UNIVERSIDADE FEDERAL DE PERNAMBBUCO
CENTRO DE INFORMÁTICA

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA METODOLOGIA DE APLICAÇÃO DE TÉCNICAS DE AMOSTRAGEM EM ALGORITMOS DE SELEÇÃO DE PROTÓTIPOS

Filipe Pinheiro dos Santos Cantarelli

Trabalho de Graduação

Recife

AGOSTO DE 2014

UNIVERSIDADE FEDERAL DE PERNAMBBUCO
CENTRO DE INFORMÁTICA

Filipe Pinheiro dos Santos Cantarelli

UMA METODOLOGIA DE APLICAÇÃO DE TÉCNICAS DE AMOSTRAGEM EM ALGORITMOS DE SELEÇÃO DE PROTÓTIPOS

Trabalho apresentado ao Programa de GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO do CENTRO DE INFORMÁTICA da UNIVERSIDADE FEDERAL DE PERNAMBUCO como requisito parcial para obtenção do grau de Bacharel em CIÊNCIA DA COMPUTAÇÃO.

Orientador: Prof. Dr. George Darmiton da Cunha Cavalcanti

Recife

Agosto de 2014

AGRADECIMENTOS

Agradeço primeiramente à Deus por ter me concedido a vida. Em seguida, agradeço a minha noiva Aline que com seu amor e companheirismo fez com que minha vida se tornasse mais alegre e completa, e sem isto este trabalho não poderia ter sido concluído. Agradeço à minha família pelo carinho, apoio e ensinamentos durante todos os anos da minha vida. Agradeço ao meu orientador pelo tempo dedicado e ensinamentos.

RESUMO

A classificação por ensemble de classificadores é uma das soluções do problema de classificação mais utilizadas atualmente. Apesar dos esforços nesta área, pouco tem sido feito na aplicação destas técnicas para o problema de seleção de instâncias.

Este trabalho possui dois objetivos. O Primeiro é introduzir à metodologia de aplicação de técnicas de amostragem em algoritmos de seleção de instâncias, assim como dar uma breve descrição de métodos utilizados em ambas as áreas. O segundo é fazer uma análise experimental dos métodos a fim de validar a metodologia.

Lista de Figuras

Figura1- ProcessoGeral 14
Figura2 - Seleção de limiar 16
Figura3 – Adaboost&Madaboost 19
Figura4 - Multiboost.....22
Figura 5 - Reweightboost 24
Figura6 - CNN 25
Figura 7 - RNN 26
Figura8 - MSS 27
Figura9 - Drop3 29
Figura10 - ISAR 30
Figura 11 - CRIS 31

Lista de Tabelas

Tabela1- Bases selecionadas 33
Tabela2 - CNN 34
Tabela3 - Adaboost&Madaboost com CNN.....34
Tabela4 - Multiboost e ReweightBoost com CNN 35
Tabela5 - RNN, k = 1 35
Tabela6 - Adaboost&Madaboost com RNN.....36
Tabela7 - Multiboost&Reweighboost com RNN.....36
Tabela8 - MSS.....37
Tabela9 - Adaboost&Madaboost com MSS.....37
Tabela10 - Multiboost&ReweighBoost com MSS.....38
Tabela11 - Drop3.....38
Tabela12 - Adaboost&Madaboost com DROP3.....39
Tabela13 - Multiboost&ReweightBoost com DROP3 39
Tabela14 - ISAR 40
Tabela15 - Adaboost&Madaboost com ISAR.....40
Tabela16 - Multiboost&ReweightBoost com ISAR 41
Tabela17 - Cris.....42
Tabela18 - Adaboost&Madaboost com CRIS.....42
Tabela19 - Multiboost&ReweightBoost com CRIS.....43

Sumário

Capítulo 1 – Introdução 8
1.1 Considerações iniciais 8
1.2 Motivação 9
1.3 Objetivo 10
1.4 Estrutura do Documento 10
Capítulo 2 - Metodologia 11
2.1 Escolha do limiar 15
2.2 Algoritmos de *boosting* 17
2.3 Adaboost e Madaboost 18
2.4 Multiboost. 20
2.5 ReweightBoost 23
Capítulo 3 - Algoritmos de seleção de instâncias. 24
3.1 CNN 24
3.2 RNN 25
3.3 MSS 26
3.4 DROP3 28
3.5 ISAR 29
3.6 CRIS 31
Capítulo 4 - Experimentos 32
4.1 Resultados 33
4.1.1 CNN 33
4.1.2 RNN 35
4.1.3 MSS 37
4.1.4 DROP3 38
4.1.5 ISAR 40
4.1.6 CRIS 41
4.2 Análise dos resultados 43
Capítulo 5 – Conclusões 45
5.1 Dificuldades 45
5.2 Trabalhos futuros 45
Referências 47

Capítulo 1 – Introdução

Este capítulo está dividido em três partes: (i) a primeira apresenta uma visão geral sobre a área de seleção de instâncias e métodos de boosting; (ii) a segunda descreve a motivação para o desenvolvimento deste trabalho; (iii) a tarceira apresenta o objetivo deste trabalho; e por fim, (iv) a quarta detalha a estrutura do documento.

1.1 Considerações iniciais

A quantidade imensa de dados que estão surgindo em diversas áreas de pesquisa cria novos desafios para métodos de mineração de dados e extração de conhecimento. Este volume cada vez crescente de dados faz com que métodos existentes se tornem inaplicáveis em muitos problemas do mundo real. Para estes conjuntos de dados a escalabilidade se torna um problema.

Uma das metodologias mais utilizadas no tratamento de grandes quantidades de informação é a de redução de dados. Dentre os métodos mais utilizados para redução de dados está a seleção de instâncias, que consiste em remover instâncias errôneas e/ou redundantes de um conjunto de treinamento com o intuito de obter um conjunto de dados que seja tratável.

A seleção de instâncias escolhe um subconjunto do total de dados disponíveis para realizar a tarefa de mineração de dados. Para isso, o subconjunto deve manter propriedades relacionadas à tarefa de extração de conhecimento que seria realizada no conjunto original. Podemos identificar dois modelos principais de seleção de instâncias: Seleção de instâncias como método para seleção de protótipos para algoritmos baseados em protótipos(como o k-nn) e seleção de instâncias para obter o conjunto de treinamento para um algoritmo de aprendizagem [1].

O problema de seleção de instâncias para aprendizagem pode ser definido como “a extração do menor subconjunto de instâncias que nos permite classificar uma amostra de teste com a mesma(ou maior) taxa de acerto”.

Outra tarefa bastante comum em mineração de dados é classificação. Um problema de classificação de K classes e N amostras de treinamento consiste de um conjunto de instâncias cujas classes são conhecidas. Cada classe pode ser interpretada como um inteiro no conjunto Y = {1,..., K}. Um classificador é uma função que mapeia uma instância a um elemento de Y. A tarefa de classificação é encontrar a função que mapeia corretamente cada instância para sua classe.

Em um *ensemble* de classificadores cada classificador mapeia uma instância ao conjunto de classes. Uma das metodologias mais bem sucedidas para construir *ensembles* de classificadores é *boosting*.

Métodos de *boosting* constroem um *ensemble* de classificadores iterativamente, modificando a distribuição das instâncias de treinamento a cada iteração. Um vetor de peso, w, é utilizado. W_ié uma medida que diz quão difícil é classificar corretamente a instância x_i. Inicialmente todas as instâncias recebem o mesmo peso. No decorrer do processo os pesos são adaptados de acordo com a classificação de cada instância, o peso é diminuído para instâncias que são facilmente classificadas e aumentado para instâncias que são dificilmente classificadas. Muitos métodos de *boosting* existem, eles diferem principalmente, além de outros detalhes, no modo como o vetor de pesos é atualizado.

1.2 Motivação

Apesar de algoritmos de seleção de instâncias e algoritmos de aprendizagem por ensemble serem áreas bastante abordadas na literatura científica, pouco trabalho tem sido feito na aplicação de técnicas de ensemble em seleção de instâncias.

Sebban et. Al [2] introduziu o PSBoost como uma das primeiras tentativas de juntar seleção de instâncias e *boosting*. No PSBoost cada hipótese fraca é

substituída por uma instância de treinamento. O peso derivado do *boosting* define a relevância da instância e um teste estatístico decide se a instância pode ser removida ou não do conjunto final.

Lazzarevid e Obradovic[3] criaram, utilizando uma abordagem diferente, uma técnica chamada de *boosting* progressivo. Ao invés de amostrar a mesma quantidade de dados a cada iteração principal do *boosting*, o método começa inicialmente com uma amostra pequena e, à medida que o tempo passa, aumenta o tamanho da amostra utilizada no método de aprendizado.

Gárcia-Pedrajas[4] utilizou seleção de instâncias para construir um *ensemble* de classificadores. O método proposto consiste na seleção de instâncias a cada passo de *boosting* para obter os membros do ensemble.

Recentemente Gárcia-Pedrajas publicou [5] uma nova metodologia para aplicação de técnicas de construção de ensemble em algoritmos de seleção de protótipos. Esta técnica será estudada extensivamente neste documento.

1.3 Objetivo

Este trabalho tem como objetivo o estudo de técnicas de aplicação de *boosting* em algoritmos de seleção de instâncias, para isto serão estudados alguns algoritmos de *boosting* assim como algoritmos de seleção de instâncias.

1.4 Estrutura do Documento

O documento se estrutura da seguinte maneira: No capítulo 2 serão apresentados a metodologia e os algoritmos de boosting utilizados; No capítulo 3 serão apresentados os algoritmos de seleção de instâncias utilizados neste trabalho; No capítulo 4 serão mostrados os experimentos realizados e, por fim, no capítulo 5 será apresentada a conclusão.

Capítulo 2 - Metodologia

Um problema de classificação com K classes e N amostras de treinamento nada mais é que um conjunto de amostras cuja classificação é conhecida. Seja $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$ um conjunto de treinamento onde cada instância x_i pertence ao domínio X. Cada classe y_i é um elemento do conjunto Y. Um classificador multiclasses é uma função f: X -> Y que mapeia uma instância de X à um elemento de Y.

A tarefa de classificação é aproximar a função desconhecida f(x) dado o conjunto de treinamento T. Num *ensemble* de classificadores nós temos um conjunto de classificadores $F = \{f_1, f_2, \dots, f_m\}$, onde cada classificador mapeia um vetor de instâncias de X para um elemento de Y. A construção de classificadores *ensemble* envolve duas etapas: construir os classificadores individuais, f_i, e construir uma regra de combinação que encontre a classificação de x baseada nos resultados dos classificadores $\{f_1(x), f_2(x), \dots, f_m(x)\}$.

A grande maioria das técnicas foca na obtenção de grupos de classificadores que são bastante precisos e que sejam mais diferentes possíveis, isto é, os classificadores devem ter uma correlação baixa. Estes dois objetivos, em essência, são conflitantes pois visto que os classificadores devem ter uma alta precisão eles devem concordar com mais frequência na classificação de uma determinada instância. Uma visão correta é fazer com que exista diversidade no conjunto de classificadores, para isso muitas técnicas já foram desenvolvidas. Kuncheva [6] encontrou pelo menos 4 diferentes abordagens: (1) usar diferentes esquemas de combinação; (2) usar diferentes classificadores; (3) usar diferentes subconjuntos de características; e, por fim, (4) usar diferentes conjuntos de treinamento. Esta última é a técnica mais comumente utilizada para essa finalidade.

Os algoritmos desta última técnica podem ser subdivididos em 2: algoritmos que mudam adaptativamente a distribuição do conjunto de treinamento baseado na performance dos classificadores anteriores e algoritmos que não adaptam a distribuição, *boosting* é um dos exemplos mais

representativos do primeiro grupo. Os métodos de *boosting* mais utilizados são os da família Adaboost.

Bagging é o algoritmo mais representativo do segundo grupo. Bagging simplesmente gera diferentes conjunto de *bootstrap* do conjunto de treinamento. Vários estudos mostraram que Adaboost é capaz de reduzir do erro geral os componentes de bias e variância,mas também mostram que *Bagging* é mais eficiente em reduzir bias que Adaboost.

Boosting altera a distribuição do conjunto de treinamento para obter o *ensemble*. Essa alteração ocorre iterativamente. Depois que o classificador é treinado, a distribuição das instâncias é atualizada considerando o erro do último classificador adicionado ao *ensemble*, ou dos últimos r classificadores adicionados em alguns métodos de *boosting*, e então uma nova iteração é realizada.

Na metodologia proposta, em cada iteração, um algoritmo de seleção de instâncias é executado utilizando a distribuição não-linear do conjunto de treinamento dada pelo peso das instâncias. Depois que o processo de seleção é realizado o algoritmo deve atualizar a distribuição das instâncias. Para atualizar a distribuição o método usa o subconjunto das instâncias selecionadas e classifica todas as instâncias de treinamento através de uma regra como o K-NN. A atualização é então definida pelo erro deste classificador, e então uma nova iteração é realizada.

Quando ensembles são construídos utilizando *boosting* os pesos das instâncias podem ser utilizados de duas maneiras. Se o algoritmo de classificação aceita pesos de instâncias, eles podem ser diretamente alimentados no processo de treinamento. Os pesos também podem ser utilizados para obter amostras de treinamento através de amostragem por pesos, onde instâncias com mais pesos são amostradas com mais frequência. O primeiro método é conhecido como *reweighting*e o último como *resampling*. A maioria dos algoritmos não aceita pesos para as instâncias, então este trabalho focou na técnica de *resampling*. Levando isso em conta, amostrar uma instância mais de uma vez não faz muito sentido para algoritmos de seleção de instâncias. Então primeiro, nós obtemos uma amostra utilizando os pesos como distribuição,

e então nós removemos as instâncias repetidas antes de aplicar o algoritmo de seleção. Este método tem a vantagem de reduzir o conjunto de treinamento em cada passo, tornando o algoritmo mais rápido.

A grande diferença em relação ao *boosting* clássico e ao *boosting* no contexto de seleção de instâncias é que no boost*i*ng clássico nós obtemos a combinação de diferentes classificadores que foram construídos iterativamente pelo processo. Já em *boosting* aplicado a seleção de instâncias nós obtemos um subconjunto das instâncias obtido através de votação feita pela seleção realizada pelo algoritmo de redução a cada passo.

Em cada passo de *boosting*, a seleção feita é armazenada através de um voto dado a cada instância selecionada. Os votos recebem pesos quando o algoritmo de *boosting* usa classificadores com peso como parte do *ensemble*. Uma vez que o processo de *boosting* termina, nós temos um vetor de votos que grava quantas vezes cada instância foi selecionada. Um dos problemas que temos que resolver é como obter um limiar de seleção que é o equivalente ao limiar utilizado em *ensemble* de classificadores que determina qual a classe em ensemble de classificadores binários. Esta tarefa será descrita na próxima seção.

A figura 1 apresentada a seguir possui uma descrição breve do processo de seleção por *boosting*.

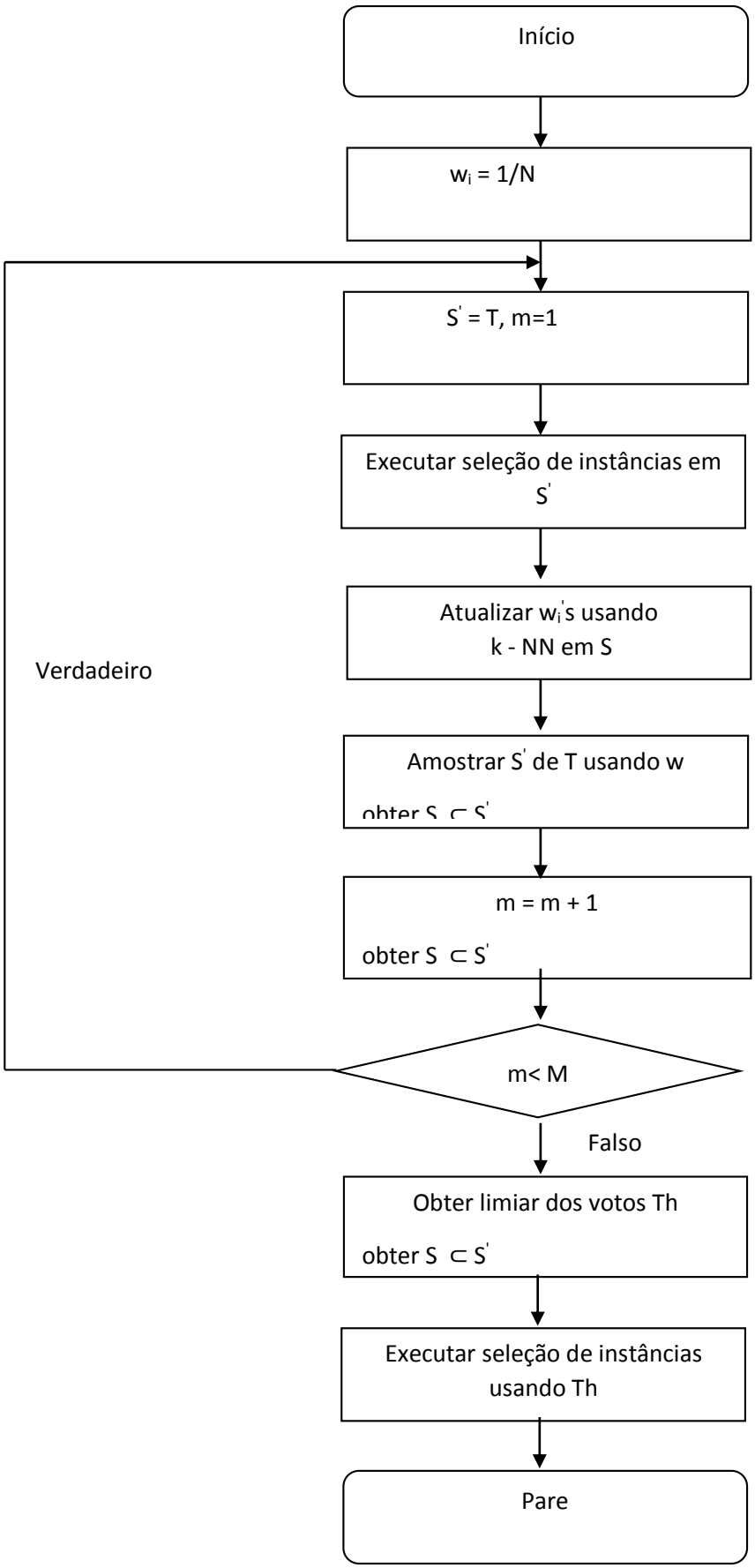

Figura1- ProcessoGeral

2.1 Escolha do limiar

O último passo do algoritmo é combinar a seleção de instâncias anteriores para selecionar o subconjunto final. Como o resultado das M iterações de *boosting*, nós temos um vetor de votos(com pesos) que indicam quantas vezes aquela determinada instância foi selecionada. Em *ensembles*, a maneira mais utilizada para obter a classe de uma nova instância é *winner-takes-all*. Adaptando este modelo nós podemos selecionar instâncias cujo número de votos indica que elas foram selecionadas com mais frequência.

Apesar deste método ser simples e rápido, ele não alcança bons resultados e se associa a um grande problema. Fixar um limiar de maioria de votos ignora o fato de que normalmente uma grande percentagem de instâncias são removidas a cada passo. O número de instâncias que são selecionadas em mais do que a metade das iterações é muito baixa, significando que a saída final do processo de *boosting* é a seleção de poucas instâncias que implicam em baixa performance. Além disso, é improvável que qualquer limiar fixo seja apropriado para todos os conjuntos de treinamento possíveis.

Então, se faz necessária a utilização de um limiar dinâmico. Dado um vetor final de votos v, que armazena os votos obtidos para cada instânciax_i, e um dado limiar, θ, selecionamos as instâncias cujos votos estão acima do limiar dado: $v_i > \theta$. Para obter o melhor limiar nós definimos um critério, $J(\theta)$, e avaliamos todos os possíveis limiares.

O processo é descrito pela figura-2 apresentada a seguir. Para avaliar um dado limiarθ, nós primeiros selecionamos o subconjunto,S_θ, de instâncias do conjunto de treinamento usandoθ: $S_\theta = \{x_i \in T : v_i > \theta\}$. Então, o critério $J(\theta)$ é avaliado utilizando S_θ para treinar um classificador, que neste documento será o k-nn. Para J, nós escolhemos um critério simples, mais especificamente uma combinação linear entre performance e redução. A qualidade de uma redução é determinada por dois fatores: sua performance e sua habilidade de reduzir. O critério então leva em consideração estes dois fatores:

$$J(\theta) = \alpha * performance + (1 - \alpha) * reducao$$

A performance de classificação pode ser medida de maneiras diferentes, dependendo do problema. Para este documento, a acurácia do classificador foi utilizada como medida de performance. Redução é sempre a medida de porcentagem de instâncias removidas. αé parâmetro utilizado para balancear a importância relativa destes fatores. Na maioria dos casos a performance de classificação é mais desejável que redução. O algoritmo é mostrado a seguir:

Entrada: Conjunto de treinamento $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, um vetor de votos v e um vetor de limiares possíveis $\{l_1, l_2, \dots, l_m\}$

Saída: O limiar com melhor avaliação

$J_b = 0$

$J_b = 0$
$for\ l = l_1\ to\ l_m\ do$
$S_l = \{x_i \in T : v_i > l\}$
$\quad c = performance\ do\ classificador\ k - NN\ usando\ S_l\ como\ cjto.de\ treinamento$
$\quad r = 1 - \frac{|S_l|}{N}$
$\quad J(l) = \alpha c + (1 - \alpha) r$
$\quad if\ J(l) \geq J_b\ then$
$J_b = J(l)$
$\quad \theta = l$
$return\ \theta$

Figura2 - Seleção de limiar

O conjunto de possíveis limiares, $\{\theta_1, \theta_2, \dots, \theta_n\}$ inclui todos os diferentes valores no vetor de votos. Mas, para balancear o processamento, apenas um subconjunto destes possíveis valores devem ser considerados, pois limiares com valores similares vão obter avaliações similares em termos de classificação e performance.

A complexidade deste passo deve ser considerada para estabilizar seu efeito na complexidade geral do algoritmo. A avaliação de cada limiar envolve obter o erro do classificador k-NN usando as instâncias selecionadas. Então, a complexidade por iteração é a mesma que a do k-NN. O máximo número de limiares considerados deve ser, então, o mesmo número de iterações. De fato, a complexidade é reduzida porquê nem todas as instâncias devem ser reavaliadas em termos de distâncias para seus vizinhos próximos.

2.2 Algoritmos de *boosting*

Nesta sessão, são apresentadas descrições detalhadas dos métodos utilizados nos experimentos. Apesar dos métodos de boosting padrões serem aplicados com quase nenhuma modificação, todos os algoritmos são mostrados por questão de clareza. Nem todo método de *boosting* pode ser adaptado para o problema de seleção de instâncias. Algoritmos de *boosting* que podem ser utilizados devem preencher dois quesitos:

1. O espaço de entrada não deve ser modificado. A seleção final das instâncias vai ser realizada no espaço original de características. Métodos que constroem ensembles que modificam o espaço de entrada, como os baseados em projeção não lineares[7],florestas de rotação[8], subespaços aleatórios [9,10]
2. A saída não deve ser modificada. O processo de aprendizado é um de seleção de instâncias e não de classificação. Então, qualquer método que modifique a saída não vai fazer sentido. Logo, métodos como Brownboost[11]ouLogiBoost[12]não se aplicam.

Para evitar a restrição de banco de dados, os métodos de *boosting* feitos para problemas binários não foram considerados. Então métodos como ModestAdaboost[13] GentleBoost[14]não foram utilizados.

Mesmo com as restrições descritas acima muitos métodos de *boosting* podem ser utilizados. Nos experimentos, foram utilizados Adaboost, Madaboost, MultiBoost e ReweightBoost.

Todos os métodos de *boosting* devem avaliar o erro a cada passado usando o vetor de pesos w. Essa avaliação é feita usando o classificador k-NN e a seleção é feita no próximo passado. Os algoritmos mostrados abaixo, este classificador é representado por C^{S_m}.

2.3 Adaboost e Madaboost

A versão do adaboost utilizada aqui é de [15]. Este algoritmo recebe um conjunto de treinamento $T = \{(x_1, y_1), \dots, (x_n, y_n)\}$, um algoritmo de seleção de instâncias, IS, e o número de iterações M, e retorna a seleção final: $S \subset T, S = \{x_i \in T : \sum_{m=1}^{M} \alpha_m * S_m^i > \theta$.

Esta é uma extensão padrão do algoritmo original Adaboost para problemas multi-classe.$s_M$salva a seleção de instâncias feita no passo m, se $s_M^I = 1$ então a instância x_i foi selecionado no passo m, caso contrário um valor de 0 será encontrado.

Adaboost, assim como *bagging*, forma um *ensemble* de classificadores aplicando um algoritmo de aprendizado básico sucessivamente à conjuntos de treinamento derivados através de amostragem do conjunto de treinamento base. Quando o primeiro conjunto T_1 de treinamento é formato, cada peso é inicializado para 1/n(n é o tamanho do conjunto). Em geral, o peso w_i^j para a instância i na iteração j é uma função da performance de classificadores anteriores $\{C^{S_1}, C^{S_2}, \dots, C^{S_{m-1}}\}$ nos conjuntos de treinamentos derivados anteriormente $\{T_1, T_2, \dots, T_{m-1}\}$. O algoritmo Adaboost utiliza estes pesos durante a amostragem, formando um conjunto de amostra *bootstrap* da distribuição do conjunto de treinamento definida pelos pesos.

O algoritmo apresentado é uma variante de de Bauer e Kohavi[15]. Esta variante é escolhida porque:

1. Usa um passo de atualização de peso que é menos sujeito a underflow numérico que o processo original de dois passos.
2. Impede underflow numérico
3. Continua produzindo mais membros do ensemble depois do ponto em que e >= 0.5. Esta abordagem melhora performance preditiva de acordo com [16, 17].

$S_1 = T, com todos os pesos w_i = \frac{1}{N}$
$for m = 1\ to M do$
$S_m = IS(S')$
$e_m = \sum_{x_j \in T: C^{S_m}(x_j) \neq y_j} w_j$
$if\ e_m > 0.5\ then$
$a_m = 0$
$S' = bootstrap(T)$
$Reset W para \frac{1}{N}$
$Remover instâncias repetidas de S'$
$else if\ e_m = 0\ then$
$a_m = 10$
$S' = bootstrap(T)$
$Reset W para \frac{1}{N}$
$Remover instâncias repetidas de S'$
$else$
$a_m = \frac{1}{2} \ln \frac{1 - e_m}{e_m}$
$foreach\ x_j \in T do$
$if C^{S_m}(x_j) = y_j then$
$w_j = \frac{w_j}{2(1 - e_m)}$
$else$
$w_j = \frac{w_j}{2e_m}$, esta linha só é executada para o Adaboost
$Normalizar\ w\ para\ que\ a\ soma\ dos\ elementos\ seja\ 1$
$Obter\ S' como\ uma\ amostra\ com\ repetição\ de\ T\ usando\ w$
$Remover\ instâncias\ repetidas\ de\ S'$

Figura3 – Adaboost&Madaboost

Note uma pequena modificação. Apesar de em [15] pararem a execução e tratar $C^{S_{m-1}}$, como possuindo peso infinito se e = 0, o algoritmo apresentado atribui o valor de a_m para o valor 10 nesta circunstância e reamostra do conjunto de treinamento para reiniciar o processo de boosting. Isto permite a possibilidade de que mais de um membro do ensemble seja derivado com erro 0, habilitando estes a votarem um conta o outro e também sendo utilizados para critério de desempate.

A diferença entre o Adaboost e o Madaboost é a inclusão da linha 21 como descrito na figura 3.

2.4 Multiboost.

As observações de que *bagging* e Adaboost parecem operar por diferentes mecanismos, tem diferentes efeitos, e que ambos possuem os melhores efeitos obtidos através do treinamento dos primeiros membros do *ensemble*, sugerem que eles podem ser combinados de modo a obter melhorias. Como os mecanismos diferem, a combinação deles pode aumentar a performance de cada outro separadamente. Dado que:

1. *Bagging* remove principalmente variância, enquanto Adaboost reduz tanto bias como variância e;
2. Existe evidência de que *bagging* é mais efetivo que Adaboost em reduzir variância [17]

A combinação dos dois pode ser capaz de reter a redução do *bias* proposta por Adaboost e adicionar a redução de variância proposta por bagging. Como a maior parte do efeito dos dois é obtido pelos primeiros membros do *ensemble* é possível que mesmo pequenos *ensembles* formados pela combinação das abordagens possam obter o melhor benefício de cada abordagem isoladamente.

Um método para combinar as duas abordagens pode ser simplesmente desenvolver dois ensembles, um contendo membros formados por boost*i*ng e o outro contendo membros formados por *bagging*. Mas, isto levanta uma pergunta não trivial de como os votos dos ensembles devem ser combinados. Adaboost atribui um peso aos votos dos ensembles enquanto *bagging* não o faz, fazendo com que os votos se tornem imensuráveis. Uma maneira mais interessante de resolver este problema é utilizar *bagging* num subconjunto de *ensembles* cada um formado por uma aplicação de Adaboost. Ao invés de utilizar *bagging* diretamente, o que reduziria o número de exemplos de treinamento disponíveis para formar cada subconjunto do ensemble, *wagging* é mais recomendado de acordo com [17]. Manter todos os exemplos no conjunto de treinamento é visto como importante, de acordo com [18] Adaboost se beneficia, utilizando *reweighting*, do acesso a todas as instâncias de treinamento durante a indução de cada membro do ensemble.

Resumindo, MultiBoosting é um método composto pela aplicação de *wagging*(variante de *bagging*) ensembles formados por Adaboost. Uma decisão

deve ser tomada para determinar quantos subconjuntos do ensemble devem ser gerados durante uma iteração, e o tamanho destes subconjuntos.

O método de multiboost é demonstrado a seguir. O método periodicamente reinicializa os pesos utilizando a distribuição de Poisson aproximada por:

$$-\log(\frac{Random(1,999)}{1000})$$

Onde Random(1, 999) retorna um inteiro entre 1 e 999.

Para atribuir os passos quando os pesos são reinicializados, um vetor de inteiros, I, é construído. Os valores do vetor são $I_i = i * \frac{M}{n}, para\ i = 1, 2, \dots, n-1$ e$I_i = M\ para\ i \geq n$ onde n é o valor inteiro obtido em $n = \sqrt{M}$. Deste modo multiboost pode ser pensado como uma combinação de Adaboost e Wagging.

O algoritmo é apresentado na figura 4 a seguir.

$S_1 = T, comtodosospesosw_i = \frac{1}{N}$

$k = k + 1$

$form = 1\ toMdo$

$if\ I_k = m\ then$

$Reset\ w\ com\ os\ pesos\ de\ uma\ distribuição\ continua\ de\ Poisson$

$k = k + 1$

$S_m = IS(S')$

$e_m = \sum_{x_j \in S: C^{S_m}(x_j) \neq y_j} w_j$

$if\ e_m > 0.5\ then$

$a_m = 0$

$S' = bootstrap(T)$

$Reset\ w\ com\ os\ pesos\ de\ uma\ distribuição\ continua\ de\ Poisson$

$k = k + 1$

$elseif\ e_m = 0\ then$

$a_m = 10$

$S' = bootstrap(T)$

$Reset\ w\ com\ os\ pesos\ de\ uma\ distribuição\ continua\ de\ Poisson$

$else$

$a_m = \frac{1}{2}\ln\frac{1 - e_m}{e_m}$

$foreach\ x_j \in Sdo$

$if C^{S_m}(x_j) = y_j then$

$w_j = \frac{w_j}{2(1 - e_m)}$

$else$

$w_j = \frac{w_j}{2e_m}$

$if\ w_j < 10^{-8}\ \ then\ w_j = 10^{-8}$

$Normalizar\ w\ para\ que\ a\ soma\ dos\ elementos\ seja\ 1$

$Obter\ S'como\ uma\ amostra\ com\ repetição\ de\ T\ usando\ w$

$Remover\ instâncias\ repetidas\ de\ S'$

Figura4 - Multiboost

2.5 ReweightBoost

De acordo com [19]para melhorar a performance de um classificador obtido com Adaboost e um algoritmo de aprendizado, existem essencialmente duas abordagens:

1. Adicionar mais classificadores fracos.
2. Usar classificadores básicos mais complexos.

Estas abordagens sofrem dos seguintes problemas: a construção do classificador é lenta, o uso do classificador é lento e o classificador precisa de mais memória.

Então, se torna interessante ter a possibilidade de obter classificadores mais precisos sem incrementar complexidade computacional nem requisitos de armazenamento.

O método proposto em [19] se baseia na combinação de vários classificadores fracos em um classificador não fraco. O método possui um parâmetro, r, o nível de reuso. É o número de classificadores das iterações anteriores que serão utilizados. Por exemplo, se r = 5 e o número de iterações é 7, sete classificadores serão gerados. Sendo os cinco primeiros gerados utilizando todos os classificadores até então. E os últimos dois utilizando os cinco últimos classificadores gerados. O algoritmo é dado na figura 5 a seguir

$S' = T, comtodosospesos w_i = \frac{1}{N}$
$form = 1\ toMdo$
$S^m = IS(S')$
$H_m^r = C^{S_m} + C^{S_{m-1}} + C^{S_{m-2}} + \cdots + C^{S_{\max(m-r,1)}}$
$e_m = \sum_{x_j \in T: C^{S_m}(x_j) \neq y_j} w_j$
$a_m = \frac{1}{2} \ln \frac{1 - e_m}{e_m}$
$foreach\ x_j \in Tdo$
$w_j = w_j e^{-a_m I(C^{S_m}(x_j) \neq y_j)}$, *onde I é a função indicadora*
Normalizar w para que sua soma seja 1
Obter S' como uma amostra com repetições de T usando w
Remover instâncias repetidas de S'

Figura 5 - Reweightboost

Capítulo 3 - Algoritmos de seleção de instâncias.

São muitos os algoritmos que se destinam a resolver o problema de seleção de instâncias. Alguns destes, que são amplamente conhecidos na literatura, serão apresentados nesta seção. Todos os algoritmos presentes foram implementados e utilizados para a avaliação da técnica apresentada neste trabalho. É importante notar também que a avaliação de algoritmos de seleção de instâncias não é o foco deste trabalho, logo uma visão menos abrangente será abordada neste capítulo

Para critério de organização e entendimento foram adotadas algumas notações:

1. T é o conjunto de treinamento cuja classificação é conhecida.
2. S é o conjunto final de instâncias selecionadas pelo algoritmo

3.1 CNN

A regra CNN, é uma das regras ad hoc que apareceram na literatura, sua criação teve como motivação considerações estatísticas pertinentes a regra do vizinho próximo(Nearest-Neighbour). Para entender seu funcionamento se faz necessário o entendimento do NN, então uma breve revisão do NN será tomada e logo em seguida será explicado o funcionamento do CNN.

A regra NN classifica uma instância com a mesma classe da instância mais próxima a ela que já se tem conhecimento, ou seja, dado um conjunto de treinamento com n instâncias classificadas, uma nova instância recebe a classe da instância mais próxima do conjunto. Uma das propriedades mais interessantes da NN é que de acordo com [20]sob algumas hipóteses de regularização simples sobre a estatística do processo, para qualquer métrica e para uma variedade de funções de custo, o risco incorrido é menor que duas vezes o risco de Bayes. Do ponto de vista prático, o NN pode se tornar impraticável para muitas aplicações devido ao seu alto custo de armazenamento. A regra CNN foi proposta de modo a manter a abordagem básica e evitar o custo alto de armazenamento.

A regra CNN define o conceito de subconjunto consistente. O subconjunto consistente é nada mais do que o subconjunto do conjunto de treinamento que quando utilizado como base na regra NN classifica todas as instâncias do conjunto de treinamento corretamente. Um subconjunto mínimo consistente é um subconjunto consistente com a menor quantidade de elementos. Em geral, o algoritmo nem sempre encontrará o menor.

O algoritmo procede com S contendo a primeira instância de T. Em seguida cada elemento de T é classificado individualmente utilizando o conjunto S, à medida que uma determinada instância é classificada erroneamente ela é adicionada à S. O processo é repetido até que todos os dados sejam classificados corretamente

O CNN possui problemas ao trabalhar com dados ruidosos, pois como os ruídos não são classificados corretamente pelo conjunto de treinamento, na maioria dos casos, eles são adicionados ao conjunto final. Como há uma redução nos dados em si, significa que a razão ruído/dado aumenta.

O CNN na maioria das vezes assegura a classificação correta de todo conjunto T, mas alguns casos específicos faz com que o algoritmo não termine(dependendo da implementação) ou gere um conjunto final não considerado consistente. Além disso, não há uma garantia de que o conjunto final seja mínimo.

Utilizar uma permutação aleatória de T.

$S = \{\}$

$foreach\ t_i \in T\ do$
$\quad if\ class(t_i)\ \neq class(elemento\ mais\ próximo\ de\ t_i\ em\ S) then$
$S = S \cup \{t_i\}$

Figura6 - CNN

3.2 RNN

Alguns podem se perguntar se o procedimento do CNN é eficiente reduzindo o tamanho do conjunto de treinamento ou se encontra um subconjunto consistente mínimo. As respostas dessas perguntas foram respondidas por GATES [21] que mostrou que apesar do CNN mostrar reduções por volta de 90%

ainda é possível reduzir a maioria dos resultados. A técnica introduzida é chamada RNN(ReducedNearestNeighbour) e nos experimentos conseguiu obter taxas melhores que o CNN em todos os casos. Isto significa que o CNN não seleciona, na maioria das vezes, o subconjunto mínimo. O quão distante do mínimo ele está é uma pergunta difícil de ser respondida.

O algoritmo pode ser visto como uma extensão do CNN. Dado que o CNN gere um subconjunto consistente, este subconjunto consistente é utilizado pelo RNN para encontrar o subconjunto mínimo. Apesar do RNN ser bem sucedido na maioria dos casos, ele nem sempre encontra o conjunto mínimo.

A técnica tem como entrada o conjunto gerado pelo algoritmo CNN. A cada iteração é removida uma instância do conjunto e é checada se essa remoção altera a correta classificação do conjunto de treinamento original. Se houver erro de classificação, a instância é reinserida e o processo continua. Até que todas as instâncias tenham sido testadas.

Em termos de complexidade este algoritmo é mais caro que o CNN, mas na grande maioria das vezes gera um conjunto menor que o do CNN. Então seu preço é pago na economia do tempo de classificação.

$S = CNN(T)$
$foreach\ s_i \in S do$
$S = S - \{s_i\}$
$foreach\ t_i \in T do$
$if classe(t_i) \neq classe(vizinho de t_i em S)\ then$
$\quad S = S \cup \{s_i\}$
$\quad break$
$return\ S$

Figura 7 - RNN

3.3 MSS

Criado por BARANDELA [22] o algoritmo é uma extensão do algoritmo de Ritter [23] que por sua vez é uma extensão do CNN. O algoritmo é uma modificação para encontrar um subconjunto seletivo. Um subconjunto é seletivo se é consistente e todas as instâncias do conjunto de treinamento original estão

próximas a uma instância seletiva da mesma classe do que a qualquer instância do conjunto de treinamento de outra classe.

Apesar de não haver nenhuma prova, é muito provável que o algoritmo de Ritter encontra o menor subconjunto seletivo porque realiza uma busca exaustiva implícita. De fato, foi comprovado que o problema de selecionar o menor subconjunto é NP-Completo[24]. Apesar de desafiador, foi demonstrado que minimalidade nem sempre é uma boa propriedade para problemas reais na prática. Talvez um subconjunto um pouco maior de instâncias pode representar mais fielmente as fronteiras de decisão ótimas do problema original. Apesar disto, um método de redução deve sempre balancear uma boa redução no tamanho do conjunto com uma boa performance de classificação resultante.

O MSS se baseia nas 3 seguintes ideias:

1. Uma instância x_j é um vizinho relativo de outra instância x_i, ambas da mesma classe, se x_j é mais próxima de x_i do que o inimigo mais próximo de x_i;
2. O conjunto de todos os vizinhos relativos de uma instância x_i é representado por Y_i, a vizinhança relativa de x_i.
3. O subconjunto seletivo modificado é definido como um subconjunto do conjunto de treinamento que contém, para cada x_i pertencente ao conjunto de treinamento, aquele elemento de Y_i que está mais próximo de uma classe diferente da de x_i (ou seja, mais próximo do inimigo mais próximo).

O algoritmo segue na figura seguinte:

$$
\begin{aligned}
&Ordenar\ as\ instâncias\ T_i\ de\ acordo\ com\ D_i(distância\ para\ inimigo\ mais\ próximo\\
&S = T\\
&for\ i = 1\ to\ N\ do\\
&\quad add = false\\
&\quad for\ j = i\ to\ N\ do\\
&\qquad if\ t_j \in S \wedge d(t_i, t_j) < D_j\ then\\
&\qquad\quad S = S - \{t_j\}\\
&\qquad\quad add = TRUE\\
&\quad if\ add\ then\ MSS = MSS \cup \{t_i\}\\
&\quad if\ S = \emptyset\ then\ break\\
&return\ MSS
\end{aligned}
$$

Figura8 - MSS

3.4 DROP3

O DROP3 é um dos algoritmos da família DROP, previamente apresentados como RT criados por Wilson & Martinez [25]. Ele foi proposto como uma melhoria no algoritmo DROP2.

O DROP2 ordena T numa tentativa de remover pontos centrais antes de pontos de borda. Um problema com este método é que instâncias ruidosas também são pontos de “borda”, e fazem com que a ordem de remoção mude drasticamente. Uma instância ruidosa no centro de um cluster pode fazer com que muitos pontos daquele cluster sejam considerados pontos de borda, e alguns deles pode ficar em S mesmo após o ponto ruidoso ser removido.

DROP3 se utiliza de uma iteração para filtragem de ruído antes de ordenar as instâncias em S. Isto é feito utilizando uma regra similar a ENN. Qualquer instância que é classificada erroneamente pelos k vizinhos mais próximos é removida. Este processo remove instâncias ruidosas, assim como pontos de borda, fazendo com que os pontos de decisão se tornem suaves. Isto impede de acontecer overfitting dos dados.

Após remover as instâncias ruidosas de S desta maneira, as instâncias são ordenadas por distância para seu inimigo mais próximo em S, deste modo pontos distantes da real fronteira de decisão são removidos primeiro. Isto faz com que pontos internos a clusters possam ser removidos no início do processo, mesmo que haja pontos ruidosos por perto.

O algoritmo é dado a seguir:

$S = T$
$foreach\ s_i \in S\ do$
    $if\ class(s_i) \neq classe(k - vizinhos\ mais\ próximos\ em\ T)$
        $S = S - \{s_i\}$
$foreach\ s_i \in S\ do$
    $encontrar\ os\ k + 1 - vizinhos\ mais\ próximos\ de\ s_i em\ S\ \neq s_i$
    $Adicionar\ s_i$ à $lista\ de\ associados\ de\ cada\ k + 1\ vizinhos$
$Ordenar\ S\ de\ acordo\ com\ a\ distância\ para\ o\ inimigo\ mais\ próximo. Decrescente.$
$foreach\ s_i \in S\ do$
    $x = número\ de\ associados\ classificados\ corretamente\ utilizando\ S$
    $y = número\ de\ associados\ classificados\ corretamente\ utilizando\ S - \{s_i\}$
    $if\ y \geq x\ then$
        $foreach\ a_i \in Associado(s_i)$
            $encontrar\ um\ novo\ vizinho\ v_i\ para\ a_i$
            $Associado(s_i) = Associado(s_i) \cup \{v_i\}$
        $S = S - \{s_i\}$
$return\ S$

Figura9 - Drop3

3.5 ISAR

Método proposto por Cristiano e George [26], foi criado tendo em foco as seguintes propriedades: tolerância a ruído, alta performance, alta redução e insensibilidade a ordem de apresentação dos dados. É composto de quatros passos principais: remoção de ruídos, construção de ranking, seleção de instâncias e processo de classificação.

ISAR aplica um tipo de remoção de ruídos como um passo de pre-processamento. O algoritmo de filtragem remove do conjunto de treinamento qualquer instância que é classificada erroneamente pelos k vizinhos mais próximos, assim como o DROP3. Este filtro remove instâncias ruidosas e pontos

de borda. O algoritmo é dado a seguir:

$S = T$
$foreach\ s_i \in S\ do$
    $if\ class(s_i) \neq classe(k - vizinhos\ mais\ próximos\ em\ T)$
        $S = S - \{s_i\}$
$T = S$
$R = \{\dots\}$
$Z = \{0, 0, \dots, 0\}$
$foreach\ t_i \in T\ do$
    $foreach\ t_{j:j\neq i} \in T\ do$
$z_j = z_j + \delta(t_i, t_j) * net(t_j, t_i, T)$
$T = sort(T, Z), ordena\ T\ de\ forma\ decrescente\ em\ S$
$foreach\ t_i \in T\ do$
    $if\ isRlvt(t_i, R)\ then$
        $R = R \cup \{t_i\}$
$return\ R$

Figura10 - ISAR

Onde a função net é dada por:

$$net(t_j, t_i, T) = \frac{\exp\left(-d(t_i, t_j)\right)}{\sum_{k=1:k\neq j}^{N} \exp(-d(t_j, t_k))}$$

E a função delta é dada por:

$$\delta(t_i, t_j) = \begin{cases} 1, class(t_i) = class(t_j) \\ -1, casocontrário \end{cases}$$

Uma instânciat_i é considerada relevante para o método ISAR se não existe t_r tal que:

$$class(t_r) = class(t_i) \wedge d(t_i, t_r) \leq radius(t_r)$$

A classificação de uma instância não conhecida é dada pela atribuição da classe da instância que satisfaz:

$$n = \arg\min \frac{d(t, t_r)}{raio(t_r)}$$

3.6 CRIS

Algoritmo proposto por Cristiano e George [27]tomando como base o ISAR. O algoritmo é composto por três partes: cálculo de ranking, cálculo de raio e seleção de instâncias.

O algoritmo é mostrado a seguir

$Y = \emptyset$
$S = \{0, 0, \dots, 0\}$
$R = \{0, 0, \dots, 0\}$
$foreach\ x_i \in X\ do$
  $foreach\ x_{j:j\neq i} \in X\ do$
    $if\ class(x_j) = class(x_i)\ then$
      $\delta = 1$
    $else$
      $\delta = -1$
$s_j = s_j + \delta * \frac{\exp\left(-d(x_i, x_j)\right)}{\sum_{k=1:k\neq j}^{N} \exp\left(-d(x_j, x_k)\right)}$
$foreach\ s_i \in S\ do$
$s_i = \frac{s_i - \min(S)}{\max(S) - \min(S)}$
$for\ i = 1\ to\ |X|\ do$
$r_i = \frac{s_i}{s_i + s_{nei}} * d(x_i, x_{nei})$
$I = rank(S)$
$foreach\ i \in I\ do$
  $if \nexists y \in Y \mid (x_i \in coverageSet(y))\ then$
    $Y = Y \cup \{x_i\}$
$return\ Y$

Figura 11 - CRIS

exponencial normalizada, sendo incrementado caso as instâncias sejam da mesma classe e decrementada caso contrário. Todas as instâncias de T são avaliadas. Após o cálculo de todos os scores eles são normalizados entre 0 e 1. Os raios das hiperesferas são então computadas para cada instância tomando como base a distância e o score do inimigo mais próximo. As instâncias são então ordenadas por ordem decrescente de score para dar início ao processo de seleção. A seleção é dada iterativamente, uma instância é selecionada se e somente se ela não está contida na hiperesfera de alguma instância já selecionada que possua a mesma classe.

Capítulo 4 - Experimentos

Para fazer uma comparação justa entre os algoritmos foram selecionados 15 conjuntos da plataforma UCI Machine Learning Repository. Uma descrição das bases é dada na tabela 1.

Em relação aos parâmetros dos algoritmos, cada algoritmo tem os mesmos parâmetros recomendados pelos autores. Em relação ao *boosting* é utilizado um tamanho de ensemble M = 30, que é um valor comum. Para Reweightboost foi utilizado um valor de r = 5. Os algoritmos de seleção utilizados não têm hiperparâmetros associados. O valor αusado na combinação linear de seleção de limiar foi $\alpha = 0.75$.

A técnica utilizada para validar os valores foi a *stratified* k-*fold cross validation*, ela foi empregada em todos os testes realizados com o valor de k = 10. A técnica consiste em subdividir o conjunto inicial em partes com aproximadamente o mesmo tamanho e a mesma distribuição de classes. Em seguida unimos nove partes para utilizar de treinamento e a parte restante para utilizar como teste. Este processo é repetido dez vezes até que todas as partes tenham sido utilizadas para testes. A taxa de erro e redução são as médias aritméticas das dez iterações.

Tanto para classificação como para a determinação dos limiares e também para a iteração de *boosting* foi utilizado o classificador k-NN com k = 3 com exceção dos algoritmos ISAR, CNN e MSS que utilizaram k = 1.

Foram utilizadas duas medidas de distância: distância euclidiana e a *heterogeneous euclidean overlap metric* (HEOM). A distância euclidiana foi utilizada quando os bancos possuíam todos atributos reais. A HEOM foi utilizada quando o banco possuía algum dado categórico ou quando ele possuía dados não informados.

A medida HEOM é dada por:

$$HEOM(x, y) = \sqrt{\sum_{i=1}^{m} d_i(x_i, y_i)^2}$$

Ela é uma medida heterogênea que trabalha tanto com atributos numéricos como categóricos. A função de distância é dada por:

$$d_i(x_i, y_i) = \begin{cases} 1, x \text{ ou } y \text{ desconhecidos} \\ overlap(x_i, y_i) \text{ se for nominal} \\ \dfrac{|x_i - y_i|}{range\ de\ i} \end{cases}$$

E a função overlap é dada por:

$$overlap(x_i, y_i) = \begin{cases} 0, x = y \\ 1, caso\ contrário \end{cases}$$

Todos os bancos tiveram seus valores reais normalizados entre 0 e 1.

Tabela1- Bases selecionadas

Base	Instâncias	Reais	Categóricas	Inputs	Classes	Missing?
anneal	798	6	32	38	6	S
arrhythmia	452	279	0	279	16	S
balance	625	4	0	4	3	N
card	690	6	9	15	2	S
ecoli	336	7	0	7	8	N
haberman	306	3	0	3	2	N
heart	270	13	0	13	2	N
ionosphere	351	34	0	34	2	N
liver	345	6	0	6	2	N
glass	214	10	0	10	7	N
pima	768	8	0	8	2	S
wine	178	13	0	13	3	N
tic-tac-toe	958	0	9	9	9	N
vehicle	946	18	0	18	4	N
vote	435	0	16	16	2	S

4.1 Resultados

Os resultados a seguir são dispostos por algoritmos.

4.1.1 CNN

Tabela2 - CNN

	CNN		
Base	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	96.54	90.34	17
Arrhythmia	52.91	45.88	25
Balance	70.94	71.6	5
Ionosphere	86.46	81.04	2
Liver	61.5	56.32	1
Tic-Tac-Toe	89.62	78.32	11
Vehicle	65.44	60.38	10
Pima	68.47	63.93	10
Card	75.1	72.43	10
Heart	73.52	69.78	2
Ecoli	74.76	71.66	1
Haberman	61.03	59.99	1
Glass	63.18	59.65	0.9
Wine	90.77	88.15	0.3
Vote	91.05	88.5	2
	73.5885714	69.2479	6.55

Tabela3 - Adaboost&Madaboost com CNN

	Adaboost			Madaboost		
Base	Precisão(%)	Redução(%)	Tempo(ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	96.29	80.22	8069	96.41	73.24	18149
Arrhythmia	56.78	37.07	14038	58.51	34.44	17809
Balance	71.61	61.76	1981	71.73	60.92	3326
Ionosphere	87.17	75.79	539	86.32	70.63	934
Liver	62.23	41.74	569	61.57	39.53	723
Tic-Tac-Toe	89.74	74.88	9741	90.41	66.64	27048
Vehicle	66.37	48.27	8535	66.5	37.34	12783
Pima	71.39	47.23	9439	71.18	45.40	13962
Card	77.38	61.61	7507	77.64	56.26	12711
Heart	77.41	52.17	592	78.08	47.06	889
Ecoli	75.81	62.86	303	76.19	60.25	459
Haberman	63.73	38.53	393	65.78	37.32	453
Glass	69.27	46.56	162	65.84	41.76	209
Wine	89.04	82.52	37	92.25	78.56	73
Vote	91.29	78.22	895	93.81	76.06	1416
	76.3673333	59.2953	4186.67	76.81	55.02	7396.27

Tabela4 - Multiboost e ReweightBoost com CNN

	Multiboost			ReweightBoost		
Base	Precisão(%)	Redução(%)	Tempo(ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	95.43	84.24	13695	96.10	77.33	20028
Arrhytmia	55.81	29.89	17873	57.80	7.76	15234
Balance	72.34	59.17	3388	75.14	64.81	3666
Ionosphere	85.67	76.12	831	87.71	71.16	922
Liver	60.61	46.02	703	61.39	21.57	647
Tic-Tac-Toe	88.44	73.73	21008	89.41	68.55	27909
Vehicle	67.28	51.09	12818	68.44	46.22	10806
Pima	70.74	46.40	13792	70.70	47.64	10237
Card	77.6	58.68	12175	75.88	68.62	9975
Heart	75.82	55.28	851	75.68	59.06	881
Ecoli	78.26	60.73	443	77.11	68.44	404
Haberman	61.24	43.79	447	66.96	56.74	419
Glass	70.1	48.03	205	65.96	25.46	239
Wine	92.61	80.38	59	96.74	90.99	97
Vote	93.26	80.58	1134	92.20	81.26	1288
	75.1392857	59.6087	6628.133	77.148	57.0407	6850.133

4.1.2 RNN

Tabela5 - RNN, k = 1

	RNN, k = 1		
Base	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	94.49	90.46	244
Arrhytmia	46.40	44.96	290
Balance	71.07	70.65	47
Ionosphere	83.95	81.29	10
Liver	60.09	55.98	21
Tic-Tac-Toe	81.06	78.39	272
Vehicle	66.15	60.88	77
Pima	63.72	64.18	54
Card	75.89	73.02	80
Heart	71.86	69.54	14
Ecoli	73.40	71.36	13
Haberman	62.19	61.01	20
Glass	67.68	58.00	8
Wine	90.70	88.89	2
Vote	89.64	88.45	15
	72.0464286	69.1864	82.2857143

Tabela6 - Adaboost&Madaboost com RNN

	Adaboost			Madaboost		
Base	Precisão(%)	Redução(%)	Tempo(ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	96.07	80.21	11988	97.16	72.28	27027
Arrhytmia	48.81	37.23	14900	49.18	32.65	19402
Balance	72.93	62.52	1278	72.79	61.52	2133
Ionosphere	85.20	75.64	546	86.29	70.80	943
Liver	62.03	38.79	729	58.90	36.10	949
Tic-Tac-Toe	81.00	74.54	10879	81.09	68.42	30350
Vehicle	65.20	46.77	9100	65.68	39.57	13359
Pima	65.99	47.33	4818	65.91	44.60	7138
Card	77.88	61.06	9005	78.23	56.97	14067
Heart	75.15	48.41	760	72.04	49.67	1114
Ecoli	75.36	63.10	410	76.42	60.16	661
Haberman	62.50	37.92	504	63.50	35.77	593
Glass	70.07	43.97	254	69.22	40.18	338
Wine	90.69	83.05	56	90.04	79.33	107
Vote	90.14	76.92	939	89.73	75.17	1561
	74.9338462	58.4631	3021.385	74.603077	55.2508	5639.462

Tabela7 - Multiboost&Reweighboost com RNN

	Multiboost			ReweightBoost		
Base	Precisão(%)	Redução(%)	Tempo(ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	96.28	85.29	20077	95.86	74.75	29387
Arrhytmia	49.81	32.66	18297	53.29	8.01	15963
Balance	74.58	58.55	1943	77.37	67.39	2137
Ionosphere	86.19	74.19	766	85.96	70.99	1023
Liver	62.99	40.96	919	60.27	15.70	881
Tic-Tac-Toe	80.85	77.23	22048	79.99	70.94	31062
Vehicle	67.53	51.26	13121	67.28	42.78	11248
Pima	67.74	41.66	6763	69.35	25.44	5250
Card	77.90	61.83	13434	78.80	67.22	11180
Heart	70.16	56.16	1030	70.10	62.49	1067
Ecoli	74.62	60.57	574	75.78	68.52	571
Haberman	63.25	48.33	563	61.33	48.41	558
Glass	68.04	49.61	303	70.01	36.94	351

Wine	89.38	81.22	81	91.85	86.34	136
Vote	88.38	78.81	1221	88	80.48	1423
	74.7392308	60.0292	4828.154	75.083846	57.2031	5145.154

4.1.3 MSS

Tabela8 - MSS

	MSS, k = 1		
Base	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	94.67	82.47	222
Arrhythmia	51.1	41.16	96
Balance	70.53	58.79	14
Ionosphere	88.67	75.15	16
Liver	63.13	43.58	4
Tic-Tac-Toe	80.44	16.43	93
Vehicle	94.67	82.47	222
Pima	65.73	50.83	31
Card	77.76	61.85	71
Heart	77.87	57.21	10
Ecoli	77.25	64.54	6
Haberman	62.14	49.67	2
Glass	66.56	56.13	2
Wine	91.73	81.58	1
Vote	91.4	77.32	29
	75.875	58.7043	54.6

Tabela9 - Adaboost&Madaboost com MSS

	Adaboost			Madaboost		
Base	Precisão(%)	Redução(%)	Tempo(ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	94.57	85.38	16520	94.27	83.16	22090
Arrhythmia	50.62	40.76	12324	50.83	41.48	14850
Balance	71.44	60.75	1170	71.44	61.85	1482
Ionosphere	88.31	81.39	421	89.02	79.55	631
Liver	64.05	49.48	699	63.98	45.16	865
Tic-Tac-Toe	79.8	69.23	19058	80.64	45.09	29803
Vehicle	94.57	85.38	16520	94.27	83.16	22090
Pima	64.36	52.76	5080	66.08	52.55	6670
Card	77.44	66.36	7379	77.77	64.6	10910
Heart	78.13	60.19	698	77.86	58.89	960

Ecoli	77.58	68.08	358	77.24	65.83	485
Haberman	63.12	47.78	431	63.18	48.54	460
Glass	65.73	56.44	178	67.08	56.54	221
Wine	91.68	84.25	48	90.02	83.49	83
Vote	91.87	78.48	1936	92.42	75.88	2465
	76.8846667	65.78066667	5521.333333	77.07333333	63.05133333	7604.333333

Tabela10 - Multiboost&ReweighBoost com MSS

	Multiboost			ReweightBoost		
Base	Precisão(%)	Redução(%)	Tempo(ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	95.25	84.35	20979	92.7	87.87	23910
Arrhytmia	50.86	41.35	14984	51.1	41.16	9584
Balance	71.69	61.78	1504	72.03	57.28	1570
Ionosphere	88.16	77.99	613	88.37	74.33	702
Liver	63.82	43.68	862	62.77	35.06	705
Tic-Tac-Toe	79.9	64.65	29417	80.05	16.43	7536
Vehicle	95.25	84.35	20979	92.7	87.87	23910
Pima	66.1	51.61	6781	65.73	50.83	4808
Card	77.91	62.87	10858	77.76	61.85	9333
Heart	78.15	56.84	899	76.63	51.16	856
Ecoli	77.56	64.51	473	77.25	64.54	451
Haberman	62.04	49.68	449	62.5	59.94	510
Glass	67.08	55.61	216	66.56	56.13	240
Wine	91.05	81.63	68	92.29	71.26	98
Vote	91.31	80.01	1963	92.27	83.53	98
	76.05857143	64.06066667	7403	76.714	59.9493333	5620.733333

4.1.4 DROP3

Tabela11 - Drop3

	DROP3, k = 3		
Base	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	91.43	89.31	1523
Arrhythmia	59.24	95.38	376
Balance	87.10	82.95	259
Ionosphere	86.57	93.91	143
Liver	65.48	74.98	36
Tic-Tac-Toe	87.17	80.25	762
Vehicle	67.05	75.82	387

Pima	73.68	83.91	286
Card	85.27	85.28	440
Heart	80.18	84.45	57
Ecoli	85.2	85.28	63
Haberman	72.12	86.08	36
Glass	58.89	77.06	17
Wine	96.75	85.04	14
Vote	95.33	93.2	389
	78.295	84.2642857	319.2

Tabela12 - Adaboost&Madaboost com DROP3

	Adaboost			Madaboost		
Base	Precisão(%)	Redução(%)	Tempo (ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	89.98	86.47	24782	89.04	87.69	28371
Arrhythmia	58.49	94.92	5190	50.85	94.94	5282
Balance	84.25	89.54	3991	85.65	88.61	88.61
Ionosphere	76.69	94.6	1160	66.06	96.28	1321
Liver	64.63	86.92	1200	62.86	85.31	1251
Tic-Tac-Toe	90.37	64.19	26930	86.8	62.7	41718
Vehicle	67.62	81.32	18505	65.9	79.88	19707
Pima	71.35	85.2	7302	73.87	80.83	7700
Card	82.72	87.91	10402	83.47	87.16	11714
Heart	77.93	91.08	1036	79.3	91.76	1137
Ecoli	83.79	92.27	733	82.39	91.54	833
Haberman	67.61	90.02	674	66.64	88.12	711
Glass	58.4	86.47	383	59.43	84.57	395
Wine	91.13	89.42	167	83.74	91.92	213
Vote	87.58	91.9	2615	93.67	88.9	3600
	76.836	87.482	7004.666667	75.31133333	86.68066667	8269.440667

Tabela13 - Multiboost&ReweightBoost com DROP3

	Multiboost			ReweightBoost		
Base	Precisão(%)	Redução(%)	Tempo(ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	90.72	88.21	23523	83.82	92.17	29182
Arrhytmia	59.04	94.06	3478	52.26	91.31	3973
Balance	85.72	86.29	3549	84.36	88.81	5046
Ionosphere	71.93	94.71	895	60.4	96.58	1564
Liver	63.36	87.4	1047	62.21	88.64	843

Tic-Tac-Toe	84.79	81.21	37328	82.77	76.15	34038
Vehicle	67.42	80.48	17614	63.54	85.7	11354
Pima	70.62	88.18	6225	71.65	94.42	5129
Card	84.06	88.19	9330	82.9	89.86	9734
Heart	80.76	91.05	905	76.57	93.54	1079
Ecoli	81.05	91.53	627	81.65	93.68	959
Haberman	62.54	95.32	535	72.65	95.98	629
Glass	55.19	82.64	312	55.34	86.88	336
Wine	82.96	92.83	151	81.8	92.23	284
Vote	92.76	93.32	2127	89.09	95.79	4712
	74.29714286	89.028	7176.4	73.40066667	90.7826667	7257.466667

4.1.5 ISAR

Tabela14 - ISAR

	ISAR		
Base	Precisão(%)	Redução(%)	Tempo (ms)
Anneal	82.67	88.89	477
Arrhythmia	49.17	60.68	155
Balance	75.13	88.24	42
Ionosphere	78.11	82.31	24
Liver	53.92	63.74	13
Tic-Tac-Toe	90.17	92.52	70
Vehicle	63.67	66.51	109
Pima	70.42	76.7	66
Card	82.33	85.26	160
Heart	74.35	83.41	20
Ecoli	80.88	84.66	143
Haberman	72.55	82.58	9
Glass	53.26	69.98	5
Wine	85.16	89.57	4
Vote	90.17	92.52	70
	72.2707143	79.6464286	91.1333333

Tabela15 - Adaboost&Madaboost com ISAR

	Adaboost			Madaboost		
Base	Precisão(%)	Redução(%)	Tempo(ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	92.12	79.73	19638	91.02	82.07	23100
Arrhythmia	55.83	13.71	15702	55.87	18.51	15598

Balance	77.18	47.49	2757	76.48	48.38	2908
Ionosphere	85.89	69.38	758	77.8	71.98	814
Liver	60.53	21.57	1596	61.91	17.79	1608
Tic-Tac-Toe	88.18	92.14	1325	92.75	87.05	1602
Vehicle	66.02	45.33	20388	68.14	42.65	20945
Pima	68.9	57.12	8935	69.45	43.01	9110
Card	83.29	69	10843	83.41	68.64	11625
Heart	77.42	64.16	958	71.59	58.77	988
Ecoli	83.07	70.22	10731	82.48	70.59	11477
Haberman	70.61	69.79	585	71.55	84.63	603
Glass	64.33	37.17	305	66.45	41.46	306
Wine	88.23	86.09	93	88.1	86.39	107
Vote	88.18	92.14	1325	92.75	87.05	1602
	76.652	61.00266667	6395.933333	76.65	60.598	6826.2

Tabela16 - Multiboost&ReweightBoost com ISAR

	Multiboost			ReweightBoost		
Base	Precisão(%)	Redução(%)	Tempo (ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	91.19	79.73	20383	87.58	87.58	17095
Arrhytmia	56.16	14.21	15272	54.66	46.95	6620
Balance	74.97	50.73	2489	80	98.06	1468
Ionosphere	75	74.13	695	74.48	86.85	568
Liver	59.04	16.57	1438	58.08	67.83	936
Tic-Tac-Toe	92.92	87.87	1200	86.98	95.72	1189
Vehicle	67.09	51.28	20311	64.62	75.22	10462
Pima	69.67	61.36	8638	69.76	92.65	4228
Card	82.38	83.57	10157	81.66	86.1	7134
Heart	73.77	81.01	881	70.22	96.09	601
Ecoli	81.54	84.18	10126	81.85	87.26	6695
Haberman	73.01	84.29	531	69.76	91.62	318
Glass	66.4	43.48	281	61.53	44.23	216
Wine	84.5	90.27	92	75.35	94.3	120
Vote	92.92	87.87	1200	86.98	95.72	1189
	74.83142857	66.03666667	6246.266667	73.56733333	83.0786667	3922.6

4.1.6 CRIS

Tabela17 - Cris

	CRIS		
Base	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	93.8	7.34	467
Arrhythmia	57.19	6.72	143
Balance	75.52	68.75	71
Ionosphere	86.11	56.35	26
Liver	58.83	21.87	22
Tic-Tac-Toe	73.75	38.09	227
Vehicle	68.64	19.44	160
Pima	68.54	33.39	116
Card	84.34	41.21	141
Heart	75.74	40.14	19
Ecoli	82.86	40.39	21
Haberman	61.42	39.24	11
Glass	64.87	32.34	9
Wine	90.65	49.04	5
Vote	90.59	63.35	74
	74.4471429	35.3078571	100.8

Tabela18 - Adaboost&Madaboost com CRIS

	Adaboost			Madaboost		
Base	Precisão(%)	Redução(%)	Tempo (ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	90.81	83.58	28052	92.55	78.57	127170
Arrhythmia	54.08	37.46	17907	54.98	21623	21623
Balance	71.68	62.96	4113	73.28	57.91	5575
Ionosphere	77.87	78.14	920	87.15	53.33	1768
Liver	59.87	42.94	1536	58.65	31.39	1935
Tic-Tac-Toe	86.83	29.55	46298	87.21	22.33	62678
Vehicle	65.34	48.07	20315	64.05	34.37	34698
Pima	68.86	46.39	14025	69.88	31.52	19254
Card	81.1	60.11	15281	82.16	49.9	26840
Heart	74.67	74.67	1049	73.84	45.87	1705
Ecoli	81.17	65.06	885	81.84	56.64	1805
Haberman	64.58	34.2	1096	63.74	34.41	1186
Glass	66.27	49.23	402	61.49	35.14	567
Wine	89.14	79.35	218	83.01	81.78	328
Vote	90.3	58.03	5663	90.64	56.99	6752
	74.838	56.64933333	10517.33333	74.96466667	1486.21	20925.6

Tabela19 - Multiboost&ReweightBoost com CRIS

	Multiboost			ReweightBoost		
Base	Precisão(%)	Redução(%)	Tempo(ms)	Precisão(%)	Redução(%)	Tempo(ms)
Anneal	92.23	75.21	109592	88.3	78.44	136120
Arrhytmia	54.9	22.69	20965	57.82	1	22670
Balance	74.79	52.11	5256	75.44	48.32	5120
Ionosphere	87.98	56.9	1868	81.54	54.93	1916
Liver	62.34	12.94	1868	63.63	6.32	2219
Tic-Tac-Toe	87.69	19.6	58913	87.95	20.69	39618
Vehicle	64.88	31.32	33265	69.75	5.91	32174
Pima	70.85	26.21	18743	71.91	23.36	15806
Card	83.27	46.8	24999	83.42	55.71	23830
Heart	73.93	40.99	1578	76.79	38.29	1783
Ecoli	82.38	45.46	1606	80.84	56.46	1581
Haberman	64.5	33.88	1119	69.41	53.88	1158
Glass	66.53	32.38	526	67.94	14.33	683
Wine	88.57	71.09	275	94.31	81.72	414
Vote	90.62	59.8	5517	90.09	64.38	6645
	75.34571429	41.82533333	19072.66667	77.276	40.2493333	19449.13333

4.2 Análise dos resultados

O interesse era comparar a performance de diferentes algoritmos de *boosting*, então o objetivo era checar se os algoritmos de *boosting* apresentam maior performance que o algoritmo padrão. A performance, como dito anteriormente é entendida como uma combinação de dois fatores: precisão e redução. Mas como este trabalho é sobre melhorar o processo de seleção de instâncias, nós consideramos uma melhoria se a precisão aumentar consideravelmente.

A maior observação a se fazer é que de todos os experimentos, os algoritmos de boosting foram capazes de melhorar a precisão de teste do método base.

Para CNN, as tabelas mostram que Adaboost, Madaboost e Multiboost se comportam de maneira similar. Isto não é uma surpresa, pois estes métodos tem a mesma base. Madaboost é uma simples modificação de Adaboost e Multiboost é apenas aplicar Adaboost por várias iterações reinicializando os pesos de acordo com a iteração. Reweightboost demonstrou um aumento claro em termos de performance.

Para RNN, um comportamento semelhante ao CNN foi observado. Aqui podemos notar que o Reweightboost é um método mais agressivo em relação aos outros dois em termos de redução.

Para o MSS nós percebemos uma melhoria clara nas direções de performance e redução para todos os algoritmos observados.

Para o DROP3 nós percebemos uma melhora de redução para todos os algoritmos. Mais uma vez, a técnica Reweightboost se mostrou agressiva em termos de redução.

Para o ISAR, um comportamento semelhante ao CNN é encontrado. Neste caso, o ReweightBoost apresentou um grande aumento tanto em termos de performance como de redução.

Para o CRIS, houve um aumento considerável de performance de redução em todos os casos.

Capítulo 5 – Conclusões

O desenvolvimento desta metodologia apresenta um pontapé inicial de pesquisas nessa área. O método apresentou ser eficaz em relação a todos os algoritmos apresentados, a diversidade dos métodos indica que a metodologia funciona em qualquer algoritmo de seleção de instâncias.

Uma das grandes desvantagens destas técnicas é o seu alto custo computacional, alguns métodos chegam a ser 100 vezes mais lentos que os métodos base. Este alto custo, e as vezes, a pouca alteração de precisão para alguns algoritmos tornam estas técnicas menos atrativas. Otimizações destas técnicas devem ser desenvolvidas.

Pelo alto custo computacional, testes foram realizados com bases relativamente pequenas. Sabemos que técnicas de amostragem funcionam de maneira mais estável quando uma grande quantidade de dados são utilizados. Ficando assim, inviável a execução de testes mais representativos.

5.1 Dificuldades

Problemas foram encontrados durante o desenvolvimento deste trabalho, o mais notável é a falta de implementações públicas acessíveis para todos os algoritmos aqui listados, assim como na falta de um padrão unificador, como uma biblioteca, para os algoritmos disponíveis. Apesar de alguns destes algoritmos já serem bastante difundidos na comunidade, encontrar uma implementação que possa ser integrado à necessidade da pesquisa é desafiador.

Também houveram problemas de descrição de algoritmos, todos os algoritmos de *boosting* foram descritos no artigo base [5] de forma incorreta. O algoritmo Floatboost não foi apresentado aqui, pois possuía uma descrição que não batia com o problema a ser resolvido pelo artigo.

5.2 Trabalhos futuros

Muitos trabalhos podem ser realizados para a melhoria desta metodologia. Como os algoritmos iteram na execução de um algoritmo de seleção interna, sua complexidade básica é dada pela complexidade deste algoritmo. Um dos mais importantes seria tentar otimizar o processo para fazer com que seja aplicável a bases maiores.

Trabalhos também podem ser feitos na linha de criar um algoritmo de seleção de instâncias mais simples e que integre fortemente com o estilo do processo. Isto evita, por exemplo, do algoritmo de seleção e o da metodologia estarem fazendo o mesmo retrabalho em certos aspectos. E melhora o tempo de execução.

Também podem ser investigados algoritmos de seleção que aceitem pesos como parâmetros, estes pesos podem ser reflexos do peso de *boosting* que é mantido durante o processo.

Outras técnicas de amostragem também podem ser testadas, como *bagging* e *Random forests*. Para estas o método tem que ser adaptado.

Referências

[1] CANO, J. R.; HERRERA, F.; LOZANO, M..Using Evolutionary Algorithms as Instance Selection for Data Reduction in KDD: An Experimental Study: An Experimental Study.I**eee Trans. Evolutionary Computation,** v. 7, n. 6, p.561-575, dez. 2003.

[2] SEBBAN, Marc; NOCK, Richard; LALLICH, Stéphane. Stopping Criterion for Boosting-Based Data Reduction Techniques: from Binary to Multiclass Problems. **Journal Of Machine Learning Research,** v. 3, p.863-885, 2002.

[3] LAZAREVIC, Aleksandar; OBRADOVIC, Zoran. Data Reduction Using Multiple Models Integration. **Principles Of Data Mining And Knowledge Discovery: 5th European Conference, PKDD 2001, Freiburg, Germany, September 3–5, 2001 Proceedings,** p.301-313, 2001.

[4] GARCIA-PEDRAJAS, N. Constructing Ensembles of Classifiers by Means of Weighted Instance Selection. **Neural Networks, Ieee Transactions On,** v. 20, n. 2, p.258-277, 13 fev. 2009

[5] GARCÍA-PEDRAJAS, Nicolás. Boosting instance selection algorithms, 2014.

[6] KUNCHEVA, Ludmila I.. Combining classifiers: Soft computing solutions. **In: S.k. Pal (eds.) Pattern Recognition: From Classical To Modern Approaches,**, p.427-452, 2001.

[7] GARCÍA-PEDRAJAS, Nicolás. Supervised projection approach for boosting classifiers. **Pattern Recognition,** v. 42, n. 9, p.1742-1760, 16 jun. 2009.

[8] RODRIGUEZ, J J; KUNCHEVA, L I; ALONSO, C J. Rotation Forest: A New Classifier Ensemble Method. **Pattern Analysis And Machine Intelligence, Ieee Transactions On,**., v. 28, n. 10, p.1619-1630, out. 2006.

[9] HO, Tin Kam. The random subspace method for constructing decision forests. **Pattern Analysis And Machine Intelligence, Ieee Transactions On,** , v. 20, n. 8, p.832-844, ago. 1998.

[10] GARCÍA-PEDRAJAS, Nicolás; ORTIZ-BOYER, Domingo. Boosting random subspace method. **Neural Networks,** v. 21, n. 9, p.1344-1362, nov. 2008.

[11] FREUND, Yoav. An Adaptive Version of the Boost By Majority Algorithm. **In Proceedings Of The Twelfth Annual Conference On Computational Learning Theory,** p.102-113, 2000.

[12] FRIEDMAN, Jerome; HASTIE, Trevor; TIBSHIRANI, Robert. Additive Logistic Regression: a Statistical View of Boosting. **Annals Of Statistics,** v. 28, p.1-2000, jan. 1998.

[13]VEZHNEVETS, A.; VEZHNEVETS, V**,** *Modest AdaBoost-teaching AdaBoost to generalize better*. **Graphicon** v.12, n. 5,p. 987-997, 129, 2005.

[14] FRIEDMAN, Jerome; HASTIE, Trevor; TIBSHIRANI, Robert. Additive Logistic Regression: a Statistical View of Boosting. **Annals Of Statistics,** v. 28, p.1-2000, jan. 1998.

[15] WEBB, Geoffrey I.. MultiBoosting: A Technique for Combining Boosting and Wagging. **Machine Learning,** p.159-196, 2000.

[16] BREIMAN, Leo. Bias, variance and arcing classifiers. **(Technical Report 460). Statistics Department,** Berkeley, CA, p.1-1, abr. 1996.

[17] BAUER, Eric; KOHAVI, Ron. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. **Machine Learning,** v. 36, p.105-139, 1999.

[18] QUINLAN, J. R.. Bagging, Boosting, and C4.5. **In Proceedings Of The Thirteenth National Conference On Artificial Intelligence,** p.725-730, 1996.

[19] RODRÍGUEZ, Juan J.; MAUDES, Jesús. Boosting Recombined Weak Classifiers. **Pattern Recognition Letters, 29,** p.1049-1059, 2007.

[20] HART, P. The condensed nearest neighbor rule. **Information Theory, Ieee Transactions On,** v. 14, n. 3, p.515-516, 1968.

[21] GATES, G. The reduced nearest neighbor rule. **Information Theory, Ieee Transactions On,** v. 18, n. 3, p.431-433, maio 1972.

[22] BARANDELA, Ricardo; FERRI, Francesc J.; SÁNCHEZ, J. Salvador. *Decision Boundary Preserving Prototype Selection for Nearest Neighbor Classification*. **Int. J. Patt. Recogn. Artif. Intell.,** R, v. 19, n. 6, p.787-806, set. 2005.

[23] RITTER, G; WOODRUFF, H; LOWRY, S. An algorithm for a selective nearest neighbor decision rule. **Information Theory, Ieee Transactions On,** v. 21, n. 6, p.665-669, nov. 1975.

[24] WILFONG, G.. Nearest Neighbor Problems. **Proceedings Of The 7th Annual Symposium On Computational Geometry,** W, v. 2, n. 4, p.383-416, 1992.

[25] WILSON, D. Randall; MARTINEZ, Tony R.. Reduction Techniques for Instance-Based Learning Algorithms. **Machine Learning,** p.257-286, 2000.

[26] PEREIRA, C de Santana; CAVALCANTI, G. D. C. Instance selection algorithm based on a Ranking Procedure. **Neural Networks (ijcnn), The 2011 International Joint Conference On,** p.2409-2416, 2011.

[27] PEREIRA, C de Santana; CAVALCANTI, G. D. C. Competence Enhancement for Nearest Neighbor Classification Rule by Ranking-Based Instance Selection. **Tools With Artificial Intelligence (ictai), 2012 Ieee 24th International Conference On,** S, v. 1, p.763-769, 2012.