Problemas de Regressão utilizando Combinação de Preditores baseado no Coeficiente de Correlação

Luzia Vidal de Souza[1], Aurora T. R. Pozo[2], Joel M. C. da Rosa[3]

[1]Departamento de Desenho – Universidade Federal do Paraná (UFPR)

[2]Departmento de Informática – Universidade Federal do Paraná (UFPR)

[3]Departamento de Estatística - Universidade Federal do Paraná (UFPR)
Curitiba, Paraná – Caixa Postal 19981 – Brasil

{luzia, joel}@ufpr.br, aurora@inf.ufpr.br

Abstract: *Ensembles of machines currently receive a lot of attention; they combine predictions from different forecasting methods as a procedure to improve the accuracy. This paper explores Boosting technique to obtain an ensemble of regressors and proposes a new formula for updating the weight and for the final hypothesis. This new formula is based on the correlation coefficient. To validate this method, experiments were accomplished, the algorithm has been compared with the results obtained by published works. The results show advantages in the use of the proposed approach.*

Resumo: Algoritmos de combinação de preditores têm recebido muita atenção dos pesquisadores; eles combinam as previsões obtidas por diferentes preditores de maneira a melhorar a acurácia do preditor. Neste trabalho a técnica *Boosting* é explorada para obter uma combinação de regressores e propõe uma nova fórmula para a atualização dos pesos e para a obtenção da hipótese final. Esta nova fórmula é baseada nos coeficientes de correlação. Para validar a metodologia, experimentos foram realizados, os resultados obtidos pelo algoritmo foram comparados aos resultados obtidos em outros trabalhos publicados e mostram melhorias em relação aos demais métodos.

1. Introdução

A técnica de *Boosting* é utilizada, com freqüência, por pesquisadores na área de Aprendizagem de Máquina, para melhorar a acurácia dos algoritmos de classificação. Esta técnica tem sido aplicada, recentemente e com sucesso, aos problemas de regressão e consiste em efetuar repetidas execuções de um algoritmo de aprendizagem básico, modificando a distribuição de pesos no conjunto de treinamento e combinando os preditores obtidos num único preditor mais eficiente. A primeira proposta do algoritmo foi apresentada por [Schapire 1990] para resolver problemas de classificação binários. Em 1996, [Freund & Schapire 1996] propuseram uma versão do algoritmo chamada de *Adaboost*, na qual o algoritmo ajusta um modelo logístico aditivo no qual o número de iterações é o número de funções usadas na representação aditiva que serão utilizadas para aproximar a função estimada. O algoritmo gera em cada passo uma distribuição sobre as observações da amostra, dando um peso maior às observações classificadas incorretamente no passo anterior. Várias outras versões desta técnica foram criadas, entre elas destaca-se os algoritmos de *Boosting* para problemas de classificação com

múltiplas classes chamadas *AdaBoost.M1* e *AdaBoost.M2* [2]. Em 1997 surge também a versão do algoritmo para problemas de regressão, a qual foi chamada *AdaBoost.R* [Freund e Schapire, 1997]. Em 1997 Drucker desenvolveu o algoritmo *Adaboost.R2* [Drucker 1997], uma versão Ad-hoc do *AdaBoost* para problemas de regressão e realizou alguns experimentos tendo obtido resultados promissores. Recentemente os pesquisadores [Friedman, Hastie & Tibshirani 2001], [Duffy 2000], [Zemel & Pitassi 2001] e [Ridegway 1999] comparam o algoritmo *Boosting* com um algoritmo de gradiente descendente, que otimiza a função de perda (resíduos) e mostraram que quando a função de perda utilizada é a exponencial, o algoritmo *Adaboost* se assemelha ao método de Newton. A proposta de [Solomatine & Shrestha 2004], *AdaBoost.RT,* apresenta uma forma de atualização dos pesos utilizando o erro relativo ao invés do erro absoluto. Após o estudo das várias versões do algoritmo *Boosting* e de várias medidas para a atualização dos pesos, verificou-se que o coeficiente de correlação entre os valores observados no conjunto de treinamento e os valores previstos por um algoritmo base poderiam fornecer bons resultados se utilizados na atualização dos pesos, uma vez que o coeficiente de correlação é uma medida que relaciona uma variável à outra, desta forma, as alterações sofridas por uma delas são acompanhadas por modificações nas outras. Desta forma surgiu o algoritmo *Boosting Correlation Improovment* (BCI) que considera os coeficientes de correlação e os utiliza para a atualização da distribuição de pesos e para a combinação final dos preditores. A Seção 2 explica sucintamente as metodologias empregadas. A Seção 3 descreve os experimentos realizados e seção 4 apresenta as conclusões.

2. Metodologia

2.1 Combinação de Preditores

Os dois algoritmos mais populares de combinação de preditores são Bagging e *Boosting*. A idéia principal dos dois algoritmos é a de utilizar a saída final de diferentes preditores para agregá-los em um único preditor. O algoritmo *Bagging* é uma técnica *Bootstrap* [9] que treina cada classificador utilizando diferentes conjuntos de treinamento, gerados aleatoriamente e com reposição de N exemplos, onde N é o número de exemplos do conjunto de treinamento. Desta forma, muitos exemplos aparecem mais de uma vez no conjunto de treinamento, enquanto que outros podem não aparecer nenhuma vez. Cada um dos conjuntos de treinamento é utilizado para gerar um preditor ou classificador diferente [Blake & Merz 1998], os quais serão combinados ao final do algoritmo. O algoritmo *Boosting* será detalhado nas seções seguintes.

2.2 *Adaboost* para problemas de classificação

O algoritmo *Adaboost* utiliza como entrada um conjunto de treinamento (x_1, y_1), ..., (x_m, y_m) onde cada x_i é um vetor de características, pertencente a algum espaço X e y_i pertence ao conjunto $Y = \{-1, +1\}$, ou seja, o algoritmo é utilizado para problemas de classificação com duas classes. O *Adaboost* chama um classificador base, repetidamente, num conjunto de T execuções, $t = 1, ...,T$. Uma das principais idéias do algoritmo é modificar a distribuição, ou conjunto de pesos, sobre o conjunto de treinamento. O peso desta distribuição no exemplo de treinamento i na execução t é denotado por D_t(i). Inicialmente os pesos são todos iguais, mas a cada execução, os pesos dos exemplos classificados incorretamente são incrementados de forma que o classificador "base" seja forçado a atuar com maior intensidade sobre estes exemplos no conjunto de treinamento, a taxa de erro é calculada contando o número de

classificações incorretas, conforme Eq. (1). Observa-se que o erro é calculado de acordo com a distribuição D_t sobre a qual o classificador fraco foi treinado.

$$e_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i) \qquad (1)$$

Uma vez que a hipótese h_t tenha sido recebida (obtida na iteração t do algoritmo base), o *Adaboost* escolhe um parâmetro α_t pertencente ao conjunto dos números reais, o qual mede a importância que é dada à hipótese h_t. O valor de α_t para problemas binários, bem como a atualização do vetor de pesos D_t, estão detalhados no pseudo-código do algoritmo é apresentado na Figura 1 [Schapire & Freund 1996]. Para formular a hipótese final H, cada hipótese h_t contribui com uma certa confiança dada por α_t.

Algoritmo *Adaboost*:
Dado: S= $\{(x_1, y_1), ..., (x_m, y_m); x_i \in X, y_i \in \{-1, +1\}\}$

Inicialize $D_1(i) := \frac{1}{m} \quad \forall\ (x_i, y_i) \in S$

For $t = 1, ... T,$ Treine o classificador base usando a distribuição D_t e obtenha a hipótese fraca $h_t: X \rightarrow \{-1, +1\}$. Calcule α_t: $\alpha_t = \frac{1}{2}\ln\frac{1-e_t}{e_t}$, onde e_t é a taxa de erro do classificador h_t.

Atualize a distribuição: $D_{t+1}(i) = \frac{D_t(i)}{Z_t} = \begin{cases} e^{-\alpha_t}, & \text{se } h_t(x_i) = y_i \\ e^{\alpha_t}, & \text{se } h_t(x_i) \neq y_i \end{cases}$

$$= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}, \text{ onde } Z_t \text{ é o fator de normalização}$$

End For

Saída: Hipótese Final: $H(x) = sign\left(\sum_{1...T} \alpha_t * h_t(x)\right)$

Figura 1. Algoritmo Adaboost para problemas de classificação binários

2.3 *Boosting* para Problemas de Regressão – *AdaBoost.R*

Nos problemas de regressão o que se deseja é encontrar uma aproximação para determinados valores de uma variável contínua. Dada uma função $f(x)$, deve-se encontrar uma função $g(x)$, tal que $f(x_i) = g(x_i)$, $\forall x_i \in X$, onde X é um conjunto de valores do intervalo considerado. A função $f(x)$ não é necessariamente conhecida, o que se conhece é um conjunto de pontos $\{x, f(x) | x \in X\}$. Sobre um conjunto de treinamento S = $\{(x_1, f(x_1)),..., (x_m, f(x_m))\}$, aplica-se um algoritmo básico de aprendizado, na tentativa de encontrar uma função g próxima da função f no domínio desejado. Há muitas maneiras de se medir o quão próxima a função g está da função f. Uma maneira de verificar esta proximidade é verificar quando o erro médio quadrático se torna pequeno, ou garantir que a função g esteja próxima da função f em todo o domínio. Ao se utilizar o algoritmo *Boosting,* pode-se obter uma aproximação ainda maior. Após se utilizar um algoritmo de aprendizagem básico, repetidas vezes, em diferentes amostras, obtém-se várias hipóteses que serão então combinadas, de forma a obter uma melhor aproximação para a função f. O algoritmo *AdaBoost.R* [Schapire & Freund 1998], funciona de

maneira semelhante ao *AdaBoost*. A idéia é usar alguma ordenação dos valores para diferenciar as predições corretas das incorretas, como se fosse um problema de classificação. Nos problemas de regressão não é possível obter valores de predição exatos como nos problemas de classificação. As discrepâncias entre os valores preditos e observados são inevitáveis. A medida desta discrepância é que vai permitir dizer se o valor previsto é ou não aceitável, define-se para isto uma função de perda. De acordo com [Hastie, Tibishirani & Friedman 2001] utilizando a função de perda exponencial Eq. (2), o algoritmo *Adaboost* apresenta um melhor desempenho. Para manter a notação utilizada nos artigos citados neste trabalho, considere deste ponto em diante que: $f_t(x_i)$ é o valor obtido pelo preditor base e $y(x_i)$ é o valor real.

$$L_t(x_i) = 1 - \exp\left(-\frac{|f_t(x_i) - y(x_i)|}{\max_{i=1\ldots m}|f_t(x_i) - y(x_i)|}\right) \quad (2)$$

Da mesma forma que no algoritmo *Adaboost*, o algoritmo base recebe os exemplos (x_i, y_i) do conjunto de treinamento para gerar a hipótese h: $X \rightarrow Y$, onde $Y = [0, 1]$, tal que para cada valor x o preditor forneça um valor aproximado para y. O algoritmo procura gerar uma hipótese h, com o menor erro de previsão possível. Os problemas de regressão são então reduzidos à problemas de classificação binários [Allwein, Schapire & Singer 2000] para então se aplicar o *Adaboost.* O problema é que cada exemplo do problema de regressão é expandido à vários problemas de classificação, com isto o número de iterações *Boosting* cresce linearmente. A hipótese final h_f é determinada, calculando a mediana ponderada das hipóteses fracas, conforme a Eq. (3), onde α_t é um valor de confiança dado à hipótese h_t.

$$h_f(x) = inf\left\{y \in Y : \sum_{t:h_t(x)\leq y} log(1/\alpha_t) \geq \frac{1}{2}\sum_i log(1/\alpha_t)\right\} \quad (3)$$

2.4 O Algoritmo *AdaBoost.RT*

O algoritmo *AdaBoost.RT* para problemas de regressão foi apresentado por [Solomatine & Shrestha 2004], neste trabalho os autores propõem uma metodologia, cuja idéia principal é a introdução de uma constante ϕ, como um valor limiar de erro relativo que deve classificar a predição como correta ou incorreta, esta taxa de erro é calculada contando o número de predições corretas e incorretas, conforme Eq. (4).

$$i : \left|\frac{f_t(x_i) - y_i}{y_i}\right| > \phi \quad (4)$$

Este valor de ϕ é então utilizado para a atualização do vetor de pesos, Eq. (5).

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} x \begin{cases} \alpha_t, se \left|\frac{f_t(x_i) - y_i}{y_i}\right| \leq \phi \\ 1, caso\ contrário \end{cases} \quad (5)$$

A principal diferença entre o algoritmo *Adaboost.RT* e os demais algoritmos de *Boosting* é no cálculo da função de perda, que usa o erro relativo ao invés do erro absoluto, o que faz com seja dada maior ênfase aos exemplos que possuem maiores erros de previsão, de forma que na

próxima iteração estes erros sejam reduzidos. A outra diferença deste algoritmo em relação aos demais algoritmos de *Boosting* é quanto ao critério de parada, neste caso pode-se definir o número de iterações necessárias, enquanto que na maioria dos demais algoritmos, o critério de parada utilizado, é o de verificar quando a taxa de erro torna-se maior ou igual a 0.5, por fim a combinação final dos preditores obtidos é feita pela média ponderada dos preditores obtidos e do valor de confiança dado à cada preditor, enquanto que nos outros algoritmos a combinação final é feita utilizando a mediana geométrica. A fórmula de combinação final dos preditores está representada na Eq. (6).

$$f_{fin}(x) = \sum_t \log\left(\frac{1}{\alpha_t}\right) \frac{f_t(x)}{\sum_t \log\left(\frac{1}{\alpha_t}\right)} \tag{6}$$

Uma desvantagem deste algoritmo é a necessidade de se encontrar um valor ótimo para ϕ, se os valores de ϕ forem muito baixos, poucos exemplos serão corretamente classificados, por outro lado, se os valores de ϕ forem muito altos, poucos exemplos obterão uma boa previsão. Assim, para encontrar valores ótimos para ϕ, é necessário utilizar algum procedimento de minimização do erro relativo absoluto, antes de se aplicar o algoritmo *AdaBoost.RT*. Os experimentos realizados pelos autores desta técnica mostraram que o algoritmo efetua predições melhores que as realizadas por um único preditor com uma confiança de 99%, porém para que seja comparado aos demais algoritmos de *Boosting* propostos, há necessidade de se realizar mais experimentos.

2.5 *Boosting* usando coeficientes de Correlação *(BCI)*

A abordagem proposta para o algoritmo *Boosting* tem uma fundamentação empírica ao utilizar o coeficiente de correlação para a atualização do vetor de pesos, o que influencia diretamente na minimização da função de perda, pois este coeficiente fornece uma relação entre as variáveis em questão, que neste caso são os valores previstos e valores observados no conjunto de treinamento. O mesmo coeficiente foi também utilizado na combinação final dos preditores obtidos por um algoritmo básico. A definição do coeficiente de correlação é dada pela Eq. (7). Onde m é o número de observações da amostra, $\bar{x}$ é a média da amostra X e $\bar{y}$ é a média da amostra Y.

$$\rho(x, y) = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2 \sum_{i=1}^{m}(y_i - \bar{y})^2}} \tag{7}$$

O método proposto segue a metodologia do algoritmo *Adaboost.R*. Inicializa a distribuição de pesos com valores iguais para todos os exemplos da amostra. Assim, dado o conjunto de observações no tempo $T = \{x_1, x_2, ..., x_m\}$, o vetor de pesos para a primeira iteração é: $P_1(i) = 1/m$ para todo $x_i \in T$. Utiliza-se então um algoritmo de aprendizagem básico para obter os valores previstos. O erro é então calculado, utilizando a função de perda exponencial apresentada na Eq. (2). Onde $f_t(x_i)$ representa os valores previstos pelo algoritmo base na iteração t e $y(x_i)$ são os valores observados. A escolha da utilização da função exponencial para calcular a perda obtida pelo algoritmo foi feita em função dos resultados obtidos com a

realização de vários experimentos, que comparavam o desempenho do algoritmo utilizando as funções de perda linear Eq. (8), quadrática Eq. (9) e exponencial Eq. (2).

$$L_i = \frac{\left|f_t(x_i) - y(x_i)\right|}{\max_{i=1,\ldots m}\left|f_t(x_i) - y(x_i)\right|} \tag{8}$$

$$L_i = \frac{\left|f_t(x_i) - y(x_i)\right|^2}{\max_{i=1,\ldots m}\left|f_t(x_i) - y(x_i)\right|^2} \tag{9}$$

O cálculo dos coeficientes de correlação entre os valores observados e os valores previstos são feitos de acordo com a Eq. (10).

$$\rho_t(f_t(x), y(x)) = \frac{\sum_{i=1}^{m}\left(f_t(x_i) - f_t(\bar{x})\right)\left(y(x_i) - y(\bar{x})\right)}{\sqrt{\sum_{i=1}^{m}\left(f_t(x_i) - f_t(\bar{x})\right)^2 \sum_{i=1}^{m}\left(y(x_i) - y(\bar{x})\right)^2}} \tag{10}$$

onde m é o número de observações do conjunto de treinamento, $f_t(\bar{x})$ é a média da amostra obtida por $f_t(x)$ e $y(\bar{x})$ é a média dos valores observados no conjunto de treinamento. A atualização dos pesos é feita multiplicando-se o coeficiente de correlação obtido na iteração t do algoritmo *BCI* pelos vetores de peso P_t e de perda L_t, a atualização é feita de acordo com a Eq. (11).

$$P_{t+1}(x) = \rho_t\left(f_t(x), y(x)\right) * P_t * L_t(x_i) \tag{11}$$

Finalmente, após terem sido completadas as T iterações do algoritmo BCI, a combinação final dos preditores é feita conforme a Eq. (12). Na Figura 2 é apresentado o pseudo-código do algoritmo BCI.

$$F(x_{i-1}) = \frac{\sum_{t=1}^{T}\rho_t\left(f_t(x), y(x)\right) * f_t(x)}{\sum_{t=1}^{T}\rho_t\left(f_t(x), y(x)\right)},\ i = 1, \ldots, m\text{-}1;\ F(x_m) = \frac{\sum_{t=1}^{T} f_t(x_i)}{T},\ I = m \tag{12}$$

3. Experimentos

Para validar a metodologia proposta, o algoritmo BCI foi implementado para os problemas de regressão múltipla. O algoritmo "base" utilizado foi o *Model Tree*, descrito em Breiman et al. [Breiman et al. 1984], através de implementação computacional feita utilizando o *software* estatístico *R*[1]. Para avaliar o desempenho do método, a base de dados utilizada bem como o algoritmo base (*Model Tree*) foram os mesmos apresentados no trabalho de Solomatine [Solomatine & Shrestha 2004], retiradas do *UCI repository* [Breiman 1997] e do trabalho de Drucker [Drucker 1997], que são bases de dados públicas. Desta maneira os resultados puderam ser comparados aos resultados obtidos através das Redes Neurais Artificiais (multy-

[1]http://www.r-project.org/

layer perceptron), *Model Tree* e *Adaboost.RT* publicadas no mesmo trabalho. As bases de dados utilizadas foram: *Housing*, *Auto-Mpg* e CPU e Friedman #1.

Algoritmo – BCI
Dado $T = \{x_1, x_2, ..., x_m\}$
Inicializar $P_1(i) = 1/m$, para todo $x_i \in T$
For $t = 1, ...,T,$ execute um algoritmo básico de aprendizado, e determine f_t, modelo de previsão para a iteração t. Calcule a perda para cada exemplo:

$$L_i = 1 - \exp\left(- \frac{|f_t(x_i) - y(x_i)|}{\max_{i=1...m} |f_t(x_i) - y(x_i)|} \right)$$, onde $y(x_i)$ são os valores observados.

Calcule o coeficiente de correlação entre $f_t(x_i)$ e $y(x_i)$.

$$\rho_t(f_t(x), y(x)) = \frac{\sum_{i=1}^{m} \left(f_t(x_i) - f_t(\bar{x})\right)\left(y(x_i) - y(\bar{x})\right)}{\sqrt{\sum_{i=1}^{m} \left(f_t(x_i) - f_t(\bar{x})\right)^2 \sum_{i=1}^{m} \left(y(x_i) - y(\bar{x})\right)^2}}$$

Faça a atualização dos pesos: $P_{t+1}(x_i) = \rho_t\left(f_t(x), y(x)\right) * P_t * L_t(x_i)$

End for

Saída Final: obter a combinação final dos preditores, $F(x)$:

$$F(x_{i-1}) = \frac{\sum_{t=1}^{T} \rho_t\left(f_t(x), y(x)\right) * f_t(x_i)}{\sum_{t=1}^{T} \rho_t\left(f_t(x), y(x)\right)}, i = 1, ..., m\text{-}1 \qquad F(x_m) = \frac{\sum_{t=1}^{T} f_t(x_i)}{T}, i = m$$

Figura 2. Algoritmo BCI (*Boosting Correlation Improovment*)

3.1 Descrição das Bases de Dados

Os conjuntos de dados foram divididos em treinamento e teste, conforme é apresentado na tabela (1). Na tabela (2) estão as variáveis preditoras, não preditoras (classes) e o tipo da variável de saída (Prevista).A base de dados denominada CPU, refere-se à performance relativa de CPU's de 29 fabricantes diferentes de computadores. A base de dados *Housing* possui dados referentes aos valores medianos de moradias no subúrbio da cidade de Boston. A base *Auto-Mpg* contém dados para a previsão do consumo de combustível por milha, medida em galões, de acordo com as características de cada automóvel, tais como número de cilindros, potência do motor, aceleração e outros itens. A base de dados *Friedman* foi gerada de acordo com os trabalhos de Friedman [Friedman 2001] e Drucker [Drucker 1997], é um problema de predição não linear, com 10 variáveis independentes da distribuição uniforme [0,1] e gerada de acordo com a Eq. (13).

$$y = 10 sen(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + n \qquad (13)$$

Onde n possui uma distribuição Normal padrão. Somente 5 variáveis preditoras são necessárias, mas o preditor tem a função de distinguir as variáveis que não possuem habilidade de previsão (x_6 a x_{10}) de acordo com as variáveis preditoras (x_1 a x_5).

Tabela 1. Conjuntos de Dados

Conjuntos de Dados	Instâncias	Treinamento	Teste
CPU	209	137	72
Housing	506	337	169
Auto-Mpg	398	262	136
Friedman #1	1500	990	510

Tabela 2. Número e tipo de variáveis

Base de Dados	Nº de Atributos (Preditores)	Nº de Atributos (não Preditores)	Tipo da Variável Prevista
CPU	6	2	Contínua
Housing	13	0	Contínua
Auto_Mpg	7	1	Contínua
Friedman	5	1	Contínua

3.2 Preparação dos Dados

Os conjuntos de treinamento e teste foram selecionados aleatoriamente e sem reposição do conjunto de dados original e de acordo com os dados da tabela (3). Foram gerados 10 conjuntos de treinamento e teste. O algoritmo BCI foi implementado utilizando o software R utilizando árvores de indução. Cada exemplo aparece no conjunto de treinamento proporcionalmente ao seu peso, desta forma, os exemplos que apresentam as maiores taxas de erro, aparecem mais vezes no conjunto de treinamento gerado. Este novo conjunto de treinamento é então utilizado para gerar o modelo de regressão. Após o modelo ter sido gerado, ele é utilizado para calcular os valores previstos do conjunto de teste. Este procedimento é repetido 10 vezes para cada conjunto de treinamento e teste (validação cruzada). O número de iterações *Boosting* também é 10, desta forma são executadas 10 iterações do algoritmo *Boosting* para cada um dos 10 conjuntos de treinamento. Os valores previstos são então combinados de acordo com a equação proposta no algoritmo BCI.

3.3 Resultados

Os resultados foram comparados aos obtidos pelo *Model Tree*, *ANN* (*Artificial Neural Network*), *Bagging*, *Adaboost.R2* e *Adaboost.RT,* que utilizam também utilizam o *Model Tree* como algoritmo base. Os dados apresentados na tabela 3 são os mesmos apresentados no trabalho de Solomatine [Solomatine & Shrestha 2004], foi acrescentada a coluna 1, que contém os resultados obtidos pelo algoritmo (BCI) apresentado neste trabalho. Os valores apresentados são resultados obtidos da média dos RMSE de 10 diferentes conjuntos, independentes de dados.

Tabela 3. Comparação do RMSE nos 10 conjuntos de teste

Bases	BCI	Model Tree	Bagging	ANN	AdaBoost.R	AdaBoost.RT
CPU	**22,5**	34,7	32,6	**13,9**	24,5	26,5
Housing	**1,12**	3,62	3,24	3,54	3,23	3,23

Auto-Mpg	**0,85**	3,01	2,86	3,79	2,84	2,96
Friedman#1	**0,7**	2,19	2,06	1,51	1,82	1,72

Como se pode observar na tabela (3), o algoritmo *BCI* apresenta o menor *RMSE* dos conjuntos de teste, exceto para a base de dados CPU, na qual o desempenho das redes neurais foi melhor, porém apresenta resultado melhor quando, comparado aos demais algoritmos de *Boosting*, mostrando uma redução de erros que varia entre 50% e 70%. Para analisar o desempenho relativo dos algoritmos, é apresentada na tabela 4 o desempenho médio relativo (em percentual) de uma técnica em relação às demais para todos os conjuntos de dados considerados. A comparação apresentada na tabela 4 é chamada de *scoring-matrix*, onde o elemento SM_{ij} representa o desempenho médio do i-ésimo algoritmo (cabeçalho da tabela 4) em relação ao j-ésimo algoritmo (coluna 1 da tabela 4) calculado para N conjuntos de dados da de acordo com a equação 14 e para $i \neq j$.

$$SM_{ij} = \frac{1}{N}\sum_{k=1}^{N}\frac{RMSE_{kj} - RMSE_{ki}}{max\left(RMSE_{kj}, RMSE_{ki}\right)} \qquad (14)$$

Tabela 4. *Scoring Matrix* para os diferentes algoritmos apresentados

Algoritmo	BCI	Model Tree	Bagging	ANN	AdaBoost.R2	AdaBoost.RT
BCI	0	61	58,2	40,4	51,2	52,8
Model Tree	-61	0	-6,8	-18,1	-11,5	-14,3
Bagging	-58,2	6,8	0	-12,8	-9,4	-8
ANN	-40,4	18,1	12,8	0	6,6	7,3
AdaBoost.R2	-51,2	11,5	9,4	-6,6	0	1,6
AdaBoost.RT	-52,8	14,3	8	-7,3	-52,8	0

Como se pode observar na tabela 4, os valores apresentados pelo algoritmo BCI são superiores aos apresentados pelos demais algoritmos, mostrando assim um melhor desempenho do algoritmo na tarefa de regressão.

4. Conclusões e trabalhos futuros

Neste artigo foi apresentado uma nova versão do algoritmo *Adaboost*, utilizando os coeficientes de correlação entre os valores observados e os valores previstos. Este coeficiente de correlação foi utilizado na atualização dos pesos e na combinação final dos preditores obtidos por diversas execuções de um algoritmo "base". A abordagem do algoritmo *Adaboost* para problemas de regressão é recente e tem sido foco de investigação de muitos pesquisadores já que os resultados obtidos com sua utilização têm sido melhores do que os obtidos através da utilização de um único preditor. O algoritmo proposto BCI foi implementado utilizando o *software* estatístico R. As bases de dados utilizadas foram as mesmas utilizadas por Solomatine [Solomatine & Shrestha 2004]. Os resultados foram comparados aos resultados obtidos através dos algoritmos de Redes Neurais Artificiais, *Model Tree*, *Bagging* e os algoritmos de *Boosting*: *Adaboost.R2* e Adaboost.*RT* e mostraram a superioridade do algoritmo BCI na maioria dos casos. A vantagem de se utilizar a metodologia proposta, além de apresentar menores taxas de erro em relação a outros algoritmos é a possibilidade de utilizar um algoritmo fraco para obter as hipóteses, o que reduz drasticamente o tempo computacional. Outros resultados obtidos com a utilização do algoritmo BCI podem ser encontrados em

[Souza, Pozo & Chaves 2006]. Em um próximo estudo, pretende-se utilizar RNA como algortimo base, já que o mesmo apresenta bons resultados.

Referências

Allwein, E. L, Schapire, R. E. and Singer, Y. (2000). "Reducing multiclass to binary: A unifying approach for margin classifiers". Journal of Machine Learning Research, v1, p. 113-141.

Blake, C. L. & Merz, C. J. (1997). "UCI Repostitory of machine learning databases", Irvine, CA: Universtity of California, Dep. of Information and Computer Science, 1998. Available: http://www.ics.uci.edu/~mlearn/MI.Repository.html.

Breiman, L. & Friedman, J. (1984). "Classification and Regression Trees". New York: Belmont Wadsworth Int. Group.

Breiman, L. (1997). "Prediction Games and Arcing Algorithms". Neural Computation, vol 11 (7), pp 1493-1518.

Drucker, H. (1997). "Improving regression using boosting techniques", In: International Conference on Machine Learning, Orlando: Proceeding of International Conference on Machine Learning, ICML97.

Duffy, N. & Helmbold, D. P. (2000), "Leveraging for regression". Procedures of the 13th Annual Conference on Computer Learning Theory, p. 208-219, San Francisco: Morgan Kaufmann.

Efron, B., Tibshirani, R. J. (1993). "An introduction to the bootstrap". Chapman & Hall, New York, NY.

Freund Y.; Schapire, R.E. (1998). "A short Introduction to Boosting". Journal of Japanese Society for Artificial Intelligence, v.14(5), p. 771

Friedman, J.; Hastie, T.; Tibshirani. (2001). "Greedy Function Approximation: A Gradient Boosting Machine". In: Annals of Statistics, v.29(5), p. 1189-1232.

Hastie, T.; Tibshirani, R.; Friedman, J. (2001). "The elements of Statistical Learning". Springer Science+Business Media. New York, 10013, USA.

Ridgeway, G. Madigan, D. & Richardson, T. (1999). "Boosting methodology for regression problems". Procedure of 7th International Workshop on Artificial Intelligence and Statistics, p. 152-161. Fort Lauderdale, FL.

Schapire, R. E. (1990). "The Strenght of weak learnability", In: Machine Learning, p. 197-227.

Schapire, R. E. (1998). "Improved boosting algorithms using confidence rated predictions", In: Proceedings of the Eleventh Annual Conference on Computational Learning Theory, p.80 - 91.

Schapire, R. E. and Freund, Y. (1996) "Experiments with a new boosting algorithm", In: Machine Learning: Proceedings of the 13th International Conference, 1996, p.148-156

Solomatine, D. P., Shrestha, D. L. (2004). "Adaboost.RT: a Boosting Algorithm for Regression Problems". IEEE, p. 1163-1168.

Souza, L. V., Pozo, A. T. R & Chaves Neto, A (2006). "Using Correlation to Improve Boosting Technique:An Application for Time Series Forecasting", In: Proceedings of 18th IEEE International Conference on Tools with Artificial Inteligence.

Zemel, R. & Pitassi, T. (2001). **A gradient-based boosting algorithm for regression problems**. Leen, T. K. Dietterich, T. G., & Tresp, V. (Eds.), Advances in Neural Information Processing Systems 13. MIT press.