Pós-Graduação em Ciência da Computação

Valter Eduardo da Silva Júnior

UMA NOVA ABORDAGEM DO REAL ADABOOST RESISTENTE A *OVERFITTING* PARA CLASSIFICAÇÃO DE DADOS BINÁRIOS

Dissertação de Mestrado

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

RECIFE
2016

Valter Eduardo da Silva Júnior

UMA NOVA ABORDAGEM DO REAL ADABOOST RESISTENTE A *OVERFITTING* PARA CLASSIFICAÇÃO DE DADOS BINÁRIOS

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Renata Maria Cardoso Rodrigues de Souza*
Co-Orientador: *Getúlio José Amorim do Amaral*

RECIFE
2016

Catalogação na fonte
Bibliotecário Jefferson Luiz Alves Nazareno CRB 4-1758

S586n Silva Júnior, Valter Eduardo da.
Uma nova abordagem do real adaboost resistente a overfitting para classificação de dados binários / Valter Eduardo da Silva Júnior – 2016.
58f.: fig., tab.

Orientadora: Renata Maria Cardoso Rodrigues de Souza
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, Recife, 2016.
Inclui referências.

1. Inteligência artificial. 2. Algorítimos computacionais. 3. Classificação. I. Souza, Renata Maria Cardoso Rodrigues de. (Orientadora). II. Titulo.

006.3 CDD (22. ed.) UFPE-MEI 2016-160

Valter Eduardo da Silva Júnior

Uma nova abordagem do Real AdaBoost resistente a overfitting para classificação de dados binários

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 26/08/2016

BANCA EXAMINADORA

_____

Prof. Dr. Sergio Ricardo de Melo Queiroz
Centro de Informática / UFPE

_____

Profa. Dra. Cláudia Regina Oliveira de Paiva Lima
Departamento de Estatística/UFPE

_____

Profa. Dra. Renata Maria Cardoso Rodrigues de Souza
Centro de Informática /UFPE
(**Orientadora**)

Dedico esta dissertação a toda minha família,
companheiros de estudo e professores que me deram apoio
durante esta caminhada.

Agradecimentos

Muitas foram as pessoas que estiveram comigo e contribuiram de forma direta e indireta para a realizações deste trabalho. A todas elas, muito obrigado.

Obrigado a Profa. Dra. Renata Maria Cardoso Rodrigues de Souza, por toda dedicação e comprometimento como orientadora, sempre disposta a ajudar, e tambem pela confiança e amizade do coorientador Prof.Dr. Getúlio José Amorim do Amaral pelas ajudas nas contribuições que este trabalho fosse concluido com êxito.

Agradeço à Universidade Federal de Pernambuco por me proporcionar a oportunidade de desenvolver parte dessa pesquisa em sua universidade.

Agradeço também ao meu colega de laboratório, Daniel Bion Barreiros, pelas valiosas ajudas durante as etapas experimentais e pelos conhecimentos compartilhados.

Aos meus pais Valter Eduardo da Silva e Valéria Maria de Luna e irmãos pela educação, carinho e apoio de sempre.

À minha esposa Cinthia Fernandes Alves e Silva pelo carinho e compreensão no dia a dia.

A especialmente a minha filha Sophia Fernandes Alves e Silva, que me deu mais motivação, animo e felicidade na minha vida.

Por fim, ao CNPq (Conselho Nacional de Desenvolvimento CIentífico e Tecnológico) pelo apoio financeiro deste trabalho, por meio da bolsa de mestrado concedida.

Às vezes, quando você inova, você comete erros. É melhor admiti-los rapidamente, e continuar a melhorar as suas outras inovações.

—STEVE JOBS

Resumo

O estudo da Inteligência Artificial (IA) e de suas técnicas tem trazido grandes resultados para a evolução da tecnologia em diversas áreas. Técnicas já conhecidas como as Redes Neurais (RN) e Árvores de Decisão (AD) vêm sendo aprimoradas por técnicas de Boosting como o Adaptive Boosting (AdaBoost). Esta técnica é uma das que apresenta maior perspectiva de crescimento devido a seu potencial, flexibilidade e simplicidade para ser implementada em diferentes cenários, como por exemplo, no tratamento para reconhecimento de padrões. Desde o seu surgimento surgiram várias variantes do AdaBoost, as mais conhecidas são o Real AdaBoost (RAb) e Gentle AdaBoost (GAb), no intuito de obter um melhor desempenho. Um problema peculiar do Real AdaBoost é relacionado a base de dados com ruído. Vários artigos sugerem que o Real AdaBoost é sensível a um nível alto de ruído. A partir desse problema será proposto uma nova variante do Real AdaBoost com o objetivo de reduzir esta sensibilidade ao ruído visto ao aparecimento de overfitting nas bases de testes dos problemas com ruído. A nova variante do Real Adaboost será chamada de Noise Real AdaBoost (NRAb), onde será aplicada em base de dados simuladas e reais e será utilizado Real AdaBoost e Gentle AdaBoost para comparar o desempenho em relação a nova variante proposta nesta dissertação.

Palavras-chave: Algoritmos de Boosting. Classificação. Overfitting. Aprendizado de máquinas. Erros fragmentários. Arvores de Decisão

Abstract

The study of Artificial Intelligence (AI) and its techniques have brought great results for the evolution of technology in various fields. Known techniques such as Neural Networks (RN) and Decision Trees (AD) have been enhanced by Boosting techniques such as Adaptive Boosting (AdaBoost). This technique is one that has greater prospect of growth potential due to its flexibility and simplicity to be implemented in different scenarios, such as in treatment for pattern recognition. Since its inception AdaBoost were several variants, the best known are the Real AdaBoost (RAB) and Gentle AdaBoost (GAB) in order to get better performance. A peculiar problem of Real AdaBoost is related to noise with database. Several articles suggest that Real AdaBoost is sensitive to a high noise level. From this problem a new variant of Real AdaBoost in order to reduce this sensitivity to noise seen the emergence of overfitting in the problems with noise test bases will be proposed. The new variant of the Real AdaBoost will be called Noise Real AdaBoost (NRAb), which will be applied to simulated and real data base and will be used Real AdaBoost and Gentle AdaBoost to compare performance against the new variant proposed in this dissertation.

Keywords: Boosting Algorithms. Classification. Overfitting. Learning Machines. Fragmentary Errors. Decision Trees

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

IA	Inteligência Artificial	7
AD	Árvores de Decisão	19
AdaBoost	Adaptive Boosting	7
CART	Classification and Regression Trees	19
GAb	Gentle AdaBoost	7
ID3	Induction of Decision Trees	19
NRAb	Noise Real AdaBoost	7
RAb	Real AdaBoost	7
RN	Redes Neurais	7
Bagging	Bootstrap Aggregating	23
SVM	Suport Vector Machines	13
RL	Regressão Logistica	13
AUC	Área da Curva ROC	55
UCI	University of California, Irvine	42

Sumário

1

Introdução

Os computadores vêm evoluindo significativamente nos últimos anos, seus recursos de processamento, ganhando novas habilidades, bem como aumento de capacidade de processamento e possibilidades de interagir com os humanos. A área de classificação na área da computação tem sido uma área muito pesquisada, sendo vista como uma etapa inicial de muitas aplicações e técnicas de diversos tipos, como Suport Vector Machines (SVM), Regressão Logistica (RL), Rede Neurais, entre outras.

Dada uma base de dados com suas variáveis explicativas, um classificador ideal deve ser capaz de identificar as classes do alvo com o menor erro possível. Nesse sentido, os algoritmos de maior sucesso nesta área são os métodos baseados no processo iterativo, sendo o AdaBoost uma das técnica mais utilizada e que será abordada nesta dissertação. O algoritmo AdaBoost foi um dos primeiros algoritmos de Boosting a ser amplamente usado em sistemas de tempo real, por causa da sua simplicidade e adaptabilidade. A propriedade do algoritmo AdaBoost de maximizar as margens de classificação nem sempre conduz à melhor generalização. RäTSCH; ONODA; MüLLER. (1998), GROVE; SCHUURMANS. (1998a) e QUINLAN (1996) observam que o algoritmo AdaBoost, o qual cria uma combinação linear de classificadores, frequentemente apresenta um desempenho inferior que um classificador individual na presença de ruído, que são dados com erro aleatórios ou pontos aberrantes (*outliers*). Além disso, DIETTERICH. (2000), RäTSCH; ONODA; MüLLER. (2001) e SERVEDIO. (2003) mostraram que o algoritmo Adaboost pode conduzir ao sobreajuste (overfitting) das amostras, na presença de dados aleatoriamente ruidosos. HAWKINS. (2004) indica que o sobreajuste aparece quando o número de classificadores aumenta de forma excessiva, ou seja, produz-se uma combinação de classificadores muito complexa, o que leva a uma deterioração do desempenho.

Durante o processo de treinamento, o algoritmo AdaBoost modifica os pesos das amostras, focando naquelas que são classificadas incorretamente e atribui pesos maiores para elas. Este procedimento pode provocar efeitos indesejáveis quando as amostras são altamente ruidosas. Isto é evidenciado quando os classificadores possuem um bom desempenho com amostras de treinamento, no entanto, o desempenho é inferior com amostras de teste.

Neste trabalho, com o propósito de evitar os problemas de overfitting descritos anteri-

ormente apresentamos um algoritmo Real AdaBoost modificado. Em particular, visamos um desempenho melhorado na sua classificação final. Analisamos e comparamos a abordagem desenvolvida com outras abordagens convencionais para validar a metodologia proposta.

O objetivo principal deste trabalho é desenvolver um novo algoritmo, menos sensível ao ruído. O algoritmo desenvolvido será aplicado na classificação binária de bases de dados simulada e reais, evitando efeito do sobreajuste de amostras. Os estudos técnicos desenvolvidos foram:

- Investigar as abordagens relacionadas com o processo de atualização de pesos nos algoritmos de Real AdaBoost, a fim de equacionar possíveis soluções aos problemas de sobreajuste de amostras.
- Desenvolver e implementar um algoritmo de Real Boosting Adaptativo com maior resistência ao ruído, que atualize os pesos da amostras em função de novos fatores.
- Teste do método desenvolvido na tarefa de classificação binária.
- Avaliar o desempenho do algoritmo desenvolvido e compará-lo com os algoritmos Real AdaBoost e Gentle AdaBoost.

A presente dissertação é organizada em 4 capítulos e a descrição de cada um segue abaixo

- **Capítulo 2.** Apresentamos os métodos de ensemble, seus princípios e propriedades. Explicamos os principais algoritmos e suas variações. Apresentamos também algumas variações no AdaBoost e as dividimos de acordo com a atualização de pesos, maximização da margem e a importância atribuída ao classificador.
- **Capítulo 3.** Apresentamos uma descrição detalhada da abordagem desenvolvida nesta dissertação de Mestrado. Assim, introduzimos um algoritmo Real AdaBoost modificado com o propósito de superar os problemas de sobreajuste de amostras ruidosas.
- **Capítulo 4.** São apresentados os resultados dos experimentos realizados para comprovação do aperfeiçoamento obtido na abordagem proposta. Além disto, é proporcionado um estudo comparativo entre a abordagem proposta (descrita no Capítulo anterior) e uma seleção de trabalhos representativos, que foi apresentada na revisão bibliográfica (apresentada no Capítulo 2).
- **Capítulo 5.** Neste capítulo se apresentam as conclusões e contribuições do trabalho.

2

Métodos de *Ensemble*

Ensemble é uma palavra francesa que significa junto, comitê. A ideia geral é representar um conjunto de soluções combinadas para um único problema. Assim, ensemble é um conjunto de classificadores que tomam decisões individuais, mais que são combinados para classificar novas amostras. Um ensemble consiste em um comitê, por exemplo de classificadores, cujas hipóteses individuais são induzidas separadamente e as decisões referentes a cada hipótese são combinadas através de um método de consenso para a classificação de novos dados. O objetivo principal é aumentar a capacidade de generalização do modelo individual HANSEM; SALAMON (1990). Um classificador com acurácia é aquele que tem uma taxa de erro melhor que a atualização de uma adivinhação aleatória para novos valores de entrada. E dois classificadores são diversos quando eles cometem erros diferentes para um nova entrada DIETTERICH. (2000).

A Figura 2.1 representa o modelo de um Método *Ensemble*. Basicamente, vários classificadores individuais são instanciados e recebem um conjunto de características como entrada. Cada classificador realiza suas predições para os dados recebidos e gera algumas regras de aprendizado. A partir de então, um método de combinação desses resultados é aplicado e um novo classificador é formado.

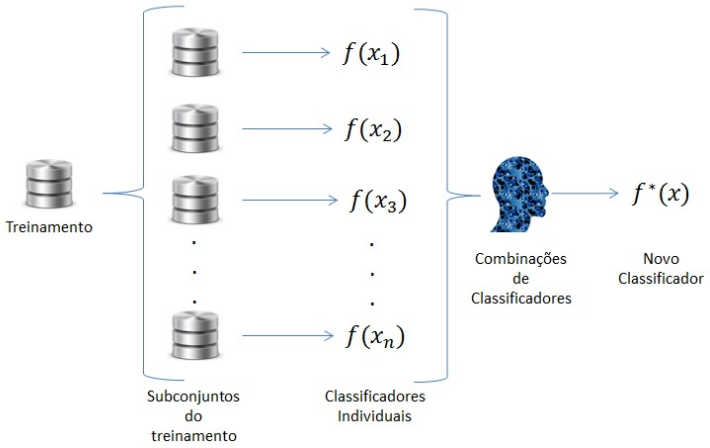


Figura 2.1 Modelo conceitual do aprendizado ensemble: vários conjuntos de dados são criados, para cada conjunto um classificador é treinado e então eles são combinados gerando um novo classificador.

Ao combinar soluções de classificadores independentes temos uma melhoria no resultado, uma vez que a combinação de estimativas diminui a variância e, o erro médio esperado de um conjunto de modelos é melhor que o erro médio esperado de cada um dos modelos. Métodos ensemble são indicados para serem utilizados em classificadores fracos, isto é, classficadores que realizam predições ligeiramente melhores que adivinhação.

2.1 Construindo um classificador *ensemble*

Existem muitos métodos para construir classificadores *ensembles*, as mais comuns são: enumeração de hipóteses, manipulação dos exemplos de treinamento, manipulação das características de entrada e manipulação dos objetivos finais DIETTERICH. (2000). Em nosso trabalho, utilizamos a abordagem de manipulação dos exemplos de treinamento por ser uma das mais comuns OPITZ; MACLIN (1999) e ZHOU; WU; TANG (2002) e também pode ser útil para tratar grandes volumes de dados, particionando os dados em sub-conjuntos e treinando classificadores com diferentes partições e então combinar as saídas.

Manipulação dos exemplos de treinamento Nesta abordagem os exemplos de treinamento são manipulados para gerar múltiplas hipóteses. O algoritmo de aprendizagem é executado várias vezes, cada vez com um subconjunto diferente dos exemplos de treinamento. Esta técnica funciona especialmente bem para algoritmos instáveis, ou seja, cuja classificação pode sofrer grandes alterações em razão de pequenas alterações nos dados de treinamento.

Os classificadores ensemble predizem as classes por meio de uma combinação dos votos dos classificadores: média, média ponderada, votação simples e ponderação das saídas ROKACH (2005).

As técnicas de combinações de classificadores citadas acima, mais utilizadas são:

- **Combinando classificadores com votação simples**: Neste tipo de abordagem, cada classificador individual faz uma predição e a opção mais votada é escolhida como resposta. Este tipo de combinação apesar de ser bem simples produz bons resultados. Seja um ensemble com 5 classificadores, para que uma amostra seja classifcada incorretamente é necessário que três dos cinco classificadores façam uma escolha errada. Se a probabilidade de erro de cada classificador individual for de 0.2, e se os classificadores cometem erros independentes, a probabilidade de erro do ensemble será de $0.2/5 = 0.04$. É fato, que na prática, os erros não serão totalmente independentes, uma vez que os classificadores serão gerados com a mesma base de treinamento. Mas, ainda assim, como são usadas para cada classificador um subconjunto da base de treinamento, os classificadores serão pelo menos um pouco diferentes, reduzindo a correlação dos erros.

- **Ponderação das saídas**: Considera que existem classificadores mais influentes na decisão final. De acordo com o nível de precisão de cada classificador individual,

cada predição recebe um maior peso.

2.1.1 Classificadores base

Os classificadores base são funções que, avaliando uma instância ou amostra, retornam um rótulo. Estes classificadores não necessariamente devem apresentar um desempenho ótimo. A seguir apresentam-se alguns dos classificadores base mais usados em aprendizado de máquina.

- **Rede Neural**

  As redes neurais artificiais consistem em um método de solucionar problemas de inteligência artificial, construindo um sistema que tenha circuitos que simulem o cérebro humano. De forma similar ao aprendizado humano, as redes neurais adquirem o conhecimento a partir do seu ambiente e o armazenam usando as conexões entre neurônios (pesos sinápticos) HAYKIN. (1998). São mais que isso, são técnicas computacionais que apresentam um modelo inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência. Uma grande rede neural artificial pode ter centenas ou milhares de unidades de processamento, enquanto que o cérebro de um mamífero pode ter muitos bilhões de neurônios.

  Apesar da complexidade da redes neurais não permitir uma única definição, as linhas seguintes seguem como uma tentativa das inúmeras definições ou interpretações do que seja realmente uma rede neural.

  Uma rede neural é uma estrutura de processamento de informação distribuída paralelamente na forma de um grafo direcionado, com algumas restrições e definições próprias. Um grafo direcionado é um objeto geométrico que consiste de um conjunto de pontos, chamados nós, ao longo de um conjunto de segmentos de linhas direcionadas entre eles.

  Os nós deste grafo são chamados elementos de processamento. Suas arestas são conexões, que funcionam como caminhos de condução instantânea de sinais em uma única direção, de forma que seus elementos de processamento podem receber qualquer número de conexões de entrada. Estas estruturas podem possuir memória local, e também possuir qualquer número de conexões de saída desde que os sinais nestas conexões sejam os mesmos. Portanto, estes elementos tem na verdade uma única conexão de saída, que pode dividir-se em cópias para formar múltiplas conexões, sendo que todos carregam o mesmo sinal como ilustra a Figura 2.2.

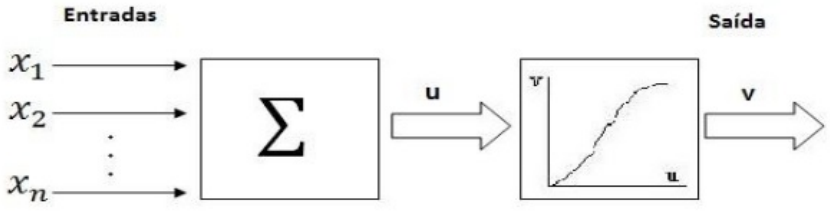


Figura 2.2 Representação de um neurônio artificial.

A rede neural artificial é um sistema de neurônios ligados por conexões sinápticas e dividido em neurônios de entrada, que recebem estímulos do meio externo, neurônios internos ou hidden (ocultos) e neurônios de saída, que se comunicam com o exterior. A forma de arranjar perceptrons em camadas é denominado *Multilayer Perceptron*. O multilayer perceptron foi concebido para resolver problemas mais complexos, os quais não poderiam ser resolvidos pelo modelo de neurônio básico. Um único perceptron ou uma combinação das saídas de alguns perceptrons poderia realizar uma operação XOR, porém, seria incapaz de aprendê-la. Para isto são necessárias mais conexões, os quais só existem em uma rede de perceptrons dispostos em camadas. Os neurônios internos são de suma importância na rede neural pois provou-se que sem estes torna-se impossível a resolução de problemas linearmente não separáveis. Em outras palavras pode-se dizer que uma rede é composta por várias unidades de processamento, cujo funcionamento é bastante simples. Essas unidades, geralmente são conectadas por canais de comunicação que estão associados a determinado peso. As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. O comportamento inteligente de uma Rede Neural Artificial vem das interações entre as unidades de processamento da rede.

A maioria dos modelos de redes neurais possui alguma regra de treinamento, onde os pesos de suas conexões são ajustados de acordo com os padrões apresentados. Em outras palavras, elas aprendem através de exemplos. Arquiteturas neurais são tipicamente organizadas em camadas, com unidades que podem estar conectadas às unidades da camada posterior.

A Figura 2.3 ilustra bem a estrutura do Multilayer Perceptron

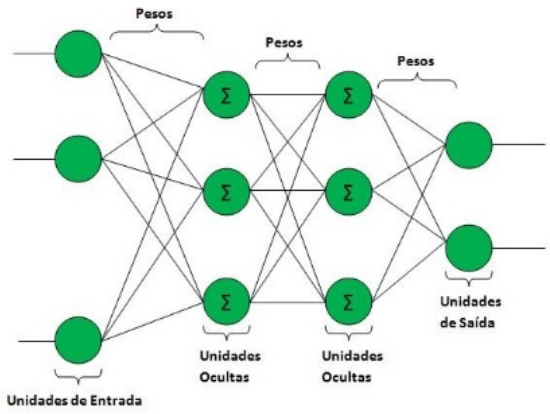


Figura 2.3 Estrutura Multilayer Perceptron (Multiplas Camadas).

A rede neural passa por um processo de treinamento a partir dos casos reais conhecidos, adquirindo, a partir daí, a sistemática necessária para executar adequadamente o processo desejado dos dados fornecidos. Sendo assim, a rede neural é capaz de extrair regras básicas a partir de dados reais, diferindo da computação programada, onde é necessário um conjunto de regras rígidas pré-fixadas e algoritmos.

- **Árvores de Decisão (Decision Tree)**

  Em geral, os algoritmos de indução de Árvores de Decisão (AD) podem ser entendidos como *Top Down Induction of Decision Trees*. Uma arvore de decisão é uma estrutura de dados definida recursivamente como:

  - um nó folha que corresponde a uma classe:
  - ou um nó de decisão, que contém um teste sobre algum atributo. Para cada resultado do teste, existe uma aresta para uma subárvore. Cada subárvore tem a mesma estrutura da árvore.

  A indução de árvores de decisão tem sido aprimorada há algum tempo. Na década de 1960, HUNT; STONE; MARIN (1966) usaram métodos de busca exaustiva em árvores de decisão para modelar o aprendizado de conceitos humanos. Já na década de 1978, foi publicado o trabalho de QUINLAN (1979) com Induction of Decision Trees (ID3). Na década de 1980, houve a primeira publicação em massa, por BREIMAN et al. (1984) do *software* Classification and Regression Trees (CART) (presente atualmente em vários produtos comerciais), justamente com o artigo de QUINLAN (1986) sobre ID3. Desde então, uma variedade de melhorias têm sido incorporadas às arvores de decisão, tais como tratamento de ruído, tratamento de atributos contínuos e ausente, árvores oblíquas (não paralelas aos eixos) e heurísticas de controle de *overfitting*.

O indutor ID3 pode ser considerado um algoritmo básico para a construção de árvore de decisão sem poda, na qual é conduzida uma busca gulosa (*greedy*), ou seja, o algoritmo não considera escolhas anteriores QUINLAN (1986). O algoritmo básico de construção de uma árvore de decisão é bem simples: utilizando o conjunto de treinamento, um atributo é escolhido de forma a particionar os exemplos em subconjuntos, de acordo com valores desse atributo. Para cada subconjunto, outro atributo é escolhido para particionar novamente cada um deles. Cada escolha de atributo representa um teste realizado em um nó interno da árvore, ou seja, cada nó interno executa um teste em apenas um atributo e tem dois ou mais ramos, cada um representando um possível resultado do teste. Este processo prossegue enquanto um dado subconjunto criado contenha uma mistura de exemplos com relação aos rótulos de classe. Uma vez obtido um subconjunto uniforme todos os exemplos naquele subconjunto pertencem à mesma classe, sendo rotulado com o mesmo nome da respectiva classe. Um novo exemplo é rotulado da seguinte maneira: começando do nó raiz, testes são realizados e, de acordo com seus resultados, o exemplo caminha para baixo na árvore até encontrar um nó folha, recebendo, então, seu rótulo.

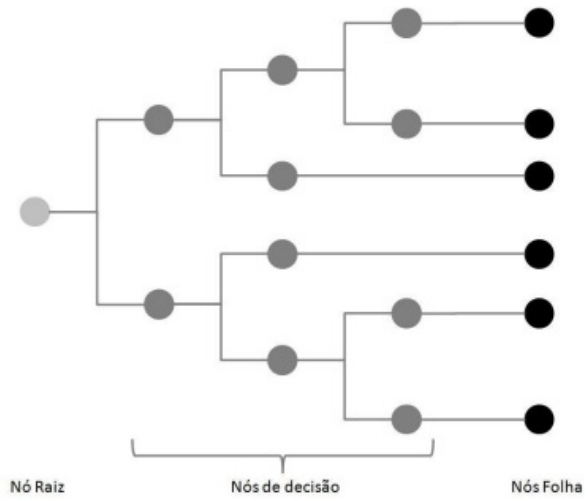


Figura 2.4 Modelo de árvore de decisão.

- ***Decision Stump*** *Decision Stump* são Árvores de Decisão que consistem de apenas duas folhas, ou seja, um nível apenas de árvore. Assim cada classificador fraco gerado por ele é capaz de analisar apenas uma decisão do conceito geral do problema analisado.

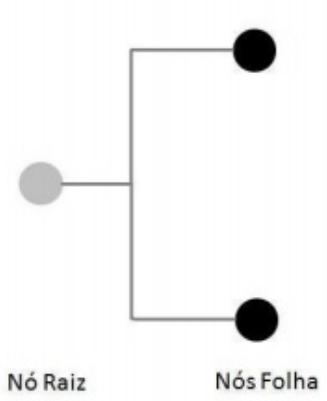


Figura 2.5 Modelo decision stump.

É considerado um dos mais simples algoritmos fracos e bastante utilizado em estudos com o método Boosting. Portanto, essa análise simplificada que o algoritmo é capaz de realizar pode não ser tão eficiente para solucionar problemas muito complexos ou com muitos parâmetros analisados ou até mesmo ocorrer problemas de underfitting durante a etapa de treinamento. Por outro lado, sua simplicidade implica em classificações mais rápidas.

2.2 Medindo desempenho de um classificador

Dado um classificador construído com base na amostra de treinamento $L = \{x_i; y_i\}_{i=1}^{N}$, a quantidade ε é a probabilidade de ser fazer uma classificação errada em uma amostra independente da que foi usada. Essa probabilidade também é conhecida com taxa de erro (error rate), e é uma boa medida do desempenho de um classificador. Outra maneira de pensar na taxa de erro é a seguinte: dada a amostra $L = \{x_i; y_i\}_{i=1}^{N}$, construa um classificador d. Numa nova amostra (independente), a taxa de erro ε é a proporção de objetos classificados erroneamente por d. Existem algumas maneiras de estimar ε:

- Usando um conjunto independente, chamado de conjunto de teste
- Usando Validação Cruzada

2.2.1 Estimação por Amostra de Teste

A estimação por amostra de teste consiste em dividir o conjunto de dados de treinamento $L = \{x_i; y_i\}_{i=1}^{N}$ em duas amostras independentes e disjuntas, amostras L_1, de tamanho N_1, e L_2 de tamanho N_2, onde $N_1 + N_2 = N$. Usamos L_1 para construir o classificador d(ou seja, L_1 é agora a amostra de treinamento) e o testamos na amostra L_2, que funciona agora como amostra de teste. A estimativa de ε é então

$$\hat{\varepsilon}(d) = \frac{1}{N_2} \sum_{i=1}^{N_2} I(d(x_i) \neq y_i) \tag{2.1}$$

BREIMAN et al. (1984) sugere que os N_2 objetos em L_2 sejam escolhidos aleatoriamente da amostra de treinamento original. Assim, na equação 2.1 a soma é sobre os (x_i, y_i) em L_2. Não há um consenso sobre quão grande deve ser N_2, mas geralmente usa-se menos da metade do total de casos (por exemplo, $N_1 = \frac{2}{3}N$ e $N_2 = \frac{1}{3}N$).

Em amostras grandes, o procedimento acima é razoável, mas quando N é pequeno, perde-se muita informação, já que as N_2 observações em L_2 não são usadas para construir o classificador. RIPLEY (1996) usa o fato que $R = \sum_{i=1}^{N_2} I(d(x_i) \neq y_i) \sim B(N_2, \varepsilon)$ para mostrar que, para estimar ε com alguma eficiência, N_2 deve ser relativamente grande. Isso nos leva à segunda forma de estimar ε, conhecida como validação cruzada.

2.2.2 Estimação por validação Cruzada

Considere o procedimento de estimar ε dado anteriormente, usando a divisão do conjunto de treinamento em dois conjuntos L_1 e L_2. Podemos usar L_1 para construir os classificador d e estimar ε aplicando d em L_2. Chame esta estimativa de $\hat{\varepsilon_1}$. Agora fazendo o oposto, ou seja, usando L_2 para construir d e testando-o em L_1, obtemos a estimativa $\hat{\varepsilon_2}$. Se combinarmos essas estimativas, obteremos um estimador $\hat{\varepsilon} = \frac{1}{2}(\hat{\varepsilon_1} + \hat{\varepsilon_2})$, que é não viesada, ou seja, o valor esperado $E(\hat{\varepsilon}) = \varepsilon$.

A estimação de ε por validação cruzada consiste em expandir o conceito acima da seguinte maneira. Suponha que o conjunto original de tamanho N seja dividdo em V partes, ou seja, temos $L_1, L_1, ..., L_V$, cada um com tamanho N_v. Para cada parte v, construa o classificador $d^{(v)}$ usando as $v-1$ partes e teste na restante. Então a estimativa de ε usando a parte v pode ser calculada por 2.1, substituindo N_2 por N_v, $v = 1, 2, ..., V$ e fazendo a soma sobre os casos em N_v. Então a estimatia de ε por validação cruzada é dada por

$$\hat{\varepsilon} = \frac{1}{V} \sum_{v=1}^{V} \varepsilon(d^{(v)}) \tag{2.2}$$

Esta estimativa é novamente não viesada.

O caso $V = N$ é conhecido como leave-one-out, pois em cada passo, usamos $N-1$ observações para construir o classificador, e testar na observação restante. É interessante notar que, na estimação por validação cruzada, todos os casos são usados para construir o classificador, e cada caso é usado exatamente uma vez para testar o classificador, fato que é colocado em BREIMAN et al. (1984). A desvantagem deste método é computacional, pois é necessário muito mais tempo de computação para executar este processo.

A estimação da taxa de erro por validação cruzada não deve ser confundida com a técnica de validação cruzada para seleção de modelos. Na última, calcula-se uma medida de bondade de ajuste baseada em amostras por validação cruzada, e procura-se o modelo que maximize este

critério.

2.3 Bagging

Publicado no meio acadêmico pela primeira vez por BREIMAN (1996), o método Bootstrap Aggregating (Bagging) tem o propósito de combinar preditores gerados por um mesmo algoritmo base com o objetivo de reduzir a variância de funções preditivas. Conhecido popularmente por Bagging, o método apresenta bons resultados em algoritmos base instáveis como rede neurais e, principalmente, árvores de decisão CREAMER; FREUND. (2005). O procedimento é classificado como instável se pequenas mudanças nos dados acarretam em grandes alterações nos classifcadores e na previsão final.

No caso de classificação, os preditores são combinados através de voto, ou seja, a classe mais votada é a escolhida, e no caso de regressão, usa-se comumente a média dos preditores. A partir de seu surgimento, houve muito interesse em explicar como bagging funciona. Em BREIMAN (1996), houve uma primeira tentativa, através de uma argumentação de que o preditor fornecido pelo bagging teria menor erro quadrático médio, no caso de regressão. BREIMAN. (1998) decompõe a taxa de erro como soma das variâncias e mostra empiricamente (através de dados simulados) que o bagging atua reduzindo violentamente a variância de um classificador instável. FRIEDMAN; HALL (2000) estudaram a decomposição de estimadores em partes lineares e de ordem superior para argumentar que bagging funciona reduzindo a variabilidade da componente não linear, deixando a parte linear intacta.

De todas as explicações para o funcionamento do bagging, o que é consenso é que este procedimento pode melhorar o desempenho de preditores instáveis, que são basicamente preditores com alta variância. BUJA; STUETZLE (2000) apontam o fato de que, em procedimentos estáveis (por exemplo, análise discriminante), o preditor via bagging não traz melhora e pode até ter desempenho ligeiramente inferior.

2.4 Boosting

O método Boosting surgiu na aprendizado de máquina. Dentro dessa área, foi lançado um problema teórico chamado de problema de Boosting, que pode ser informalmente exposto da seguinte maneira: "Suponha que existe um método de classificação que é ligeiramente melhor do que uma escolha aleatória, para qualquer distribuição em Ω. Esse método é chamado de classificador fraco (weak learner). A existência de um classificador fraco implica a existência de uma classificador forte (strong learner), com erro pequeno sobre todo o espaço Ω?".

Esse problema foi resolvido por SCHAPIRE. (1990), que mostrou que era possível obter um classificador forte a partir de um fraco. A partir de então, foram desenvolvidos vários algoritmos dentro do contexto de Boosting. Um dos mais recentes e bem sucedidos deles é o algoritmo conhecido como AdaBoost (*Adaptative Boosting*), que funciona pertubando a

amostra de treinamento gerando a cada iteração (de forma determinística, mas adaptativa) uma distribuição sobre as observações da amostra, dando maior peso (maior probabilidade de estar na amostra pertubada) às observações classificadas erroneamente no passo anterior. Existe um outro método de combinação de preditores, conhecido por Bagging (*Bootstrap Aggregating*), que funciona pertubando essa amostra de treinamento aleatoriamente por meio de re-amostragem, gerando a cada iteração um classificador e o classificador final é obtido pela agregação desses classificadores SCHAPIRE; FREUND. (1999a).

Desde o seu desenvolvimento como uma resposta a um problema teórico, os algoritmos do tipo Boosting têm recebido grande atenção, tanto na comunidade estatística quanto na de Aprendizagem de Máquina. A comunidade estatística busca entender como e por que Boosting funciona, abordando aspectos como consistência, enquanto na comunidade de aprendizagem de máquina a abordagem é mais focada nos próprios algoritmos e em sua funcionalidade RUBESAM (2004).

O algoritmo AdaBoost é o mais famoso dos algoritmos de Boosting, e foi apresentado por FREUND; SCHAPIRE. (1996). Os autores fizeram uma análise do algoritmo em termos de limites para as probabilidades de erro na amostra de treinamento e nas amostras de teste (o erro de um classificador em casos novos é chamado, na literatura de aprendizagem de máquinas, de erro de generalização).

Um dos limites teóricos mostrados implica que o erro na amostra de treinamento decai exponencialmente com o número de iterações do algoritmo. Empiricamente, observa-se que, após algumas iterações, o erro na amostra de treinamento cai a zero, confirmado o resultado teórico.

Inicialmente, observou-se que, quando se continua a executar o algoritmo AdaBoost, o erro na amostra teste continua a decrescer, indicando que o algoritmo é resistente a super ajuste (*overfitting*) BUKLMANN; HOTHORN (2007). O super ajuste é o problema que surge quando um modelo tem desempenho bom no conjunto de treinamento, mas em dados novos, que não foram usados no ajuste do modelo, tem desempenho ruim. Isso ocorre geralmente porque o modelo se torna complexo demais (ou seja, número excessivo de parâmetros) e passa a ajustar peculiaridades do conjunto de treinamento. Por exemplo, em regressão logística, a adição de variáveis sempre melhora o desempenho no conjunto usado para estimar o modelo, mas em algum ponto isso começa a se tornar prejudicial e o desempenho em um conjunto de teste é ruim. Em redes neurais, se o algoritmo de otimização é executado indefinidamente, o erro sempre diminui no conjunto, mas em certo ponto ele começa a aumentar no conjunto de teste. Existem métodos para determinar o ponto de parada nesse caso, como por exemplo o método de parada precoce (*early stopping*), que cessa a otimização quando o erro começa a aumentar no conjunto de teste.

FRIEDMAN; HASTIE; TIBSHIRANI (2001) mudaram totalmente o modo como Boosting é visto, pelo menos na comunidade estatística. Eles colocaram Boosting como uma aproximação do ajuste de um modelo aditivo na escala logística, usando a máxima verossimi-

lhança da Bernoulli como critério. Sugeriram uma aproximação mais direta, o que levou ao algoritmo LogitBoost, um algoritmo para ajustar a regressão logística aditiva que dá resultados praticamente idênticos ao AdaBoost de FREUND; SCHAPIRE. (1995).

Mais recentemente, notou-se que, se um algoritmo de Boosting for executado por um tempo (números de iterações) muito grande, da ordem de dezenas de milhares, isso ocasionará super ajuste. FRIEDMAN; HASTIE; TIBSHIRANI (2000) dá um exemplo em que isso ocorre. Algumas abordagens para esse problema foram tentadas. JIANG (2000) mostrou que, sob certas condições de regularidade, como o número ideal de iterações, o algoritmo AdaBoost é consistente em processo, no sentido de que, durante o treinamento, ele gera um sequência de classificadores com erro que converge para o erro do classificador (regra) de Bayes.

2.4.1 Adaboost

O algoritmo Adaboost para classificação binária é o algoritmo boosting mais conhecido. O classificador base retorna valores em $\{-1,+1\}$ e pode ser, por exemplo, uma árvore de regressão ou uma rede neural. Será apresentado a seguir a versão desse algoritmo, dada em FRIEDMAN; HASTIE; TIBSHIRANI (2001).

Suponha que temos um conjunto de treinamento $L = (x_1,y_1),...,(x_N,y_N)$, onde as classes estão rotuladas $\{-1,+1\}$, ou seja, $C = \{-1,+1\}$. Defina $F(x) = \sum_{m=1}^{M} c_m f_m(x)$, onde M é o número de vezes que o algoritmo é executado (iterações), f_m é um classificador base que retorna valores $\{-1,+1\}$, os valores c_m são constantes e a predição correspondente a cada valor de x é a função sinal de $F(x)$, ou seja, $sign(F(x))$. A função $sign(.)$ retorna 1 se $sign(.) > 0$ e retorna -1 se $sign(.) < 0$. O algoritmo Adaboost ajusta classificadores base f_m em amostras ponderadas do conjunto de treinamento, dando maior peso, ou ponderação, aos casos que são classificadores erroneamente. Os pesos são ajustados adaptativamente em cada ietração e o classificador final é uma combinação linear dos classificadores f_m.

O algoritmo AdaBoost consiste em três passos:

1. Dado $(x_1,y_1),...,(x_N,y_N)$ em que $x_i \in \mathbf{X}$ e $y_i \in Y = \{-1,+1\}$. Incialize os pesos $w_i^m = 1/N$, $i = 1,2,...,N$.

2. Repita para $m = 1,2,...,M$:

(a) Ajuste o classificador $f_m(x) \in \{-1,+1\}$ usando os pesos w_i e os dados de treinamento;

(b) Calcule

$$\begin{aligned} e_m &= \frac{\sum_{i=1}^{N} w_i^m I[Y_i \neq f_m(X_i)]}{\sum_{i=1}^{n} w_i^m} \\ c_m &= \frac{1}{2} ln\left(\frac{1-e_m}{e_m}\right) \end{aligned} \tag{2.3}$$

(c) Faça

$$w_i^{m+1} = \frac{w_i^m}{z^m} \times \begin{cases} e^{-c_m}, & \text{se} \quad y_i = f_m(x_i) \\ e^{c_m}, & \text{se} \quad y_i \neq f_m(x_i) \end{cases} \tag{2.4}$$

em que z^m é um fator de normalização

$$z^m = \sum_{i=1}^{N} w_i^m e^{(-c_m y_i f_m(x_i))} \tag{2.5}$$

3. A predição é dada por $sign(F(x)) = sign(\sum_{m=1}^{M} c_m f_m(x))$.

No algoritmo acima, e_m representa a média ponderada dos erros com pesos $w = (w_1, ..., w_n)$. Em cada iteração, o algoritmo aumenta os pesos w_i das observações classificadas erroneamente por um fator que depende dos erros e_m das observações do conjunto de treinamentoo (passo 2(c)).

FRIEDMAN; HASTIE; TIBSHIRANI (2000) mostram que o algoritmo AdaBoost pode ser derivado como algoritmo iterativo para ajustar um modelo aditivo logístico, otimizando um critério que até segunda ordem é equivalente à log-verossimilhança da binomial. Os proxímos tópicos irão mostrar as variantes do Adaboost que será utilizado nesta dissertação.

2.4.1.1 Treinamento do AdaBoost

Nesta seção apresentamos o procedimento de treinamento no algoritmo AdaBoost. Com propósitos de ilustração, consideramos apenas 3 classificadores desempenhando uma classificação binária. Nesta representação consideramos dois tipos de amostras, representadas por círculos, onde o tamanho de cada símbolo indica o peso atribuído a cada amostra. Inicialmente, todos os pesos possuem o mesmo valor, então temos símbolos (quadrados e triângulos) possuem do mesmo tamanho, como se apresenta na Figura 2.6.

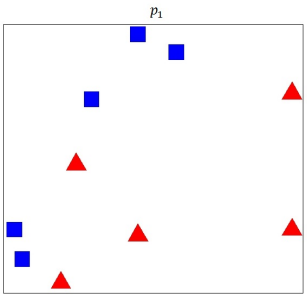


Figura 2.6 Amostras de Treinamento.

Após a primeira iteração, os pesos das amostras classificadas incorretamente são aumentados, visando que o classificador seguinte foque nestas amostras (Figura 2.7). Para dar maior ênfase, os tamanhos dos símbolos são escalados de acordo com sua classificação. Assim, as amostras com pesos maiores, produto de uma classificação incorreta, possuem símbolos maiores, no entanto, as amostras classificadas corretamente possuem símbolos menores.

Com o objetivo de minimizar a taxa de erro, o ajuste dos pesos obriga ao seguinte classificador focar nas amostras com pesos maiores. Após várias iterações, o algoritmo de Boosting

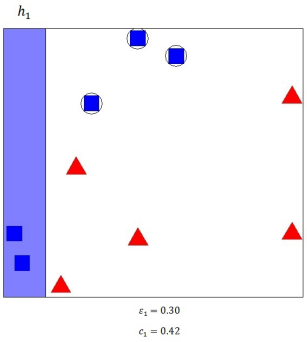
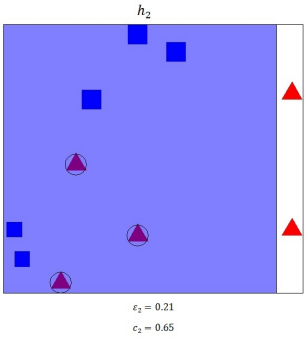
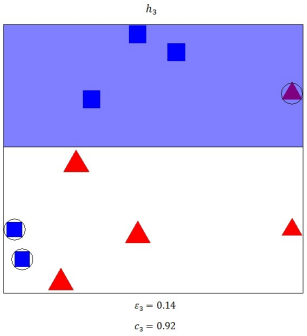


Figura 2.7 Primeira curva de separação.

combina esses classificadores fracos num único classificador, chamado de forte, esperando maior precisão que cada um dos classificadores fracos. Neste momento, os valores de c_m são computados, atribuindo uma importância a cada classificador, e proporcionando uma maior influência aos melhores classificadores, como se apresenta na Figura 2.8a.



(a) Segunda curva de separação.



(b) Terceira curva de separação.

Figura 2.8 Vista Geral do treinamento do algoritmo AdaBoost.

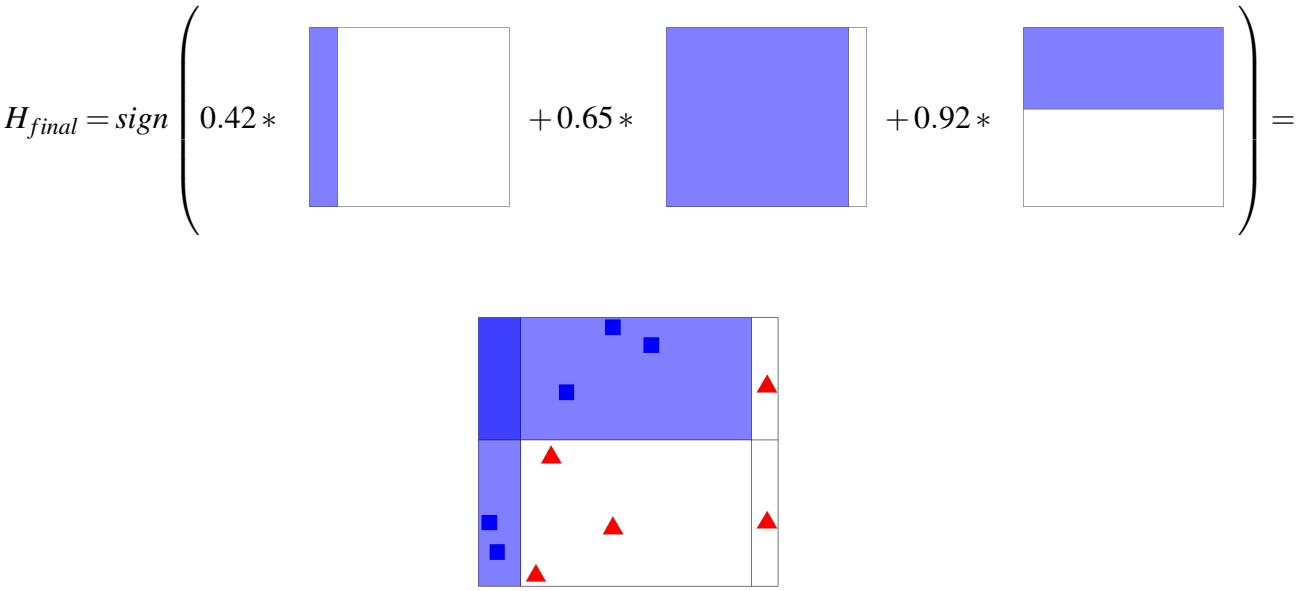


Figura 2.9 Combinação linear dos classificadores fracos, no final do treinamento do AdaBoost.

2.4.2 Adaboost e suas variantes

A primeira modificação real do algoritmo AdaBoost possivelmente tenha sido o Real AdaBoost SCHAPIRE; SINGER. (1999), o qual otimiza a função de custo com menor taxa de erro, mediante a otimização da função $E\left[exp(-y(F(x)+h_i(x))\right]$. Esta modificação é chamada de Real porque os classificadores atribuem uma probabilidade de pertencer a uma classe, mediante um valor real que considera a distribuição atual dos pesos. Abaixo segue o algoritmo:

Entrada: Os dados de treinamento, uma vez que os pares de N de (x_i, y_i), onde o vetor x_i é os atributos e y_i é a saida desejada, sendo binária -1 ou +1 e T é o número de iterações.

Saída: Uma função $f_T(x_i)$ que pode ser usado para classificar um vetor x_i de atributos. Se $f_T(x_i) < 0$ classificar x_i como -1. Se $f_T(x_i) > 0$ classificar x como +1.

Inicialização: Associar uma probabilidade $p_i = \frac{1}{N}$ com o i-ésimo exemplo.

Iteração: Para $t = 1, ..., T$, calcular a hipótese h_t e uma atualização das probabilidades $p_1, ..., p_N$ pelo seguintes passos:

a) Selecionar aleatoriamente com reposição um subconjunto S_t do treinamento. Os i-ésimos exemplos é selecionado com pobabilidade p_i.

b) Para cada classificador h_j, calcular os seguintes valores:

$$\begin{aligned}
P_r^+ &= \sum p_i \quad \text{sobre os exemplos} \quad x_i \quad \text{se} \quad h_j(x_i) = 1, y_i = 1 \\
P_r^- &= \sum p_i \quad \text{sobre os exemplos} \quad x_i \quad \text{se} \quad h_j(x_i) = -1, y_i = -1 \\
P_w^+ &= \sum p_i \quad \text{sobre os exemplos} \quad x_i \quad \text{se} \quad h_j(x_i) = -1, y_i = 1 \\
P_w^- &= \sum p_i \quad \text{sobre os exemplos} \quad x_i \quad \text{se} \quad h_j(x_i) = 1, y_i = -1
\end{aligned} \tag{2.6}$$

$$G(j) = \sqrt{P_r^+(j)P_w^-(j)} + \sqrt{P_w^+(j)P_r^-(j)}$$

Escolha o h_t do classificador h_j que minimize $G(j)$. Se $G(j) \geq 0.5$ volte ao passo a.

c) Calcule os pesos c_t^+, c_t^- e a função g_t como segue abaixo:

$$c_t^+ = \frac{1}{2} ln\left(\frac{P_r^+(t)+\varepsilon}{P_w^-(t)+\varepsilon}\right), \quad c_t^- = \frac{1}{2} ln\left(\frac{P_w^+(t)+\varepsilon}{P_r^-(t)+\varepsilon}\right), \quad g_t(x) = \begin{cases} c_t^+, \text{se} & h_t(x) = 1 \\ c_t^-, \text{se} & h_t(x) = -1 \end{cases}$$

onde ε é um número positivo pequeno. O valor recomendado é: $\varepsilon = \frac{1}{4N}$ em que N é o tamanho da amostra.

d) Atualize as probabilidades

$$p_i^{(t+1)} = \frac{p_i^{(t)} e^{-y_i g_t(x_i)}}{Z_t} \tag{2.7}$$

onde Z_t é a normalização escolhida de modo que a $\sum_{i=1}^{N} p_i = 1$

Terminação:

$$f_T(x) = \sum_{t=1}^{T} g_t(x)$$

Em contraste, o algoritmo Gentle AdaBoost é uma versão mais robusta e estável. Proposto por FRIEDMAN; HASTIE; TIBSHIRANI (2000), o algoritmo utiliza o metodo de Newton em cada passo da otimização. Ao invés de ajustar a probabilidade estimada das classes, utiliza uma regressão dos mínimos quadrados ponderados para minimizar a função custo $E\left[exp(-yF(x))\right]$.

Similarmente, FRIEDMAN; HASTIE; TIBSHIRANI (2000) introduziram o LogitBoost, substituindo a função de custo exponencial do AdaBoost pela função de regressão Logística. O LogitBoost pode ser visto como uma generalização da regressão logística clássica, dado que adota um modelo aditivo em função de T termos: $F(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$. Em cada estágio de iteração, o LogitBoost ajusta uma função de distribuição para cada classe separadamente. Esta é necessária quando implementamos o classificador fraco usando uma árvore de regressão. O LogitBoost minimiza: $\sum_i log(1 + exp(-y_i F(x_i)))$. A variante Real AdaBoost e Gentle será utilizada em comparação a técnica NRAb proposta neste trabalho.

3

Noise Real Adaboost

3.1 Introdução

O algoritmo Real Adaboost vem se mostrando bastante eficiente no que se diz a respeito a classificação binária. Porém, esta técnica apresenta deficiências em dados com alto ruído, onde a cada iteração o algoritmo aumenta o seu peso e com isso tem um aprendizado melhor dos ruídos, porém ocorrendo sobreajustes.

Nesse sentido, o estudo da tolerância ao ruído no algoritmo Adaboost possui uma importância relevante, principalmente quando o banco de dados é extremamente grande ou ruidoso.

Neste Capítulo, propomos uma modificação do Algoritmo Real AdaBoost com uma modificação na atualização dos pesos e analisamos como esta atualização influência no erro empírico. O objetivo dessa abordagem é evitar problemas de sobreajuste introduzindo uma atualização de pesos mais robusta.

3.1.1 Real Adaboost considerando o erro empírico

DIETTERICH. (2000), RäTSCH; ONODA; MüLLER. (2001) e SERVEDIO. (2003) demonstraram que o algoritmo Adaboost possui uma alta tendência de sobreajustes às amostras com dados com nível de ruído alto. HAWKINS. (2004) indica que o sobreajuste aparece quando o número de classificadores aumenta de forma excessiva, ou seja, produz-se uma combinação de classificadores muito complexa, o que leva a uma perda do desempenho.

Apesar do classificador ter sido treinado para reduzir o erro empírico, pode acontecer que, quando é aplicado nos dados de teste, um erro de generalização grande seja gerado. Acredita-se que seja causado pelos dados de treinamento com alto nível de ruído, que provoca sobreajuste. Varios pesquisadores propuseram estratégias na modificação dos pesos atribuídos às amostras a fim de evitar esse fenômeno.

Uma extensão que foi encontrada para superar o AdaBoost padrão foi o Real AdaBoost. Utiliza-se um conjunto de treinamento $x_i \in X$, $i = 1,...,N$, com rótulos $y_i \in \{-1,1\}$ para ajustar

estes classificadores. É usada a distrubuição de probabilidades p_i para descrever os pesos das amostras e esses pesos são modificados de acordo com a Equação 2.7.

VAPNIK. (1982) define o erro empírico de um classificador h_t a respeito às amostras X de treinamento como:

$$E = \sum_{i=1}^{N} |(h_t(x_i) \neq Y_i)| \tag{3.1}$$

Esse erro possui o seguinte limitante de erro FREUND; SCHAPIRE. (1995):

$$E \leq \sum_{i=1}^{N} exp(-y_i h_t(x_i)) \tag{3.2}$$

O objetivo é minimizar estes limitantes para atualizar os pesos das amostras, e assim se obtém a Equação 3.3, a qual é uma expressão analítica para calcular as importâncias de c_t^+ e c_t^- do classificador fraco de cada iteração.

$$c_t^+ = \frac{1}{2} ln \left(\frac{P_r^+(t) + \varepsilon}{P_w^-(t) + \varepsilon} \right) \quad c_t^- = \frac{1}{2} ln \left(\frac{P_w^+(t) + \varepsilon}{P_r^-(t) + \varepsilon} \right) \tag{3.3}$$

O procedimento de atualização dos pesos poderia causar uma distorção considerável sobre a distribuição de pesos FREUND. (1999). Esta distorção é causada pelo erro acumulativo adquirido em iterações repetidas. Com o objetivo de descobrir que fatores influenciam no erro empírico E_T, considera-se que este pode ser dividido em pequenos erros, que aqui chamaremos de erros fragmentários, de tal forma que a soma de "V" dos erros fragmentários descreva o erro empírico, como ilustra na Equação 3.4.

$$E_t = E_{t_1} + E_{t_2} + E_{t_3} ... + E_{t_V} = \sum_{v=1}^{V} E_{tv} \tag{3.4}$$

Será ilustrado detalhadamente o Teorema do limitante do erro do Real AdaBoost de acordo com SCHAPIRE; SINGER. (1999).

Teorema 1: Consideramos um classificador fraco , de tal forma que, quando ativado pelo Real AdaBoost em cada iteração sejam gerados classificadores com erros $E_1, E_2, E_3, ..., E_T$ que são os erros fragmentados. O erro de cada iteração pode ser dividido em "V" erros fragmentários $E_t = E_{t_1} + E_{t_2} + E_{t_3} ... + E_{t_V}$. Então o erro de generalização $E_g = P(y_i \neq F(x_i))$ do classificador final F, possui o seguinte limite superior.

$$E_g \leq 2^T \prod_{t=1}^{T} \left(\sqrt{P_r^+(t) P_w^-(t)} + \sqrt{P_w^+(t) P_r^-(t)} \right) \tag{3.5}$$

Prova 1: Primeiro observar que a seguinte desigualdade detém: $E \leq e^{-y_i f_T(x_i)}$ pela Equação 3.2. Portanto,

$$E \leq \frac{1}{N}\sum_{i=1}^{N} e^{-y_i f_T(x_i)}$$
$$E \leq \frac{1}{N}\sum_{i=1}^{N} e^{-y_i \sum_{t=1}^{T} g_t(x_i)}$$

A qual pode ser reescrita da seguinte forma:

$$E = \frac{1}{N}\sum_{i=1}^{N}\prod_{t=1}^{T} e^{-y_i g_t(x_i)} \tag{3.6}$$

p_i^t é denotado pelo valor p_i da primeira iteração t. Assim de acordo com a equação 2.7, consideramos:

$$\frac{p_i^{t+1}}{p_i^t} = \frac{e^{-y_i g_t(x_i)}}{Z_t}$$

Substituindo na equação 3.6

$$\begin{aligned} E &= \frac{1}{N}\sum_{i=1}^{N}\prod_{t=1}^{T} Z_t \frac{p_i^{t+1}}{p_i^t} \\ &= \frac{1}{N}\sum_{i=1}^{N}(\prod_{t=1}^{T} Z_t)\frac{p_i^{T+1}}{p_i^T} \end{aligned}$$

Considerando que $p_i^1 = 1/N$

$$E = (\prod_{t=1}^{T} Z_t)(\sum_{i=1}^{N} p_i^{T+1})$$

Escolhemos um Z_t como um fator de normalização, tal que $\sum_{i=1}^{N} p_i^{T+1} = 1$. Então

$$E \leq \prod_{t=1}^{T} Z_t \tag{3.7}$$

O limite de E é em termos de Z_t. Ela permanece sendo expressa em termos dos valores de P. Definir $Q_t(i) = p_i e^{-y_i g_t(x_i)}$. Temos:

$$\begin{aligned} Z_t &= \sum_{i=1}^{N} Q_t(i) = \\ &= \sum_{h_t(x_i)=1 \wedge y_i=1} Q_t(i) + \sum_{h_t(x_i)=-1 \wedge y_i=-1} Q_t(i) + \sum_{h_t(x_i)=-1 \wedge y_i=1} Q_t(i) + \sum_{h_t(x_i)=1 \wedge y_i=-1} Q_t(i) \\ &= e^{-c_t^+} P_r^+(t) + e^{c_t^-} P_r^-(t) + e^{-c_t^-} P_w^+(t) + e^{c_t^+} P_w^-(t) \end{aligned}$$

$$Z_t = e^{-c_t^+} P_r^+(t) + e^{c_t^-} P_r^-(t) + e^{-c_t^-} P_w^+(t) + e^{c_t^+} P_w^-(t) \quad (3.8)$$

cálculo direto mostra que Z_t é minimizado por $c_t^+ = \frac{1}{2} ln\left(\frac{P_r^+(t)}{P_w^-(t)}\right)$ e $c_t^- = \frac{1}{2} ln\left(\frac{P_w^+(t)}{P_r^-(t)}\right)$, substituindo na Equação 3.8, que dá:

$$Z_t = P_r^+(t)\sqrt{\frac{P_w^-(t)}{P_r^+(t)}} + P_r^-(t)\sqrt{\frac{P_w^+(t)}{P_r^-(t)}} + P_w^+(t)\sqrt{\frac{P_r^-(t)}{P_w^+(t)}} + P_w^-(t)\sqrt{\frac{P_r^+(t)}{P_w^-(t)}}$$

Aplicando a propriedade da radiação, onde o quociente de radicais de mesmo índice é igual ao quociente de radicandos, ficamos com:

$$\begin{aligned} Z_t = {} & P_r^+(t)\left[(P_w^-(t))^{1/2}(P_r^+(t))^{-1/2}\right] + P_r^-(t)\left[(P_w^+(t))^{1/2}(P_r^-(t))^{-1/2}\right] \\ & + P_w^+(t)\left[(P_r^-(t))^{1/2}(P_w^+(t))^{-1/2}\right] + P_w^-(t)\left[(P_r^+(t))^{1/2}(P_w^-(t))^{-1/2}\right] = \end{aligned}$$

Ficamos com:

$$\begin{aligned} Z_t = {} & \left[(P_w^-(t))^{1/2}(P_r^+(t))^{1/2}\right] + \left[(P_w^+(t))^{1/2}(P_r^-(t))^{1/2}\right] \\ & + \left[(P_r^-(t))^{1/2}(P_w^+(t))^{1/2}\right] + \left[(P_r^+(t))^{1/2}(P_w^-(t))^{1/2}\right] \end{aligned}$$

Portanto encontramos a seguinte expressão:

$$Z_t = 2\left(\sqrt{P_r^+(t)P_w^-(t)} + \sqrt{P_w^+(t)P_r^-(t)}\right)$$

Substituindo na Equação 3.7, encontramos a Equação 3.5:

$$E \leq 2^T \prod_{t=1}^{T}\left(\sqrt{P_r^+(t)P_w^-(t)} + \sqrt{P_w^+(t)P_r^-(t)}\right)$$

O erro de treinamento pode ser composto de vários erros fragmentários, de tal forma que esta modificação deveria apresentar um comportamento similar ao algoritmo Real AdaBoost no sentindo adaptativo. A aplicação da soma de erros fragmentários beneficia o desempenho do algoritmo, minimizando o erro empírico de forma mais adequada e evitando problemas de sobreajustes diante de ruído.

3.1.2 Coeficiente de iteração no AdaBoost Real

Nesta seção será apresentado a modificação na geração do peso do Adaboost Real, segundo RäTSCH; ONODA; MüLLER. (2001) mostraram que o Real AdaBoost não consegue ser

robusto na presença do ruído e mostra um tendência de sobreajuste na generalização. Será considerado que as amostras ruídos possuem algumas características: distribuição das probabilidades sobrepostas, amostras atípicas e amostras rotuladas incorretamente. Estes três tipos de fenômeno aparecem com constante frquência no mundo real.

Os trabalhos de GROVE; SCHUURMANS. (1998b), QUINLAN (1996) , BAUER; KOHAVI. (1999), BREIMAN. (1999a) , DIETTERICH. (2000) e RäTSCH; ONODA; MüLLER. (2001) demonstram que o algoritmo Real AdaBoost é sensível ao ruído. A ideia original considera que, na presença de dados ruidosos, é necessário diminuir a influência de alguns classificadores fracos, ou seja, é considerado os erros fragmentários, de modo que cada um deles é associado a um evento diferente.

É proposto um método diferente para atualizar a distribuição de pesos, considerando a frequência com que uma amostra é classificada de forma incorreta. Esta frequência será denotada por γ_i. Consideramos também dois eventos (β_1, β_2) relacionados à frequência de amostras classificadas incorretamentes γ_i, de forma tal que cada evento seja expresso como se apresenta na Equação 3.9:

$$\begin{aligned} \beta_1 &= \gamma_i \geq k \\ \beta_2 &= \gamma_i < k \end{aligned} \tag{3.9}$$

onde k é o número máximo de frequência de observações classificadas erroneamente.

Assim como SCHAPIRE; SINGER. (1999), supondo que o novo intervalo do classificador fraco h_t esteja restrito a $\{-1, 0, +1\}$, ou seja, um classificador fraco pode realizar uma estimativa de tal forma que o rótulo seja, -1 ou $+1$, bem como poderia "desistir", atribuindo um novo rótulo 0. Na Equação 3.10 é introduzido uma função $\lambda(x)$ que compara a estimativa do classificador fraco de não decidir em amostras que são ruidosas.

$$\lambda(x_i) = \begin{cases} y_i, & h_t(x_i) = y_i \\ 0, & \gamma_i < k \wedge h_t(x_i) \neq y_i \\ y_i, & \gamma_i \geq k \wedge h_t(x_i) \neq y_i \end{cases} \tag{3.10}$$

Os pesos aplicados às amostras são incrementados unicamente se estas atingem uma frequência de classificação incorreta igual a k, baseados na função $\lambda(x)$, como mostra a Equação 3.11

$$p_i^{(t+1)} = \frac{p_i^{(t)} exp^{(-g_t(x_i)\lambda(x_i))}}{Z_t} \tag{3.11}$$

As amostras classificadas incorretamente γ_i irão produzir quatro erros fragmentarios, os quais são somados para obter o erro total. Então, o fator de normalização Z_t pode ser calculado da seguinte forma:

$$Z_t = \sum_{i=1}^{N} Q_t^*(i) = \\ = e^{-c_t^+} P_r^+(t) + e^{c_t^-} P_r^-(t) + e^0 P_w^+(t) + e^0 P_w^-(t) \tag{3.12}$$

onde, $Q_t^*(i) = p_t e^{-\lambda(x_i) g_t(x_i)}$.

Note-se que o conjunto de novas variáveis γ_i procura prevenir o incremento excessivo dos pesos nas amostras ruidosas, através da frequência de ocorrência de classificações erradas. Em outras palavras, os pesos unicamente são incrementados naquelas amostras de fato classificadas de forma incorreta.

O conjunto de novas variáveis γ_i assume seus valores entre $\{0, 1, 2, .., k\}$, começando em zero e sendo incrementado pela unidade quando a amostra associada é classificada incorretamente. Em contrapartida, a variável γ_i mantém o mesmo valor quando a amostra associada é classificada de forma correta. A Equação 3.13 apresenta o passo de atualização de variável γ_i.

$$\gamma(i) = \begin{cases} \gamma_i + 1, & h_t(x_i) \neq y_i \\ 0, & \gamma_i \geq k \\ \gamma_i, & \text{caso contrário} \end{cases} \tag{3.13}$$

No capítulo seguinte, iremos introduzir nossa abordagem em dados simulados e dados reais.

4

Resultados e Discussões

Neste capítulo é de objetivo principal descrever os experimentos realizados para comprovar a melhoria no algoritmo proposto no Capítulo 3 e será apresentado um estudo comparativo entre a abordagem proposta e uma seleção de trabalhos representativos, que foram apresentados na revisão bibliográfica (apresentada na Seção 2.4.2).

Na seção 4.1, será exposta uma avaliação do desempenho da abordagem proposta utilizando diferentes conjuntos de dados gerados artificialmente, os resultados da validação cruzada nas bases do UCI se encontra na seção 4.2 e já na seção 4.3 será aplicado os testes de Friedman e Wilcoxon nas bases do UCI.

4.1 Avaliação do algoritmo Noise Real AdaBoost

Os algoritmos Real Adaboost e Gentle Adaboost são comparados com o algoritmo desenvolvido neste trabalho, o qual chamamos de Noise Real Adaboost. Como classificador base, usamos a árvore de classificação CART, devido à sua simplicidade.

A fim de avaliar o desempenho da nossa abordagem, foram utilizados distintos conjuntos de dados. Por um lado, um grupo de conjuntos foi gerado artificialmente; outro grupo de dados foram obtidos de diferentes sites da internet. Cada conjunto de dados foi dividido aleatoriamene em conjuntos de treinamento e teste numa razão de 75:25. O valor máximo atribuído à variável γ_i é limitado pelo valor de $k = 25$ e será executado 10 vezes as técnicas na base de treinamento de cada base experimental para reduzir o efeito da aleatoriedade nos resultados.

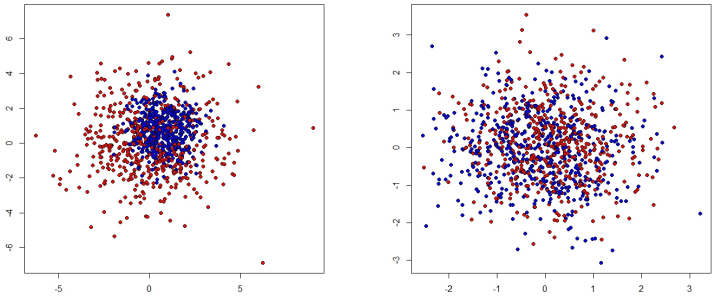
4.1.1 Conjunto de dados TwoNorm e RingNorm

Os conjuntos de dados TwoNorm e RingNorm foram utilizados por BREIMAN. (1998).

- **RingNorm**, possui duas classes de dados. A primeira classe é preenchida de 500 dados rotulados (positivos) que possuem uma distribuição gaussiana multivariada, com média zero e matriz de covariância 4 vezes a matriz identidade. De forma similar, a segunda classe contém 500 dados rotulados (negativos), os quais seguem

uma distribuição gaussiana multivariada com matriz de covariância igual à identidade e média $(a,a,...,a)$, com $a = \frac{2}{\sqrt{500}}$, como se apresenta na Figura 4.1a

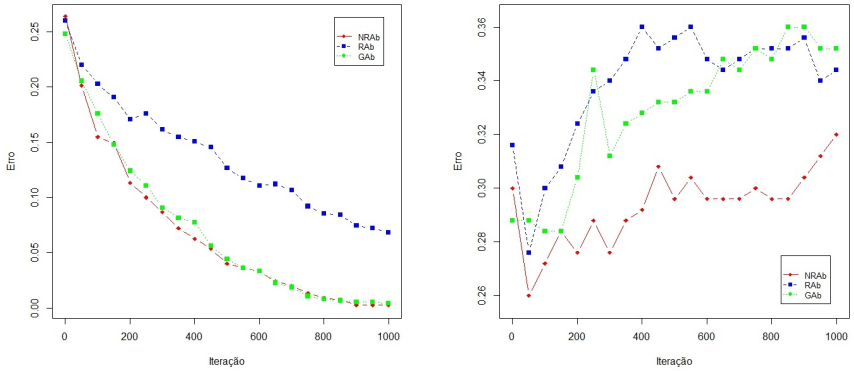
- **TwoNorm**, possui duas classes de dados. A primeira classe contém 500 dados rotulados (positivos), o s quais seguem uma distribuição gaussiana com uma matriz de covariância igual à identidade e média $(a,a,...,a)$. A segunda também contém 500 dados rotulados (negativos), seguindo uma distribuição gaussiana multivariada cm matriz de covariância igual à identidade e média $(-a,-a,...,-a)$, com $a = \frac{2}{\sqrt{500}}$, como se apresenta na Figura 4.1b



(a) RingNorm **(b)** TwoNorm

Figura 4.1 Conjunto de dados gerado artificialmente BREIMAN. (1998).

Na Figura 4.2a, apresentam-se as curvas de desempenho na fase de treinamento produzidas pelo Gentle Adaboost e nossa abordagem, utilizando CART como classificador fraco. A Figura 4.2b ilustra o desempenho de amostras de avaliação. Comparando as curvas, é possivel perceber o efeito benéfico da introdução do coeficiente de iteração, já que este leva a um menor erro no conjunto de dados de teste.



(a) Treino no RingNorm usando CART. **(b)** Teste no RingNorm usando CART.

Figura 4.2 Curvas de treinamento e de teste para o conjunto de dados RingNorm.

Os resultados obtidos para o conjunto de dados RingNorm são sumarizados na Tabela 4.1. A primeira coluna indica o número de alternativas ou combinações de classificadores fracos adotados. Na segunda coluna é indicado o tipo de algoritmo utilizado para o treinamento dos classificadores fracos. Nas colunas restantes, são apresentadas: a taxa de erro no treinamento, a taxa de erro no teste com o respectivo desvio padrão obtido em cada cenário do experimento. A técnica proposta apresentou erros menores nas três etapas das iterações no teste e no treinamento.

Tabela 4.1 Resultado de Gentle Adaboost (GAb), Noise Real AdaBoost (NRAb) e Real Adaboost (RAb) usando CART como classificador fraco. Conjunto de Dados RingNorm.

Classificadores	Algoritmo	Erro de Treinamento	Erro de Teste
100	GAb	0.1787±0.0187	0.2880±0.0080
	NRAb	**0.1600±0.0266**	**0.2720±0.0106**
	RAb	0.2027±0.0107	0.3000±0.0136
500	GAb	0.0493±0.0524	0.3400±0.0224
	NRAb	**0.0440±0.0540**	**0.2960±0.0106**
	RAb	0.1267±0.0282	0.3560±0.0275
1000	GAb	0.0040±0.0644	0.3520±0.0257
	NRAb	**0.0027±0.0625**	**0.3200±0.0127**
	RAb	0.0680±0.0464	0.3440±0.0232

Analisando a Tabela 4.1 podemos verificar que a técnica proposta apresentou erros menores nas três partes das iterações em 100, 500 e 1000 e que também apresenta o menor erro na iteração menor (100) do que a total (1000).

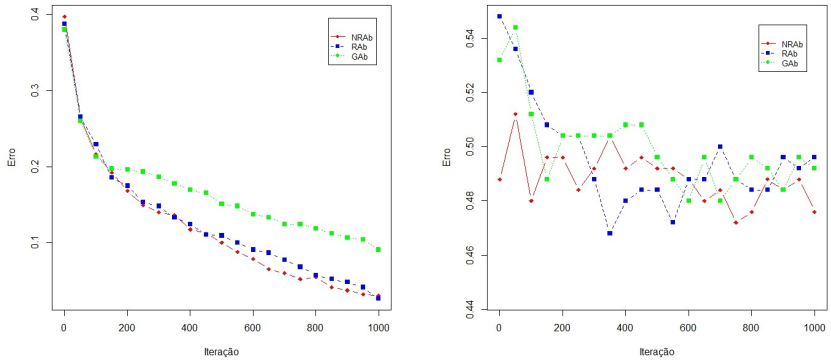
Similarmente, o conjunto de dados TwoNorm BREIMAN. (1998) também foi utilizado para avaliar o desempenho da nossa abordagem. A Tabela 4.2 apresenta os resultados produzidos neste experimento. A alternativa de combinação de diferentes classificadores é apresentada na primeira coluna, mostrando o desempenho para três tipos de combinação diferentes (100, 500 e 1000 classificadores fracos). Neste caso, foram avaliadas duas modificações do AdaBoost (Real AdaBoost e Gentle AdaBoost). A técnica RAb apresentou erro menor na etapa 500 e a técnica proposta NRAb apresentou erro menores na primeira e última etapa (100 e 1000).

Tabela 4.2 Resultado de Gentle Adaboost (GAb), Noise Real AdaBoost (NRAb) e Real Adaboost (RAb) usando CART como classificador fraco. Conjunto de Dados TwoNorm.

Classificadores	Algoritmo	Erro de Treinamento	Erro de Teste
100	GAb	0.2307±0.0354	0.5160±0.0123
	NRAb	**0.2147±0.0388**	**0.4880±0.0202**
	RAb	0.2147±0.308	0.5200±0.0177
500	GAb	0.1493±0.0391	0.4977±0.0134
	NRAb	**0.1053±0.0581**	0.4950±0.0119
	RAb	0.1107±0.0572	**0.4840±0.0201**
1000	GAb	0.0907±0.0485	0.4920±0.0133
	NRAb	0.0307±0.0715	**0.4760±0.0124**
	RAb	**0.0267±0.0682**	0.4960±0.0159

Na Tabela 4.2 a técnica NRAb apresentou erro menores nas iterações 100 e 1000 na base de teste do conjunto de dados TwoNorm erros menores em 100 e 500 na base de treinamento.

Alguns resultados produzidos no comparativo entre o desempenho da nossa abordagem e as abordagens clássicas são apresentados na Figura 4.3. Na Figura 4.3a são apresentados os resultados obtidos no treinamento das diferentes abordagens e na Figura 4.3b são ilustrados os resultados produzidos na etapa de teste. A comparação entre as classificações está de acordo com o comportamento descrito na Tabela 4.2 e a taxa de erro na etapa de treinamento apresenta uma ligeira diminuição em comparação com o Real Adaboost. Porém, esta diminuição não fica muito evidente na etapa de teste, na qual as curvas de erro do RAb possuem quedas maiores, porém se aproximando na iteração 1000 o NRAb apresenta uma diminuição superando o RAb.



(a) Treino no RingNorm usando CART.

(b) Teste no RingNorm usando CART.

Figura 4.3 Curvas de treinamento e de teste para o conjunto de dados RingNorm.

4.1.2 Conjunto de dados em espiral

Este experimento comparativo utilizou o conjunto de dados Espiral, visando a analisar o desempenho da nossa abordagem em situações nas quais as amostras apresentam ruído ou

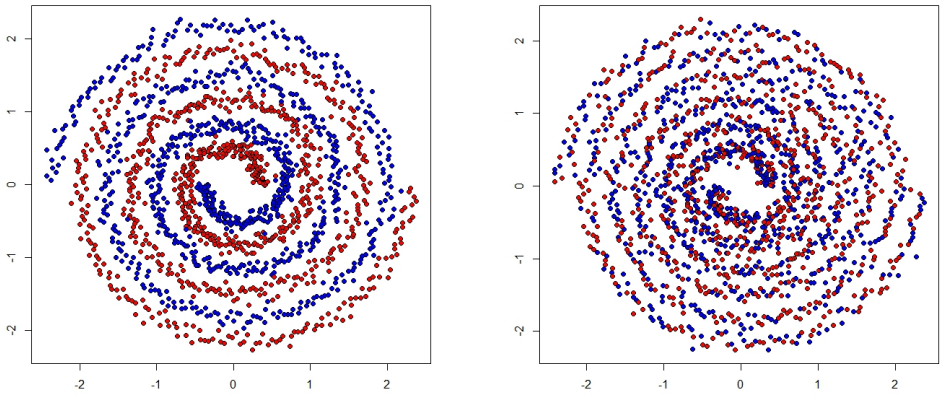
existem outliers. Na Figura 4.4, são apresentados os conjuntos de dados gerados artificialmente. As espirais foram geradas de acordo com YUAN-CHENG; JING-YU. (2010), considerando os números aleatórios n_1, $n_2 \in [0,N]$, ângulos aleatórios ϕ_1 e $\phi_2 \in [0,2\pi]$, raios locais aleatórios r_1 e $r_2 \in [0,K]$, $K > 0$ e o passo D da espiral sobre o anel. O conjunto de dados positivos foi gerado da seguinte forma:

$$\begin{cases} x_p = (D_{n_1} + 1)cos(2\pi.n_1) + r_1.cos\phi_1 \\ y_p = (D_{n_1} + 1)sin(2\pi.n_1) + r_1.sin\phi_1 \end{cases} \quad (4.1)$$

Por outro lado, o conjunto de dados negativos foi gerado assim:

$$\begin{cases} x_n = (D_{n_2} + 1)cos(2\pi.n_2 + \pi) + r_2.cos\phi_2 \\ y_n = (D_{n_2} + 1)sin(2\pi.n_2 + \pi) + r_2.sin\phi_2 \end{cases} \quad (4.2)$$

O ajuste de D e K pode determinar a existência ou não de sobreposição das amostras positivas e das amostras negativas; bem como ajustar N (o número de rotações da espiral) pode determinar a complexidade do conjunto de dados. As Figuras 4.4a e 4.4b, apresentam um conjunto de dados sem ruído e outro com 20% de ruído, respectivamente.



(a) Espiral sem ruído. **(b)** Espiral com 20% de ruído.

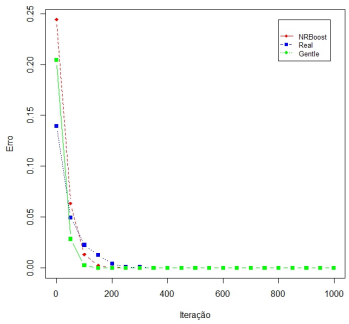
Figura 4.4 Dados gerados artificialmente, Espiral.

A fim de avaliar o desempenho da nossa proposta, foram geradas 1600 amostras rotuladas (800 positivas e 800 negativas). Este conjunto de dados segue a distribuição das Equações 4.1 e 4.2. Nos experimentos, utilizam-se CART como classificadores fracos e cada treinamento foi executado 10 vezes. Os resultados destes experimentos se encontram na Tabela 4.3. A organização das colunas segue a seguinte ordem: ruído inserido nos dados, algoritmo utilizado para o treinamento e taxa de erro produzida no teste da combinação linear de classificadores. O NRAb apresenta erros menores nas bases com a simulação de ruídos e já na base sem ruído ele se assemelha com a técnica RAb, fato no qual já era esperado visto que o NRAb é derivado do RAb e que só apresenta melhora em bases com ruídos.

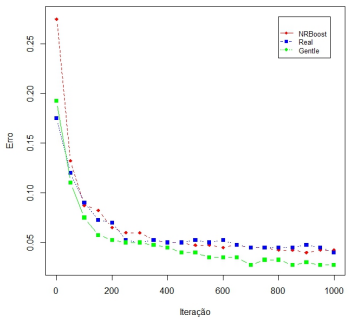
Tabela 4.3 Resultado de Gentle Adaboost (GAd), Noise Robust AdaBoost (NRAb) e Real Adaboost (RAb) usando CART. Conjunto de dados Espiral.

Ruído	Algoritmo	Erro de Teste
0%	GAb	**0.0275±0.0270**
	NRAb	0.0425±0.0306
	RAb	0.0400±0.0252
10%	GAb	0.1125±0.0303
	NRAb	**0.1050±0.0288**
	RAb	0.1150±0.0317
20%	GAb	0.1275±0.0269
	NRAb	**0.1235±0.0268**
	RAb	0.1375±0.0302

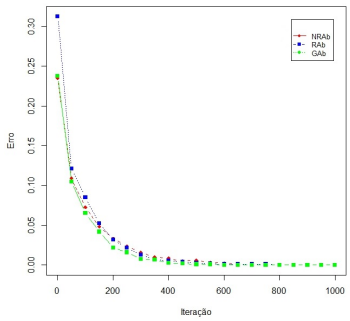
Alguns exemplos de curvas produzidas no experimento de avaliação de desempenho com o uso do coeficiente de iteração são apresentados na Figura 4.5. As Figuras 4.5b, 4.5d e 4.5f comparam o desempenho da nossa abordagem frente ao desempenho do AdaBoost Gentle e AdaBoost Real sem ruido e com a adição de ruído de 10% e 20%, utilizando o classificador fraco (CART). As figuras mostram que nossa abordagem leva uma uma pequena vantagem em relação ao Gentle AdaBoost e uma boa vantagem em relação ao Real AdaBoost que é a variante que adicionamos o coeficiente de iteração.



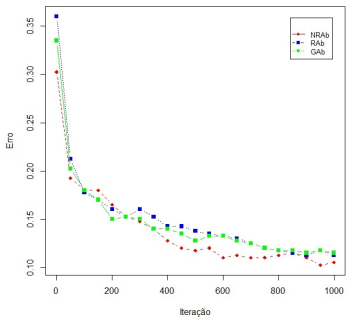
(a) Treinamento sem ruído.



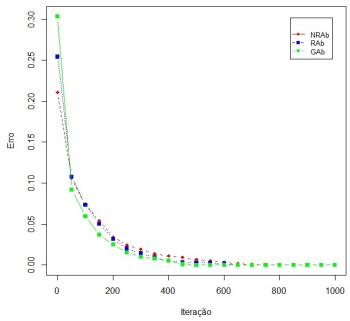
(b) Teste sem ruído.



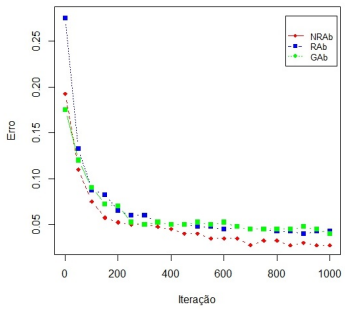
(c) Treinamento com 10% ruído.



(d) Teste com 10% ruído.



(e) Treinamento com 20% ruído.



(f) Teste com 20% ruído.

Figura 4.5 Curvas do treinamento e teste para o conjunto de dados Espiral

4.1.3 Conjunto de dados do UCI.

Será utilizado 6 conjuntos de dados da University of California, Irvine (UCI), cada conjunto de dados é dividido num conjunto de treinamento e outro de teste, numa proporção de 75:25. O valor máximo atribuído à variável γ_i é limitado pelo valor de $k = 25$. Os detalhes destes conjuntos de dados são apresentados na Tabela 4.4.

Tabela 4.4 Descrição do conjunto de dados UCI.

Dados	#variáveis	#amostras
australian	14	690
b-cancer	10	603
bank	16	45211
ionosphere	34	347
liver	6	345
spam	1899	4000

Todos estes conjuntos de dados foram rotulados como positivos ou negativos $([-1, 1])$ e todos os testes foram rodados para 1000 iterações em 10 vezes contínuas. Os resultado produzidos neste experimento são resumidos na Tabela 4.5 .A primeira coluna indica os 6 conjuntos de dados utilizados no treinamento dos classificadores. Na segunda coluna, é indicado o algoritmo

utilizado. Nas colunas restantes, apresentamos: o erro médio obtido após 10 repetições do treinamento e o erro produzido na etapa de teste . Cada valor é acompanhado pelo respectivo desvio padrão. Em todas as bases selecionadas do UCI a técnica NRAb apresentou erros menores em comparação as técnicas propostas RAb e GAb.

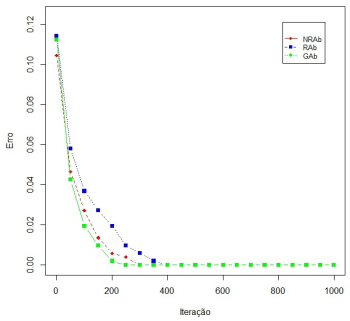
Tabela 4.5 Resultado de GentleBoost (GAb), Real AdaBoost (RAb) e Noise Real AdaBoost (NRAb). Conjuntos de Dados do UCI.

Dados	Algoritmo	Erro de Teste
australian	GAb	0.1503±0.0068
	NRAb	**0.1445±0.0106**
	RAb	0.1618±0.0068
b-cancer	GAb	0.0140±0.0078
	NRAb	**0.0070±0.0066**
	RAb	0.0210±0.0066
bank	GAb	0.0912±0.0015
	NRAb	**0.0903±0.0041**
	RAb	0.0960±0.0013
ionosphere	GAb	0.0795±0.0082
	NRAb	**0.0682±0.0109**
	RAb	0.0795±0.0117
liver	GAb	0.2069±0.0181
	NRAb	**0.1724±0.0190**
	RAb	0.1839±0.0169
spam	GAb	0.0408±0.0041
	NRAb	**0.0382±0.0068**
	RAb	0.0408±0.0059

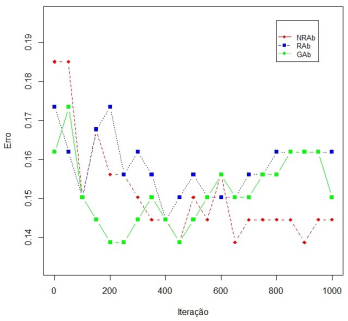
Nas Figuras 4.6 segue as curvas do treinamento e teste de cada base de dados do UCI utilizado para o experimento das três técnicas, Gentle AdaBoost, Real Adaboost e Noise Real AdaBoost.

Analisando as curvas dos testes das bases do UCI, é nítido que a técnica proposta tem um ganho em relação as utilizadas para comparação em boa parte das iterações em alguns casos diminuindo o efeito do *overfitting* como mostra as Figuras 4.6b, 4.6d e 4.6h.

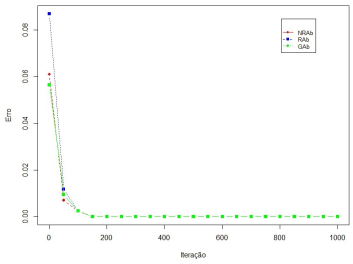
Figura 4.6 Curvas do treinamento e teste para o conjunto de dados do UCI.



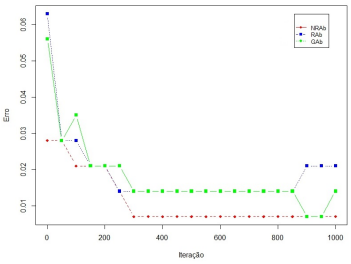
(a) Treinamento Australian.



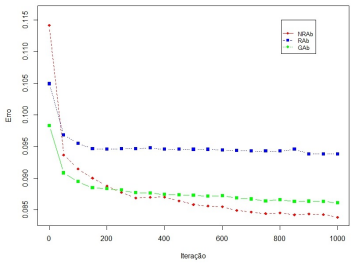
(b) Teste Australian.



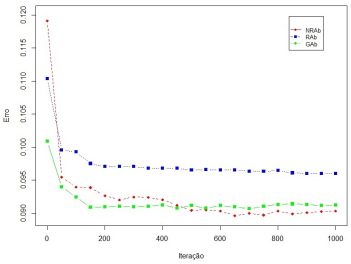
(c) Treinamento b-cancer.



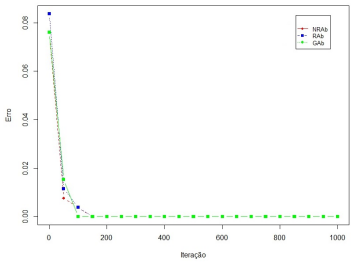
(d) Teste b-cancer.



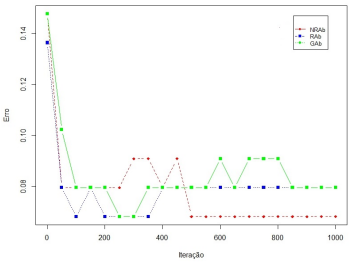
(e) Treinamento bank.



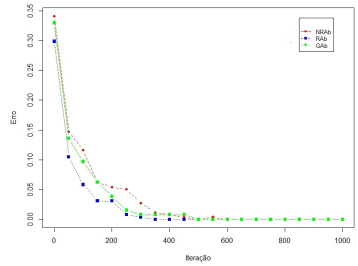
(f) Teste bank.



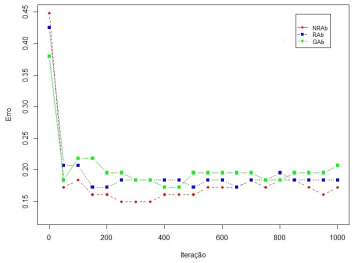
(g) Treinamento ionosphere.



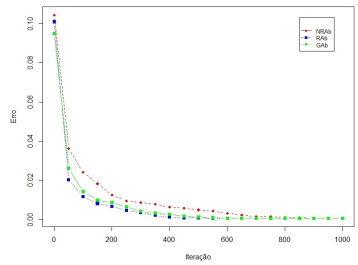
(h) Teste ionosphere.



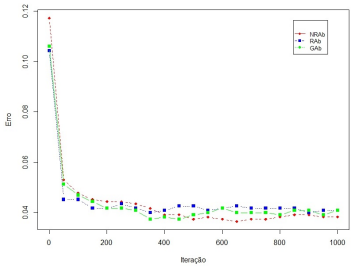
(i) Treinamento liver.



(j) Teste liver.



(k) Treinamento spam.



(l) Teste spam.

4.2 Resultados da Validação Cruzada nas bases do UCI

Será utilizado validação cruzada de amostra aleatoriamente estratificada com $C = 10$ partes e com adição aleatória de 20% de ruído na base de treinamento e na base de teste não ocorre esta adição, abaixo segue as Tabelas 4.6, 4.7, 4.8, 4.9, 4.10 e 4.11, onde apresentam os erros dos testes de cada parte das divisões nas bases do UCI. Em todas as tabelas a nova variante do Real AdaBoost o NRAb apresentou um erro médio menor do que o GAb e RAb.

Tabela 4.6 Validação Cruzada com os resultados de GentleBoost (GAb), Real AdaBoost (RAb) e Noise Real AdaBoost (NRAb) no conjuntos de dados **Australian**.

Partes	Algoritmo	Erro de Teste
1	GAb	0.1739
	NRAb	**0.1516**
	RAb	0.1739
2	GAb	0.1739
	NRAb	**0.1516**
	RAb	0.1739
3	GAb	**0.1304**
	NRAb	0.1309
	RAb	0.1884
4	GAb	0.1956
	NRAb	**0.1723**
	RAb	0.1739
5	GAb	0.1304
	NRAb	**0.1033**
	RAb	0.1159
6	GAb	0.2391
	NRAb	**0.1929**
	RAb	0.2536
7	GAb	0.1304
	NRAb	**0.1103**
	RAb	0.1449
8	GAb	0.1812
	NRAb	0.1792
	RAb	**0.1739**
9	GAb	0.1739
	NRAb	**0.1516**
	RAb	0.1739
10	GAb	0.1449
	NRAb	**0.1232**
	RAb	0.1447
Média	GAb	0.1674±0.0347
	NRAb	**0.11488±0.0285**
	RAb	0.1697±0.0382
Mediana	GAb	0.1739
	NRAb	**0.1516**
	RAb	0.1739

Tabela 4.7 Validação Cruzada com os resultados de GentleBoost (GAb), Real AdaBoost (RAb) e Noise Real AdaBoost (NRAb) no conjuntos de dados **b-cancer**.

Partes	Algoritmo	Erro de Teste
1	GAb	0.1059
	NRAb	**0.0904**
	RAb	0.0962
2	GAb	0.1214
	NRAb	**0.0968**
	RAb	0.1051
3	GAb	0.1015
	NRAb	**0.0799**
	RAb	0.0871
4	GAb	0.1058
	NRAb	**0.0824**
	RAb	0.0883
5	GAb	0.1108
	NRAb	**0.0900**
	RAb	0.0956
6	GAb	0.1176
	NRAb	**0.0906**
	RAb	0.0982
7	GAb	0.1129
	NRAb	**0.0956**
	RAb	0.1017
8	GAb	0.1049
	NRAb	**0.0847**
	RAb	0.0912
9	GAb	0.1165
	NRAb	**0.0976**
	RAb	0.1029
10	GAb	0.1086
	NRAb	**0.0880**
	RAb	0.0934
Média	GAb	0.1106±0.0064
	NRAb	**0.0896±0.0060**
	RAb	0.0960±0.0061
Mediana	GAb	0.1097
	NRAb	**0.0902**
	RAb	0.0959

Tabela 4.8 Validação Cruzada com os resultados de GentleBoost (GAb), Real AdaBoost (RAb) e Noise Real AdaBoost (NRAb) no conjuntos de dados **bank**.

Partes	Algoritmo	Erro de Teste
1	GAb	0.0951
	NRAb	**0.0894**
	RAb	0.0909
2	GAb	0.0998
	NRAb	**0.0914**
	RAb	0.0960
3	GAb	0.0969
	NRAb	0.0927
	RAb	**0.0911**
4	GAb	0.0902
	NRAb	**0.0856**
	RAb	**0.0856**
5	GAb	0.0942
	NRAb	**0.0834**
	RAb	0.0856
6	GAb	0.0967
	NRAb	**0.0920**
	RAb	0.0938
7	GAb	0.1002
	NRAb	0.0936
	RAb	**0.0925**
8	GAb	0.0951
	NRAb	0.0871
	RAb	**0.0852**
9	GAb	0.1002
	NRAb	0.0938
	RAb	**0.0933**
10	GAb	0.1000
	NRAb	0.0991
	RAb	**0.0973**
Média	GAb	0.0968±0.0033
	NRAb	**0.0908±0.0046**
	RAb	0.0911±0.0044
Mediana	GAb	0.0535
	NRAb	**0.0424**
	RAb	0.0535

Tabela 4.9 Tabela de Validação Cruzada com os resultados de GentleBoost (GAb), Real AdaBoost (RAb) e Noise Real AdaBoost (NRAb) no conjuntos de dados **ionosphere**.

Partes	Algoritmo	Erro de Teste
1	GAb	0.0857
	NRAb	**0.0679**
	RAb	0.0857
2	GAb	0.0428
	NRAb	**0.0271**
	RAb	0.0428
3	GAb	0.1285
	NRAb	**0.1086**
	RAb	0.1285
4	GAb	0.1000
	NRAb	0.0679
	RAb	**0.0571**
5	GAb	0.1571
	NRAb	**0.1494**
	RAb	0.1571
6	GAb	0.2142
	NRAb	**0.1766**
	RAb	0.2142
7	GAb	**0.1142**
	NRAb	0.1222
	RAb	**0.1142**
8	GAb	**0.0000**
	NRAb	**0.0000**
	RAb	**0.0000**
9	GAb	0.1428
	NRAb	**0.1358**
	RAb	0.1571
10	GAb	0.0857
	NRAb	0.0679
	RAb	**0.0571**
Média	GAb	0.1071±0.0599
	NRAb	**0.0923±0.0557**
	RAb	0.1014±0.0647
Mediana	GAb	0.1071
	NRAb	**0.0883**
	RAb	0.1000

Tabela 4.10 Tabela de Validação Cruzada com os resultados de GentleBoost (GAb), Real AdaBoost (RAb) e Noise Real AdaBoost (NRAb) no conjuntos de dados **liver**.

Partes	Algoritmo	Erro de Teste
1	GAb	0.3235
	NRAb	**0.3076**
	RAb	0.3382
2	GAb	0.3823
	NRAb	**0.3496**
	RAb	0.3676
3	GAb	0.3529
	NRAb	**0.2657**
	RAb	0.3088
4	GAb	0.1764
	NRAb	**0.1538**
	RAb	0.1911
5	GAb	0.2500
	NRAb	0.2237
	RAb	**0.2205**
6	GAb	0.3970
	NRAb	**0.3496**
	RAb	0.4264
7	GAb	0.4117
	NRAb	**0.3915**
	RAb	0.3970
8	GAb	**0.3382**
	NRAb	0.3636
	RAb	0.3676
9	GAb	0.2500
	NRAb	**0.2237**
	RAb	0.2352
10	GAb	0.2205
	NRAb	**0.2097**
	RAb	0.2205
Média	GAb	0.3102$\pm$0.0809
	NRAb	**0.2839$\pm$0.0796**
	RAb	0.3073$\pm$0.0844
Mediana	GAb	0.3308
	NRAb	**0.2866**
	RAb	0.3235

Tabela 4.11 Tabela de Validação Cruzada com os resultados de GentleBoost (GAb), Real AdaBoost (RAb) e Noise Real AdaBoost (NRAb) no conjuntos de dados **spam**.

Partes	Algoritmo	Erro de Teste
1	GAb	0.0641
	NRAb	0.0568
	RAb	**0.0565**
2	GAb	0.0760
	NRAb	**0.0682**
	RAb	0.0695
3	GAb	0.0684
	NRAb	**0.0547**
	RAb	0.0554
4	GAb	0.0489
	NRAb	**0.0330**
	RAb	0.0336
5	GAb	0.0532
	NRAb	0.0423
	RAb	**0.0413**
6	GAb	0.0706
	NRAb	**0.0537**
	RAb	0.0576
7	GAb	0.0597
	NRAb	**0.0547**
	RAb	0.0565
8	GAb	0.0695
	NRAb	**0.0527**
	RAb	0.0576
9	GAb	0.0521
	NRAb	**0.0403**
	RAb	0.0456
10	GAb	0.0597
	NRAb	**0.0527**
	RAb	0.0576
Média	GAb	0.0622±0.0090
	NRAb	**0.0509±0.0099**
	RAb	0.0531±0.0101
Mediana	GAb	0.0619
	NRAb	**0.0532**
	RAb	0.0565

4.3 Testes Estatísticos

A análise comparativa dos classificadores limitou-se a aplicação do teste de Friedman para k amostras relacionadas, o qual se trata de um teste não-paramétrico, ou seja, não requer o conhecimento da distribuição da variável da população. Este teste permite ranquear os algoritmos RAb, GAb e NRAb, permitindo obter a resposta de qual teve o melhor comportamento. O principal objetivo do teste de Friedman é conferir se os classificadores gerados possuem diferenças expressivas, sendo caracterizada hipótese nula; ou seja, quando não existem diferenças entre os algoritmos. Para verificar se existe ou não correlação entre os dados, deve-se fazer o somatório das variâncias dos ranques, para então, através deste somatório calcular a probabilidade do valor ser superior ou igual à variância obtida utilizando a distribuição Qui-quadrada com $k-1$ graus de liberdade. O resultado numérico final deste teste apresenta um nível de significância (p-valor). Quando tal valor for menor que 0.05, então é aconselhado rejeitar a hipótese nula. A hipótese nula para o teste é H_0 : GAb = NRAb = RAb e a altermativa H_1 : GAb $\neq$ NRAb $\neq$ RAb.

Tabela 4.12 p-valores obtidos no teste de Friedman na validação cruzada.

Dados	p-valor	Decisão
australian	0.0168	Rejeita H_0
b-cancer	0.0524	Não Rejeita H_0
bank	0.0004	Rejeita H_0
ionosphere	0.0608	Não Rejeita H_0
liver	0.0070	Rejeita H_0
spam	0.0002	Rejeita H_0

Para um valor de p < 0.05, na Tabela 4.12 é possível concluir que há diferença significativa entre os classificadores nas bases **australian**, **bank**, **liver** e **spam**, ou seja, as outras bases **b-cancer** e **ionosphere** não rejeita a hipótese nula, visto que todos os testes obtiveram resultados superiores a 0.05.

Sabendo que existe diferença estatística pelo metódo de Friedman entre os classificadores nas bases citadas acima, será aplicado o teste de Wilcoxon pareado que também é um teste não-paramétrico para comparação entre dois classificadores. A princípio são calculados os valores numéricos da diferença entre cada par, sendo possível três condições: aumento $(+)$, diminuição $(-)$ ou igualdade $(=)$. Uma vez calculadas todas as diferenças entre os valores obtidos para cada par de dados, essas diferenças são ordenadas pelo seu valor absoluto (sem considerar o sinal), substituindo-se então os valores originais pelo posto que ocupam na escala ordenada. O teste da hipótese de igualdade entre os grupos é baseado na soma dos postos das diferenças negativas e positivas. Este teste para dados pareados, ao invés de considerar apenas o sinal das diferenças entre os pares, considera o valor dessas diferenças, sendo assim um teste

não-paramétrico dos mais poderosos e "populares".

O objetivo do teste dos sinais de Wilcoxon é comparar as performances de cada sujeito (ou pares de sujeitos) no sentido de verificar se existem diferenças significativas entre os seus resultados nas duas situações.

As Tabelas 4.13, 4.14, 4.15 e 4.16 mostra que no teste da hipótese alternativa H_1 :GAb>NRAb rejeita a hipotese nula onde H_0 :GAb=NRAb indicando que a nova variante NRAb tem um desempenho superior do que o GAb, já na hipótese alternativa H_1 :RAb>NRAb o p-valor do teste aponta que não rejeita a hipotese nula que é H_0 :RAb=NRAb, ou seja, a variante proposta NRAb apresenta erro menores nas partições da validação cruzada.

Tabela 4.13 p-valores obtidos no teste de Wilcoxon na validação cruzada da base **australian**.

Hipótese alternativa	p-valor	Decisão
H_1 :GAb>NRAb	0.0161	Rejeita H_0
H_1 :RAb>NRAb	0.0053	Não Rejeita H_0

Tabela 4.14 p-valores obtidos no teste de Wilcoxon na validação cruzada da base **bank**.

Hipótese alternativa	p-valor	Decisão
H_1 :GAb>NRAb	0.0009	Rejeita H_0
H_1 :RAb>NRAb	0.0009	Não Rejeita H_0

Tabela 4.15 p-valores obtidos no teste de Wilcoxon na validação cruzada da base **liver**.

Hipótese alternativa	p-valor	Decisão
H_1 :GAb>NRAb	0.0124	Rejeita H_0
H_1 :RAb>NRAb	0.0019	Não Rejeita H_0

Tabela 4.16 p-valores obtidos no teste de Wilcoxon na validação cruzada da base **spam**.

Hipótese alternativa	p-valor	Decisão
H_1 :GAb>NRAb	0.0009	Rejeita H_0
H_1 :RAb>NRAb	0.0124	Não Rejeita H_0

5

Conclusões e Trabalhos Futuros

5.1 Conclusões

Neste trabalho, foi apresentado e introduzido um novo procedimento de atualização no algoritmo Real AdaBoost, visando melhorar seu desempenho em dados ruidosos. Os resultados experimentais demonstram e validam o método proposto, assim, as principais conclusões que podem ser destacadas são:

- **Em relação à investigação das abordagens relacionadas com a atualização de pesos:** A atualização da distribuição de pesos no algoritmo Real AdaBoost possui uma importância relevante no processo de seleção de características. Quando a distribuição de pesos associados às amostras são atualizadas adequadamente pode-se obter resultados que melhoram o desempenho do algoritmo Real AdaBoost, como se mostrou na subseção 3.1.1 e 3.1.2.

- **Em relação ao desenvolvimento e implementação de um novo algoritmo baseado na atualização de pesos:** Neste trabalho, mostramos que o erro de treinamento pode ser dividido nos chamados erros fragmentários 3.1.1. Este fato abre muitas possibilidades para avaliar diferentes tipos de fatores que poderiam afetar a performance do algoritmo. Consequentemente, o assim chamado Noise Real AdaBoost utilizou um coeficiente de iteração para obter erros fragmentários e, dessa forma, dividir o erro empírico. Esta abordagem apresentou vantagens na rapidez de convergência quando comparado com os algoritmos clássicos e, em muitos casos, obteve erros de treinamento mais baixos em menor número de rodadas. Assim, foi possível usar um número menor de classificadores fracos para construir um classificador forte ótimo, o qual é uma propriedade desejável nas aplicações em tempo real, onde a velocidade de teste é crítica. A efetividade e precisão do procedimento foram comprovadas através de conjuntos de dados artificiais e reais. Nesse sentido, esta abordagem conseguiu melhorar o desempenho dos algoritmos de Real AdaBoost na presença de dados ruidosos.

5.2 Trabalhos Futuros

Como sugestão de trabalhos futuros podemos enumerar:

- Expandir o algoritmo proposto para o problema de classificação de multi-classe.
- Adicionar um critério de parada (*early stopping*) em uma base de validação, para evitar a perda de generalização do algoritmo.
- Gerar um *dashboard* de acompanhamento dos resultados das iterações na base de validação, por exemplo: taxa de erro, Área da Curva ROC (AUC) e taxa das classes por faixa de escore.
- Adicionar uma melhoria para base de dados de classe desbalanceadas aplicando alguns técnicas conhecidas como *undersampling* ou *oversampling*.
- Apresentar uma otimização para obtenção do valor k que é número máximo de frequência de observações classificadas erroneamente de acordo com o número de registro e variável explicativas da base de treinamento.

Referências

BAUER, E.; KOHAVI., R. An empirical comparison of voting classification algorithms: bagging, boosting, and variants. , [S.l.], p.105–139, 1999.

BREIMAN, L. . **Bagging Predictors.** [S.l.]: Machine Learning., 1996. 123-140p. v.24.

BREIMAN., L. **Arcing classifiers.** [S.l.]: Annals of Statistics., 1998. 801-824p.

BREIMAN., L. Prediction games and arcing algorithms. , [S.l.], p.1493–1517, 1999a.

BREIMAN, L. et al. CLassification and Regression Trees. , [S.l.], 1984.

BUJA, A.; STUETZLE, W. The effect of bagging on variance, bias and mean squared error. , [S.l.], 2000.

BUKLMANN, P.; HOTHORN, T. **Boosting Algorithms**: regularization, prediction and model fitting. [S.l.]: Statistical Science., 2007. 477-505p. v.22, n.4.

CREAMER, G.; FREUND., Y. Using Adaboost for equity investiment scoredcards. , [S.l.], 2005.

DIETTERICH., T. G. **Ensemble methods in machine learning**. [S.l.]: Proceedings of the First International Workshop on Multiple Classifier Systems, MCS '00, London, UK. Springer-Verlag., 2000. 1-15p.

FREUND., Y. **An adaptive version of the boost by majority algorithm.** [S.l.]: ACM New York, NY., 1999. 102-113p.

FREUND, Y.; SCHAPIRE., R. E. **A decision-theoretic generalization of on-line learning and an application to boosting.** [S.l.]: Proceedings of the Second European Conference on Computational Learning Theory, EuroCOLT '95, London, UK, Springer-Verlag., 1995. 23-37p.

FREUND, Y.; SCHAPIRE., R. E. Experiments with a new Boosting algorithm. In: **Anais...** Machine Learning., 1996. p.148–156.

FRIEDMAN, J. H.; HASTIE, T. J.; TIBSHIRANI, R. J. **Additive logistic regression**: a statistical view of boosting (with discussion). [S.l.]: The Annals of Statistics., 2000. 337-407p. v.28.

FRIEDMAN, J. H.; HASTIE, T. J.; TIBSHIRANI, R. J. **The Elements of Statistical Learning.** [S.l.]: Springer Verlag., 2001.

FRIEDMAN, J.; HALL, P. On bagging and nonlinear estimation. , [S.l.], 2000.

GROVE, A. J.; SCHUURMANS., D. Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence. **Proc. of the Int. Conf. on Neural Information Processing.**, [S.l.], p.692–699, 1998.

GROVE, A. J.; SCHUURMANS., D. Boosting in the limit: maximizing the margin of learned ensembles. , [S.l.], p.692–699, 1998.

HANSEM, L. K.; SALAMON, P. Neural Network Ensembles. , [S.l.], p.993–1001, 1990.

HAWKINS., D. M. The problem of overfitting. **Journal of Chemical Information and Computer Sciences.**, [S.l.], p.1–12, 2004.

HAYKIN., S. **Neural Networks**: a comprehensive foundation. 2nd.ed. [S.l.]: Prentice Hall PTR., 1998.

HUNT, E. B.; STONE, P. J.; MARIN, J. Experiments in induction. , [S.l.], 1966.

JIANG, L. Process consistency for adaboost. , [S.l.], 2000.

OPITZ, D.; MACLIN, R. Popular ensemble methods: an empirical study. , [S.l.], v.11, p.169–198, 1999.

QUINLAN, J. Discovering rules by induction from large collections of examples. , [S.l.], 1979.

QUINLAN, J. R. Induction of decision trees. , [S.l.], p.81–106, 1986.

QUINLAN, J. R. Boosting first-order learning. , [S.l.], p.143–155, 1996.

RIPLEY, B. D. Pattern Recognition and Neural Networks. , [S.l.], 1996.

ROKACH, L. **Ensemble methods for classifiers.** [S.l.]: The Data Mining and Knowledge Discovery Handbook., 2005. 957-980p.

RUBESAM, A. **Estimação Não Paramétrica Aplicada a Problemas de Classificação via Bagging e Boosting**. 2004. 127p. Dissertação (Mestrado em Ciência da Computação) — Dissertação de mestrado do Instituto de Matemática, Estatística e COmputação Científica da Universidade Estadual de Campinas, Campinas.

RäTSCH, G.; ONODA, T.; MüLLER., K.-R. Soft Margins for AdaBoost. **Proc. of the Int. Conf. on Neural Information Processing.**, [S.l.], p.506–509, 1998.

RäTSCH, G.; ONODA, T.; MüLLER., K.-R. Soft Margins for AdaBoost. **Machine Learning**, [S.l.], p.287–320, 2001.

SCHAPIRE., R. E. **The strength of weak learnability.** [S.l.]: Machine Learning., 1990. 197-227p.

SCHAPIRE, R. E.; FREUND., Y. Boosting: foundations and algorithms. , [S.l.], 1999a.

SCHAPIRE, R. E.; SINGER., Y. **Improved boosting algorithms using confidence-rated predictions.** [S.l.]: Machine Learning., 1999. 297-336p.

SERVEDIO., R. A. Smooth boosting and learning with malicious noise. **Machine Learning**, [S.l.], p.633–648, 2003.

VAPNIK., V. **Estimation of Dependences Based on Empirical Data; Springer Series in Statistics (Springer series in Statistics).** [S.l.]: Springer-Verlag New York., 1982.

YUAN-CHENG, X.; JING-YU., Y. Using clustering to speed up adaboost and detecting noisy data. **Journal of Software.**, [S.l.], p.1889–1897, 2010.

ZHOU, Z. H.; WU, J.; TANG, W. Ensembling neural networks: many could be better than all. , [S.l.], p.239–263, 2002.