps* formados são usados como novos conjuntos de treinamento, dando origem a vários preditores, como apresentado na Figura 1. A fim de gerar o preditor final, que é utilizado para predizer o conjunto de teste, todos os preditores são combinados por meio de uma votação não ponderada (no caso de classificação) ou do cálculo da média (no caso de regressão).

Geralmente, o método *bagging* aumenta o desempenho de algoritmos de AM instáveis, ou seja, algoritmos cuja saída seja um preditor 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 (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

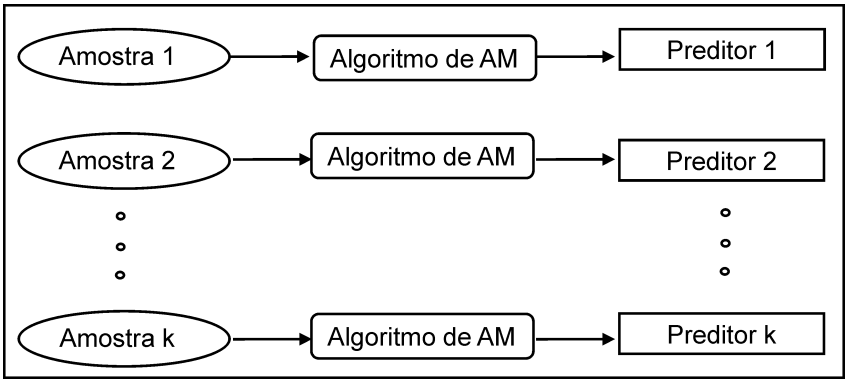

Figura 1: *Bagging* – Geração Paralela de Preditores.

reduzida com o uso de *bagging*. Para isso, duas alternativas baseadas na teoria de Aprendizado Bayesiano foram testadas. A primeira hipótese é que *bagging* funciona bem por ser uma aproximação do ótimo procedimento do modelo Bayseano médio, com uma prévia aproximação implícita. A segunda é que *bagging* funciona bem, 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 Mineração de Dados (Gama 1999). Para isso, adiciona-se um *loop* antes de selecionar o *bootstrap* e mandá-lo para o algoritmo de aprendizado e, por fim, fazer a combinação dos preditores gerados.

2.2 *Boosting*

O método *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 um algoritmo "fraco", cujos preditores devem ser apenas um pouco melhor que um preditor que prediz de forma aleatória.

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

O uso do método *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 encontrada em problemas reais.

Um dos algoritmos *boosting* mais utilizado é o *AdaBoost* (*Adaptative Boosting*), apresentado por Freund e Schapire (Freund & Schapire 1997). Dado um conjunto de treina-

mento 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 k, os pesos $D_k(m)$ são alterados e um novo conjunto T'_k é gerado, com base em T'_{k-1}. A partir de cada um desses conjuntos de treinamento um preditor é gerado. Note que *AdaBoost* gera preditores em série, como é ilustrado na Figura 2, de maneira diferente do bagging, que os gera em paralelo.

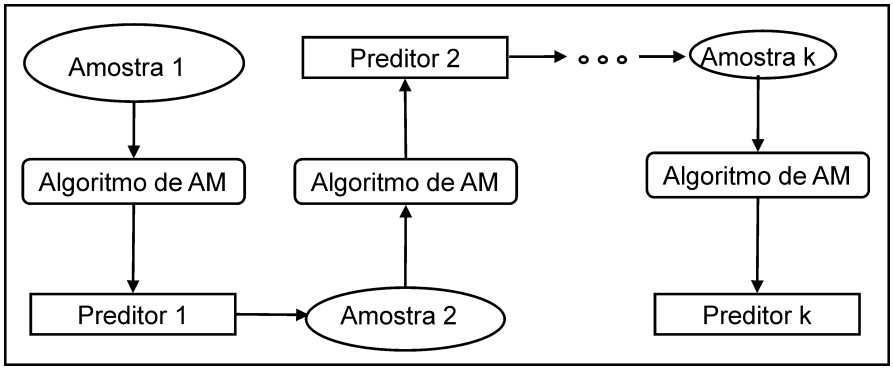

Figura 2: Geração de Preditores em Série.

O *AdaBoost* pode usar os pesos para construir um novo conjunto de treinamento T'_n, sendo que 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, a cada iteração os pesos são ajustados de acordo com o desempenho do preditor anterior. Os exemplos que foram preditos de forma errada têm o peso aumentado e os que foram preditos corretamente têm o peso diminuído. Com isso, os preditores focalizam os exemplos mais difíceis de serem preditos, ou seja, aqueles que foram preditos de forma errônea pelo preditor anterior. Desta maneira, o algoritmo *AdaBoost* torna os preditores cada vez mais "fortes".

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

2.3 *Stacking*

No *stacking*, também chamado de *stacked generalization*, diferentes algoritmos de Aprendizado de Máquina geram preditores com o conjunto de treinamento original. Logo após, as descrições dos exemplos de treinamento são estendidas para incluir os resultados da predição desses preditores, formando um novo conjunto de treinamento. Esse novo conjunto é então analisado por outros algoritmos de Aprendizado de Máquina a fim de produzir futuros preditores, e assim por diante.

Wolpert (1992) propôs o framework *stacked generalization*, que é uma arquitetura em

camadas. Preditores 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 preditor no nível mais alto fornece a predição final.

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

Trabalhos em arquitetura *stacking*, como (Gama 1999; Merz 1998), concentram-se em arquiteturas de duas camadas, como é ilustrada na Figura 3.

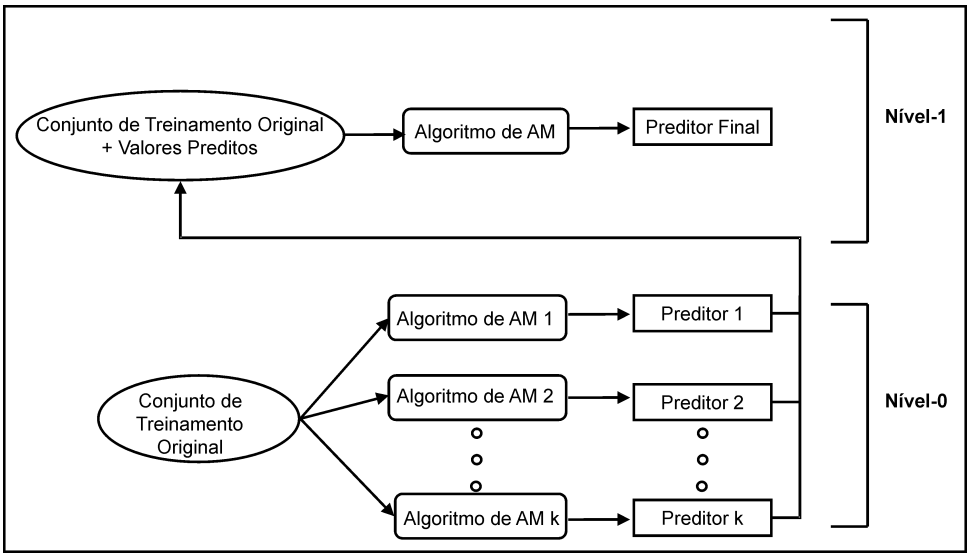

Figura 3: Arquitetura *Stacking* – Treinamento.

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

A fase de treinamento começa com a geração do conjunto de k pares $T' = \{(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)\}$, a partir de um conjunto de treinamento T, com n exemplos. Tais pares podem ser gerados de duas maneiras diferentes. A primeira maneira, proposta por (Fan, Chan, & Stolfo 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 elementos de T retirando-se os elementos contidos em x_i.

Os algoritmos de nível-0 são utilizados para gerar preditores tendo como conjuntos de

treinamento cada conjunto y_i. Os preditores 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 um exemplo para o conjunto de treinamento do algoritmo de nível-1. Os exemplos devem conter, além da classe correta, os valores preditos pelos preditores 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 preditor de nível-1, com a utilização de um único algoritmo no nível-1. A resposta desse preditor será a resposta do preditor combinado.

Para explorar completamente o conjunto de treinamento original, todos os preditores de nível-0 são novamente treinados 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 4, quando um novo exemplo é apresentado, ele é calculado por todos os preditores de nível-0. Um vetor de predições é formado e utilizado no nível-1, que fornece a predição final do exemplo.

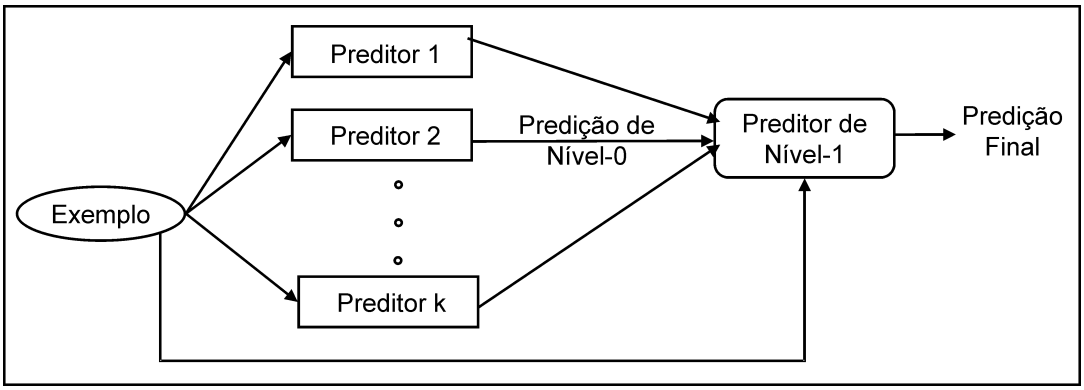

Figura 4: Arquitetura *Stacking* – Aplicação.

Na próxima seção esses três métodos serão aplicados a problemas de regressão.

3 Algoritmos para Combinação de Regressores

Para realizar a combinação de regressores, isto é, preditores com atributo-meta numérico, estão disponíveis, na ferramenta Weka, somente a implementação do método *bagging* e do *stacking* (Witten & Frank 2000), sendo necessária a implementação do método *bagging* para a ferramenta Cubist e do método *boosting* para ambas as ferramentas.

O método *bagging* foi implementado, neste trabalho, para combinar regressores gerados pela ferramenta Cubist (RuleQuestResearch 2003). Para essa implementação do método *bagging* foram desenvolvidos dois *scripts* em PERL: um para geração de amostras e outro para combinação de regressores.

O primeiro *script*, descrito no Algoritmo 1, implementa a geração das amostras, criando todos os arquivos necessários para a posterior execução do algoritmo de regressão

em todas as amostras. Conforme o que foi apresentado na Seção 2.1, cada amostra, ou *bootstrap*, é formada tomando aleatoriamente elementos do conjunto de dados original, com substituição, sendo as amostras do mesmo tamanho do conjunto de dados original.

Algoritmo 1 Algoritmo da Implementação do *Bagging* – Geração de Amostras

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 o atribui a variável *numExemplos*
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**

Saída: Arquivos `.data`, `.names` e `.test` gerados para cada amostra

Dessa maneira, o *script* para geração de amostras recebe como entrada três arquivos (`.data`, `.names` e `.test`), que são utilizados pelo algoritmo de regressão para gerar um regressor, além do número de amostras que devem ser geradas e o nome dos arquivos de saída. Vale ressaltar que o arquivo `.data` possui os dados do conjunto de treinamento, o `.names` contém a descrição dos atributos do conjunto de dados e o `.test` os dados do conjunto de teste.

A combinação dos regressores propriamente dita é realizada por um segundo *script*, cujo funcionamento é apresentado no Algoritmo 2. Esse *script* para combinação de regressores recebe como entrada o número de regressores que serão combinados, além dos arquivos gerados pelo algoritmo de regressão. Esses arquivos contém os valores reais e os preditos por determinado regressor para cada exemplo do conjunto de teste.

A resposta do regressor combinado é a média das predições de cada regressor e o erro é calculado através do módulo da diferença entre esse valor e a predição real. Dividindo-se o erro dos regressores combinados, que é a soma dos erros dos exemplos, pelo número total de exemplos tem-se a taxa de erro da combinação de regressores. Observe que com esse procedimento a compreensibilidade do modelo é prejudicada.

Algoritmo 2 Algoritmo da Implementação do *Bagging* – Combinação de Regressores

Entradas:

nome dos arquivos de entrada {gerados pelo Cubist}
constante inteira $numAmostras$ {número de regressores a serem combinados}

1: Lê os arquivos de entrada
2: **para** $i := 1$ a $numExemplos$ **faça**
3: $preditoFinal := 0$
4: $erroFinal := 0$
{Para cada regressor}
5: **para** $i := 1$ a $numAmostras$ **faça**
6: $predito :=$ valor predito pelo regressor
7: $preditoFinal := preditoFinal + predito$
8: **fim-para**
{Calcula valor predito pelo regressor combinado para o exemplo}
9: $preditoFinal := \dfrac{preditoFinal}{numAmostras}$
10: $valorReal :=$ valor real da classe do exemplo
{Calcula o erro para o exemplo}
11: $erro := |valorReal - preditoFinal|$
12: $erroFinal := erroFinal + erro$
13: **fim-para**
{Calcula o erro do Regressor Combinado}
14: $erroFinal := \dfrac{erroFinal}{numExemplos}$

Saída: Taxa de Erro do Regressor Combinado – $erroFinal$

Assim, nos experimentos apresentados neste relatório, a combinação utilizando o método *bagging* com a ferramenta Cubist ocorreu de acordo com os seguintes passos:

1. Geração de Amostras: foram geradas amostras a partir dos conjuntos de treinamento, formando grupos compostos por 3, 5, 7 e 10 amostras para cada Base de Dados. A geração de tais amostras foi realizada utilizando o *script* descrito no Algoritmo 1.

2. Geração de Regressores: para cada amostra foi gerado um regressor utilizando a ferramenta Cubist.

3. Combinação de Regressores: os regressores gerados a partir de cada grupo de amostras foram combinados utilizando o conjunto de teste, obtendo assim, quatro regressores combinados, um para cada grupo de amostras do item 1.

Com relação à combinação de regressores usando o método *boosting* não é possível utilizar o algoritmo *AdaBoost*, que trata somente de problemas de classificação. Para tanto, o método *boosting* foi implementado, tanto para a ferramenta Cubist quanto para a Weka, considerando as particularidades dos problemas de regressão.

No método *boosting* implementado, a cada exemplo é atribuído um peso. Inicialmente, todos os exemplos recebem um peso $w_i = 1/N$, sendo N o número de exemplos do conjunto de treinamento. Esses pesos são atualizados após a geração de cada regressor, sendo os pesos dos exemplos preditos de maneira correta diminuídos.

Por se tratar de um problema de regressão, como já citado, não é trivial verificar se a predição está correta ou não. Essa verificação foi realizada com base na metodologia que utiliza um limiar para definir se um dado exemplo foi corretamente coberto pelo regressor. Com a utilização desse limiar é possível calcular o número de exemplos para o qual a cabeça da regra é verdadeira ou falsa. O ponto crítico nessa estratégia é a definição do valor do limiar. O limiar utilizado leva em consideração o cálculo do desvio padrão do erro da regra encontrado no conjunto de dados (Pugliesi & Rezende 2003).

Dessa maneira, a cada iteração e atualização dos pesos dos exemplos, um novo conjunto de treinamento é formado, sendo que a probabilidade de determinado exemplo ser selecionado para esse novo conjunto é proporcional ao seu peso. Os exemplos que não foram preditos corretamente pelo regressor têm maior probabilidade de pertencer ao novo conjunto. Essa seleção é realizada utilizando o método da roleta. De acordo com esse método, um exemplo irá ocupar, em uma roleta, uma área proporcional ao seu peso. Assim, exemplos com pesos maiores ocuparão áreas maiores e, conseqüentemente, terão maior probabilidade de serem selecionados. Maiores detalhes sobre esse método pode ser encontrado em (Carvalho, Braga, & Ludermir 2003). A implementação do método da roleta, aqui utilizado, foi realizada por meio de uma reta numérica. Cada exemplo possui um intervalo nessa reta, com extensão igual ao seu peso. Para selecionar um exemplo, é sorteado, de maneira aleatória, um número pertencente a essa reta. O exemplo selecionado será aquele cujo intervalo possui o número sorteado.

Por fim, após determinado número de regressores serem gerados, estes são combinados de maneira semelhante à combinação realizada no método *bagging*. Porém, as iterações

para gerar novos regressores a serem combinados é interrompida caso o erro do último regressor atinja um determinado valor.

Assim, os experimentos com o método *boosting*, bem como os realizados com o *bagging*, podem ser divididos em três fases: geração de amostras, geração de regressores e combinação de regressores. Porém, no caso do *boosting*, as amostras, bem como os regressores, não são gerados paralelamente e sim em série. A cada iteração, uma nova amostra (ou conjunto de treinamento) e, conseqüentemente, um novo regressor são gerados, de acordo com o desempenho do regressor anterior, como ilustrado na Figura 2.

Tanto para a geração de amostras quanto para a combinação dos regressores, foram implementados programas na forma de *script* em PERL. O *script* utilizado para gerar as amostras é ilustrado no Algoritmo 3.

Algoritmo 3 Algoritmo da Implementação do *Boosting* – Geração de Amostras

Entrada:

nome dos arquivos de entrada {arquivos .data, .names e .rules}

1: Lê os arquivos de entrada
2: Verifica o tamanho do conjunto de treinamento e o atribui a variável $numExemplos$
3: **para** $i := 1$ a $numExemplos$ **faça**
4: Verifica se o exemplo foi predito corretamente pelo conjunto de regras {Calcula o erro do preditor}
5: **se** exemplo foi predito erroneamente **então**
6: $erro := erro + w_i$
7: **fim-se**
8: **fim-para**
9: $\beta := erro/(1 - erro)$ {Atualiza os pesos}
10: **para** $i := 1$ a $numExemplos$ **faça**
11: **se** exemplo foi predito corretamente **então**
12: $w_i := w_i * \beta$
13: **fim-se**
14: **fim-para**
15: Normaliza os pesos dos exemplos
16: Seleciona os exemplos para o novo conjunto de treinamento, utilizando os pesos como probabilidade (método da roleta)

Saída: Novo conjunto de treinamento

O primeiro passo para a realização dos experimentos com o método *boosting* é a geração de um regressor a partir de todos os dados do conjunto de treinamento original. Tendo esse primeiro regressor, o *script* para geração de amostras é utilizado.

Esse *script* recebe como entrada três arquivos, `.data`, `.names` e `.rules`, que devem estar na sintaxe padrão de exemplos e regras de regressão do DISCOVER, definida em (Pugliesi, Dosualdo, & Rezende 2003). Vale ressaltar que o arquivo `.data` contém os

exemplos de treinamento, o `.names` possui as descrições dos atributos e no arquivo `.rules` encontram-se o regressor gerado convertido para a sintaxe padrão de regras de regressão. Com esses arquivos, o *script* verifica quais exemplos foram preditos de forma incorreta e quais foram preditos de foram correta pelo regressor, atualiza os pesos dos exemplos e seleciona os exemplos do novo conjunto de treinamento com base nesses pesos.

Com o novo conjunto de treinamento formado é iniciada uma nova iteração. A partir desse conjunto será gerado um regressor e, de acordo com seu desempenho, um novo conjunto de treinamento, e assim sucessivamente. Essa fase de geração de amostras e regressores é finalizada após o número de iterações requeridas pelo usuário ser atingido ou um regressor obter um erro maior que 0,5 ou igual a zero. Em seguida, é realizada a combinação dos regressores.

A implementação da etapa de combinação dos regressores nos experimentos realizados com o método *boosting* é similar à combinação com o método *bagging*. A resposta do regressor combinado é dada pela média das respostas dos regressores gerados. O *script* utilizado nesta etapa recebe como entrada o número de regressores que devem ser combinados e os arquivos contendo as predições geradas pelas ferramentas Cubist ou Weka.

Já os experimentos com o método *stacking*, utilizado na combinação de regressores heterogêneos, foram realizados com a ferramenta Weka, visto que esta ferramenta disponibiliza esse método implementado além de viabilizar a utilização de diferentes algoritmos para problemas de regressão. Com a ferramenta Cubist não foi utilizado o método *stacking*, pois essa possui apenas um algoritmo de regressão.

4 Experimentos Realizados

Nesta seção são apresentados os experimentos realizados com a combinação de regressores homogêneos por meio dos métodos *bagging* e *boosting*, bem como a combinação de regressores heterogêneos utilizando o *stacking*. Os regressores combinados em tais experimentos foram gerados pelas ferramentas Cubist e Weka. Os experimentos foram realizados com três Bases de Dados: Abalone, CPU Performance e Housing.

4.1 Ferramenta Cubist e Weka

Os experimentos apresentados neste relatório técnico foram realizados com o auxílio das ferramentas Cubist e Weka.

A ferramenta Cubist é freqüentemente utilizada para gerar modelos baseados em regras para problemas de regressão. Essa ferramenta está disponível para os sistemas operacionais Unix e Windows 98/Me/2000/XP. Outras características que favorecem o uso dessa ferramenta é a facilidade na sua utilização e na compreensão dos modelos gerados, além de suportar a análise de Bases de Dados com milhares de registros e dezenas ou centenas de atributos numéricos ou nominais (RuleQuestResearch 2003).

A ferramenta Weka (*Waikato Environment for Knowledge Analysis*) foi desenvolvida

na Universidade de Waikato, na Nova Zelândia, implementada em Java e testada nos sistemas operacionais Linux, Windows e Macintosh (Witten & Frank 2000). Ela pode ser vista como uma coleção de algoritmos de Aprendizado de Máquina que podem ser utilizados para resolver problemas de Mineração de Dados do mundo real. Além de algoritmos para classificação, regressão, associação e *clustering*, a ferramenta Weka também fornece suporte às fases de pré-processamento e de visualização de dados.

O principal algoritmo da ferramenta Weka utilizado foi o *M5*, já que este é um algoritmo de regressão e todas as Bases de Dados utilizadas possuem como classe um atributo numérico. O *M5* foi utilizado nos experimentos com os métodos *bagging* e *boosting*, bem como foi o algoritmo de nível-1 para o método *stacking*, que teve como algoritmos de nível-0 o *Linear Regression* e o *Locally Weighted Regression* também disponíveis na ferramenta Weka.

4.2 Bases de Dados Utilizadas

Nesta seção, são apresentadas as descrições das três Bases de Dados utilizadas nos experimentos apresentados neste relatório, Abalone, CPU Performance e Housing, sendo todas retiradas do Repositório da UCI (Blake & Merz 1998).

4.2.1 Abalone

O objetivo dos experimentos com a Base de Dados Abalone é predizer a idade de um molusco da califórnia a partir de medidas físicas. A idade desse molusco é determinada pelo número de anéis existentes no interior do mesmo e, para se contar tais anéis, é preciso cortar o molusco e visualizá-lo no microscópio, o que é uma tarefa lenta e cansativa. Com essa Base de Dados espera-se predizer o número de anéis, que determina a idade, a partir de outras medidas, mais fáceis de obter, como peso e diâmetro.

Como pode ser visto na Tabela 1, a Base de Dados Abalone possui 4.177 exemplos, com 7 atributos contínuos e 1 nominal, além da classe (número de anéis), que é um valor numérico. Por ter sido retirada do Repositório da UCI, não foi necessário realizar limpeza nos dados antes de iniciar o processo de extração de padrões. Foi realizada apenas uma divisão dos dados em dois conjuntos: conjunto de treinamento (com 75% dos dados) e conjunto de teste (com 25% dos dados).

Nº de Exemplos	4.177
Nº de Atributos Contínuos	7
Nº de Atributos Nominais	1

Tabela 1: Descrição da Base de Dados Abalone.

4.2.2 CPU Performance

Os experimentos com a Base de Dados CPU Performance têm como objetivo prever o poder de processamento de um computador, com base em características como quantidade mínima e máxima de memória principal. Assim, cada exemplo representa uma configuração diferente de computador. A descrição dos dados utilizados nesses experimentos é apresentada na Tabela 2.

Nº de Exemplos	209
Nº de Atributos Contínuos	6
Nº de Atributos Nominais	0

Tabela 2: Descrição da Base de Dados CPU Performance.

Assim, a base CPU Performance, utilizada nos experimentos apresentados neste relatório, possui 6 atributos, sendo todos contínuos, além da classe, que representa o desempenho da CPU (*Central Processing Unit*). Os 209 exemplos foram divididos em conjunto de treinamento e de teste, tendo 75% e 25% dos dados originais, respectivamente.

4.2.3 Housing

O valor de moradias em subúrbios de Boston pode ser predito com a Base de Dados Housing. Cada exemplo dessa base descreve uma área com características como taxa de crime e número médio de quartos por moradia. Na Tabela 3 é apresentada a descrição dos dados dessa base.

Nº de Exemplos	506
Nº de Atributos Contínuos	12
Nº de Atributos Nominais	1

Tabela 3: Descrição da Base de Dados Housing.

Nesses experimentos foi utilizada a Base de Dados Housing sem modificações, possuindo 506 exemplos, 12 atributos contínuos, 1 nominal e a classe numérica, representando o valor das moradias. Os dados da Base de Dados Housing foram divididos em conjunto de treinamento e de teste, seguindo a mesma porcentagem de 75% e 25%, respectivamente.

4.3 Análise dos Resultados

Para cada uma das três Bases de Dados descritas na Seção 4.2 foram realizados experimentos com a combinação de 3, 5, 7 e 10 regressores, utilizando, além das ferramentas Cubist e Weka, as implementações dos métodos *bagging* e *boosting*. Para o método *stacking* esses valores foram as quantidades de regressores gerados por cada algoritmo de nível-0.

Todos os resultados apresentados nesta seção foram obtidos utilizando-se a média final dos erros de combinação dos regressores.

O método *bagging* foi que o apresentou melhores resultados em todos os experimentos realizados. O método *boosting* não chegou a gerar resultados para algumas das combinações dos experimentos realizados, pois o regressor obteve um erro maior que 0,5 ou igual a zero, antes de gerar o número total de combinações solicitadas.

Os resultados obtidos utilizando os métodos *bagging* e *boosting* e a ferramenta Cubist, nas três Bases de Dados, são apresentados nas Tabelas 4, 6 e 8.

Já os resultados obtidos utilizando os métodos *bagging*, *boosting* e *stacking* e a ferramenta Weka, nas três Bases de Dados, são apresentados nas Tabelas 5, 7 e 9.

Na Tabela 4 o melhor resultado da combinação é com o método *bagging* usando 10 regressores, e todas as combinações usando o método *boosting* não melhoraram a precisão para a Base de Dados Abalone comparado com o resultado obtido a partir de um único regressor. Pode-se notar que o método *boosting* não gerou resultados para a combinação de 7 e 10 regressores, pois o erro do regressor 5 foi maior do que 0,5 e a combinação parou.

Pode-se notar um comportamento semelhante para a Tabela 5, na qual foi utilizada a ferramenta Weka. Neste cato, o melhor resultado também foi o da combinação utilizando o método *bagging*, porém com 5 regressores. Tanto o *boosting* quanto o *stacking* não obtiveram resultados melhores que o regressor único. O *boosting* não gerou resultados para as combinações com 5, 7 e 10 regressores, inviabilizando a geração dos resultados finais.

Regressor único	1,5	
	Bagging	*Boosting*
3 regressores	1,5017	1,9714
5 regressores	1,4918	2,3156
7 regressores	1,4918	—
10 regressores	1,4710	—

Tabela 4: Resultados dos Experimentos com Cubist para a Base de Dados Abalone.

Regressor único	1,525		
	Bagging	*Boosting*	*Stacking*
3 regressores	1,5126	1,9158	1,6526
5 regressores	1,4952	—	1,6591
7 regressores	1,5863	—	1,6542
10 regressores	1,5540	—	1,6492

Tabela 5: Resultados dos Experimentos com Weka para a Base de Dados Abalone.

Todos os resultados para as duas combinações de regressores usando a Base de Dados CPU Performance, apresentados na Tabela 6, são melhores que o obtido com um único regressor, ou seja, obtiveram menores erros. O melhor resultado para esse experimento

é o com 5 regressores utilizando o método *bagging*. Também não há resultados para a combinação de 7 e 10 regressores para o método *boosting*.

Utilizando a ferramenta Weka, Tabela 7, todos os resultados também são melhores que o obtido a partir de um único regressor. Assim como com a ferramenta Cubist, o melhor deles é o que utiliza o método *bagging* com 5 regressores. Neste caso, o método *boosting* não gerou resultados para 5, 7 e 10 regressores.

Regressor único	57,3	
	Bagging	*Boosting*
3 regressores	53,4256	54,4737
5 regressores	44,7892	56,8211
7 regressores	53,3055	—
10 regressores	46,7817	—

Tabela 6: Resultados dos Experimentos com Cubist para a Base de Dados CPU Performance.

Regressor único	58,2753		
	Bagging	*Boosting*	*Stacking*
3 regressores	44,4937	45,6213	52,2104
5 regressores	42,9138	—	52,1741
7 regressores	44,2828	—	55,1591
10 regressores	44,9673	—	52,0249

Tabela 7: Resultados dos Experimentos com Weka para a Base de Dados CPU Performance.

Na Tabela 8 são apresentados os resultados da Base de Dados Housing. O melhor resultado foi também obtido utilizando o método *bagging*. Além disso, os resultados de todas as combinações, tanto do *bagging* quanto do *boosting* foram melhores que o regressor único para todos os experimentos.

Já na Tabela 9 o melhor resultado foi obtido utilizando o método *bagging* com 10 regressores. O método *boosting* gerou resultado apenas para a combinação de 3 regressores, sendo esse um pouco menor que o regressor único. Já os resultados gerados pelo método *stacking* foram todos piores que o regressor único.

Regressor único	8,97	
	Bagging	*Boosting*
3 regressores	3,9078	6,6143
5 regressores	4,9352	5,6064
7 regressores	5,9040	4,7097
10 regressores	4,7323	—

Tabela 8: Resultados dos Experimentos com Cubist para a Base de Dados Housing.

Regressor único	4,5816		
	Bagging	*Boosting*	*Stacking*
3 regressores	6,6656	4,5456	6,0024
5 regressores	5,1377	—	6,0588
7 regressores	3,9145	—	6,0715
10 regressores	3,8390	—	6,1839

Tabela 9: Resultados dos Experimentos com Weka para a Base de Dados Housing.

No geral, o método *bagging* apresentou melhores resultados em todos os experimentos realizados. O método *boosting* não chegou a gerar resultados para todas as combinações de regressores dos experimentos realizados, pois o regressor obteve um erro maior do que 0,5 ou igual a zero antes da geração do número de combinações requerido. Pode-se notar que o método *boosting* utilizando a estratégia por nós definida teve bons resultados, tendo um erro menor que o classificador único em todos os experimentos das Bases de Dados CPU Performance e Housing, podendo ser utilizado em outros experimentos. Os erros foram maiores apenas para a Base de Dados Abalone, mas essa Base de Dados também não obteve uma melhora significativa em nenhuma das combinações realizadas.

A combinação de regressores mostrou que, no geral, ganha-se em precisão, porém perde-se em compreensibilidade, pois a compreensão pelo usuário de um único modelo é maior, e ao se combinar vários regressores não se tem mais um único regressor para ser analisado. Desse modo, não se sabe exatamente de onde vem os resultados, pois o mesmo é uma combinação dos regressores anteriores.

5 Considerações Finais

Neste relatório técnico, foi tratado o problema de Mineração de Dados preditiva para problemas de regressão. A abordagem de múltiplos regressores para análise de padrões vem tendo um expressivo crescimento e muitos casos de sucesso.

Sabe-se que o processo de Extração de Conhecimento de Bases de Dados não termina com a extração de padrões, pois após essa etapa o conhecimento deve ser avaliado, antes de ser utilizado. Nessa última etapa, denominada de pós-processamento, deve-se obter a precisão do conhecimento obtido a partir de determinado algoritmo, bem como avaliar o conhecimento quanto a sua qualidade e/ou utilidade para que, em caso positivo, seja utilizado para apoio a algum processo de tomada de decisão. É na etapa de pós-processamento que métodos de combinação de preditores podem ser utilizados a fim de aumentar a precisão do conhecimento extraído.

Como mencionado, em problemas de regressão, o cálculo da cobertura das regras é complexo, pois o atributo meta é contínuo. Assim, necessita-se de um tratamento diferente para método *boosting*. Para isso foi utilizado a abordagem proposta, implementada e validada que utiliza um limiar para verificar a predição de novos exemplos em modelos de regressão (Pugliesi & Rezende 2003; Pugliesi, Dosualdo, & Rezende 2003). Como o

boosting trabalha com pesos para identificar a probabilidade do exemplo ser selecionado e compor o novo conjunto de dados foi implementado o método da roleta que viabiliza a amostragem aleatória com peso.

Com a utilização dessa abordagem, limiar e roleta, é viabilizado a combinação de regressores em série.

Neste relatório são apresentados experimentos realizados com a combinação de regressores utilizando três Bases de Dados retiradas do Repositório da UCI. Foram realizados experimentos combinando os regressores gerados pela ferramenta Cubist por meio de implementações dos métodos *bagging* e *boosting*. Outro conjunto de experimentos foi realizado com o auxílio da ferramenta Weka, que gerou e combinou regressores utilizando os métodos *bagging* e *stacking*. Os regressores gerados por essa ferramenta também foram combinados utilizando a implementação proposta do método *boosting*.

O método *bagging* obteve melhores resultados que o *boosting* e o *stacking*. Porém, pode-se observar que, salvo algumas poucas exceções, as combinações de regressores utilizando os três métodos obtiveram erros menores que a execução de um único regressor.

Ressalta-se que a combinação de regressores apesar de, em geral, incrementar a precisão perde-se muito em compreensibilidade.

Os resultados obtidos nos experimentos realizados para este relatório confirmaram que a combinação de regressores pode aumentar a precisão do conhecimento extraído de Bases de Dados comparados a um único regressor. Entretanto, pode-se notar que não é possível determinar um padrão que mostre qual número de regressores é o melhor.

Referências

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

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

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`.

Carvalho, A. C. P. L. F., A. P. Braga, & T. B. Ludermir (2003). *Sistemas Inteligentes: Fundamentos e Aplicações* (1 ed.)., Chapter 9 – Computação Evolutiva, pp. 225–248. Barueri, SP: Manole.

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*, 155–158.

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

Fayyad, U. M., D. Haussler, & P. Stolorz (1996). KDD for science data analysis: Issues and examples. In E. Simoudis, J. W. Han, & U. Fayyad (Eds.), *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pp. 50–56. AAAI Press.

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

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

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

Mannila, H. (1997). Methods and problems in data mining. *Database Theory ICDT'97, 6th International Conference, Lecture Notes in Computer Science 1186*, 41–55.

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

Pugliesi, J. B., D. G. Dosualdo, & S. O. Rezende (2003). Sintaxe Padrão para Representar Regras de Regressão no Discover. Relatório Técnico do ICMC - Número 193. Disponível em: `ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_193.zip`.

Pugliesi, J. B. & S. O. Rezende (2003). Medidas de Erro para Regras de Regressão. *Anais do IV ENIA - Encontro Nacional de Inteligência Artificial*. no prelo.

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

RuleQuestResearch (2003). Data mining with Cubist. Disponível em: `http://www.rulequest.com/cubist-info.html`.

Sinoara, R. A., J. B. Pugliesi, & S. O. Rezende (2003, Janeiro). Combinação de classificadores homogêneos e heterogêneos. Technical Report 184, ICMC – USP, São Carlos. Disponível em: `http://citeseer.nj.nec.com/breiman00some.html`.

Till, D. (1996). *Teach Yourself Perl 5 in 21 Days*. Sams Publishing.

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

Witten, I. H. & E. Frank (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.