UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

RAMON PORTO DE SOUZA

O MÉTODO DE METACLASSIFICAÇÃO PELO ALGORITMO ADABOOST NA SHELL ORION DATA MINING ENGINE

CRICIÚMA

2016

RAMON PORTO DE SOUZA

O MÉTODO DE METACLASSIFICAÇÃO PELO ALGORITMO ADABOOST NA SHELL ORION DATA MINING ENGINE

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientadora: Profª. Dra. Merisandra Côrtes de Mattos Garcia

CRICIÚMA

2016

RAMON PORTO DE SOUZA

O MÉTODO DE METACLASSIFICAÇÃO PELO ALGORITMO ADABOOST NA SHELL ORION DATA MINING ENGINE

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Inteligência Computacional

Criciúma, 02 de dezembro de 2016.

BANCA EXAMINADORA

Profª. Merisandra Côrtes de Mattos Garcia - Doutora - (UNESC) - Orientadora

Prof. Kristian Madeira - Doutor - (UNESC)

Prof. Gilberto Vieira da Silva - Especialista - (UNESC)

RESUMO

Técnicas tradicionais têm sido inviáveis no processamento do volume crescente de dados e o *data mining* surge como solução ao automatizar e utilizar de algoritmos avançados para analisar estes dados. O *data mining* consiste na extração de padrões em uma base de dados. A classificação, uma das tarefas de *data mining* resume-se na atribuição de um novo objeto a uma classe pré-definida. Quanto mais instâncias forem classificados corretamente, mais preciso tende a ser o classificador. Dentre os algoritmos de classificação têm-se o algoritmo metaclassificador Adaboost, considerado por diferentes autores como o mais influente dentre os metaclassificadores e é capaz de induzir algoritmos de classificação usados como base a tomarem atenção em instâncias de treinamento mais difíceis de serem classificadas e com isso melhorar o resultado da classificação geral do conjunto. Algoritmos como o Adaboost são normalmente implementados em ferramentas de *data mining*. Uma delas é a Shell Orion Data Mining Engine, desenvolvida pelo Grupo de Pesquisa em Inteligência Computacional do curso de Ciência da Computação da Universidade do Extremo Sul Catarinense. O propósito desta pesquisa é disponibilizar o algoritmo metaclassificador Adaboost na ferramenta Shell Orion Data Mining Engine e verificar se ele traz resultados satisfatórios. Para isso, foram aplicadas duas bases de dados, uma binária e outra com múltiplas classes e implementadas medidas de qualidade para avaliar a qualidade dos classificadores gerados. Os resultados para a base de dados binária selecionada mostraram que o Adaboost obteve uma acurácia de 98,5507% e para a base de dados com múltiplas classes obteve acurácia de 93,3447% na Shell Orion Data Mining Engine.

Palavras-chave: *Data Mining*. Classificação. Metaclassificação. *Boosting.* Algoritmo Adaboost.

ABSTRACT

Traditional techniques have been infeasible in processing the increasing amount of data and data mining emerges as a solution by automating and using advanced algorithms to analyze this data. Data mining consists of extracting patterns from a database. Classification, one data mining task, boils down to assigning a new object to a predefined class. How much more instances are correctly classified, more accurate the classifier tends to be. Among the classification algorithms are the meta-classifier algorithm Adaboost, considered by different authors as the most influential meta-classifier, and it is able to induce classification algorithms used as base classifiers to pay attention to training instances more difficult to classify and consequently improve the overall result. Algorithms like Adaboost are usually implemented in data mining tools. One of them is the Shell Orion Data Mining Engine, developed by the Computational Intelligence Research Group of the Computer Science course at the University of Southern Santa Catarina. The purpose of this research is to provide the Adaboost meta-classifier algorithm in the Shell Orion Data Mining Engine tool and verify that it brings satisfactory results. For this, two databases, one binary and the other with multiple classes, were implemented and quality measures were implemented to evaluate the quality of the generated classifiers. The results for the selected binary database demonstrated that Adaboost obtained an accuracy of 98.5507% and for the database with multiple classes obtained accuracy of 93.3447% in the Shell Orion Data Mining Engine.

Keywords: Data Mining. Classification. Meta-classification Boosting. Adaboost Algorithm.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Adaboost	Adaptative Boosting
API	Application Programing Interface
AUC	Area Under Curve
DARPA	Defense Advanced Research Products Agency
FN	Falso Negativos
FP	Falso Positivos
ICDM	International Conference on Data Mining
IDE	Integrated Development Enviroment
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
JDBC	Java Database Connectivity
KDD	Knowledge Discovery in Databases
OO	Orientação a Objetos
PFN	Proporção de Falsos Negativos
PFP	Proporção de Falsos Positivos
SQL	Structured Query Language
SVM	Support Vector Machine
TCC	Trabalho de Conclusão de Curso
UML	Unified Modeling Language
UNESC	Universidade do Extremo Sul Catarinense
VN	Verdadeiro Negativos
VP	Verdadeiro Positivo

SUMÁRIO

1 INTRODUÇÃO

Novos métodos e abordagens são necessários para a obtenção de informação útil em bases de dados, uma vez que as técnicas tradicionais são inviáveis no processamento do volume crescente de dados que são coletados e armazenados a todo momento. Para processar esses dados e analisá-los sob diferentes perspectivas surge o *data mining,* que une técnicas clássicas a algoritmos avançados (TAN; STEINBACH; KUMAR, 2009).

O *data mining*, uma das etapas da descoberta de conhecimento em bases de dados, do inglês Knowledge Discovery in Databases (KDD), consiste na extração de padrões em uma base de dados (FAYYAD; PIATETSKY-SHAPIRO; SMITH*,* 1996, tradução nossa). A partir do que se deseja realizar no *data mining*, podem ser utilizadas inúmeras tarefas, dentre as quais destacam-se a análise de associação, agrupamento, detecção de anomalias e classificação (TAN; STEINBACH; KUMAR, 2009).

Segundo Linoff e Berry (2011, tradução nossa) a classificação resume-se na atribuição de um novo objeto a uma classe pré-definida. A tarefa de classificação é caracterizada por dois passos: o primeiro é a criação de um modelo que descreve um conjunto de classes pré-definidas, onde algumas destas serão selecionadas aleatoriamente e usadas para o aprendizado, normalmente representado por regras de classificação, árvores de decisão ou fórmulas matemáticas; o segundo passo usa este modelo para classificar os dados (HAN, KAMBER; PEI, 2012, tradução nossa). O desempenho deste modelo é avaliado por meio da contagem de registros de testes classificados correta e incorretamente. Quanto mais registros forem classificados corretamente, mais preciso tende a ser o modelo e desta forma há a necessidade de buscar por algoritmos classificatórios que sejam cada vez mais precisos e que minimizem os erros, de forma a reduzir o tempo de aprendizado (TAN; STEINBACH; KUMAR, 2009; CHAVES, 2012).

Surgindo como uma forma de minimizar estes erros e consequentemente melhorar os resultados da classificação, os métodos de grupo, também denominados de combinação de classificadores, metaclassificadores ou comitês classificadores, partem do princípio de que com a união dos resultados de vários classificadores, chamados classificadores base, chega-se a um resultado mais preciso. Assim, os metaclassificadores são métodos que usam a previsão de

múltiplos classificadores para melhorar a precisão da classificação (TAN; STEINBACH; KUMAR, 2009). A partir de como os classificadores base são gerados, existem duas abordagens de metaclassificação: o *bagging*, que explora técnicas que geram classificadores base em paralelo, e o *boosting*, que se utiliza de técnicas baseadas na geração sequencial (ZHOU, 2012, tradução nossa).

Os algoritmos de *boosting*, visam resolver o problema geral de produzir um classificador altamente preciso a partir de um conjunto de classificadores base moderadamente imprecisos (SCHAPIRE; FREUND, 2012, tradução nossa) e realizam no final de cada rodada de treinamento a atualização dos pesos dos exemplos de treinamento, de forma a diminuir o peso dos exemplos que forem classificados corretamente e aumentar dos que foram classificados incorretamente, para que os últimos participem mais vezes das rodadas, focando-se nos que são mais difíceis de classificar (TAN, STEINBACH, KUMAR, 2009).

Dentre os algoritmos de *boosting*, duas abordagens se diferem: de como as previsões são unidas e como os pesos das amostras são atualizados (TAN, STEINBACH, KUMAR, 2009). Tendo a última abordagem sido aplicada no algoritmo Adaboost, apresentado pela primeira vez em 1995 por Freund e Schapire (1997, tradução nossa) e considerado o mais influente dentre os metaclassificadores (ZHOU, 2012, tradução nossa).

O algoritmo Adaboost é um algoritmo de *boosting* adaptável, ou seja, é capaz de se adaptar aos seus classificadores base (FREUND; SCHAPIRE; ABE, 1999, tradução nossa), utilizando a soma dos pesos dos exemplos mal classificados dividido pelo total de pesos, em vez de usar a fração dos que são classificados erroneamente (WITTEN; FRANK; HALL, 2011, tradução nossa).

Esta pesquisa busca disponibilizar um metaclassificador por meio do algoritmo de *boosting* Adaboost para a tarefa de classificação na ferramenta *Shell Orion Data Mining Engine*, desenvolvida pelo Grupo de Pesquisa em Inteligência Computacional do curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, e verificar se este algoritmo traz resultados satisfatórios na tarefa de classificação desta ferramenta.

1.1 OBJETIVO GERAL

Disponibilizar na Shell Orion Data Mining Engine um metaclassificador por meio do algoritmo Adaboost.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa consistem em:

a) compreender a tarefa de classificação, metaclassificadores e Adaboost;
b) demonstrar o funcionamento matemático do algoritmo de classificação Adaboost;
c) implementar o algoritmo Adaboost para a classificação de dados;
d) aplicar o algoritmo implementado em uma base dados a ser definida;
e) disponibilizar medidas de qualidade em *data mining* para a classificação de dados.

1.3 JUSTIFICATIVA

Encontrar padrões é algo frequente, como por exemplo, padrões de cores, de formas e de rostos. Esta é a essência do *data mining* (WITTEN; FRANK, 2011, tradução nossa). O *data mining* vem suprir a necessidade do homem, que já não é capaz de analisar manualmente a expressiva quantidade de dados que se avoluma em diferentes formas de armazenamento (TAN; STEINBACH; KUMAR, 2009).

Dentre as análises que o homem precisa fazer nestes dados, há a classificação, que é a tarefa de distinguir se um objeto pertence a certa classe ou não (LINOFF; BERRY, 2011, tradução nossa). Bancos que desejem saber se um cliente pertence a um grupo onde ele pode fornecer crédito, pesquisadores que precisam saber os sintomas relacionados a certa doença, fábricas que necessitem diferenciar objetos, podem fazer uso da classificação para obter uma resposta (HAN; KAMBER; PEI, 2012, tradução nossa).

Para as diferentes aplicações que a técnica de classificação pode ser usada, não é possível gerar um classificador eficiente em todas as situações. A abordagem dos metaclassificadores parte da premissa de que não é necessário

desenvolver um algoritmo classificador forte para se obter uma classificação mais precisa e sim unir vários algoritmos existentes para que se chegue a este resultado (DZEROSKI; ZENKO, 2004, tradução nossa). Apesar das duas abordagens metaclassificadoras, *bagging* e *boosting*, serem mais precisas que um único classificador, o *boosting* tende a ser o mais preciso dentre eles (HAN; KAMBER; PEI, 2012, tradução nossa).

A abordagem de *boosting* não havia se provado verdadeira em uma situação real até a proposta do algoritmo Adaboost em 1995 (ZHOU, 2012, tradução nossa). Sendo este um método conhecido e estudado para se desenvolver metaclassificadores de alta performance (FERREIRA; FIGUEIREDO, 2012, tradução nossa).

Chaves (2012) aplicando o algoritmo Adaboost em sensores, Reis (2013) na detecção de sinais de trânsito e Mathanker et al (2011, tradução nossa) na detecção de defeitos em nozes Pecã, Luo, Lin e Chen (2011, tradução nossa) na identificação de gênero para aplicações de robô de serviço, Nascimento (2011) na detecção e contagem automática de plaquetas, ao comparar a outros algoritmos classificadores obtiveram resultados satisfatórios com o Adaboost.

Algoritmos como o Adaboost normalmente são implementados em ferramentas de *data mining*. No mercado existem inúmeras ferramentas comerciais, como SAS Enterprise, Miner, Darwin SPSS, dentre outras (GOLDSCHMIDT; BEZERRA, 2015), sendo restritivas e de alto custo, porém existem ferramentas gratuitas, como a *Shell Orion Data Mining Engine*, desenvolvida pelo Grupo de Pesquisa em Inteligência Computacional Aplicada do curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, que já possui algumas tarefas implementadas, no entanto, não possui nenhum algoritmo de *boosting*.

Pelos resultados positivos, utilidades dos metaclassificadores e a inexistência de um algoritmo que utilize a técnica de metaclassificação na ferramenta Shell Orion Data Mining Engine é que esta proposta se justifica.

1.4 ESTRUTURA DO TRABALHO

Esta pesquisa é dividida em quatro capítulos, tendo o primeiro capítulo uma introdução, os objetivos gerais e específicos e sua justificativa.

O capítulo 2 trata a descoberta de conhecimento em bases de dados (do inglês, KDD), suas tarefas e mais aprofundadamente a tarefa de *data mining,* sua subtarefa de classificação e suas implicações.

No terceiro capítulo é apresentada a abordagem da combinação de classificadores, chamada de metaclassificação, sua variante a abordagem de *boosting*, seu algoritmo mais famoso, o Adaboost e outras versões dependentes dele. Também será apresentado métodos de avaliação da qualidade de um classificador, a ferramenta Shell Orion Data Mining Engine, criada pela equipe de inteligência computacional aplicada da Universidade do Extremo Sul Catarinense, UNESC, na qual será disponibilizada o algoritmo Adaboost e trabalhos que tiveram relação com o uso do algoritmo Adaboost.

O capítulo 4 traz o trabalho desenvolvido, a metodologia usada para o desenvolvimento deste trabalho e os resultados obtidos da aplicação do módulo do Adaboost na Shell Orion Data Mining Engine.

O quinto capítulo, por fim, aborda a conclusão da pesquisa e propostas de trabalhos futuros.

2 DESCOBERTA DE CONHECIMENTO EM BASES DE DADOS

Comumente ouve-se que a humanidade está na “era da informação”, no entanto Han, Kamber e Pei (2012, tradução nossa) afirmam que se está na “era dos dados”. Novas tecnologias e tendências, como a *Internet das Coisas* e a Computação em Nuvem, vêm somente a contribuir com uma realidade presente no cotidiano: o aumento expressivo nas bases de dados, que de acordo com estimativas, pode ultrapassar 15 exabytes a cada ano (BROWN, 2014, tradução nossa).

Concomitantemente a este aumento expressivo surge a incapacidade humana de manualmente compreender, extrair informação e consequentemente obter conhecimento destes dados. Do estudo de ferramentas que automatizem este processo nasceu a Descoberta de Conhecimento em Bases de Dados (LAROSE; LAROSE, 2014, tradução nossa; GOLDSCHMIDT; BEZERRA, 2015).

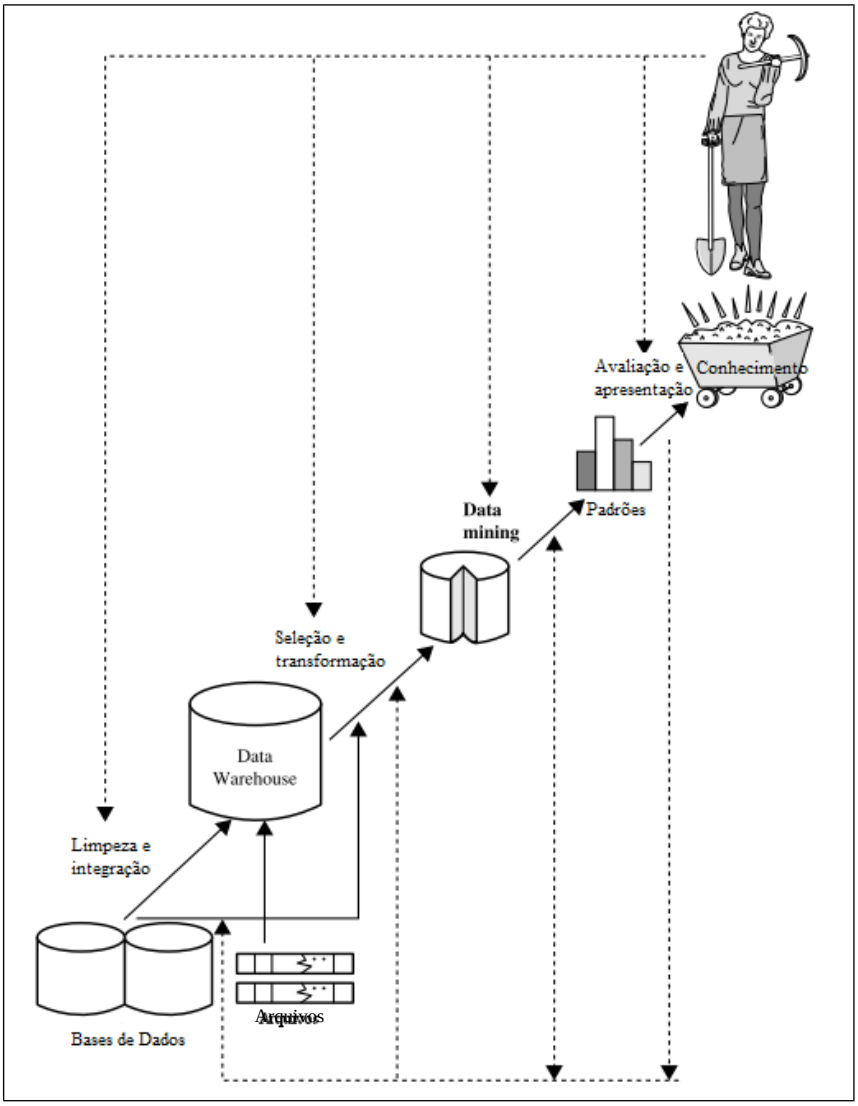
A Descoberta de Conhecimento em Bases de Dados, tradução de Knowledge Discovery in Databases (KDD), é um termo cunhado em 1989 por Piatetsky-Shapiro e abrange toda a descoberta de conhecimento útil em bases de dados (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996, tradução nossa).

O KDD constitui-se em processo de várias etapas operacionais que podem ser resumidas em sete (HAN; KAMBER; PEI, 2012, tradução nossa), figura 1, os quais são:

a) **limpeza dos dados:** para remover possíveis ruídos e dados inconsistentes;
b) **integração dos dados:** múltiplos conjuntos ou bases de dados são combinados;
c) **seleção dos dados:** os dados importantes para a análise são separados;
d) **transformação dos dados:** os dados são transformados e consolidados em formas apropriadas;
e) ***data mining*:** são utilizados algoritmos inteligentes para a extração de padrões;
f) **avaliação dos padrões:** para identificar padrões que realmente representem conhecimento útil;

g) **apresentação do conhecimento:** são utilizadas técnicas para tornar o conhecimento obtido apresentável para o usuário.

Figura 1 – Etapas operacionais do KDD



Fonte: Adaptado de Han, Kamber e Pei (2012, tradução nossa).

O subcapítulo seguinte abordará a etapa de *data mining*, por abranger o tema desta pesquisa.

2.1 DATA MINING

Data mining, uma das etapas da Descoberta de Conhecimento em Bases de Dados, consiste no processo de extrair padrões em bases de dados. O processo deve ser automático ou, ao menos, semiautomático e os dados devem possuir algum valor na obtenção de conhecimento (WITTEN; FRANK; HALL, 2011, tradução nossa).

O *data mining* é uma área multidisciplinar e mescla conceitos de aprendizado de máquina, sistemas de bancos de dados e reconhecimento de padrões (ZAKI; MEIRA JÚNIOR, 2014, tradução nossa) e possui inúmeras aplicações reais, como (GORUNESCO, 2011, tradução nossa):

a) na economia, sendo aplicada em análises de mercado, transações financeiras, dados de *e-commerce*, e como apoio à tomadas de decisões em negócios;
b) em análises médicas e farmacêuticas, com vastos dados a serem explorados;
c) em áreas científicas como astronomia, biologia, meteorologia, entre outros.

No contexto de cada aplicação do *data mining,* podem ser utilizadas inúmeras tarefas a fim de se obter os mais diversos resultados úteis. Cada tarefa é caracterizada pelo tipo de informação que se deseja obter, sendo que, dentre as tarefas existentes, destacam-se (TAN; STEINBACH; KUMAR, 2009):

a) **análise de associação:** compreende a ocorrência simultânea de itens que possam ser relacionados, sendo esta relação muitas vezes custosa computacionalmente. Como exemplo, é possível citar o famoso caso da rede supermercados que ao fazer a análise de associação em suas bases de dados encontrou a relação existente entre a venda de fraldas e de cervejas (TAN; STEINBACH; KUMAR, 2009);
b) **agrupamento:** também chamado de *clusterização,* o agrupamento busca encontrar grupos de dados que possuam dados com íntima relação entre si e objetiva maximizar a relação dos itens dentro do grupo e minimizar a relação entre grupos. Pode-se aplicar o agrupamento em uma empresa que deseje dividir seus clientes em grupos pela semelhança existentes nos seus perfis (GOLDSCHMIDT; BEZERRA, 2015);
c) **detecção de anomalias:** procura pela ocorrência de itens que fogem a normalidade do esperado (anomalias) em seu contexto. A detecção de anomalias pode ser útil, por exemplo, na detecção de uma fraude no cartão de crédito, onde há compras muito grandes em pouco tempo ou feitas fora da região do cliente (HAN; KAMBER; PEI, 2012, tradução nossa; GOLDSCHMIDT; BEZERRA, 2015);

d) **classificação:** resume-se na atribuição de um novo objeto a uma classe pré-definida (LINOFF; BERRY, 2011, tradução nossa) , sendo caracterizada por dois passos: a criação de um modelo que descreve um conjunto de classes pré-definidas e o uso deste modelo para classificar os dados (HAN, KAMBER; PEI, 2012, tradução nossa). Para exemplificar, pode-se usar um algoritmo classificatório para, classificar as galáxias encontradas por um telescópio a partir de acordo com o seu formato (TAN; STEINBACH; KUMAR, 2009).

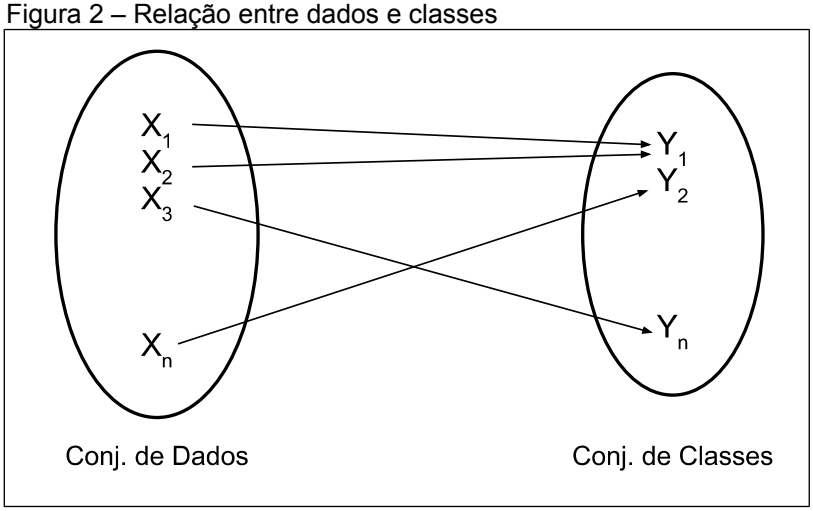
2.2 A TAREFA DE CLASSIFICAÇÃO

A classificação é um processo que se resume na atribuição de um novo objeto a uma classe pertencente a um conjunto de classes pré-definidas (LINOFF; BERRY, 2011, tradução nossa). A tarefa de classificação em *data mining* busca tornar automático este processo.

Bancos que busquem identificar se um cliente pertence a classe dos clientes aptos a receber crédito, pesquisadores que precisam saber os sintomas relacionados a certa doença ou fábricas que necessitem diferenciar objetos, podem fazer uso da tarefa de classificação para obter uma resposta (HAN; KAMBER; PEI, 2012, tradução nossa).

A tarefa de classificação é constituída de duas etapas e pode ser definida formalmente como a tarefa de aprendizado que atribui todos os conjuntos de atributos *x* a uma das classes (ou rótulos) *y* em uma função *f* (TAN; STEINBACH; KUMAR, 2009).

Figura 2 – Relação entre dados e classes



Fonte: Goldshmidt e Bezerra (2015).

Das etapas de classificação, a primeira consiste em criar um modelo de classificação, e a segunda usa este modelo para classificar os dados (AGGARWAL, 2015, tradução nossa).

Na primeira etapa um modelo, ou classificador, é criado descrevendo as classes (conceitos ou rótulos), como por exemplo, numa análise de risco onde os dados podem pertencer as classes seguro ou arriscado. Essas classes ou categorias podem ser representadas por valores discretos ou contínuos, sendo esta a principal característica que diferencia as tarefas de classificação e de regressão (TAN; STEINBACH; KUMAR, 2009).

Nesta etapa, também chamada fase de aprendizado, o modelo é gerado pelo algoritmo de classificação com a análise de um conjunto de treinamento, criado a partir das tuplas da base de dados e seus rótulos de classe associados. As tuplas podem ser descritas como um vetor n-dimensional $X = (x_1, x_2, . . . , x_n)$, onde cada tupla pertence a um rótulo de classe. Na literatura é possível encontrar o nome tuplas referenciado como objetos, amostras, instâncias ou exemplos. Este aprendizado, pela necessidade de que os rótulos sejam fornecidos (geralmente por um humano) é denominado aprendizado supervisionado, contrastando com o aprendizado não supervisionado, que é uma etapa de outra tarefa de *data mining* – o agrupamento (HAN; KAMBER; PEI, 2012, tradução nossa).

Com o modelo criado, que também é chamado de classificador ou hipótese (SCHAPIRE; FREUND, 2012, tradução nossa), é possível partir para a segunda etapa, que consiste em usar este modelo para predizer a classe de certo objeto em uma base de dados (ZAKI; MEIRA JÚNIOR, 2014, tradução nossa).

Muitos modelos de classificação já foram propostos, dentre os quais destacam-se (ZHOU, 2012 tradução nossa):

a) **árvores de decisão:** é um modelo relativamente simples que utiliza uma estrutura hierárquica em formato de árvore, partindo do nó raiz, atravessando pelos nós internos e obtendo-se a resposta conclusiva nos nós folha ou terminais, ou seja, a classe a qual aquele objeto pertence. O caminho que será feito se baseia na resposta informada para cada condição presente nos nós raiz e intermediários (TAN; STEINBACH; KUMAR, 2009);

b) **classificadores Bayesianos:** são classificadores estatísticos capazes de prever a probabilidade de um objeto pertencer a certa classe. Estes classificadores são baseados no teorema de Bayes, criado no século XVIII por Thomas Bayes (HAN; KAMBER; PEI, 2012, tradução nossa);

c) **redes neurais artificiais:** buscam simular o sistema nervoso humano, onde a função computacional de um neurônio, denominação usada para representar um nó, é definida pelos pesos das suas conexões de entrada. As alterações apropriadas fazem com que a função possa ser aprendida, e são providas pelo conjunto de treinamento sempre que as previsões forem incorretas (AGGARWAL, 2015, tradução nossa).

A existência de vários modelos se faz necessária, uma vez que, de acordo com o teorema de Wolpert e Macready (1996, tradução nossa), não existe um modelo classificador que possa ser superior a todos em todas as situações. Portanto, é importante a busca pelas mais diversas abordagens. Uma delas é a metaclassificação, que parte da premissa de que a classificação com um método que usa um conjunto de classificadores é mais eficiente ou, ao menos igual, ao o melhor classificador dentre eles (DZEROSKI; BERNARD, 2004, tradução nossa; FERREIRA; FIGUEIREDO, 2012, tradução nossa).

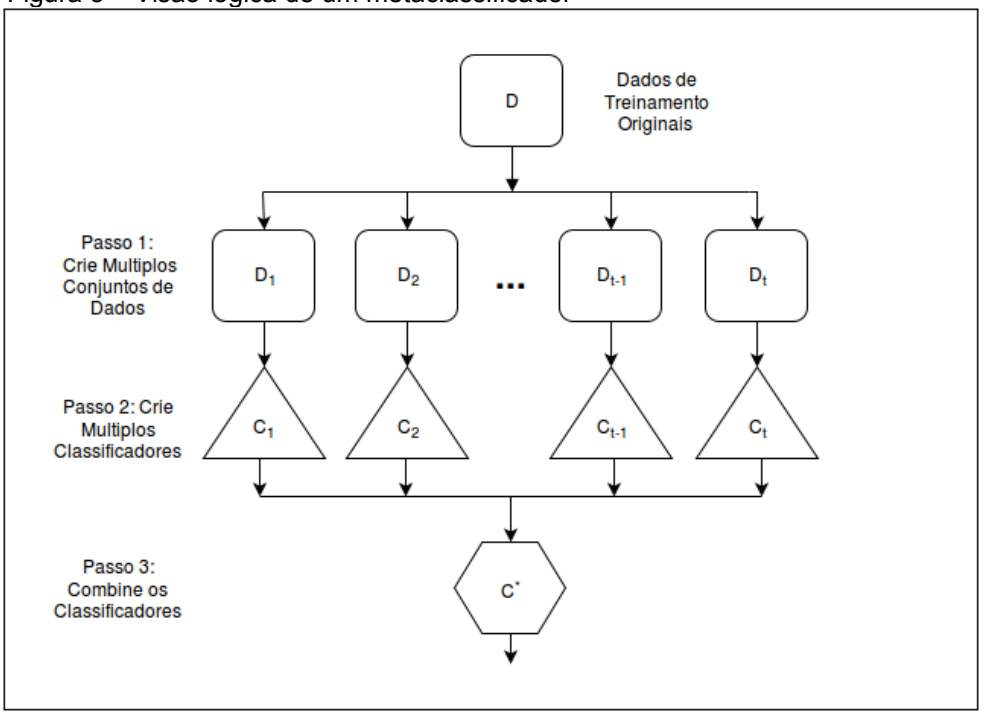
3 METACLASSIFICADORES

A tomada de decisão formada a partir de um conjunto de opiniões de especialistas de várias áreas é algo democrático e comum na sociedade humana. Fundamentada nesse princípio, a abordagem de metaclassificadores utiliza de um conjunto de classificadores, neste caso, os “especialistas”, para alcançar resultados mais precisos na classificação de dados (WITTEN; FRANK; HALL, 2011, tradução nossa).

Em função deste objetivo, algoritmos de classificação são reunidos e treinados em um conjunto de dados, gerando um grupo de classificadores, chamados de classificadores-base, que serão combinados em um classificador final (ZHOU, 2012, tradução nossa).

A figura 3 demostra de forma genérica a estrutura funcional de um metaclassificador:

Figura 3 – Visão lógica de um metaclassificador



Fonte: Tan, Steinbach e Kumar (2009).

Metaclassificadores podem ser encontrados na literatura com denominações variadas como métodos de grupo (TAN; STEINBACH; KUMAR, 2009), fusão de classificadores ou comitês de classificadores. (KANTARDZIC, 2011, tradução nossa).

Estas diferentes nomenclaturas provenientes do extensivo número de pesquisas corroboram com a importância assumida nas últimas duas décadas pelos metaclassificadores em meio as pesquisas de inteligência computacional e aprendizado de máquina. Importância que é justificada pela sua abrangência para resolver variados problemas como aprendizado incremental, dados com classes desbalanceadas e correção de erros de classificação (POLIKAR, 2012, tradução nossa) e pela capacidade de transformar classificadores fracos, que são ligeiramente melhores do que a decisão aleatória, em um classificador forte com alta capacidade de predição (HAN; KAMBER; PEI, 2012, tradução nossa).

Esta capacidade de aumentar a predição é possível pelo fato de que diferentes classificadores fazem diferentes predições e variam na sensibilidade no conjunto de testes, que então, permitem aumentar a precisão da classificação (AGGARWAL, 2015, tradução nossa), sendo que, de modo geral, os metaclassificadores costumam ser mais precisos que seus classificadores-base. Dado um metaclassificador que faça sua predição com base no voto da maioria e que seus classificadores-base são melhores do que a decisão aleatória, este metaclassificador somente errará a classificação se mais da metade dos classificadores do conjunto errar também (HAN; KAMBER; PEI, 2012, tradução nossa).

Outro ponto a justificar o uso de conjuntos de classificadores dá-se pelo fato de ser mais fácil treinar vários classificadores simples e, a partir deles, gerar um complexo, do que treinar um classificador único complexo (FERREIRA; FIGUEIREDO, 2012, tradução nossa).

Estas características já poderiam ser consideradas justificativas plausíveis para adotar a metaclassificação, contudo, pode-se induzir à reflexão de que usar um conjunto de classificadores tem um custo computacional muito superior a um classificador único. Porém, isto não ocorre, uma vez que as combinações geralmente são simples e de baixo custo computacional e na geração de classificadores únicos têm-se o trabalho de gerar muitas versões do mesmo, que são usadas para ajustes de parâmetros e seleção de modelo (ZHOU, 2012, tradução nossa).

Todavia, para que estas características descritas anteriormente tenham efeito em um sistema metaclassificador, é preciso considerar os três componentes a seguir (POLIKAR, 2012, tradução nossa):

a) **uma estratégia para escolher as amostras:** amostras que gerem resultados diferentes no treinamento dos membros do conjunto traz benefícios para a combinação de classificadores, dado que amostras diferentes geram classificadores-base diferentes que atendem a um maior número instâncias da base de dados, sendo substancial para a geração de um bom metaclassificador;

b) **uma estratégia para treinar os classificadores-base:** a estratégia de treinamento dos classificadores-base constitui-se no cerne de um metaclassificador e varia conforme a abordagem utilizada, como, por exemplo, nas abordagens de *bagging* e *boosting* (POLIKAR, 2012, tradução nossa), que serão abordados ao final deste subcapítulo. Os algoritmos-base, que podem ser completamente diferentes uns dos outros e que gerarão os modelos (AGGARWALL, 2015 tradução nossa), podem ser obtidos a partir de algoritmos de Redes Neurais Artificias, Árvores de Decisão, Máquinas de Suporte Vetorial, do inglês Support Vector Machine (SVM), dentre outros algoritmos de aprendizado, sendo comum encontrar metaclassificadores que utilizam somente um algoritmo de aprendizado, chamados de metaclassificadores homogêneos, já metaclassificadores que utilizam mais de um algoritmo são chamados de heterogêneos (ZHOU, 2012, tradução nossa);

c) **uma estratégia para combinar os classificadores-base:** a estratégia para combinar os classificadores base é o método pelo qual o classificador final será gerado, ou seja, o classificador resultante dos treinamentos dos classificadores-base formará o metaclassificador e será utilizado na fase de testes. Dependendo dos tipos de classificadores-base usados no conjunto as estratégias de combinação variam, podendo ser feita pela combinação dos rótulos de classe ou a combinação de saídas contínuas, dentre outras;

A figura 4 demonstra um pseudocódigo genérico de treinamento dos metaclassificadores, salientando as características citadas anteriormente:

Figura 4 – Pseudocódigo de um metaclassificador

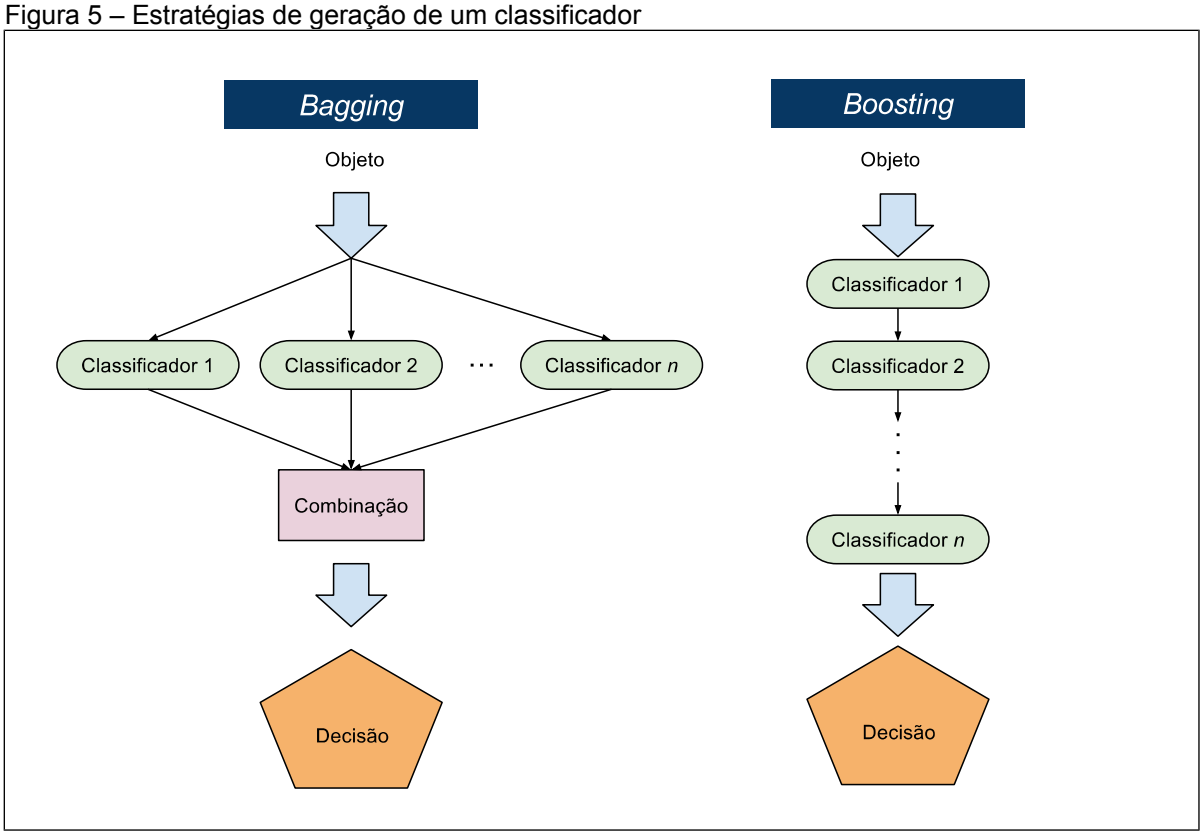
Algoritmo *Metaclassificador* (Treinando conjunto de dados: D
Algoritmos-base: $A_1 ... A_n$, instâncias de Teste T)
inicia
j = *1;*
repita
Seleciona um algoritmo Q_j de $A_1 ... A_n$;
Cria um novo conjunto de dados de treinamento $f_j(D)$ a partir de D;
Aplica Q_j em $f_j(D)$ para aprender um modelo M_j;
a**té** (finalizar)
Reporta rótulos de cada $T \in T$ baseado na combinação de predições de todos os modelos M_j aprendidos;
fim

Fonte: Aggarwal (2015, tradução nossa).

Dentre as estratégias citadas anteriormente, as mais promissoras e que possuem mais estudos na área são justamente as que manipulam o treinamento dos classificadores-base, ou seja, são as estratégias que manipulam a forma com que os algoritmos são treinados (POLIKAR, 2012, tradução nossa). Duas podem ser destacadas por terem fundamentada base teórica e serem bem conhecidas (KANTARDZIC, 2011, tradução nossa), descritas a seguir (ZHOU, 2012, tradução nossa):

a) ***bagging*:** os algoritmos de *bagging* utilizam uma abordagem paralela para a geração dos classificadores e fazem uso da independência existente entre os seus algoritmos de base de maneira a reduzir drasticamente os erros. Por trabalharem paralelamente, permitem que seus classificadores-base sejam treinados em um computador multinúcleos ou em vários computadores paralelos. Ilustrado pela figura 5;
b) ***boosting*:** os algoritmos de *boosting* geram seus classificadores de forma sequencial e possuem a motivação primária de explorar a dependência existente entre os classificadores-base (exemplificada na figura 5) e como os resultados podem ser melhorados a partir disso.

Figura 5 – Estratégias de geração de um classificador



Fonte: Adaptado de Woźniak e Graña (2014, tradução nossa).

3.1 ALGORITMOS DE BOOSTING

Boosting é uma abordagem de metaclassificação que utiliza seus classificadores-base em analogia a um grupo de especialistas de diferentes áreas capazes de se completarem em partes onde um ou outro não seja competente (WITTEN; FRANK; HALL, 2011, tradução nossa).

Sua principal característica é a capacidade de gerar um classificador com alta capacidade de predição a partir de classificadores pouco melhores que decisão aleatória (POLIKAR, 2012, tradução nossa) e se diferencia essencialmente de *bagging* na forma de utilizar as instâncias de treinamento. Enquanto *bagging* faz réplicas do conjunto dessas instâncias, *boosting* utiliza do resultado das classificações anteriores para construir novas amostras para o treinamento (ZAKI; MEIRA JÚNIOR, 2014, tradução nossa), figura 6.

Figura 6 – Pseudocódigo geral de *boosting.*

Entrada: distribuição *D;*
Algoritmo-base de treinamento A;
Número de rodadas de aprendizado T;
Processo:
1 $D_1 = D$. *//inicializa a distribuição*
2 **Para** $t = 1, \ldots, T$:
3 $h_t = A(D_t)$; *//treina o classificador na distribuição* D_t
4 $\epsilon_t = P_{x \sim D_t}(h_t(x) \neq f(x))$; *//avalia o erro de* h_t
5 $D_{t+1} = Ajuste\ da\ Distribuição(D_t, \epsilon_t)$
6 **fim**
Saída: $H(x) = Combina\ as\ saídas\ (\{h_1(x), \ldots, h_t(x)\})$

Fonte: Adaptado de Zhou (2012, tradução nossa).

A geração de novas amostras de acordo com o resultado das classificações anteriores se baseia na adição de um peso a cada instância de treinamento e, em sucessivas rodadas, estes pesos são modificados de acordo com o desempenho do classificador-base em relação à instância. O classificador final é fruto da geração sequencial dos modelos (classificadores) de um mesmo algoritmo A (AGGARWAL, 2015, tradução nossa) e da combinação dos votos de cada classificador, dado que cada voto é obtido pela função de precisão do mesmo (HAN; KAMBER: PEI, 2012, tradução nossa).

A precisão de *boosting* é afetada por dados com ruido, que se constitui de um erro intrínseco que ocorre na rotulagem de classes, pois ele considera que o erro de classificação é causado pelo componente de tendência próximo ao limite de decisão modelado incorretamente, em vez de apenas considerar que o erro é causado por dados rotulados incorretamente (AGGARWAL, 2015, tradução nossa). Além de que em alguns casos, quando *boosting* é aplicado aos dados finais, pode mostrar um resultado inferior a um classificador único, indicando que o modelo sofre de *overfitting*, que ocorre quando um classificador está ajustado excessivamente aos dados de treinamento (WITTEN; FRANK; HALL, 2011, tradução nossa).

Atualmente é possível encontrar muitas implementações do algoritmo de *boosting.* Estas implementações se diferem pela abordagem de como os pesos das amostras de treinamento são atualizados ao final de cada rodada e de como os

modelos gerados são combinados nas rodadas futuras (TAN; STEINBACH; KUMAR, 2009).

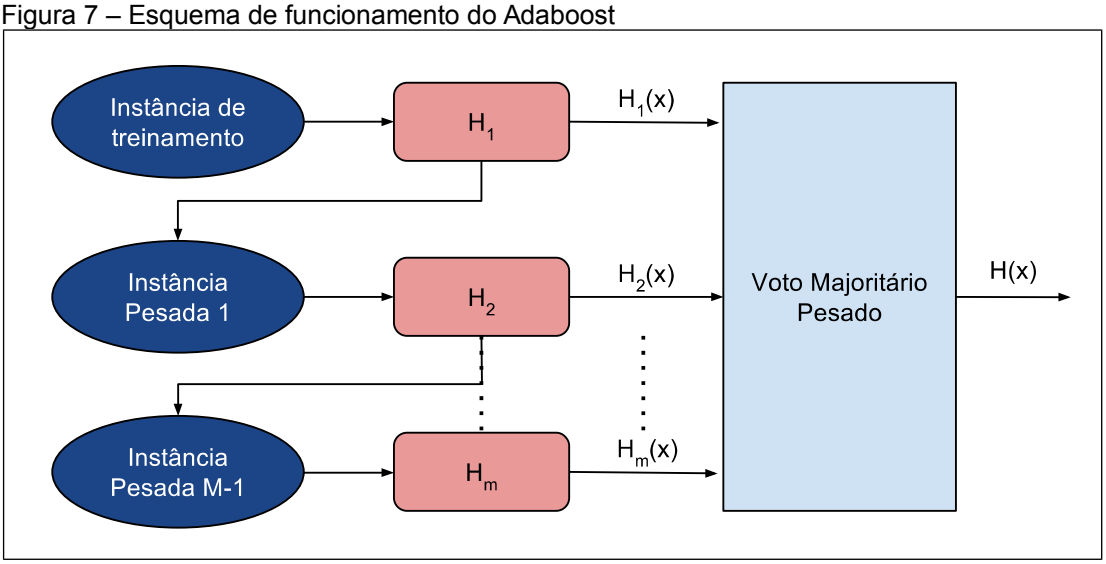
O algoritmo de *boosting* mais influente é o Adaboost, sendo o de maior destaque entre os metaclassificadores, que foi nomeado na *International Conference on Data Mining*, ICDM, conferência internacional do Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) em 2008 um dos dez algoritmos mais populares em *data mining* (ZHOU; YU, 2009, tradução nossa).

3.2 ALGORITMO ADABOOST

O algoritmo Adaboost foi proposto em 1995 por Freund e Schapire como solução a problemas existentes nos algoritmos de *boosting* anteriores, tendo sua primeira versão bem sucedida em 1999 (FREUND; SCHAPIRE, 1999, tradução nossa), posteriormente chamada de Diverse Adaboost (MATHANKER et al, 2011, tradução nossa).

Seu nome, uma contração de *Adaptive Boosting*, provém de sua capacidade de adaptar-se aos erros de classificadores pouco melhores do que decisão aleatória (FREUND; SCHAPIRE, 1995, tradução nossa).

Figura 7 – Esquema de funcionamento do Adaboost



Fonte: Ferreira e Figueiredo (2012, tradução nossa).

Em semelhança ao *boosting,* o algoritmo Adaboost também trabalha com rodadas, como exemplifica a figura 7. A cada rodada os algoritmos de treinamento são chamados iterativamente. Para a escolha dos conjuntos de dados para

treinamento dos algoritmos, o Adaboost mantém uma distribuição de instâncias de treinamento D e cada instância x_i possui um peso denominado w_i^t , que inicialmente é o mesmo em todas as instâncias da distribuição. Nas rodadas seguintes, as instâncias classificadas incorretamente terão seus pesos aumentados para que os algoritmos-base tomem sua atenção naquelas mais difíceis de serem classificadas (SHAPIRE; FREUND, 2012, tradução nossa) e permite que o Adaboost necessite de uma distribuição de treinamento muito menor do que os antigos algoritmos de *boosting* (FERREIRA; FIGUEIREDO, 2012, tradução nossa).

Na distribuição, cada instância rotulada d, representada por $(x_i, y_1), (x_i, y_2), \ldots, (x_d , y_d)$, possui uma instância x_i que é rotulada por uma classe y_i e todas as instâncias possuem valor inicial de $1/n$. Durante as t rodadas necessárias para gerar K classificadores, as instâncias têm seus pesos atualizados de acordo com o resultado das suas classificações: se classificada corretamente seu peso é aumentado e se não é classificada corretamente seu peso é diminuído. Este peso serve para medir o quão uma instância é difícil de classificar; na geração das instâncias do algoritmo seguinte; e também no cálculo do peso do algoritmo, uma vez que alguns algoritmos são mais eficientes do que outros na classificação de certas instâncias (HAN; KAMBER; PEI, 2012 tradução nossa).

Figura 8 – Pseudocódigo da geração de modelo do Adaboost

```
Atribuir peso igual para cada instância de treinamento.
Para cada iteração de t interações:
        Aplicar algoritmo de aprendizado ao conjunto de dados e
            armazenar o modelo resultante.
        Computar o erro e do modelo no conjunto de dados e armazenar
            o erro.
        Se e é igual a zero, ou maior ou igual a 0.5:
            Encerrar geração do modelo.
        Para cada instância no conjunto de dados:
            Se a instância foi classificada corretamente pelo modelo:
                Multiplicar o peso da instância por e / (1 – e).
        Normalizar o peso de todas as instâncias.
```

Fonte: Adaptado de Witten, Frank e Hall (2011, tradução nossa).

O Algoritmo Adaboost, demonstrado em pseudocódigo conforme figura 8, será descrito a seguir conforme o referencial teórico de Zaki e Meira Júnior (2014, tradução nossa) e dos autores originais Freud e Schapire (1995, tradução nossa).

Esta descrição abrange o problema da classificação binária (classificação em somente duas classes), da classificação com múltiplas classes e as diferenças existentes na forma de combinar os classificadores-base em cada uma das duas situações.

A partir de uma distribuição de dados de treinamento D, compreendendo n instâncias $\boldsymbol{x}_i$, o algoritmo Adaboost repetirá K vezes, dado que: t representa a iteração; α_t o peso do classificador M_t na t iteração; w_i^t o peso da instância $\boldsymbol{x}_i$; $\mathbf{w^t} = (w_{1,}^t w_2^t, ..., w_n^t)^T$ o vetor de probabilidade com todos os pesos das instâncias na iteração t, que se somadas resultam 1 e na primeira iteração tem seus pesos distribuídos da seguinte maneira:

$$\mathbf{w}^{\mathbf{0}} = \left(\frac{1}{n}, \frac{1}{n}, ..., \frac{1}{n} \right)^T = \frac{1}{n}\mathbf{1} \qquad (1)$$

Onde $\mathbf{1}$ representa o vetor n-dimencional de todos os 1's.

Nas sucessivas iterações, a amostra D_t de treinamento é obtida por reamostragem de $\mathbf{w}^{t-1}$ obtendo uma amostra de tamanho n, onde o ponto i é obtido pela probabilidade de w_i^{t-1}. Com essa amostra D_t o classificador M_t será treinado e também será obtido o erro de treinamento ϵ_t, conforme a expressão 2:

$$\epsilon_t = \sum_{i=0}^{n} w_i^{t-1} . I\left(M_t(x_i) \neq y_i \right) \qquad (2)$$

Sendo I uma função que retorna 1 se a classe foi atribuída *incorretamente* pelo classificador e 0 se não foi. Consequentemente, ϵ_t será a soma dos pesos das instâncias classificadas de forma errônea. O classificador que possuir ϵ_t maior ou igual a 0,5 será eliminado e o algoritmo passará para o treinamento do próximo classificador, uma vez que a precisão deste classificador está igual ou inferior à decisão aleatória. Isto também ocorre quando o ϵ_t for igual a 0, o que significa que o modelo classificou corretamente todas as instâncias e não é necessária a repesagem. Este mesmo erro de treinamento é usado no cálculo do peso do classificador M_t, como mostra a fórmula 3:

$$\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \tag{3}$$

E o peso do classificador é usado da seguinte maneira na atribuição e atualização de pesos para cada instância $\boldsymbol{x}_i$:

$$w_i^t = w_i^{t-1}.\exp\left[\alpha.I\left(M_t\left(x_i\right) \neq y_i\right)\right] \tag{4}$$

Quando todos os pesos forem atualizados (quando for necessário), parte-se para a etapa de normalização dos pesos:

$$\mathbf{w}^t = \frac{\mathbf{w}^t}{\mathbf{1}^T\mathbf{w}^t} \tag{5}$$

Ao terminar o processo de treinamento faz-se o processo de combinação dos classificadores-base para aplicação no conjunto de validação.

O peso dos classificadores é tomado em conta na hora da combinação que gerará o classificador final, de forma que a classe que será atribuída a certa instância de validação é definida pelo voto da maioria contando-se os pesos dos classificadores-base (HAN; KAMBER; PEI, 2012 tradução nossa).

O classificador resultante $\mathbf{M}^K$ da combinação, para classes binárias $\{+1,-1\}$, pode ser descrito conforme a expressão (6):

$$\mathbf{M}^K(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{K}\alpha_t M_t(\mathbf{x})\right) \tag{6}$$

Para o problema de múltiplas classes, Freund e Schapire (1995, tradução nossa) propuseram em conjunto com o algoritmo original uma extensão chamada de Adaboost.M1, que adapta o algoritmo original a este propósito tornando os classificadores-base responsáveis por atribuir uma classe a cada instância. Esta primeira variação, em contrapartida à sua simplicidade, requer que seus classificadores-base sejam muito melhores do que decisão aleatória.

Para esta extensão se faz o cálculo do voto da maioria em dada classe c_j, denotado por v_j:

$$v_j(\mathbf{x}) = \sum_{t=1}^{K} \alpha_t . I(M_t(\mathbf{x}) = c_j) \quad (7)$$

Considerando que a equação 7 apenas abrange o cálculo para determinada classe, a classe predita pela combinação dos classificadores é obtida de acordo com a expressão 8:

$$\mathbf{M}^K(\mathbf{x}) = \arg\max_{c_j} \{ v_j(\mathbf{x}) \mid j = 1,..,k \} \quad (8)$$

Pode-se destacar do funcionamento do Adaboost, dois pontos fundamentais que se diferenciam da proposta de *boosting* (POLIKAR, 2012, tradução nossa):

a) os classificadores-base do Adaboost são obtidos em uma distribuição dos dados de treinamento atualizada sucessivamente;
b) a combinação dos classificadores é feita pelo voto majoritário baseado em pesos gerados a partir dos erros dos classificadores-base na distribuição.

A figura 9 contém o algoritmo Adaboost apresentando os passos descritos anteriormente.

Figura 9 – Algoritmo Adaboost

Adaboost(*K*,D):

1 $\mathbf{w}^0 \leftarrow (1 / n) * 1 \in \mathbb{R}^n$
2 $t \leftarrow 1$
3 **enquanto** t <= K **faça**
4 $\quad \mathbf{D_t} \leftarrow$ reamostragem pesada com substituição de **D** usando $\mathbf{w}^{t-1}$
5 $\quad M_t \leftarrow$ treinar classificador em $\mathbf{D_t}$
6 $\quad \epsilon_t \leftarrow \sum_{i=0}^{n} w_i^{t-1} . I(M_t(x_i) \neq y_i)$ //taxa de erro em **D**
7 $\quad$ **se** $\epsilon_t = 0$ **então para**
8 $\quad$ **senão se** $\epsilon_t < 0.5$ **então**
9 $\quad\quad \alpha_t = \ln((1 - \epsilon_t) / \epsilon_t)$ //peso do classificadores
10 $\quad\quad$ **foreach** $i \in [1,n]$ **faça**
$\quad\quad\quad$ //atualiza os pesos
11 $\quad\quad\quad w_i^t = \begin{cases} w_i^{t-1} se\, M_t(x_i) = y_i \\ w_i^{t-1}((1-\epsilon_t)/\epsilon_t)\, se\, M_t(x_i) \neq y_i \end{cases}$
12 $\quad\quad \mathbf{w}^t = (\mathbf{w}^t / (\mathbf{1}^{\mathbf{T}}. \mathbf{w}^t))$ //normaliza os pesos
13 $\quad\quad t \leftarrow t + 1$
14 **retorna** $\{M_1, M_2, \ldots, M_k,\}$

Fonte: Zaki e Meira Júnior (2014, tradução nossa).

Este algoritmo apresentado, no entanto não é a única versão do Adaboost existente e inúmeras variantes do Adaboost foram propostas no decorrer das últimas décadas, tanto para classificação binária quanto para múltiplas classes. Dentre elas, várias se mostraram bem-sucedidas, principalmente quando aplicadas em áreas específicas. Dentre as quais podem ser destacadas para a solução de problemas de classificação binária (FERREIRA; FIGUEIREDO, 2012, tradução nossa):

a) ***emphasis boost:*** usa uma função de ênfase a fim de fazer que o algoritmo tome atenção a erros quadráticos de padrões ou a padrões próximos ao limite de classificação;
b) ***float boost:*** adiciona um estágio extra ao processo de treinamento do Adaboost, chamado de estágio de exclusão condicional, que conduz a um erro de treinamento inferior ao limite mínimo;
c) ***real Adaboost:*** trabalha com a probabilidade de um padrão pertencer a uma classe baseada no peso da distribuição atual, essas probabilidades são chamadas de valores *reais;*
d) ***gentle Adaboost:*** faz uma melhoria no algoritmo Real Adaboost usando os passos adaptativos de Newton e diferencia-se principalmente deste algoritmo pelo uso sequencial das probabilidades dos pesos para melhorar as atualizações da distribuição de treinamento;
e) ***logit boost:*** minimiza a perda logística, que é a negativa condicional log-mais-semelhante, no lugar de minimizar a perda exponencial, como ocorre no Adaboost e para isso usa dos passos adaptativos de Newton;
f) ***modest Adaboost:*** se diferencia do Adaboost por seus algoritmos-base serem treinados em uma distribuição que faz a repesagem das suas instâncias de forma invertida ao original, ou seja, as instâncias corretamente classificadas primariamente é que têm seus pesos aumentados. Por causa disso seus classificadores tendem a focar somente em seus domínios;
g) ***reweight boost:*** forma um classificador a partir do reuso de x classificadores fracos, onde esses classificadores são árvores de decisão com um único nó.

Outras variantes *multiclasse* foram propostas, como o Adaboost.MH, que faz a divisão de um problema com múltiplas classes em vários problemas binários

(SCHAPIRE; FREUND, 2012, tradução nossa); Adaboost.M2, uma segunda extensão do Adaboost original, que usa uma função de falsa perda que seja superior a decisão aleatória para recalcular os pesos das instâncias; MPBoost, faz melhorias no Adaboost.MH por meio da seleção de múltiplos pivôs; e compreendendo outras como Adaboost.MO, Adaboost.MR, Adaboost.M1W, Adaboost.SECC, BoostMA, VectorBoosting, Adaboost.ERP, Adaboost.SMO, JointBoost, ABC-Boost, Multi-Class Boosting, Adaboost.BCH, ABCBoost, Adaboost.HM e StypBoost (FERREIRA; FIGUEIREDO, 2012, tradução nossa).

Lembrando que, de acordo com o teorema de Wolpert e Macready (1996, tradução nossa), não existe um modelo classificador que possa ser superior a todos em todas as situações (como foi citado no subcapítulo 2.2, sobre a tarefa de classificação) e, portanto, as versões posteriores do Adaboost não tornam a versão original do algoritmo inferior a não ser em dada situação específica mediante avaliação de qualidade, considerando as características do algoritmo. Medidas de qualidade serão abordadas no subcapítulo seguinte.

3.2.1 Medidas de qualidade

A avaliação de modelos em *data mining* constitui uma etapa de fundamental importância a ser feita antes deles serem disponibilizados, uma vez que, geralmente, são despendidos tempo e dinheiro na geração destes modelos que podem ser desperdiçados se eles forem de baixa qualidade (LAROSE; LAROSE, 2014, tradução nossa).

A qualidade de um classificador é medida pela taxa de erro que é o número de classificações incorretas em dado conjunto. As instâncias classificadas incorretamente são denominadas erros de treinamento e as instâncias classificadas corretamente são denominadas de precisão (SCHAPIRE; FREUND, 2012, tradução nossa).

O erro de treinamento pode ser apresentado da seguinte maneira (ZAKI; MEIRA JUNIOR, 2014, tradução nossa):

$$Taxa\,de\,Erro = \frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{y}_i\right) \qquad (9)$$

E a precisão, consequentemente, da seguinte forma:

$$Precisão = \frac{1}{n}\sum_{i=1}^{n} I\left(y_i = \hat{y}_i\right) = 1 - Taxa\ de\ Erro \qquad (10)$$

A qualidade de um classificador, no entanto, não serve para medir a eficiência de um classificador em uma distribuição desconhecida. Para isso se mede a probabilidade de erros de generalização (SCHAPIRE; FREUND, 2012, tradução nossa).

Esta distribuição desconhecida é usualmente chamada de conjunto de testes (WITTEN; FRANK; HALL, 2011, tradução nossa). Geralmente os dados rotulados são divididos em três partes: dados de treinamento, usados para o treinamento do classificador; dados de validação, para melhoramentos e seleção de modelo para um conjunto de dados específico; dados de testes, para a avaliação dos classificadores gerados no treinamento. Esta divisão pode ser generalizada em dois conjuntos distintos, um agregando os dados de treinamento e validação para o ajustes da classificação e o outro os dados de testes. O uso deste último conjunto de dados no treinamento de classificadores deve ser evitado, pois pode causar *overfitting* (AGGARWAL, 2015, tradução nossa).

A proporção de divisão destes conjuntos deve ser considerada, dado que para se obter um bom classificador quanto maior o conjunto de treinamento melhor será sua generalização e, da mesma forma, quanto maior o conjunto de testes melhor será a estimativa de erro. Outro ponto a ser considerado é a necessidade de que este dois conjuntos tenham porções representativas das classes com instância divididas de forma aleatória e proporcionais entre um conjunto e outro, de forma a evitar discrepâncias injustificáveis que podem ocorrer na hora da avaliação do classificador. Esta divisão em porções representativas é chamada de *estratificação* (WITTEN; FRANK; HALL, 2011, tradução nossa).

Uma solução para qual a proporção de treinamento e para testes pode ser feita de acordo com método *holdout,* onde dois terços são cedidos para treinamento (e validação quando necessário) e o restante (um terço) para o conjunto de testes (KANTARDZIC, 2011, tradução nossa).

A avaliação do classificador em um conjunto de testes produz resultados que podem ser representados em uma matriz, chamada matriz de confusão. Na

diagonal que liga os dois pontos, verdadeiro positivo e verdadeiro negativo, têm-se os resultados classificados corretamente e nos pontos inversos, falso positivo e falso negativo, têm-se os resultados classificados incorretamente (ZAKI; MEIRA JUNIOR, 2014, tradução nossa), como ilustra a Figura 10.

Figura 10 – Matriz de Confusão

	Classe Verdadeira	
Classe Predita	Positivo (c_1)	Negativo (c_2)
Positivo (c_1)	VP	FP
Negativo (c_2)	FN	VN

Fonte: Adaptado de Gorunescu (2011, tradução nossa).

Estes pontos da matriz representados por (HAN; KAMBER; PEI, 2012, tradução nossa):

a) **Verdadeiro Positivo (VP):** refere-se ao número de instâncias positivas corretamente classificadas;
b) **Verdadeiro Negativo (VN)**: número das instâncias negativas que foram classificadas corretamente;
c) **Falso Positivo (FP):** o conjunto das instâncias negativas que foram classificadas incorretamente como positivas;
d) **Falso Negativo (FN):** refere-se ao número de instâncias positivas classificadas como negativas.

A partir dos valores destes pontos resultantes da classificação é possível obter as seguintes medidas de qualidade (LAROSE; LAROSE, 2014, tradução nossa):

a) **taxa de erro:** mede a porcentagem de instâncias classificadas incorretamente no total de instâncias classificadas;

$$Taxa\ de\ Erro = \frac{FN + FP}{VN + FN + FP + VP} \tag{11}$$

b) **acurácia:** calcula a precisão do classificador em relação a todas as instâncias classificadas;

$$Acurácia = 1 - Taxa\,de\,Erro = \frac{VN + VP}{VN + FN + FP + VP} \quad (12)$$

c) **Proporção de Falsos Positivos (PFP):** representa a taxa de falsos positivos no conjunto das instâncias classificadas como positivas;

$$PFP = \frac{FP}{FP + VP} \quad (13)$$

d) **Proporção de Falsos Negativos (PFN):** mede a relação de falsos negativos no total de instâncias classificadas como negativas;

$$PFN = \frac{FN}{FN + VN} \quad (14)$$

e) **sensibilidade:** é a proporção de instâncias classificadas como verdadeiros positivos no total de instâncias que realmente são verdadeiros positivos;

$$Sensibilidade = \frac{VP}{VP + FN} \quad (15)$$

f) **especificidade:** demonstra em relação a todas as instâncias que são realmente negativas a proporção de instâncias que foram classificadas pelo modelo como negativas.

$$Especificidade = \frac{VN}{FP + VN} \quad (16)$$

Larose e Larose (2014, tradução nossa) consideram que quanto maior a sensibilidade, especificidade e acurácia e, menor a taxa de erro, a proporção de falsos negativos e a proporção de falsos positivos, melhor será o classificador.

Outras medidas também podem ser destacadas (SOKOLOVA; LAPALME, 2009, tradução nossa; WITTEN; FRANK; HALL, 2011, tradução nossa):

h) **precisão:** proporção de verdadeiros positivos em relação a todas as instâncias calculadas como positivas:

$$Precisão = \frac{\mathrm{VP}}{\mathrm{VP}+\mathrm{FP}} \tag{17}$$

i) **F-score:** relação entre o número de instâncias positivas do conjunto de dados (representadas pelo β) e as instâncias positivas obtidas pelo classificador.

$$F-score = \frac{(\beta^2+1).\mathrm{VP}}{(\beta^2+1).\mathrm{VP}+\beta^2.\mathrm{FN}+\mathrm{FP}} \tag{18}$$

j) ***Area Under Curve* (AUC):** Capacidade do classificador evitar falsa classificação.

$$AUC = \frac{1}{2}\left(\frac{\mathrm{VP}}{\mathrm{VP}+\mathrm{FN}}+\frac{\mathrm{VN}}{\mathrm{VN}+\mathrm{FP}}\right) \tag{19}$$

k) **Estatística Kappa:** Medida da concordância entre o que foi predito e o que foi observado.

$$Kappa = \frac{\left(\frac{correto}{total}\right)-\left(\frac{concordância}{total^2}\right)}{1-\left(\frac{concordância}{total^2}\right)} \tag{20}$$

Onde:

a) *correto* representa a diagonal da matriz de confusão;

b) *concordância* representa a soma das linhas mais a soma das colunas;

c) *total* representa a soma de todos os valores.

No problema de múltiplas classes uma matriz de confusão também pode representar resultados da classificação, sendo ela uma matriz contendo o número de instâncias classificadas como *i* sendo *j* (classificada incorretamente) ou *i* sendo realmente *i* (BOROVICKA et al, 2012, tradução nossa), demostrados na diagonal da Figura 11. As medidas de qualidade, no entanto, ficam geralmente limitadas a equações “gerais” ou “por classe”(OLSON; DELEN, 2008, tradução nossa).

Figura 11 – Matriz de confusão para múltiplas classes

		Classe Verdadeira		
		A	B	C
Classe Predita	A	AA	AB	AC
	B	BA	BB	BC
	C	CA	CB	CC

Fonte: Adaptado de Felkin (2007, tradução nossa).

As métricas de qualidade para múltiplas classes são obtidas a partir dos resultados individuais para cada classe mediante ao uso de “macro-médias”, que são médias gerais obtidas a partir da soma dos resultados das equações individuais e ocasionam o tratamento igual das classes, e “micro-médias”, que também são médias, porém, por serem obtidas a partir das somas individuais de VP, VN, FP e FN geram o favorecimento das classes com maior número de instâncias, representadas por M e μ, respectivamente e o valor l representando o número de classes existentes no conjunto de dados (SOKOLOVA; LAPALME, 2009, tradução nossa):

a) **taxa de erro média:** a média das taxas de erro por classe;

$$Taxa\,de\,Erro\,Média = \frac{\sum_{i=1}^{l} \frac{FP_i + FN_i}{VP_i + FN_i + FP_i + VN_i}}{l} \quad (21)$$

b) **acurácia média:** a média resultante das acurácias por classe;

$$Acurácia\,Média = \frac{\sum_{i=1}^{l} \frac{VP_i + VN_i}{VP_i + FN_i + FP_i + VN_i}}{l} \quad (22)$$

c) **sensibilidade *M*:** média das efetividades por classe de um classificador;

$$Sensibilidade\,M = \frac{\sum_{i=1}^{l} \frac{VP_i}{VP_i + FN_i}}{l} \quad (23)$$

d) **sensibilidade μ:** efetividade do classificador em relação a soma de todas as instâncias que realmente são verdadeiros positivos;

$$Sensibilidade\,\mu = \frac{\sum_{i=1}^{l} VP_i}{\sum_{i=1}^{l} (VP_i + FN_i)} \tag{24}$$

e) **precisão *M*:** média das precisões individuais;

$$Precisão\,M = \frac{\sum_{i=1}^{l} \frac{VP_i}{VP_i + FN_i}}{l} \tag{25}$$

f) **precisão μ:** proporção de verdadeiros positivos em relação a todas as instâncias calculadas como positivas a partir da somas dos resultados individuais;

$$Precisão\,\mu = \frac{\sum_{i=1}^{l} VP_i}{\sum_{i=1}^{l} (VP_i + FP_i)} \tag{26}$$

g) **F-score *M*:** relação baseada em média por classe das instâncias positivas obtidas pelo classificador e as instâncias do conjunto de dados.

$$F - Score\,M = \frac{(\beta^2 + 1)\,Precisão_M . Sensibilidade_M}{\beta^2 . Precisão_M + Sensibilidade_M} \tag{27}$$

h) **F-score μ:** relação entre o número de instâncias positivas do conjunto de dados e as instâncias positivas obtidas pelo classificador considerando a soma dos resultados individuais;

$$F - Score\,\mu = \frac{(\beta^2 + 1)\,Precisão_\mu . Sensibilidade_\mu}{\beta^2 . Precisão_\mu + Sensibilidade_\mu} \tag{28}$$

Medidas de qualidade, como estas que foram apresentadas, geralmente são implementadas em conjunto com ferramentas de *data mining* para avaliar a qualidade do classificador gerado, como ocorre com a Shell Orion Data Mining Engine, apresentada na seção 3.3.

3.3 SHELL ORION DATA MINING ENGINE

Ferramentas de *data mining* são comumente utilizadas para a aplicação de algumas etapas do KDD, de forma a auxiliar nos processos necessários à obtenção de conhecimento. Existem inúmeras ferramentas disponíveis, sendo a grande maioria paga e de alto custo.

A Shell Orion Data Mining Engine é uma ferramenta de *data mining* desenvolvida pelo Grupo de Pesquisa em Inteligência Computacional Aplicada do curso de Ciência da Computação da Universidade do Extremo Sul Catarinense (UNESC) desde 2005 e tem sido desenvolvida pelos acadêmicos em seus Trabalhos de Conclusão de Curso (TCC) (NANDI, 2013).

Nas tabelas 1 e 2 têm-se os métodos e algoritmos já implementados na ferramenta pelos acadêmicos da universidade, desde o ano de 2005, compreendendo os métodos de associação, classificação e agrupamento em atributos nominais e, nominais e numéricos em conjunto:

Tabela 1 – Evolução da Shell Orion Data Mining Engine

Ano	Tarefa	Método	Algoritmo	Atributos	Referência
2005	Associação	Regra de Associação	Apriori	Numérico	(CASAGRANDE, 2005)
2005	Classificação	Árvore de Decisão	ID3	Nominais	(PELEGRIM, 2005)
2007	Classificação	Árvore de Decisão	CART	Nominais e numéricos	(RAIMUNDO, 2007)
2007	Agrupamento	Particionamento	K-Means	Numéricos	(MARTINS, 2007)
2007	Agrupamento	Redes Neurais	Kohonen	Numéricos	(BORTOLOTTO, 2007)
2008	Agrupamento	Lógica Fuzzy	Gustafson-kessel	Numéricos	(CASSETARI JUNIOR, 2008)
2009	Agrupamento	Lógica Fuzzy	Gath-Geva	Numéricos	(PEREGO, 2009)

Fonte: Adaptado de Nandi (2013).

Tabela 2 – Evolução da Shell Orion Data Mining Engine (continuação)

Ano	Tarefa	Método	Algoritmo	Atributos	Referência
2009	Classificação	Árvore de Decisão	C4.5	Nominais e numéricos	(MONDARDO, 2009)
2009	Classificação	Redes Neurais	RBF	Numéricos	(SCOTTI, 2010)
2010	Agrupamento	Lógica Fuzzy	RCP	Numéricos	(CROTTI JUNIOR, 2010)
2010	Agrupamento	Lógica Fuzzy	URCP	Numéricos	(CROTTI JUNIOR, 2010)
2010	Agrupamento	Lógica Fuzzy	FCM	Numéricos	(CROTTI JUNIOR, 2010)
2011	Agrupamento	Densidade	DBSCAN	Numéricos	(GAVA, 2011)
2012	Agrupamento	Enxame – Colônia de formigas	SACA	Numéricos	(GHELLERE, 2012)
2012	Agrupamento	Enxame – Colônia de formigas	Ant-Based Clustering	Numéricos	(GHELLERE, 2012)
2012	Agrupamento	Enxame – Colônia de formigas	A²CA	Numéricos	(GHELLERE, 2012)
2012	Classificação	Bayesiano	Naive Bayes	Numéricos	(NOVASKI, 2012)
2013	Associação	Regras de Associação	FP-Growth	Numéricos	(NANDI, 2013)

Fonte: Adaptado de Nandi (2013).

Por ser desenvolvida na linguagem de programação Java, a Shell Orion Data Mining Engine consequentemente herda as caraterísticas multiplataforma da linguagem, podendo ser instalada em diferentes sistemas operacionais, além de utilizar a Interface de Programação de Aplicações (*Application Programing Interface* – API) Java Database Connectivity (JDBC), o que a faz também independente da Base de Dados a ser utilizada. Esta pesquisa busca disponibilizar um metaclassificador por meio do algoritmo de *boosting* Adaboost para a tarefa de classificação na *Shell Orion Data Mining Engine* e obter resultados satisfatórios na tarefa de classificação desta ferramenta.

3.4 TRABALHOS CORRELATOS

Nos subcapítulos a seguir são abordados cinco casos do uso do Adaboost, como na identificação de gênero para aplicações de robô de serviço (LUO; LIN; CHEN, 2011, tradução nossa), na classificação de defeito em nozes-pecã

(MATHANKER et al, 2011, tradução nossa), aplicado a sensores e sistemas embarcados (CHAVES, 2012), na detecção e contagem automática de plaquetas (NASCIMENTO, 2011) e na detecção de sinais de trânsito por (REIS, 2013).

3.4.1 Gender Recognition Based on Ensemble Learning with Selective Features for Service Robotics Applications

O artigo *Gender Recognition Based on Ensemble Learning with Selective Features for Service Robotics Applications,* publicado na conferência *International Conference on Robotics and Biomimedics* da IEEE, sigla de Instituto de Engenheiros Eletricistas e Eletrônicos, em 2011, na Tailândia, por Ren C. Luo, Tzu Ta Lin e Kuan Yu Chen trata do reconhecimento de gênero com características seletivas para aplicações de robôs de serviço.

Neste Artigo Luo, Lin e Chen (2011, tradução nossa) propuseram o uso de algoritmos baseados em SVM para a geração dos conjuntos de classificadores nos os algoritmos Rotation Forest e Discrete Adaboost para a otimização na diferenciação de gênero (em masculino e feminino) em imagens de faces frontais e em ângulos aproximados de 10 graus.

Luo, Lin e Chen (2011, tradução nossa) coletaram 304 imagens frontais de face para o treinamento dos classificadores, divididas em mesma quantidade para ambos os gêneros, sendo 107 frontais e 125 em ângulos de 10 graus, provenientes da base de dados da DARPA (Defense Advanced Research Products Agency, sigle em inglês). Em seguida, implementaram, fizeram o treinamento e os testes com os algoritmos PCA, Gabor, RAW LBP, WLD, LDP2 E LDP3 e posteriormente usaram estes algoritmos como base para o Rotation Forest e o Adaboost.

As otimizações obtidas pelos algoritmos Adaboost e Rotation Forest, principalmente com o primeiro algoritmo foram promissoras. Em imagens de tamanho 36 por 36 frontais, o Adaboost obteve 88% de precisão ante 84% do melhor individual. Em imagens de tamanho 48 por 48 o Adaboost chegou a obter 90% e em ângulo, até 72% de acertos.

3.4.2 AdaBoost Classifiers for Pecan Defect Classification

O artigo *AdaBoost Classifiers for Pecan Defect Classification* de S.K. Mathanker, P.R. Weckler, T.J. Bowsera, N. Wanga, N.O. Manessb, foi aceita na revista científica *Computers and Electronics in Agriculture* em 25 de março de 2011, da Universidade do Estado de Oklahoma, Estados Unidos e trata da utilização das versões Diverse Adaboost, Real Adaboost, Gentle Adaboost e Star Adaboost do algoritmo Adaboost em comparação com SVM na classificação de defeito em nozes-pecã.

Mathanker et al (2011, tradução nossa) justificam a importância da pesquisa da classificação nozes-pecã por defeito pelo fato de que leis europeias atuais punem os produtores por doenças ou injúrias causadas pelos alimentos, que podem gerar altos custos de reparação, além de considerar o controle de qualidade dos alimentos uma tendência em todo o globo. Para isso os autores propuseram o uso de versões do Adaboost, SVM e classificador Bayesiano para diferenciação de nozes com e sem defeitos.

Cem amostras de imagem de pecãs com defeito e cem amostras de pecã sem defeito foram obtidas em raio-X e segmentadas em três métodos de segmentação: *Reverse water flow*, *Oh* e *Twice Otsu*. Os classificadores Diverse AdaBoost, Real AdaBoost, Gentle AdaBoost, Star AdaBoost, SVM Radial, SVM Linear, SVM Quadratic e Bayesiano foram treinados e posteriormente testados nestas amostras, seus tempos também foram calculados.

Os algoritmos Adaboost obtiveram bom desempenho em todos os métodos, SVM obteve bom desempenho somente no método *Twice Otsu*. A versão Real Adaboost obteve o melhor desempenho, alcançando 92,2% no método *Reverse* e também melhor tempo (10^{-6} segundos) em relação aos outros algoritmos. Nos métodos *Oh* e *Twice Otsu*, o mesmo algoritmo obteve 92,3% e 92,7%, sendo que de modo geral os algoritmos variantes do Adaboost obtiveram o melhor desempenho.

3.4.3 Estudo do Algoritmo Adaboost de Aprendizagem de Máquina aplicado a Sensores e Sistemas Embarcados

A dissertação de mestrado de Bruno Butilhão Chaves, *Estudo do Algoritmo Adaboost de Aprendizagem de Máquina aplicado a Sensores e Sistemas Embarcados*, foi apresentada em 2012 para a obtenção do Título de Mestre em Engenharia Mecânica na Escola Politécnica da Universidade de São Paulo.

Chaves (2012) tinha como objetivo o uso do algoritmo Adaboost na aplicação em sensores isolados, em sistemas complexos e em dispositivos autônomos com sensores para a obtenção uma melhor precisão e sensibilidade e na implementação deste algoritmo em processadores com classificador embarcado.

Para verificar da viabilidade técnica, o autor realizou um estudo de caso como sistema de sensores para a verificação de adulteração em combustíveis etanol. O microprocessador utilizado foi um Arduino Mega 1280 e linguagem de programação C++ e também foi utilizado um display LCD para a exibição dos resultados.

Os resultados foram promissores, atingindo um percentual de 90% de classificações corretas, indicando a viabilidade do sistema para a verificação de adulteração em etanol, além de demostrar a possibilidade de se utilizar o algoritmo Adaboost em situações não triviais.

3.4.4 Classificação Adaboost para Detecção e Contagem Automática de Plaquetas

Este trabalho de conclusão de curso, escrito por Débora Natália de Oliveira Nascimento em 2011 para obtenção do Grau de Bacharel em Engenharia de Computação pela Universidade de Pernambuco apresenta o uso do algoritmo Adaboost para a detecção e contagem automática de plaquetas.

Nascimento (2011) propôs neste trabalho o desenvolvimento de um sistema de baixo custo para a identificação e contagem de plaquetas sanguíneas com o uso do algoritmo Adaboost e técnicas de pré-processamento de imagens. Também foi objetivado que o sistema fosse simples e fácil manuseio.

Em primeiro momento foi feito o pré-processamento das imagens sanguíneas, obtidas a partir do banco de imagens da Sociedade Americana de

Hematologia. Após esta etapa, partiu-se para a seguinte, que é o desenvolvimento do sistema. Para isso foi utilizada a linguagem de programação C/C++, e dividida esta etapa em três, constituem a montagem do conjunto de algoritmos-base baseado em uma estrutura proposta por Viola e Jones (2004 apud NASCIMENTO, 2011), chamada cascata de classificadores; o treinamento do algoritmo Adaboost; e o desenvolvimento e teste do sistema.

Os resultados obtidos demostraram viabilidade do sistema de contagem de plaquetas, alcançando taxas de acertos, em imagens preprocessadas em canal verde e equalização local, de 92%, equalização global, 95%, em tons de cinza e equalização local, 90% e tons de cinza com equalização global, taxas de 98%. Os resultados também demonstraram a viabilidade da aplicação do sistema na detecção de outros tipos de células sanguíneas em trabalhos futuros.

3.4.5 Detecção de Sinais de Trânsito Através do Método de Classificação Adaboost

O artigo de Willian Augusto Dias dos Reis, *Detecção de Sinais de Trânsito Através do Método de Classificação Adaboost*, foi publicado na revista UNOPAR Cietífica Ciências Exatas e Tecnológicas da Universidade Norte do Paraná em novembro de 2013.

No artigo, o autor aplica o algoritmo Adaboost para o reconhecimento de sinais de trânsito, como forma a tornar possível uma visibilidade destacada destes sinais a partir de um painel ou um aviso sonoro e assim reduzir a desatenção do motorista à sinalização local. Na aplicação do projeto os algoritmos-base foram treinados somente para identificar a placa PARE (REIS, 2013).

Para o desenvolvimento do projeto foi utilizada a biblioteca OpenCV, que possui algumas ferramentas implementadas, dentre as quais foram utilizadas: Opencv-createsamples.cpp, para a geração e preparação das imagens que serão utilizadas para treinamento; Opencv-haartraining.cpp, para o treinamento dos classificadores, Opencv-performance.cpp, para testes de desempenho. O algoritmo de base foi Arvores Generativas de Decisão e foram criados três classificadores.

Ao aplicar os classificadores nas amostras de testes foi possível observar que a medida que o número de estágios foi aumentando, a taxa de erro foi

diminuindo e com isso, foi possível demonstrar que mesmo em testes mínimos, como os que foram apresentados, é possível construir uma ferramenta classificatória que consiga detectar sinais de trânsito.

4 O ALGORITMO ADABOOST NA TAREFA DE CLASSIFICAÇÃO DA SHELL ORION DATA MINING ENGINE

A Shell Orion Data Mining Engine é uma ferramenta gratuita concedida pelo Grupo de Pesquisa em Inteligência Computacional Aplicada do Curso de Ciência da Computação da UNESC. Esta ferramenta tem a função de auxiliar na descoberta de conhecimento em bases de dados e, por meio da adição de módulos desenvolvidos por acadêmicos em seus Trabalhos de Conclusão de Curso, é constantemente aprimorada e atualizada.

Esta pesquisa contribui com a Shell Orion a partir do desenvolvimento do algoritmo metaclassificador Adaboost na tarefa de classificação, sendo aplicado em duas bases de dados, uma binária e outra de múltiplas classes.

Para isso, foram selecionadas e descritas as bases de dados; seguidas as etapas metodológicas para o desenvolvimento do Adaboost, que incluem a modelagem matemática, modelagem UML[1], a implementação do algoritmo e a aplicação de métricas de qualidade na classificação; e avaliado os tempos de processamento do Adaboost na Shell Orion com relação ao mesmo algoritmo implementado em outra ferramenta de *data mining*.

4.1 BASES DE DADOS

As bases de dados utilizadas para a avaliação do algoritmo Adaboost foram obtidas na UCI *Machine Learning Repository*[2], que possui inúmeras bases de dados para *data mining*. Na seleção das bases de dados foram consideradas aquelas que compreendem o mundo real e, portanto, suas classes não são necessariamente balanceadas.

A primeira base de dados empregada nesta pesquisa, *Chronic Kidney Disease*, é constituída de 400 instâncias referentes a identificação da presença de doença renal crônica em pacientes do Apollo Hospitals, na Índia, durante um período de dois meses, tendo sido criada em 2015 (LICHMAN, 2016, tradução nossa). Das

[1] Unified Modeling Language. Mais informações na seção 4.2.2.

[2] Repositório de bases de dados fundado pelo Departamento de Ciência da Computação da Universidade da Califórnia, Irvine e mantido em colaboração com a Universidade de Massachusetts. Disponível gratuitamente em (http://archive.ics.uci.edu/ml/index.html).

400 instâncias presentes na base de dados, 197 não puderam ser aproveitadas por possuírem valores faltantes, restando 203 instâncias que foram utilizadas.

Chronic Kidney Disease possui 24 atributos por registro mais o atributo da classe, detalhados na tabela 3. Os valores do atributo de classe são divididos binariamente entre os pacientes que possuem doença renal crônica (*ckd*) e os que não possuem (*notckd*).

Tabela 3 – Atributos da base de dados *Chronic Kidney Disease*

Atributo	Valor	Descrição
age	Numérico	Idade em anos
bp	Numérico	Pressão Sanguínea em mm/Hg
sg	Nominal (1.005, 1.010, 1.015, 1.020, 1.025)	Gravidade Específica
al	Nominal (1 a 5)	Albumina
su	Nominal (1 a 5)	Açúcar
rbc	Nominal (normal, abnormal)	Glóbulos vermelhos no sangue
pc	Nominal (normal, abnormal)	Células de pus
pcc	Nominal (present, notpresent)	Células de pus aglomeradas
ba	Nominal (present, notpresent)	Bactéria
brg	Numérico	Glicose no sangue aleatório
bu	Numérico	Ureia no sangue
sc	Numérico	Creatinina Serum
sod	Numérico	Sódio
pot	Numérico	Potássio
hemo	Numérico	Hemoglobina
pcv	Numérico	Volume globular
wc	Numérico	Contagem de glóbulos brancos no sangue
rc	Numérico	Contagem de glóbulos vermelhos no sangue
htn	Nominal (yes, no)	Hipertensão
dm	Nominal (yes, no)	Diabetes Mellitus
cad	Nominal (yes, no)	Doença da artéria coronária
appet	Nominal (good, poor)	Apetite
pe	Nominal (yes, no)	Edema na região dos pés
ane	Nominal (yes, no)	Anemia
class	Nominal (ckd, notckd)	Presença ou ausência de doença renal crônica

Fonte: Adaptado de Lichman (2016, tradução nossa).

A segunda base de dados, *Seeds*, conta com 210 instâncias para a diferenciação de sementes de trigo em três variedades: *Kama*, *Rosa* e *Canadian*. Os dados foram obtidos a partir do raio-X do núcleo das sementes (CHARYTANOWICZ, 2010, tradução nossa). Não foi necessária a remoção de instâncias, pois a base de dados não possui valores faltantes.

Seeds possui 8 atributos contando com o atributo de classe, descritos na tabela 4, tendo sido gerada no ano de 2010, pela Universidade de Lublin em conjunto com a Cracow University of Technology.

Tabela 4 – Atributos da base de dados *Seeds*

Atributo	Valor	Descrição
a	Numérico	Área da semente na radiografia
p	Numérico	Perímetro
c	Numérico	Compacidade
l	Numérico	Comprimento do núcleo
w	Numérico	Largura do núcleo
ac	Numérico	Coeficiente de assimetria
lkg	Numérico	Comprimento da ranhura do núcleo
class	Nominal (1 a 3)	Semente: Kama(1), Rosa (2), Canadian (3)

Fonte: Adaptado de Charytanowicz (2010, tradução nossa).

4.2 METODOLOGIA

Para alcançar os resultados almejados levou-se em conta a seguinte metodologia: levantamento bibliográfico; modelagem matemática do algoritmo de classificação Adaboost; modelagem do módulo utilizando Unified Modeling Language (UML); implementação do algoritmo; implementação de medidas de qualidade em *data mining* para a classificação de dados; realização de teste e correção dos possíveis erros; aplicação da base de dados binária *Chronic Kidney Disease* e da base de dados com múltiplas classes *Seeds*; análise dos resultados obtidos.

Na etapa do levantamento bibliográfico foram fundamentados e compreendidos os temas sobre descoberta de conhecimento em bases de dados, *data mining*, a tarefa de classificação, metaclassificadores, *boosting*, o algoritmo

Adaboost, a ferramenta Shell Orion e medidas de qualidade na classificação para a dados.

4.2.1 Modelagem matemática do algoritmo de classificação Adaboost

A etapa de modelagem matemática do algoritmo Adaboost visa esclarecer seu funcionamento e proporcionar um melhor entendimento do mesmo.

No desenvolvimento da modelagem matemática utilizou-se como base o capítulo 1 do livro de Shapire e Freund (2012, tradução nossa) intitulado *Boosting: Foundations and Algorithms*. Os formalismos utilizados seguiram os apresentados por Zaki e Meira Júnior (2014, tradução nossa) no livro *Data Mining and Analysis: Fundamental Concepts and Algorithms*.

Em um conjunto de treinamento, obtido de uma base de dados binária hipotética, com 10 instâncias divididas em duas classes (+1, -1), conforme ilustra a figura 12:

Figura 12 – Conjunto de treinamento

índice	0	1	2	3	4	5	6	7	8	9
classe	1	1	1	-1	-1	-1	1	1	-1	-1

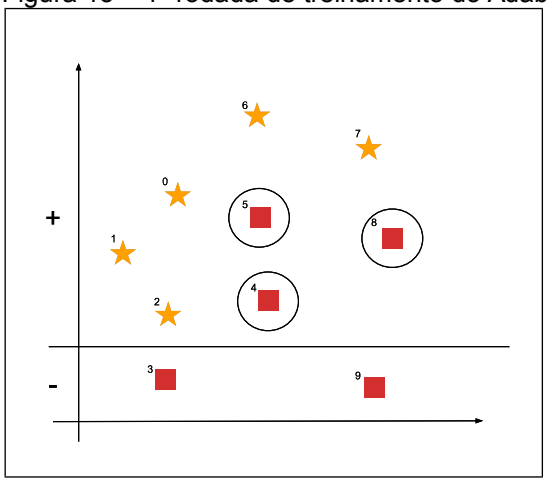
Fonte: Do autor.

Pode-se criar o vetor de probabilidades com o mesmo número de pesos correspondente ao número de instâncias, no qual, na primeira rodada ($t = 1$), é amostrado para cada peso o valor de um dividido pelo total de instâncias do conjunto:

$$\mathbf{w}^{\mathbf{0}} = \left(\frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}\right) = \frac{1}{10}\mathbf{1}$$

Com a distribuição obtida deste vetor faz-se o treinamento do primeiro classificador. Considerando um algoritmo-base que faça predições de forma linear, como o algoritmo 1-Rule (descrito na seção 4.2.3.1), e que o classificador gerado faça a predição correta de 7 das 10 instâncias (instâncias 0, 1, 2, 3, 6, 7 e 9), é possível representá-lo com o gráfico da figura 13.

Figura 13 – 1ª rodada de treinamento do Adaboost



Fonte: Adaptado de Chaves (2012).

As estrelas presentes no gráfico simbolizam as instâncias de classe positiva (de classe +1) e os quadrados simbolizam as instâncias de classe negativas (de classe -1), enquanto que a posição de cada instância retrata os valores dos atributos das mesmas. A linha divisória do gráfico separa na parte inferior as instâncias classificadas como negativas e a parte superior as classificadas como positivas. Os quadrados circulados evidenciam três instâncias negativas erroneamente classificadas como positivas.

As classificações corretas e incorretas são obtidas pelo processo de comparação do valor do atributo de classe, que é a classe verdadeira da instância, e o valor do atributo de classe predito pelo classificador. Neste processo, o erro de treinamento ϵ_1 pode ser calculado pela soma dos pesos das instâncias que foram classificadas incorretamente, sendo ele, neste caso, a soma dos pesos das instâncias 4, 5 e 8.

$$\epsilon_1 = \sum_{i=0}^{9} w_i^{1-1} . I\left(M_1(x_i) \neq y_i\right) = \left(\frac{1}{10} + \frac{1}{10} + \frac{1}{10}\right) = 0{,}3000$$

O erro de treinamento foi diferente de 0 e menor que 0,5. Caso fosse igual a 0 significaria que o classificador foi perfeito, o que não ocorreu e, portanto, não seria necessário mais rodadas, e se fosse maior que 0,5 ultrapassaria o limite inferior à decisão aleatória, tornando-o inválido para ser usado como base do Adaboost.

Como neste exemplo o erro de treinamento foi de 0,3, pode-se fazer o cálculo do peso do classificador, que é medido pelo logaritmo de 1 menos o erro de treinamento dividido pelo erro de treinamento (explanado na equação 3 presente na seção 3.2). Este peso será armazenado para ser usado na geração do classificador final ao término das rodadas do Adaboost.

$$\alpha_1 = \ln\left(\frac{1-\epsilon_1}{\epsilon_1}\right) = \ln\left(\frac{1-0{,}3000}{0{,}3000}\right) \approx 0{,}8473$$

Em seguida, o valor do erro de treinamento é utilizado para realizar a atualização dos pesos das instâncias, sendo aumentado os pesos das instâncias 4, 5 e 8, pois foram classificadas incorretamente.

$$w_4^1 = 0{,}1\left((1-0{,}3000)/0{,}3000\right) \approx 0{,}2333$$
$$w_5^1 = 0{,}1\left((1-0{,}3000)/0{,}3000\right) \approx 0{,}2333$$
$$w_8^1 = 0{,}1\left((1-0{,}3000)/0{,}3000\right) \approx 0{,}2333$$

Posteriormente, faz-se a normalização dos pesos. A normalização é feita pela divisão do peso de cada membro pela soma de todos os pesos dos membros da distribuição. A figura 14 apresenta os pesos anteriores, que foram os pesos atribuídos ao criar o vetor de probabilidades; os pesos após a atualização gerada pelas classificações incorretas; e os pesos depois da normalização.

Figura 14 – Distribuição atualizada e normalizada na 1ª rodada

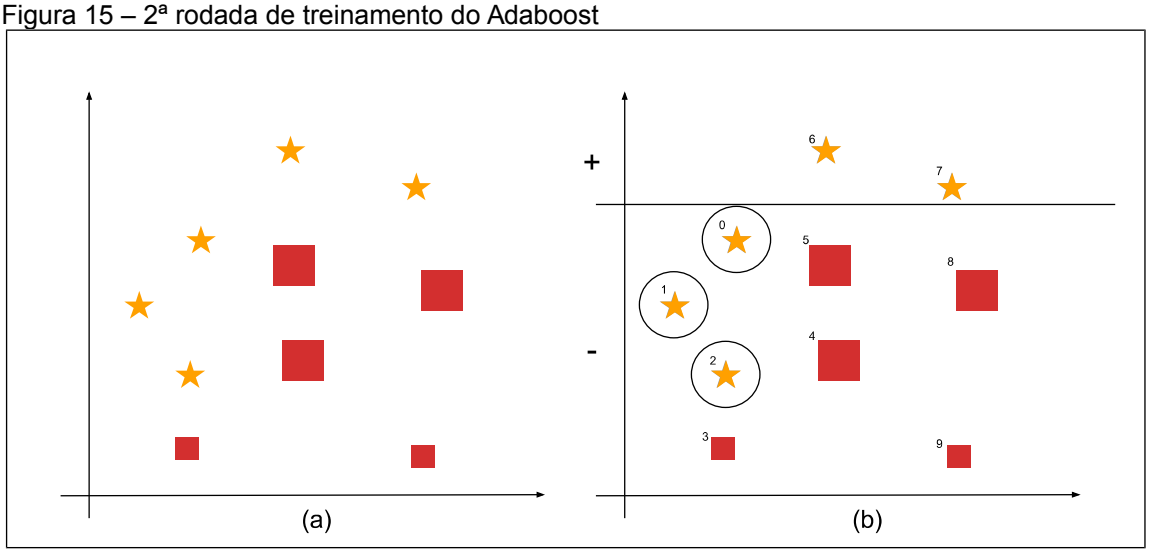
índice	0	1	2	3	4	5	6	7	8	9
pesos anteriores ($\mathbf{w^0}$)	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
pesos atualizados	0,1	0,1	0,1	0,1	0,2333	0,2333	0,1	0,1	0,2333	0,1
pesos normalizados ($\mathbf{w^1}$)	0,0714	0,0714	0,0714	0,0714	0,1667	0,1667	0,0714	0,0714	0,1667	0,0714

Fonte: Do autor.

Ao término da normalização, segue-se para a segunda rodada ($t = 2$), porém com um adendo: as instâncias classificadas incorretamente na rodada anterior (as instâncias 4, 5 e 8) têm maior relevância nesta rodada (a relevância é observável pela linha *pesos normalizados* da figura 14).

Esta relevância dos pesos é aplicada pela adição do vetor de probabilidades normalizado junto com a distribuição de treinamento diretamente no código do algoritmo que gerará o classificador ou, em algoritmos que não aceitem os pesos, é feita pela reamostragem considerando os pesos.

Figura 15 – 2ª rodada de treinamento do Adaboost



Fonte: Adaptado de Chaves (2012).

No gráfico (a) da figura 15 nota-se os quadrados referentes às instâncias classificadas erroneamente representados em tamanho maior para indicar sua relevância. No gráfico (b) tem-se a predição do segundo classificador influenciada pela predição do classificador anterior. Nota-se três círculos neste mesmo gráfico que demonstram o erro do classificador nas instâncias 0, 1 e 2, sendo possível fazer o cálculo do erro de treinamento.

$$\epsilon_2 = \sum_{i=0}^{9} w_i^1 . I\left(M_2(x_i) \neq y_i\right) = \left(0{,}0714 + 0{,}0714 + 0{,}0714\right) = 0{,}2142$$

Conseguintemente, faz-se o cálculo do peso do classificador.

$$\alpha_2 = \ln\left(\frac{1-\epsilon_2}{\epsilon_2}\right) = \ln\left(\frac{1-0{,}2142}{0{,}2142}\right) \approx 1{,}2998$$

Em seguida, atualiza-se o peso das instâncias classificadas incorretamente.

$$w_0^2 = 0{,}0714((1-0{,}2142)/0{,}2142) \approx 0{,}2619$$
$$w_1^2 = 0{,}0714((1-0{,}2142)/0{,}2142) \approx 0{,}2619$$
$$w_2^2 = 0{,}0714((1-0{,}2142)/0{,}2142) \approx 0{,}2619$$

Por fim, faz-se a normalização dos pesos, demonstrada na figura 16.

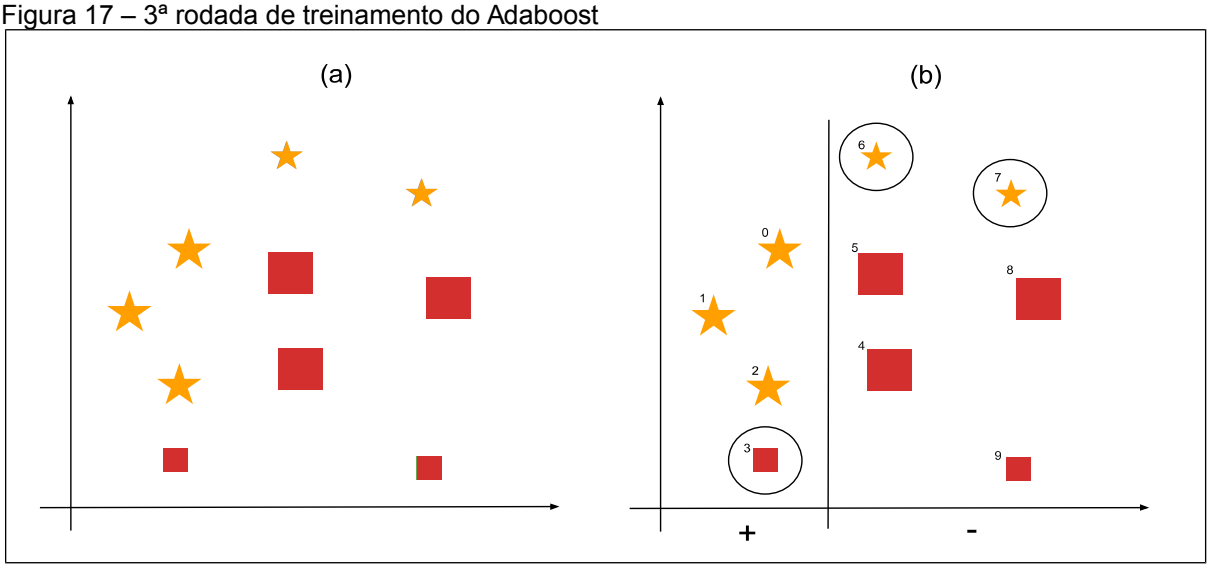
Figura 16 – Distribuição atualizada e normalizada na 2ª rodada

índice	0	1	2	3	4	5	6	7	8	9
pesos anteriores ($\mathbf{w^0}$)	0,0714	0,0714	0,0714	0,0714	0,1667	0,1667	0,0714	0,0714	0,1667	0,0714
pesos atualizados	0,2619	0,2619	0,2619	0,0714	0,1667	0,1667	0,0714	0,0714	0,1667	0,0714
pesos normalizados ($\mathbf{w^1}$)	0,1667	0,1667	0,1667	0,0454	0,1061	0,1061	0,0454	0,0454	0,1061	0,0454

Fonte: Do autor.

Isto feito, pode-se passar para a terceira rodada ($t = 3$) do Adaboost. Da mesma forma que ocorreu na rodada anterior, atribui-se maior importância às instâncias de maior peso.

Figura 17 – 3ª rodada de treinamento do Adaboost



Fonte: Adaptado de Chaves (2012).

Na figura 17 tem-se o resultado da classificação nesta rodada. O classificador errou nas instâncias 3, 6 e 7, portanto, seu erro de treinamento é:

$$\epsilon_3 = \sum_{i=0}^{9} w_i^2 . I(M_3(x_i) \neq y_i) = (0{,}0454 + 0{,}0454 + 0{,}0454) = 0{,}1362$$

Dado o erro de treinamento, o peso do classificador é obtido por:

$$\alpha_3 = \ln\left(\frac{1-\epsilon_3}{\epsilon_3}\right) = \ln\left(\frac{1-0{,}1362}{0{,}1362}\right) \approx 1{,}8472$$

Em sequência, as instâncias são atualizadas.

$$w_3^3 = 0{,}0454((1-0{,}1362)/0{,}1362) \approx 0{,}2879$$
$$w_6^3 = 0{,}0454((1-0{,}1362)/0{,}1362) \approx 0{,}2879$$
$$w_7^3 = 0{,}0454((1-0{,}1362)/0{,}1362) \approx 0{,}2879$$

Finda a atualização, segue-se com a normalização dos pesos.

Figura 18 – Distribuição atualizada e normalizada na 3ª rodada

índice	0	1	2	3	4	5	6	7	8	9
pesos anteriores ($\mathbf{w^0}$)	0,1667	0,1667	0,1667	0,0454	0,1061	0,1061	0,0454	0,0454	0,1061	0,0454
pesos atualizados	0,1667	0,1667	0,1667	0,2879	0,1061	0,1061	0,2879	0,2879	0,1061	0,0454
pesos normalizados ($\mathbf{w^1}$)	0,0965	0,0965	0,0965	0,1667	0,0614	0,0614	0,1667	0,1667	0,0614	0,0263

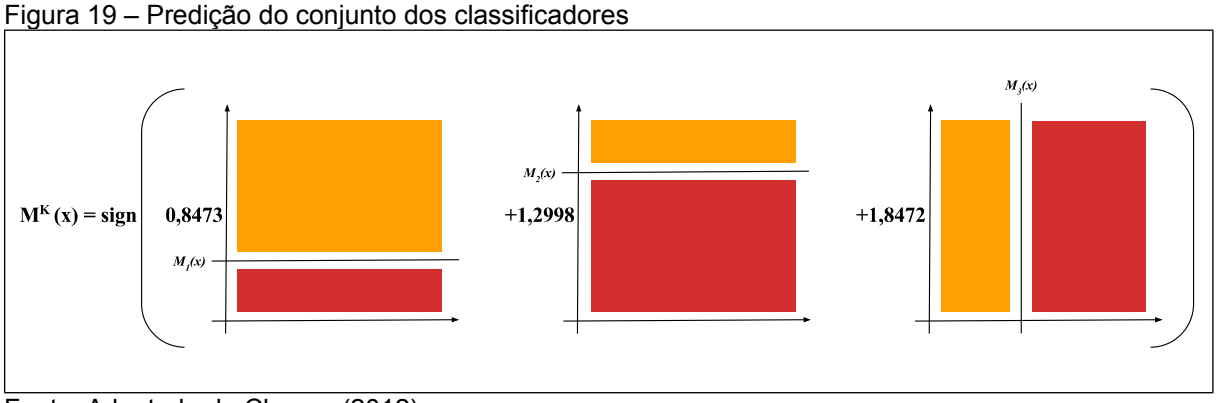
Fonte: Do autor.

É possível notar que este último classificador foi induzido a concentrar-se justamente nas instâncias que os classificadores anteriores falharam, o que pode ser justificado pela maior importância dada a elas. Também é possível notar que sua predição se torna complementar às predições dos outros dois classificadores.

Com essa função complementar entre os classificadores gerados, a combinação feita no Adaboost pode predizer corretamente todas as instâncias da

distribuição. A figura 19 representa esta combinação dos classificadores, tendo sido os pesos α_i de cada classificador $(M_i(x))$ introduzidos ao lado de cada gráfico.

Figura 19 – Predição do conjunto dos classificadores



Fonte: Adaptado de Chaves (2012).

O resultado da classificação torna-se o sinal da classe (+ e -) considerando o peso de cada membro e sua resposta. Por exemplo, a instância de índice 8 foi classificada como positiva (+1) pelo primeiro classificador, então adiciona-se para esta classe o peso de 0,8473 deste classificador; o segundo classificador classificou-a como negativa (-1), atribuindo para a classe 1,2998; por fim, o terceiro classificador também classificou a instância 8 como negativa, acrescentando o peso de 1,8472, o que resulta na equação:

$$\begin{aligned} & sign\left(M_1(x)*(+1) \;+\; M_2(x)*(-1) \;+\; M_3(x)*(-1)\right) \\ &= \; 0{,}8473 - 1{,}2998 - 1{,}8472 \\ &= \; sign\left(-2{,}2997\right) \\ &= \; \mathbf{-1} \end{aligned}$$

O resultado da combinação foi o sinal obtido da classe -1, que é a classe correta. Em um segundo exemplo, com a instância 6, que possui classe +1, o resultado também é correto.

$$\begin{aligned} & sign\left(M_1(x)*(+1) \;+\; M_2(x)*(+1) \;+\; M_3(x)*(-1)\right) \\ &= \; 0{,}8473 + 1{,}2998 - 1{,}8472 \\ &= \; sign\left(0{,}2999\right) \\ &= \; \mathbf{+1} \end{aligned}$$

Como foi mencionado ao início da modelagem, o caso supracitado foi aplicado em uma base de dados binária. Bases de dados com múltiplas classes já são consideradas no algoritmo que foi apresentado. Modificações ocorrem na geração do classificador final, onde leva-se em conta, ao invés do sinal do voto da maioria, a classe com o maior voto, tornando-se necessário apenas computar o voto da maioria para cada classe, conforme a equação 7 descrita na seção 3.2.

Para exemplo citado abaixo, o processo de geração dos classificadores-base foi suprimido para evitar redundância, dado que o algoritmo apresenta o mesmo funcionamento ocorrido na base de dados binária e evidenciada a etapa do cálculo do voto da maioria em uma situação hipotética.

Considerando uma situação hipotética, em que um conjunto de treinamento é obtido de uma base de dados com três classes (A, B e C) e três classificadores foram gerados, tendo sido os erros de treinamento 0,1314, 0,2179 e 0,2581, respectivamente e os pesos calculados em:

$$\alpha_1 = \ln\left(\frac{1-\epsilon_1}{\epsilon_1}\right) = \ln\left(\frac{1-0{,}1314}{0{,}1314}\right) \approx 1{,}8888$$

$$\alpha_2 = \ln\left(\frac{1-\epsilon_2}{\epsilon_2}\right) = \ln\left(\frac{1-0{,}2179}{0{,}2179}\right) \approx 1{,}2779$$

$$\alpha_3 = \ln\left(\frac{1-\epsilon_3}{\epsilon_3}\right) = \ln\left(\frac{1-0{,}2581}{0{,}2581}\right) \approx 1{,}0556$$

Propondo-se que para certa instância $\mathbf{x}$ o primeiro e o terceiro classificador predisseram a classe A, o segundo classificador predisse a classe B e a classe C não foi predita por nenhum deles, os votos para cada classe foram:

$$v_A(\mathbf{x}) = \sum_{t=1}^{K} \alpha_t . I(M_t(\mathbf{x}) = c_j) = 1{,}8888 + 1{,}0556 = 2{,}9444$$

$$v_B(\mathbf{x}) = \sum_{t=1}^{K} \alpha_t . I(M_t(\mathbf{x}) = c_j) = 1{,}2779$$

$$v_C(\mathbf{x}) = \sum_{t=1}^{K} \alpha_t . I(M_t(\mathbf{x}) = c_j) = 0$$

Dado que o voto da maioria é o voto da classe que obteve maior peso:

$$\begin{aligned}\mathbf{M}^{K}(\mathbf{x}) &= \arg\max_{c_j}\{v_j(\mathbf{x}) \mid j=1,..,k\} \\ &= \arg\max\{v_A, v_B, v_C\} \\ &= \arg\max\{2{,}9444,\ 1{,}2779,\ 0\} \\ &= \mathbf{A}\end{aligned}$$

Tem-se a classe A como a classe que foi predita pelo Adaboost.

A modelagem matemática que foi descrita compreendeu o processo de funcionamento do algoritmo, no entanto não abrangeu outras características que fazem parte de todo o sistema da Shell Orion, como a interação do usuário com o Adaboost por meio da ferramenta e as etapas percorridas até a visualização dos dados. Para isso se faz a modelagem do módulo utilizando UML.

4.2.2 Modelagem do módulo utilizando Unified Modeling Language

A Linguagem de Modelagem Unificada, do inglês, Unified Modeling Language (UML), é uma linguagem para modelagem visual baseada no paradigma orientado a objetos. Ela surgiu em 1995, fruto da unificação de várias linguagens gráficas para a orientação a objetos e é utilizada para definir as características do sistema antes do seu desenvolvimento (GUEDES, 2011).

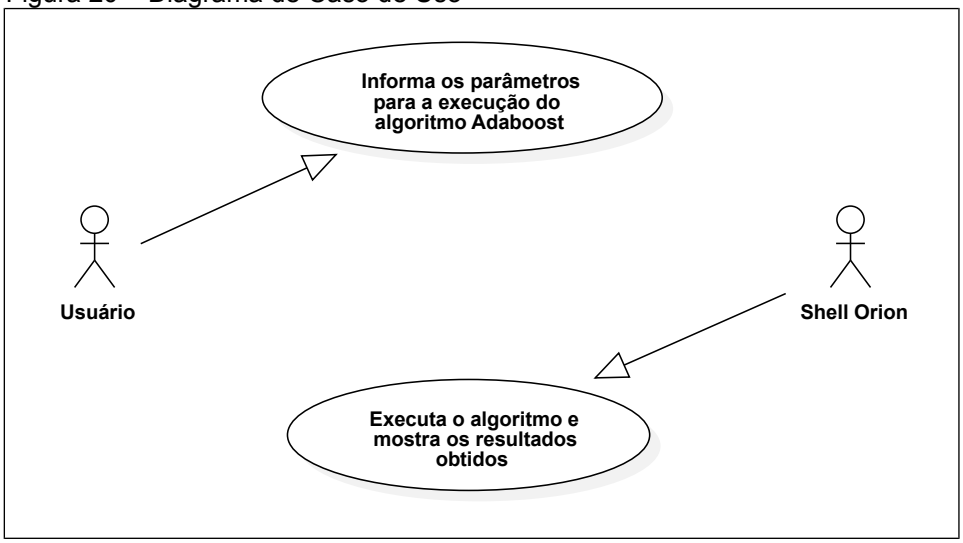
Nesta pesquisa foram desenvolvidos os diagramas de caso de uso, de atividade e de sequência, dadas as suas características, fazendo uso da ferramenta *StarUML*[3].

O diagrama de caso é uma especificação que demonstra a interação existente entre um agente externo e um sistema, sem revelar o funcionamento interno do sistema. Sua representação é feita com atores, simbolizados por bonecos, e suas associações com casos de uso (BEZERRA, 2015).

A figura 20 mostra um diagrama de uso, com o primeiro ator simbolizando o usuário, a interação em que o usuário deve informar os parâmetros para a execução do algoritmo Adaboot e, com o segundo ator, representado pela Shell Orion, mostra a participação da ferramenta com a resposta dos resultados obtidos na execução do algoritmo.

[3] Versão de avaliação, sem limite de tempo, disponível em (http://staruml.io/download)

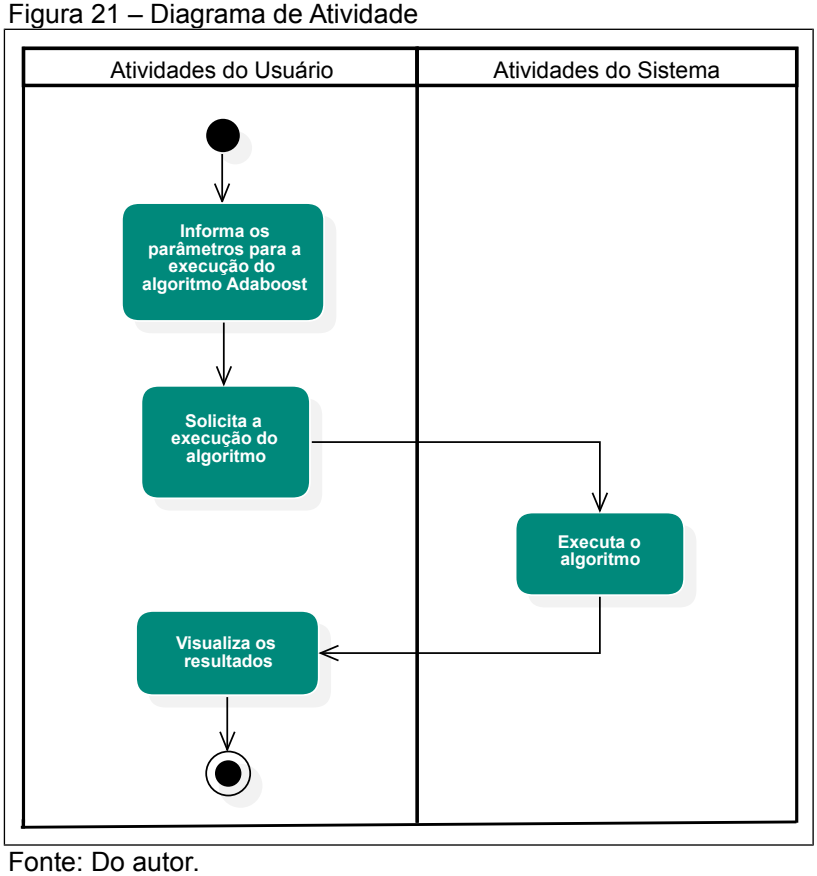
Figura 20 – Diagrama de Caso de Uso



Fonte: Do autor.

O diagrama de atividade representa o fluxo de interação em um sistema em um determinado cenário (figura 21). Ele faz uso de círculos que demonstram o início e o fim do fluxo, retângulos de pontas arredondadas para ações e seta para o fluxo (PRESSMAN, 2011).

Figura 21 – Diagrama de Atividade

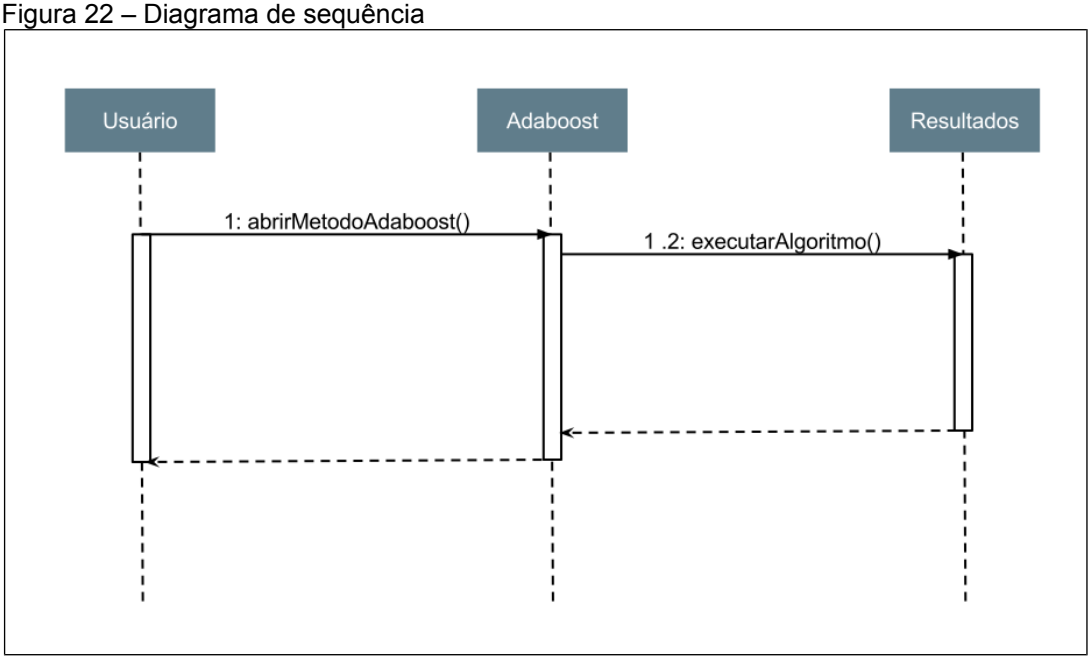


Fonte: Do autor.

O fluxo do diagrama de atividade na figura 21 mostra o processo percorrido quando o usuário informa os parâmetros para a execução do Adaboost e solicita a execução do mesmo, ao passo que o sistema executa o algoritmo de acordo com o que informado e retorna os resultados para a visualização do usuário.

O diagrama de sequência trata da correspondência de mensagens entre objetos, envolvendo as ações do usuário e os eventos em sequência da entrada até a saída do sistema (DENNIS; WIXOM; TEGARDEN, 2015, TRADUÇÃO NOSSA). Na figura 22, é possível ver a ação do usuário, e os eventos consequentes desta ação.

Figura 22 – Diagrama de sequência



Fonte: Do autor.

Considerando a modularidade da Shell Orion Data Mining Engine, o desenvolvimento da modelagem UML serviu como facilitador da compreensão e integração do algoritmo ao sistema.

4.2.3 Implementação do algoritmo AdaBoost no módulo de classificação da Shell Orion Data Mining Engine

A implementação do algoritmo Adaboost foi realizada ambiente de desenvolvimento integrado Netbeans, na versão 8.1. Esta implementação foi

inserida como um módulo ao código fonte já existente da Shell Orion Data Mining Engine.

Dado que a ferramenta e seus módulos anteriores foram desenvolvidos na linguagem de programação Java, da Oracle, optou-se por dar continuidade na mesma linguagem. O kit de desenvolvimento Java, necessário para o desenvolvimento da linguagem, foi utilizado na versão 1.8.0_101.

Uma das características permitidas pela linguagem e disponibilizada pela Shell Orion é a conexão com vários Sistemas Gerenciadores de Bases de Dados (SGBD), necessitando somente que o SGBD possua um *driver* para JDBC. No desenvolvimento do algoritmo Adaboost e realização de testes, o SGBD empregado foi MySQL, que é gratuito e amplamente utilizado (DUBOIS, 2014, tradução nossa).

Durante a implementação do Adaboost foi necessário escolher o algoritmo usado como base para que testes pudessem ser feitos para analisar o funcionamento do algoritmo. Inicialmente, considerou-se o uso de algoritmos já existentes na Shell Orion Data Mining Engine, porém, isto se provou inviável pela necessidade de refatoração dos códigos, o que introduziria *bugs* e despenderia tempo superior ao disponível para a realização desta pesquisa.

Portanto, optou-se pelo uso do algoritmo 1-Rule, também encontrado na literatura sobre o nome de Decision Stump (MARKOSKI et al, 2015, tradução nossa).

4.2.3.1 O algoritmo-base 1-Rule

A versão utilizada do 1-Rule, apresentada por Holte (1993, tradução nossa), é um classificador simples que se utiliza de apenas um atributo para fazer a classificação. Com este atributo ele gera um conjunto de regras, que podem ser compreendidas como uma única regra, o que origina a denominação 1-Rule, do inglês uma regra.

O funcionamento do algoritmo 1-Rule (figura 23) inicia na seleção de todos os valores de cada atributo, com isso é formado um conjunto onde, para cada valor do atributo atual, encontra-se a classe mais frequente e se adiciona ao conjunto de regras que formará a regra do atributo. Após a geração de todas as regras de atributos, faz-se o cálculo da acurácia das regras no mesmo conjunto usado para o treinamento e retorna-se a regra que obteve a maior acurácia. Caso

esta acurácia se repita em mais de uma regra, faz-se a seleção de forma aleatória entre elas (NEVILL-MANNING; HOLMES; WITTEN, 1995, tradução nossa).

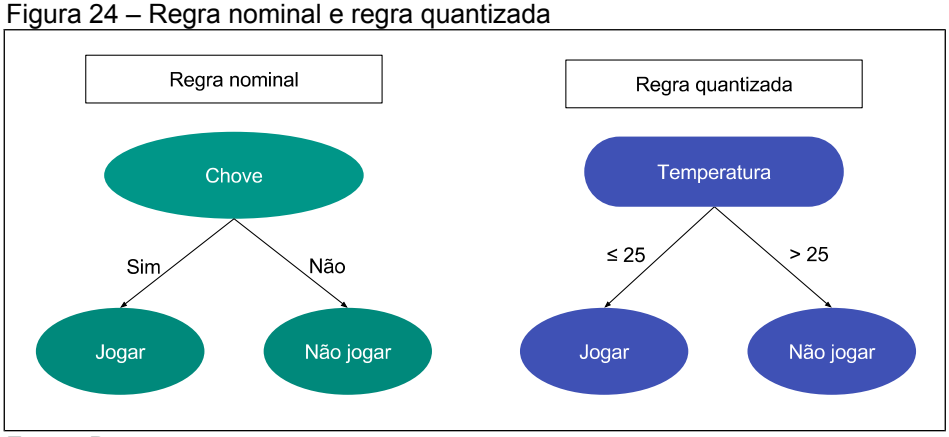
Figura 23 – Pseudocódigo do algoritmo 1-R

```
Para cada atributo A,
    Para cada valor de atributo v pertencente a A
        Selecione o conjunto de instancias i que possuam o valor v
        Encontre a classe mais frequente c neste conjunto
        Adicione a condição "se atributo A possui valor v, então a
            classe é c" para a regra de A
Retorne a regra com a maior acurácia.
```

Fonte: Adaptado de Holmes e Nevill-Manning (1995, tradução nossa).

No caso de atributos com valores numéricos, faz-se um preprocessamento antes da execução do algoritmo, chamado de quantização (figura 24), que converte os valores numéricos em nominais. Isso se faz necessário, pois, sem este processo o 1-Rule gera uma regra para cada valor de atributo e consequentemente faz o algoritmo causar *overfitting* no classificador gerado. A quantização do 1-Rule, primeiramente divide os valores do atributo, já ordenados de forma crescente, em pequenos grupos de acordo com a classe dos vizinhos do conjunto e em seguida, elimina as divisões entre os grupos que não atendam o valor mínimo da classe majoritária, que normalmente é definido em seis. Após esta etapa, atribui-se aos valores da distribuição original o grupo no qual ele faz parte. Finalizada a quantização, pode-se aplicar o algoritmo normalmente (WITTEN; FRANK; HALL, 2011, tradução nossa).

Figura 24 – Regra nominal e regra quantizada



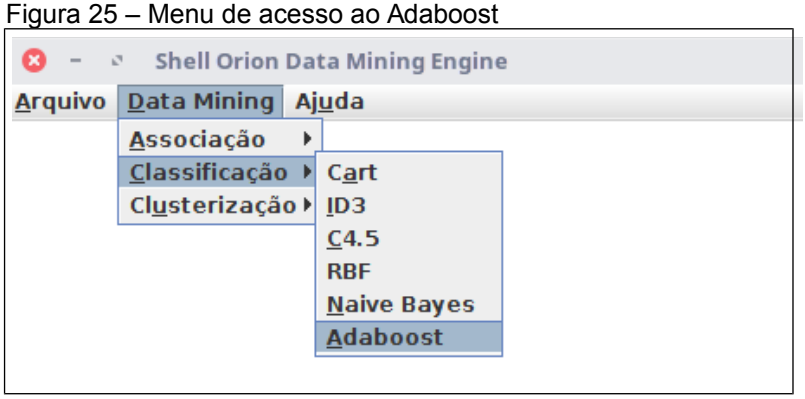
Fonte: Do autor.

O algoritmo 1-Rule e suas variantes são comumente utilizados como algoritmo-base do Adaboost, inclusive no artigo *Experiments with a New Boosting Algorithm*, em que os autores originais do Adaboost fizeram experimentos utilizando o 1-Rule como algoritmo-base do mesmo (FREUND; SCHAPIRE, 1996, tradução nossa). Obras mais recentes, como Schapire e Freund (2012, tradução nossa) e Markoski et al (2015, tradução nossa) também referenciam o algoritmo como um possível algoritmo-base do Adaboost.

Ao implementar o 1-Rule e Adaboost utilizou-se da estratégia de interfaces em Java, sendo possível que, em trabalhos futuros, novos algoritmos sejam utilizados como base, desde que atendam ao contrato da interface *Classificador* no código desenvolvido, interface que contém apenas três métodos: *treinar*, *predizer* e *testar*.

O módulo do Adaboost, foi inserido acessando o menu encontrado em *Data Mining* > *Classificação* > *Adaboost* (figura 25).

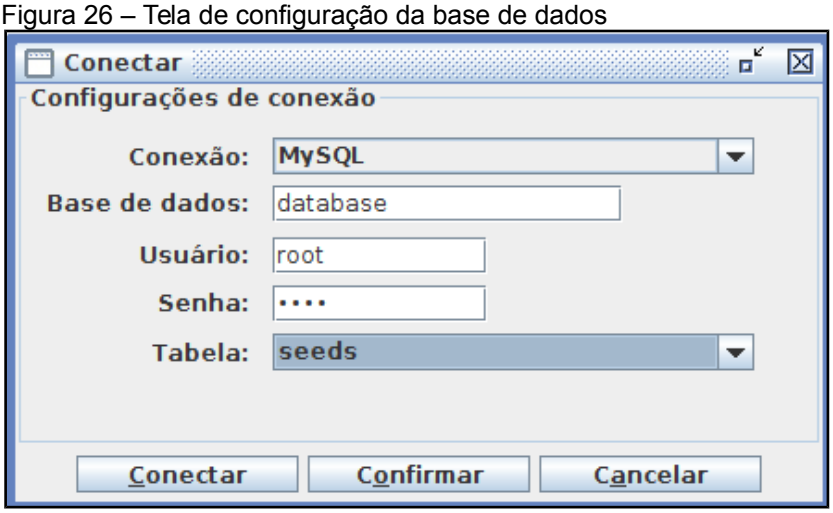
Figura 25 – Menu de acesso ao Adaboost



Fonte: Do autor.

Para que o usuário tenha acesso à tela do Adaboost, algumas configurações precisam ser informadas previamente. As primeiras configurações que precisam ser informadas se encontram na tela de configuração da base de dados (figura 26), que é aberta ao selecionar o menu *Arquivo* > *Conectar*.

Figura 26 – Tela de configuração da base de dados



Fonte: Do autor.

As configurações que podem ser informadas nesta tela são:

a) **conexão:** tipo de SGBD que será conectado;

b) **base de dados:** base de dados do SGBD, quando for necessário trabalhar com várias tabelas de bases de dados.

c) **usuário:** usuário de acesso ao SGBD;

d) **senha:** senha de acesso ao SGBD;

e) **tabela*:*** tabela que será utilizada pelo algoritmo;

Ao clicar no botão conectar, os dados que foram informados são testados e ao confirmar em salvar o usuário retornará para a tela principal da Shell Orion (figura 25), onde é possível acessar a tela do Adaboost (figura 27), que também possui os seguintes parâmetros a serem informados:

a) ***holdout* (%):** divisão da base de dados em porcentagem para treinamento e testes. Exemplo: *holdoult* em 66% dividirá a base em duas, uma de treinamento com 66% do tamanho original e outra de testes com os 33% restantes;

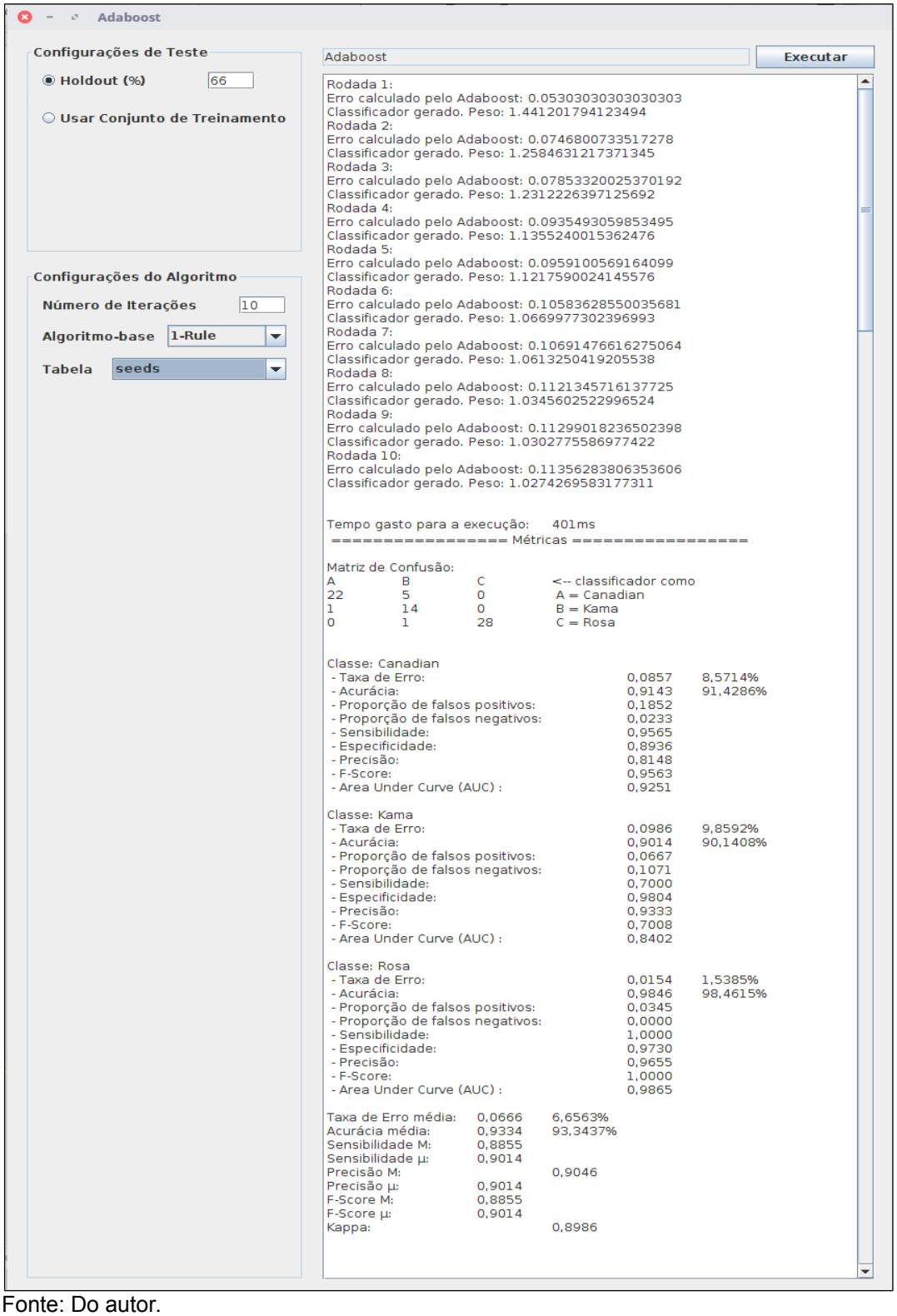
b) **usar Conjunto de Treinamento:** a base de dados será utilizada de forma integral para treinamento e para testes, sem haver uma divisão;

c) **número de Iterações:** número máximo de iterações que o Adaboost executará. Também representa o número máximo de classificadores gerados. Caso o Adaboost considerae que não são necessárias iterações ele poderá encerrar antes do número definido;

d) **Algoritmo-base:** representa o algoritmo-base que será utilizado para gerar os classificadores do conjunto, sendo disponibilizado o 1-Rule;

e) **tabela:** tabela que será utilizada para o treinamento e avaliação do Adaboost. Cada tabela do SGBD representa uma tabela da base de dados.

Figura 27 – Tela algoritmo Adaboost



Fonte: Do autor.

Os valores informados têm influência sobre os resultados do Adaboost e, quando alterados, podem geram resultados diferentes em uma mesma tabela do SGBD.

A opção “Usar o Conjunto de Treinamento” pode melhorar os resultados das métricas do algoritmo, no entanto, causa *overfitting*, que é o ajuste excessivo do algoritmo ao conjunto de treinamento, além das métricas serem consideradas inválidas para a avaliação do algoritmo (AGGARWAL, 2015, tradução nossa).

A solução para o problema da opção anterior surge com a configuração *holdout,* que por padrão está definida em 66%, ou seja dois terços da base dados empregada para treinamento e um terço para testes. Conforme Kantardzic (2011, tradução nossa) estes valores são considerados ideais para se obter resultados mais precisos (KANTARDZIC, 2011, tradução nossa). No entanto, caso o usuário deseje pode ser alterado.

Ao pressionar o botão *Executar*, a Shell Orion iniciará o processo de treinamento e, em seguida, o processo de avaliação do Adaboost na base de dados selecionada. Os resultados são apresentados no campo de texto principal, demonstrando as iterações executadas, o erro de treinamento calculado pelo Adaboost para cada classificador gerado de forma sequencial e também o peso deste classificador.

Com o fim do treinamento, é exibido o tempo gasto para o treinamento, seguido das medidas de qualidade resultantes da classificação pelo Adaboost. Encerrado todo o processo, é informado o tempo consumido na avaliação.

4.3 RESULTADOS OBTIDOS

Após a implementação do algoritmo, resultados obtidos compreendem a análise das medidas de qualidade empregadas no algoritmo e a análise do desempenho pela avaliação da acurácia do classificador gerado em uma base de dados binária e outra de múltiplas classes. Também foi realizada a análise do tempo gasto para a execução do Adaboost e a comparação dos tempos de processamento obtidos na Shell Orion com a ferramenta *Weka*.

Todos os testes foram realizados em um computador com Sistema Operacional Ubuntu 16.04 LTS 64bits, 8 GB de memória RAM e processador Intel Core i5 com 4 núcleos de 1.70GHz cada.

4.3.1 Identificação binária das classes pelo Adaboost

A primeira base de dados, preprocessada, é formada 203 instâncias usadas para identificar pacientes que possuem doença renal crônica ou não (atributo *class*). Na base preprocessada 79 possuíam doença renal crônica e 124 não possuíam.

Para o primeiro teste realizado foram informados os parâmetros *holdout* em 66% e executado com o número de iterações em 10, a fim de avaliar a taxa de acertos do Adaboost na base de dados aplicada.

O número de iterações foi definido em 10, pois, não necessariamente um maior número de rodadas aumentará a capacidade de previsão do classificador, dado que, conforme Zhou (2012, tradução nossa) informar um número muito grande de rodadas tende a gerar *overfitting* no Adaboost e complexidade desnecessária e com um número muito pequeno não traria os benefícios obtidos pelo aprendizado sucessivo que ocorre nas rodadas.

Tabela 5 – Execução do Adaboost com 10 iterações

Classe	Quantidade de instâncias	Porcentagem	Classe	Acertos (%)
ckd	23	33,333%	ckd (22 ocorrências) notckd (1 ocorrência)	95,6522%
notckd	46	66,666%	notckd (46 ocorrências) ckd (0 ocorrências)	100%

Fonte: Do autor.

Na execução (tabela 5), o algoritmo conseguiu acertar 22 em 23 as ocorrências de doença renal crônica e conseguiu acertar todos os 46 casos de não ocorrência.

Para aplicar as medidas de qualidade do classificador utilizou-se o mesmo classificador gerado no teste anterior. As medidas Proporção de Falso Positivos, Proporção de Falso Negativos, sensibilidade, especificidade, precisão e F-

score foram aplicadas para cada atributo de classe (*ckd, notckd*), já as medidas taxa de erro, acurácia, *Area Under Curve* (AUC) e estatística Kappa foram aplicadas somente uma vez na base de dados binária, pois seus cálculos levam em conta os resultados gerais e não variam de acordo com o atributo de classe.

Primeiramente foi montada a matriz de confusão para o atributo de classe *ckd*, que representa a presença de doença renal crônica (figura 28).

Figura 28 – Matriz de Confusão gerada para *ckd* de *Chronic Kidney Disease*

	Classe Verdadeira	
Classe Predita	*ckd*	*notckd*
ckd	22	1
notckd	0	46

Fonte: Do autor.

Com a matriz de confusão gerada é possível extrair dela as proporções para a classe *ckd:* verdadeiro positivos (22); verdadeiro negativos (46); falso positivos (1); falso negativos (0).

Dadas as proporções de VP, VN, FP e FN pode-se calcular as seguintes métricas para a classe *ckd:*

a) **Proporção de Falso Positivos (PFP)** $= \frac{FP}{FP+VP} = \frac{1}{1+22} \approx 0{,}0435$;

b) **Proporção de Falso Negativos (PFN)** $= \frac{FN}{FN+VN} = \frac{0}{0+46} = 0$;

c) **sensibilidade** $= \frac{VP}{VP+FN} = \frac{22}{22+0} = 1{,}0000$;

d) **especificidade** $= \frac{VN}{FP+VN} = \frac{46}{1+46} \approx 0{,}9787$;

e) **precisão** $= \frac{VP}{VP+FP} = \frac{22}{22+1} \approx 0{,}9565$;

h) **F-score** $= \frac{(\beta^2+1).VP}{(\beta^2+1).VP+\beta^2.FN+FP} = \frac{(23^2+1).22}{(23^2+1).22+23^2.0+1} \approx 0{,}9999$.

Na tabela 6, tem-se um resumo das medidas de qualidade obtidas para a classe *ckd*, que consiste na presença de doença renal crônica.

Tabela 6 – Resumo das medidas de qualidade para *ckd*

Métrica	Valor
Proporção de Falso Positivos	0,0435
Proporção de Falso Negativos	0,0000
Sensibilidade	1,0000
Especificidade	0,9787
Precisão	0,9565
F-Score	0,9999

Fonte: Do autor.

A Proporção de Falso Positivos, avaliada em 0,0435, que é referente a classe *ckd* (presença de doença renal crônica), mostra um pequeno erro ao considerar a classe correta e a Proporção de Falso Negativos evidencia que classificador não considerou nenhum caso da classe *notckd* como *ckd*, sendo perfeito na identificação.

A sensibilidade, que informa que a porcentagem de Verdadeiro Positivos que realmente são positivos, alcança o valor 1, que é o valor máximo possível. De forma semelhante, a especificidade informa um valor de 0,9787 para a porcentagem de Verdadeiro Negativos que realmente são negativos.

O F-Score, que pode ser considerado a média ponderada entre a sensibilidade e a especificidade (BOROVICKA et al, 2012, tradução nossa), obteve 0,9999 de 1 possíveis.

A fim de avaliar estas métricas para classe *notckd*, a mesma matriz gerada para a classe *ckd* foi atualizada, considerando que os valores VP, VN, FP, FN, são os valores de VN, VP, FN, FP da matriz anterior invertidos, como mostra a figura 29:

Figura 29 – Matriz de Confusão gerada para *notckd* de *Chronic Kidney Disease*

	Classe Verdadeira	
Classe Predita	*notckd*	*ckd*
notckd	46	0
ckd	1	22

Fonte: Do autor.

As medidas precisaram ser recalculadas, gerando para a classe *notckd* as proporções: verdadeiro positivos (46); verdadeiro negativos (22); falso positivos (0); falso negativos (0). Em seguida aplicou-se estas proporções no cálculo das medidas que variam em cada atributo de classe:

a) **Proporção de Falso Positivos** $= \frac{FP}{FP+VP} = \frac{0}{0+46} = 0{,}0000$;

b) **Proporção de Falso Negativos** $= \frac{FN}{FN+VN} = \frac{1}{1+22} \approx 0{,}0435$;

c) **sensibilidade** $= \frac{VP}{VP+FN} = \frac{46}{46+1} \approx 0{,}9787$;

d) **especificidade** $= \frac{VN}{FP+VN} = \frac{22}{0+22} = 1{,}0000$;

e) **precisão** $= \frac{VP}{VP+FP} = \frac{46}{46+0} = 1{,}0000$;

h) **F-score** $= \frac{(\beta^2+1).VP}{(\beta^2+1).VP+\beta^2.FN+FP} = \frac{(46^2+1).46}{(46^2+1).46+46^2.1+0} \approx 0.9787$.

As métricas calculadas também foram reunidas na tabela 7 com o propósito de facilitar a visualização dos resultados.

Tabela 7 – Resumo das medidas de qualidade para *notckd*

Métrica	Valor
Proporção de Falso Positivos	0,0000
Proporção de Falso Negativos	0,0435
Sensibilidade	0,9787
Especificidade	1,0000
Precisão	1,0000
F-Score	0,9787

Fonte: Do autor.

Os valores de PFP, em 0, e PFN próximo a zero revelam uma proporção mínima de identificação incorreta das classes. A sensibilidade e o F-score aproximaram-se do valor máximo de 1, valor esse que foi obtido pela especificidade e a precisão, indicando uma boa capacidade de predição na classe.

Em seguida, foram calculadas as medidas de qualidade que levam em conta o desempenho do classificador independentemente do atributo analisado:

a) **taxa de erro** $= \frac{FN+FP}{VN+FN+FP+VP} = \frac{0+1}{46+0+1+22} \approx 0{,}014493 = 1{,}4493\%$;

b) **acurácia** $= 1 - \text{taxa de erro} \approx 98{,}5507\%$;

c) ***Area Under Curve*** **(AUC)**

$$= \frac{1}{2}\left(\frac{VP}{VP+FN} + \frac{VN}{VN+FP}\right) = \frac{1}{2}\left(\frac{22}{22+0} + \frac{46}{46+1}\right) \approx 0{,}9894;$$

d) **Estatística Kappa**

$$= \frac{\left(\frac{correto}{total}\right) - \left(\frac{concordância}{total^2}\right)}{1 - \left(\frac{concordância}{total^2}\right)} = \frac{\left(\frac{68}{69}\right) - \left(\frac{138}{69^2}\right)}{1 - \left(\frac{138}{69^2}\right)} \approx 0{,}9851.$$

Os resultados, resumidos na tabela 8, denotam uma acurácia próxima de 100%, o que informa uma ótima capacidade de generalização para o conjunto de testes e consequentemente um erro mínimo. A *Area Under Curve*, que é a capacidade do classificador de evitar a falsa classificação foi calculada em 0,9894 e a estatística *Kappa,* obteve uma concordância entre o que foi predito e observado quase perfeita (0,9851).

Tabela 8 – Resumo das medidas gerais para *Chronic Kidney Disease*

Métrica	Valor
Taxa de erro	1,4493%
Acurácia	98,5507%
Area Under Curve (AUC)	0,9894
Estatística Kappa	0,9851

Fonte: Do autor.

Estes resultados obtidos de *Chronic Kidney Disease* foram comparados com os resultados de pesquisas compiladas no estudo *Review on Prediction of Chronic Kidney Disease Using Data Mining Techniques*, de Patil (2016, tradução nossa). Todas as pesquisas presentes aplicaram a mesma base *Chronic Kidney Disease* com os mais diferentes algoritmos e suas acurácias foram apresentadas na tabela 9.

Tabela 9 – Resultado de pesquisas na base *Chronic Kidney Disease*

Autor	Ferramenta	Algoritmo	Acurácia
Baby e Vital (2015, tradução nossa)	*Weka* e Orange	SVM	73%
		KNN	78%
Kumar (2016, tradução nossa)	*Weka*	Random Forests	100%
		SMO	97%
		Naïve Bayes	95%
		RBF	98%
		MLPC	98%
		SLG	98%
Lambordar e Narendra (2015, tradução nossa)	*Weka*	Naïve Bayes	95%
		MLP	99,75%
		SVM	62%
		J48	99%
		Conjunction Rule	94,75%
		Decision Table	99%
Ramya e Radha (2016, tradução nossa)	*R Tool* e *Weka*	BPN	80%
		RBF	85,3%
		Random Forests	78,6%
Sinha e Sinha (2015, tradução nossa)	*Weka* e *Orange*	SVM	73%
		KNN	78%
Vijayarani e Dhayanand (2015, tradução nossa)	*MATLAB*	Naïve Bayes	70,96%
		SVM	76,32%
Nesta pesquisa (Souza, 2016)	Shell Orion	Adaboost	98,55%

Fonte: Adaptado de Patil (2016, tradução nossa).

Ao analisar a acurácia obtida nas outras pesquisas se pode observar que a acurácia do Adaboost (98,55%) mostrou-se superior a 17 dos 21 algoritmos observados, tendo sido inferior somente aos algoritmos *Random Forest* (100%), *MLP* (99,75%) e *Decision Table* (99%) e J48 (99%). Isto demonstra que o classificador gerado possui qualidade e que funciona corretamente para a predição em bases de dados binárias.

4.3.2 Identificação de múltiplas classes pelo Adaboost

Considerando a capacidade do algoritmo Adaboost de lidar com bases de dados que possuam múltiplas classes, foram feitos testes em uma base de dados com esta característica. A base de dados *Seeds* conta com 210 instâncias e 8 atributos, incluindo com o atributo de classe e diferencia sementes de trigo em três variedades: *Canadian*, *Kama* e *Rosa*.

Os testes aplicados a esta base de dados foram semelhantes aos aplicados na base de dados binária. O parâmetro *holdout* foi definido em 66%, gerando 71 instâncias para testes e 139 para treinamento, e o número de iterações foi informado em 10.

Os valores obtidos foram inseridos em uma matriz de confusão para múltiplas classes (figura 30) e para cada valor de classe os índices foram obtidos, para que, posteriormente, fosse possível fazer o cálculo das métricas para múltiplas classes.

Figura 30 – Matriz de confusão de *Seeds*

		Classe Verdadeira		
		Canadian	*Kama*	*Rosa*
Classe Predita	*Canadian*	22	5	0
	Kama	1	14	0
	Rosa	0	1	28

Fonte: Do autor.

As medidas obtidas para cada classes foram disponibilizadas na tabela 10, a fim de demonstrar o desempenho de cada uma com relação as outras.

Tabela 10 – Métricas por classe do Adaboost na base de dados *Seeeds*

Classe	VP	VN	FP	FN	Acurácia (%)	PFP	PFN	Sensi-bilidade	Especifi cida-de	Preci-são	F-Score	AUC
Canadian	22	42	5	1	91,4286%	0,1852	0,0233	0,9565	0,8936	0,8148	0,9563	0,9251
Kama	14	50	1	6	90,1408%	0,0667	0,1071	0,7000	0,9804	0,9333	0,7008	0,8402
Rosa	28	36	1	0	98,4615%	0,0345	0,0000	1,0000	0,9730	0,9655	1,0000	0,9865

Fonte: Do autor.

A acurácia de todas as três classes foram superiores a 90%, tendo as sementes *Canadian* obtido 91,4286%, *Kama,* 90,1408%, e *Rosa* 98,4615%. De forma individual, *Rosa* foi a semente mais bem avaliada nas medidas de qualidade.

A partir destes valores individuais, pode-se calcular as medidas de qualidade para o classificador geral, a fim de avaliar a sua qualidade:

a) **taxa de erro média** $= \frac{\sum_{i=1}^{l} \frac{FP_i + FN_i}{VP_i + FN_i + FP_i + VN_i}}{l} \approx \frac{0,1997}{3} \approx 6,6563\%$;

b) **acurácia média** $= \frac{\sum_{i=1}^{l} \frac{VP_i + VN_i}{VP_i + FN_i + FP_i + VN_i}}{l} \approx \frac{2,8003}{3} \approx 93,3447\%$;

c) **sensibilidade** $M = \frac{\sum_{i=1}^{l} \frac{VP_i}{VP_i + FN_i}}{l} \approx \frac{2,6565}{3} \approx 0,8855$;

d) **sensibilidade** $\mu = \frac{\sum_{i=1}^{l} VP_i}{\sum_{i=1}^{l} (VP_i + FN_i)} = \frac{64}{71} \approx 0,9014$;

e) **precisão** $M = \frac{\sum_{i=1}^{l} \frac{VP_i}{VP_i + FN_i}}{l} = \frac{2,7137}{3} \approx 0,9046$;

f) **precisão** $\mu = \frac{\sum_{i=1}^{l} VP_i}{\sum_{i=1}^{l} (VP_i + FP_i)} = \frac{64}{71} \approx 0,9014$;

g) **F-score** $M = \frac{(\beta^2 + 1) Precisão_M . Sensibilidade_M}{\beta^2 . Precisão_M + Sensibilidade_M}$

$\approx \frac{(71^2 + 1).0,9046 .0,8855}{71^2.0,9046 + 0,8855} \approx 0,8855$;

h) **F-score** $\mu = \frac{(\beta^2 + 1) Precisão_\mu . Sensibilidade_\mu}{\beta^2 . Precisão_\mu + Sensibilidade_\mu}$

$\approx \frac{(71^2 + 1).0,9014 .0,9014}{71^2.0,9014 + 0,9014} \approx 0,9014$;

i) **Estatística Kappa** $= \dfrac{\left(\dfrac{correto}{total}\right) - \left(\dfrac{concordância}{total^2}\right)}{1 - \left(\dfrac{concordância}{total^2}\right)} = \dfrac{\left(\dfrac{64}{71}\right) - \left(\dfrac{142}{71^2}\right)}{1 - \left(\dfrac{142}{71^2}\right)} \approx 0{,}8986$.

De forma semelhante ao que ocorreu com os testes com a base de dados binária, as métricas calculadas e demonstradas para múltiplas classes foram reunidas na tabela 11 para facilitar a visualização dos dados.

Tabela 11 – Resumo das métricas de validação para múltiplas classes

Métrica	Valor
Taxa de erro Média	6,6563%
Acurácia Média	93,3447%
Sensibilidade M	0,8855
Sensibilidade μ	0,9014
Precisão M	0,9046
Precisão μ	0,9014
F-Score M	0,8855
F-Score μ	0,9014
Estatística Kappa	0,8986

Fonte: Do autor.

Tanto os valores micro (μ), que são as médias geradas dos valores obtidos por classe, quanto os valores macro (M), que são médias calculadas de valores gerais, foram satisfatórios na base S*eeds*, sendo todos os valores superiores a 0,88 e considerando o maior valor alcançável o valor 1. As acurácias por classe de 91,4286% para *Canadian*, 90,1408% para *Kama* e 98,4615% para *Rosa*, além a acurácia média de 93,3447%, demostram a eficiência do Adaboost na predição da base de dados para múltiplas classes.

O Adaboost disponibilizado na Shell Orion foi comparado com vinte algoritmos de quatro pesquisas pela análise da acurácia dos mesmos. Os dados das pesquisas foram extraídos e organizados em ordem alfabética na tabela 12.

Tabela 12 – Resultado de pesquisas na base *Seeds*

Autor	Ferramenta	Algoritmo	Acurácia
Lalis (2015, tradução nossa)	Não utilizou	XAC	89,27%
Sabanci e Akkaya (2016, tradução nossa)	*Weka*	K-NN	95,71%
		Multilayer Perceptron	97,14%
		J48	91,90%
		NaïveBayes	91,43%
Sujatha (2013, tradução nossa)	*Weka*	ZeroR	33,33%
		OneR	83,81%
		DecisionTable	87,14%
		NaïveBayes	91,43%
		PART	92,86%
		SMO	93,81%
		J48	91,91%
		RandomTree	91,91%
Wei et al (2014, tradução nossa)	Não utilizou	ECBMDC	92,82%
		SubBag-SVM	92,90%
		SubBag-KNN	92,26%
		Random Forest	92,22%
		SVM	93,15%
		K-NN	92,22%
		NaïveBayes	89,26%
Nesta pesquisa (Souza, 2016)	Shell Orion	Adaboost	93,34%

Fonte: Do autor.

Entre os algoritmos analisados, apenas Multilayer Perceptron (97,14%), K-NN (95,71%) e SMO (93,81%) tiveram acurácia superior ao classificador gerado pelo Adaboost (93,34%), o qual foi superior aos dezessete outros algoritmos e evidencia sua capacidade de predizer corretamente as classes da base de dados *Seeds*.

Em suma, a aplicação do Adaboost na base de dados *Chronic Kidney Disease* revelou o correto funcionamento do algoritmo em bases de dados binárias, tendo sido eficaz na predição correta das classes, alcançando 98,5507% de acurácia. Quando aplicado na base de dados com múltiplas classes *Seeds*, o Adaboost revelou ser capaz de identificar as três classes e obteve 93,3447% de acurácia. Isto demonstra que o Adaboost é capaz de identificar as classes tanto em bases de dados binárias quanto em bases de dados *multiclasses*.

4.3.3 Tempos de Processamento do Algoritmo Adaboost

A avaliação do tempo de processamento do Adaboost, foi feita pela análise do tempo de treinamento do algoritmo na Shell Orion Data Mining Engine e a análise do tempo de treinamento na ferramenta *Weka*[4].

Weka é uma ferramenta de *data mining* desenvolvida na linguagem de programação Java (o que permite que a ferramenta seja multiplataforma) e é frequentemente utilizada em pesquisas da área. Sua distribuição é feita sob a licença *General Public License*[5] (GNU) e sua manutenção é feita pela Universidade de *Wakaito,* na Nova Zelândia (WITTEN; FRANK; HALL, 2011, tradução nossa).

A análise na ferramenta Weka levou em conta a adequação dos parâmetros de forma que eles fossem os mesmos que foram aplicados na Shell Orion, e considerou-se para a etapa de avaliação, diferentes quantidades de instâncias, que foram replicadas de uma base de dados com 470 instâncias. A estratégia de replicação dos dados foi feita pelo crescimento geométrico[6], que é definido pela equação 29.

$$C_k = N.2^{k-1} \tag{29}$$

Onde:

a) C_k : é o número de instâncias que serão analisadas.

b) N : o tamanho original da base de dados.

c) k : a quantidade de iterações.

A base de dados foi replicada oito vezes (gerando 9 iterações), alcançando o número de 120320 instâncias para a análise dos tempos de processamento nas duas ferramentas. Os resultados obtidos no crescimento geométrico foram apresentados na tabela 13, iniciando-se as iterações na base de dados original.

---

[4] *Weka* é disponibilizada gratuitamente em (http://www.cs.waikato.ac.nz/ml/weka/)

[5] General Public License. Informações em (http://www.gnu.org)

[6] Uma sequência de termos, chamada de progressão geométrica ou sequência geométrica, gerada a partir do termo inicial *A*, com os termos conseguintes gerados pela multiplicação do termo anterior por um valor constante *R*, denominado razão (GERSTING, 1995).

Tabela 13 – Definição do tamanho das cargas de dados

Iteração	Quantidade de instâncias	Quantidade de atributos
1	470	3
2	940	3
3	1880	3
4	3760	3
5	7520	3
6	15040	3
7	30080	3
8	60160	3
9	120320	3

Fonte: Do autor.

O número de atributos foi definido em três, pois o intuito dos testes era apenas avaliar o tempo de execução, tendo os resultados da classificação sido desconsiderados. A tabela 14 mostra o tempo de processamento gasto em cada carga de instâncias na Shell Orion.

Tabela 14 – Tempo de processamento do Adaboost na Shell Orion

Quantidade de instâncias	Shell Orion
470	00 m: 00 s. 137 ms
940	00 m: 00 s. 242 ms
1880	00 m: 00 s. 382 ms
3770	00 m: 00 s. 876 ms
7520	00 m: 02 s. 239 ms
15040	00 m: 07 s. 145 ms
30080	00 m: 30 s. 288 ms
60160	02 m: 58 s. 325 ms
120320	15 m: 57 s. 353 ms

Fonte: Do autor.

Em seguida foi feito o cálculo do tempo de processamento na *Weka* com os mesmos dados em crescimento geométrico, exibidos conforme o número de instâncias na tabela 15, já com os resultados da Shell Orion inseridos para uma melhor visualização.

Tabela 15 – Tempo de processamento do Adaboost na *Weka*

Quantidade de instâncias	Shell Orion	*Weka*
470	00 m: 00 s. 137 ms	00 m: 00 s. 020 ms
940	00 m: 00 s. 242 ms	00 m: 00 s. 030 ms
1880	00 m: 00 s. 382 ms	00 m: 00 s. 040 ms
3770	00 m: 00 s. 876 ms	00 m: 00 s. 080 ms
7520	00 m: 02 s. 239 ms	00 m: 00 s. 140 ms
15040	00 m: 07 s. 145 ms	00 m: 00 s. 140 ms
30080	00 m: 30 s. 288 ms	00 m: 00 s. 310 ms
60160	02 m: 58 s. 325 ms	00 m: 00 s. 570 ms
120320	15 m: 57 s. 353 ms	00 m: 01 s. 353 ms

Fonte: Do autor.

Foi possível notar a diferença existente entre os tempos de processamento nas duas ferramentas, tendo a *Weka* sido mais rápida principalmente quando aplicado quantidades maiores de instâncias (de 15 minutos e 56 segundos no maior caso), demonstrando a influência gerada pelo crescimento geométrico na Shell Orion. Porém, mesmo com a Shell Orion sendo mais lenta, as duas ferramentas distribuem os tempos normalmente a medida que a carga de dados aumenta.

Para provar a hipótese de que os tempos distribuem normalmente, aplicou-se o teste de *Shapiro-Wilk* com o auxílio da ferramenta IBM SPSS[7] na versão 24. Este teste, elaborado por Samuel Shapiro e Martin Wilk em 1965, utiliza-se do critério obtido do cálculo de uma estatística *W* para verificar se uma distribuição é significantemente diferente de uma distribuição normal. O valor resultante que for superior ao valor *p* do critério, definido em $p > 0,05$, significa que os tempos se distribuem de forma *gaussiana* e estão dentro de uma distribuição normal, caso contrário isso indica uma distribuição fora da normalidade (FIELD, 2009, tradução nossa).

[7] *Statistical Package for the Social Sciences* (SPSS) é uma ferramenta que conta com diversos cálculos estatísticos e de análise exploratória dos dados. Disponível em versão para avaliação em (https://www-01.ibm.com/software/br/analytics/spss/)

Tabela 16 – Teste de Shapiro-Wilk nos tempos de processamento da Shell Orion e da Weka

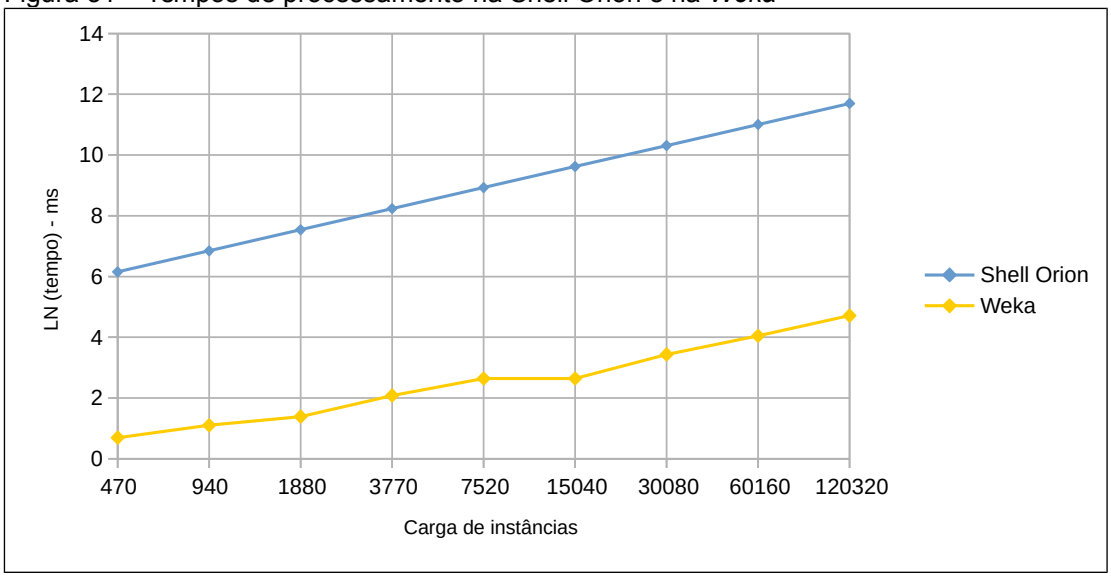
Ferramenta	Estatística	Grau de Liberdade	Significância
Shell Orion	0,721	9	0,020
Weka	0,738	9	0,040

Fonte: Do autor.

Os dados apresentados na tabela 16 informam que os dois testes de ambas as ferramentas não atenderam ao valor *p* do critério, dado que os valores de significância (*p*), 0,02 obtidos pela Shell Orion e 0,04 obtidos pela *Weka*, são inferiores a 0,05 e, portanto, indicam que as duas distribuições estão fora da normalidade.

Considerando estes resultados, utilizou-se da transformação dos tempos de processamento obtidos pelo logaritmo natural, em que, por exemplo, um tempo de 137ms é calculado por ln(137) = 4,919980926. Os valores resultantes foram mostrados graficamente na figura 31.

Figura 31 – Tempos de processamento na Shell Orion e na *Weka*



Fonte: Do autor.

Ao fim dos cálculos, aplicou-se novamente o teste de *Shapiro-Wilk*, onde se obteve para ambos os casos uma distribuição *gaussiana*. Os resultados (tabela 17) demonstram que, tanto na ferramenta Shell Orion, quanto na ferramenta *Weka*, foram obtidas distribuições dentro da normalidade ao atender o critério de $p > 0,05$, com 0,914 para a Shell Orion e 0,835 para a *Weka*.

Tabela 17 – Teste de Shapiro-Wilk após a transformação logarítmica dos tempos

Ferramenta	Estatística	Grau de Liberdade	Significância
Shell Orion	0,972	9	0,914
Weka	0,964	9	0,835

Fonte: Do autor.

Em seguida, com o teste F de Levene, proposto em 1960 por Levene, buscou-se avaliar a homogeneidade das variâncias obtidas. Este teste resultou em *p* = 0,270, sendo possível a hipótese nula e a homogeneidade existente entre as variâncias.

Dada a possibilidade da hipótese nula, aplicou-se o teste *t* de *Student* com o objetivo de comparar as médias obtidas do logaritmo natural dos tempos (figura 32).

Figura 32 – Test *t* de Student

Teste *t* de Student

	N	Média	Desvio Padrão	Erro Padrão da Média	Significância (*p*)
Shell Orion	9	8,9256	1,8981	0,6327	< 0,0001
Weka	9	2,5247	1,3608	0,4536	

Fonte: Do autor.

O valor *p* definido em *p* < 0,001, inferior a *p* = 0,05, demostra que as diferenças entre as médias foi significativa e que a hipótese nula é inválida, não havendo igualdade entre as médias do logaritmo natural dos tempos da Shell Orion e da *Weka.* Portanto, em média, a Shell Orion apresenta um tempo maior de processamento quando comparada com a *Weka*

Estes testes de tempo de processamento tiveram a finalidade de avaliar o tempo consumido pelos classificadores gerados pelas ferramentas em diferentes quantidades de instâncias.

5 CONCLUSÃO

O *data mining* surge como ferramenta fundamental para o processamento de dados e extração de conhecimento das bases de dados que se multiplicam e avolumam-se constantemente.

A tarefa de classificação mostra-se aplicável nas mais variadas áreas e presente em diversos estudos. Esta pesquisa desenvolvida se baseou na estratégia de metaclassficação, por meio do algoritmo Adaboost, para gerar classificadores com alta capacidade de predição com o auxílio de algoritmos simples, como o 1-Rule, para a identificação das classes, tantos em problemas binários quanto problemas com várias classes.

As dificuldades encontradas, apresentaram-se na escolha e implementação do algoritmo-base, onde optou-se primariamente pelo uso dos algoritmos já existentes na Shell Orion, o que se provou inviável pela falta de padronização dos projetos, além do tempo reduzido e a possibilidade de introduzir *bugs* códigos-fonte dos algoritmos. Dificuldades também foram encontradas na escolha das bases de dados, tendo sido necessárias duas bases de dados, uma binária e outra com mais de duas classes, e no processo de uso dos pesos gerados pelo Adaboost no algoritimo-base. Dificuldades que foram superadas com o auxílio da modelagem matemática do Adaboost.

Embora dificuldades tenham sido encontradas, os objetivos propostos nesta pesquisa foram atingidos, dado que os resultados das métricas de qualidade sinalizam o funcionamento correto do Adaboost.

Os resultados obtidos com a base de dados binária, em que o Adaboost obteve 98,55% de acurácia, quando comparado com outras pesquisas que utilizaram a mesma base, mostraram-se superiores a 17 dos 21 algoritmos analisados. As medidas de qualidade nesta base de dados para a classe *ckd,* que indica a presença de doença renal crônica*,* a Proporção de Falso Positivos (0,0435) e Proporção de Falso Negativos (0,0000) denotam uma taxa falso positivos e falso negativos chegando a zero; a sensibilidade alcançou 1, o maior valor possível; a especificidade, a precisão e o F-score aproximaram-se de 1. Para a classe *notckd* que indica a ausência de doença renal crônica, não ocorreu nenhum falso positivo e a Proporção de Falso Negativos (0,0435) atingiu valor próximo a zero. A sensibilidade (0,9787) e o F-score (0,9787) aproximaram-se do valor 1, que foi

alcançado pela precisão e especificidade. As medidas gerais do classificador, acurácia (98,5507%), *Area Under Curve* (0,9894) e estatística Kappa (0,9851), junto com as medidas individuais indicam a qualidade do classificador gerado para a base de dados binária.

Com a base de dados com múltiplas classes, em que obteve acurácia de 93,34%, o Adaboost foi superior a 17 dos 20 algoritmos analisados. As medidas de qualidade sensibilidade M (0,8855), sensibilidade μ (0,9014), precisão M (0,9046), precisão μ (0,9014), F-score M (0,8855), F-score μ (0,9014) e estatística Kappa (0,8986) aproximaram-se do valor máximo e melhor possível 1, evidenciam o funcionamento satisfatório do algoritmo neste tipo de problema.

Visando dar continuidade ao projeto da Shell Orion Data Mining Engine e utilizando-se do conhecimento adquirido no desenvolvimento desta pesquisa, foram propostas sugestões de trabalhos futuros:

a) pesquisar sobre os algoritmos variantes do Adaboost, a fim de avaliar a sua aplicabilidade nas bases de dados selecionadas e comparar os resultados;
b) refatorar o código dos algoritmos de classificação já existentes na Shell Orion para que eles possam ser utilizados como algoritmo-base do Adaboost e aplicar medidas de qualidade para avaliar os classificadores gerados dessas novas combinações;
c) estudar e aplicar outras medidas de qualidade na classificação de dados tanto para problemas binários como para problemas de múltiplas classes;
d) Inserir o cálculo do intervalo de confiança para estimativas pontuais das medidas de qualidade;
e) aplicar outras práticas de desenvolvimento na linguagem de programação Java com o objetivo de reduzir o tempo gasto na geração dos classificadores do Adaboost na Shell Orion.

REFERÊNCIAS

AGGARWAL, Charu C. **Data Mining:** The Textbook. New York: Springer, 2015.

BABY, P. S.; VITAL, T. P. Statistical analysis and predicting kidney diseases using machine learning algorithms. **International journal of engineering research & technology**, 2015. v. 4, n. 7, p. 206–210. Disponível em: <http://dx.doi.org/10.17577/IJERTV4IS070234>. Acesso em: 9 nov. 2016.

BEZERRA, Eduardo. **Princípios De Análise E Projeto De Sistemas Com Uml.** 3.ed. Rio de Janeiro: Elsevier, 2015.

BOROVICKA, Tomas et al. Selecting representative data sets.In: **Advances in Data Mining Knowledge Discovery and Applications.** Rijeka: InTech, 2012. p. 43-70.

BROWN, M. S. **Data Mining for Dummies**. New Jersey: John Wiley & Sons, 2014.

CHARYTANOWICZ, M. *et al.* A complete gradient clustering algorithm for features analysis of x-ray images. **Advances in intelligent and soft computing**, 2010. v. 69, p. 15–24. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.711.8820&rep=rep1&type=pdf>. Acesso em: 10 nov. 2016.

DENNIS, Alan; WIXOM, Barbara Haley; TEGARDEN, David. **System analysis & design:** An object-oriented approach with UML. 5. ed. Hoboken: John Wiley & Sons, 2015.

DZEROSKI, Saso; BERNARD, Zenko. Is Combining Classifiers Better than Selecting the Best One? **Machine Learning**, Ljubljana, v. 54, n. 3, p. 255–273, abr. 2004. Disponível em: <http://kt.ijs.si/bernard/papers/04-dze_zen-mlj04.pdf>. Acesso em: 21 mar. 2016.

DUBOIS, Paul. **MySQL Cookbook.** 3. ed. Sebastopol: O'Reilly, 2014. 866 p.

FAYYAD, U; PIATETSKY-SHAPIRO, G; SMYTH, P. **From data mining to knowledge discovery in databases.** AI magazine, v. 17, n. 3, p. 37–53, 1996. Disponível em: <http://dx.doi.org/10.1609/aimag.v17i3.1230>. Acesso em: 21 mar. 2016.

FELKIN, Mary. In: **Quality measures in data mining**. Berlim: Springer, 2007. p. 282-306.

FERREIRA, Artur J.; FIGUEIREDO, Mário A. T. Ensemble learning. In: **Ensemble machine learning**. New York: Springer, 2012. p. 1-34.

FIELD, Andy. **Discovering Statistics Using SPSS**. 3. ed. London: SAGE, 2009. 857 p.

FOWLER, Martin. **UML Distilled:** a brief guide to the standard object modeling language. 3. ed. Boston: Addison-Wesley Professional, 2004. 208 p.

FREUND, Yoav; SCHAPIRE, Robert E. A desicion-theoretic generalization of on-line learning and an application to boosting. In: **Computational learning theory**. Berlin: Springer Berlin Heidelberg, 1995. p. 23-37. Disponível em <http://dx.doi.org/10.1006/jcss.1997.1504>. Acesso em: 10 mai. 2016.

FREUND, Yoav; SCHAPIRE, Robert E. A short introduction to boosting. **IJCAI International Joint Conference on Artificial Intelligence**, v. 2, n. 5, p. 1401–1406, set. 1999. Disponível

em <http://www1.cs.columbia.edu/~jebara/4772/papers/IntroToBoosting.pdf>. Acesso em: 4 abr. 2016.

FREUND, Yoav; SCHAPIRE, Robert E. Experiments with a New Boosting Algorithm. In: **International Conference on Machine Learning**. 1996, Murray Hill: AT&T Laboratories, 1996. p.148–156. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.6252>. Acesso em 3 out. 2016.

GERSTING, Judith; SCHNEIDER, G. M. **Fundamentos Matemáticos para a Ciência da Computação**. 3. ed. Rio de Janeiro: LTC, 1995.

GUEDES, Gilleanes T. A. **UML 2:** uma abordagem prática. 2. ed. São Paulo: Novatec, 2011.

GOLDSCHMIDT, Ronaldo; BEZERRA, Eduardo. **Data Mining: Conceitos, técnicas, algoritmos, orientações e aplicações**. São Paulo: Elsevier Brasil, 2015.

GORUNESCU, Florin. **Data Mining:** Concepts, Models and Techniques. România: Springer, 2011.

HAN, Jiawei; KAMBER, Micheline; PEI, Jian. **Data mining:** Concepts and techniques. 3.ed. Massachusetts: Morgan kaufmann, 2012.

HOLMES, Geoffrey; NEVILL-MANNING, Craig G. **Feature selection via the discovery of simple classification rules.** Hamilton: University of Wakaito, 1995.

HOLTE, Robert C. Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. **Machine Learning** v. 11, n. 1, p. 63–91, 1993.

KAMALAKKANNAN, V.; RAMYACHITRA, D. Analysis of Different Classification Algorithms Applied to Anneal Dataset Using Data Mining Techniques. **International Journal of Future Innovative Science and Engineering Research**, v. 2, n. 1, p. 127–134, 2016. Disponível em: <http://www.istpublications.com/temp/16_V_Kamalakkannan_Et_al_.pdf>. Acesso em 14 nov. 2016.

KANTARDZIC, Mehmed. **Data Mining:** Concepts, Models, Methods, and Algorithms**.** 2. ed. New Jersey: John Wiley & Sons, 2011.

KUMAR, M. Prediction of chronic kidney disease using random forest machine learning algorithm. **International journal of computer science and mobile computing**, 2016. v. 5, n. 2, p. 24–33. Disponível em: <http://ijcsmc.com/docs/papers/February2016/V5I2201606.pdf>. Acesso em 9 mov. 2016.

LAMBODAR, J.; NARENDRA, K. K. Distributed data mining classification algorithms for prediction of chronic- kidney-disease. **International journal of emerging research in management & technology**, 2015. v. 4, n. 11, p. 110–118. Disponível em: <http://ermt.net/docs/papers/Volume_4/11_November2015/V4N11-109.pdf>. Acesso em 9 mov. 2016.

LAROSE, Daniel T.; LAROSE, Chantal D. **Discovering knowledge in data:** an introduction to data mining. New Jersey: John Wiley & Sons, 2014.

LICHMAN, M. **UCI Machine Learning Repository.** Irvine: University of California, School of Information and Computer Science, 2016. Disponível em: <http://archive.ics.uci.edu/ml>.

Acesso em: 10 nov. 2016.

LINOFF, Gordon S.; BERRY, Michael J. A. **Data Mining Techniques:** For Marketing, Sales, and Customer Relationship Management. 3.ed. Indianapolis: Wiley Publishing, 2011. 888 p.

LUO, Ren C.; LIN, Tzu Ta; CHEN, Kuan Yu. Gender recognition based on ensemble learning with selective features for service robotics applications. **2011 IEEE International Conference on Robotics and Biomimetics** p. 1159–1164, 2011. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6181444>. Acesso em: 15 mai. 2016.

MARKOSKI, Branko et al. Application of AdaBoost algorithm in basketball player detection. **Acta Polytechnica Hungarica** v. 12, n. 1, p. 189–207, 2015. Disponível em: <http://www.uni-obuda.hu/journal/Markoski_Ivankovic_Ratgeber_Pecev_Glusac_5 7.pdf>. Acesso em 1 nov. 2016.

MATHANKER, S. K. et al. AdaBoost classifiers for pecan defect classification. **Computers and electronics in agriculture**, v. 77, n. 1, p. 60-68, 2011. Disponível em <http://dx.doi.org/10.1016/j.compag.2011.03.008>. Acesso em 27 mar. 2016.

NANDI, Júlio C. B. **A técnica de associação pelo algoritmo frequent pattern-growth (fp-growth) na shell orion data mining engine.** 2013. 123 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina.

NASCIMENTO, Débora Natália de Oliveira. **Classificação AdaBoost para Detecção e Contagem Automática de Plaquetas**. 2011. 55 f. Trabalho de Conclusão de Curso (Graduação em Engenharia da Computação) – Universidade de Pernanbuco, Recife. Disponível em: <http://tcc.ecomp.poli.br/20112/TCC_20112_DeboraNascimento.pdf>. Acesso em: 20 mai. 2016.

NEVILL-MANNING, Craig G.; HOLMES, Geoffrey; WITTEN, Ian H. The development of Holte's 1R classifier. In: **Artificial Neural Networks and Expert Systems, 1995. Proceedings., Second New Zealand International Two-Stream Conference on**. IEEE, 1995. p. 239-242. Disponível em: <http://doi.org/10.1109/ANNES.1995.499480>. Acesso em: 3 nov. 2016.

OLSON, David L.; DELEN, Dursun. **Advanced data mining techniques**. Berlim: Springer Science & Business Media, 2008.

POLIKAR, Robi. Ensemble learning. In: **Ensemble machine learning**. New York: Springer, 2012. p. 1-34.

PRESSMAN, Roger S.. **Engenharia de Software:** Uma Abordagem Profissional. 7. ed. Porto Alegre: Mcgraw-hill, 2011.

RAMYA, S.; RADHA, N. Diagnosis of chronic kidney disease using machine learning algorithms. **International Journal of Innovative Research in Computer and Communication Engineering**, v. 4, n. 1, p. 812-820, 2016. Disponível em: <http://www.ijircce.com/upload/2016/january/49_3_Diagnosis.pdf>. Acesso em: 9 nov. 2016

REIS, William Augusto Dias dos. Detecção de Sinais de Trânsito Através do Método de Classificação Adaboost. **UNOPAR Científica Ciências Exatas e Tecnológicas**, Paraná, v.

12, n. 1, p. 27-34, nov. 2013. Disponível em <http://pgsskroton.com.br/seer/index.php/exatas/article/view/375>. Acesso em: 27 mar. 2016.

SABANCI, K.; AKKAYA, M. Classification of Different Wheat Varieties by Using Data Mining Algorithms. **International Journal of Intelligent Systems and Applications in Engineering**, v. 4, n. 2, p. 40, 27 maio 2016. Disponível em: <http://ijisae.atscience.org/article/view/5000171369>. Acesso em 14 nov. 2016.

SCHAPIRE, Robert E.; FREUND, Yoav. **Boosting:** Foundations and algorithms. London: MIT press, 2012.

SINHA, Parul; SINHA, Poonam. Comparative study of chronic kidney disease prediction using knn and svm. **International journal of engineering research & technology**, 2015. v. 4, n. 12, p. 608–612. Disponível em: <http://dx.doi.org/10.17577/IJERTV4IS120622>. Acesso em: 9 nov. 2016.

SOKOLOVA, Marina; LAPALME, Guy. A systematic analysis of performance measures for classification tasks. **Information Processing and Management** v. 45, n. 4, p. 427–437, 2009. Disponível em: <http://dx.doi.org/10.1016/j.ipm.2009.03.002>. Acesso em: 13 jun. 2016.

SUJATHA, R. Evaluation of Classifiers to Enhance Model Selection. **International Journal of Computer Science and Engineering Technology** v. 4, n. 1, p. 16–21 , 2013. Disponível em: <http://www.ijcset.com/docs/IJCSET13-04-01-014.pdf>. Acesso em 14 nov. 2016.

TAN, Pang-Ning; STEINBACH, Michael; KUMAR, Vipin. **Introdução ao datamining:** mineração de dados. Rio de Janeiro: Ciência Moderna, 2009. 900 p.

VIJAYARANI, S.; DHAYANAND, S. Data mining classification algorithms for kidney disease prediction. **International journal on cybernetics & informatics**, 2015. v. 4, n. 4, p. 13–25. Disponível em: <http://www.airccse.org/journal/ijci/papers/4415ijci02.pdf>. Acesso em: 9 nov. 2016.

WEI, H.; LIN, X.; XU, X.; LI, L.; ZHANG, W.; WANG, X. A novel ensemble classifier based on multiple diverse classification methods. **11th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2014** p. 301–305 , 2014. Disponível em: <http://dx.doi.org/10.1109/FSKD.2014.6980850>. Acesso em 14 nov. 2016.

WITTEN, Ian H.;FRANK, Eibe; HALL, Mark A. **Data mining:** practical machine learning tools and techniques. 3. ed. Massachusetts: Elsevier, 2011.

WOLPERT, David H.; MACREADY, William G. No free lunch theorems for optimization. **Evolutionary Computation, IEEE Transactions on**, v. 1, n. 1, p. 67-82, 1997. Disponível em <http://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf>. Acesso em: 4 abr. 2016.

WOŹNIAK, Michał; GRAÑA, Manuel; CORCHADO, Emilio. A survey of multiple classifier systems as hybrid systems. **Information Fusion**, v. 16, p. 3–17, 2014. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S156625351300047X>. Acesso em: 27 mar. 2016.

ZAKI, Mohammed J; MEIRA JÚNIOR, Wagner. **Data Mining and Analysis:** Foundations and Algorithms. New York: Cambridge University Press, 2014.

ZHOU, Zhi-Hua; YU, Yang. Adaboost. In: **The Top Ten Algorithms in Data Mining.** Boca Raton: CRC Press, 2009. p. 127-147.

ZHOU, Zhi-Hua. **Ensemble Methods:** Foundations and Algorthms. Boca Raton: CRC Press, 2012. 236 p.

APÊNDICE A – ARTIGO

O Método de Metaclassificação pelo Algoritmo Adaboost na Shell Orion Data Mining Engine

Ramon P. Souza[1], Merisandra Côrtes de Mattos Garcia[2]

[1]Acadêmico do Curso de Ciência da Computação – Unidade Acadêmica de Ciências, Engenharias e Tecnologias – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma– SC

[2]Professora do Curso de Ciência da Computação – Unidade Acadêmica de Ciências, Engenharias e Tecnologias – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma– SC

ramonporto92@hotmail.com, mem@unes.net

Resumo. *O armazenamento facilitado por novas tecnologias têm inviabilizado o uso de técnicas tradicionais na obtenção de conhecimento em bases de dados. Este artigo demonstra a modelagem matemática e o desenvolvimento do algoritmo metaclassificador e de boosting Adaboost para a tarefa de classificação em bases de dados. Tarefa esta que consiste na atribuição de classes já definidas a objetos ou instâncias, dado que a versão utilizada deste algoritmo atende tanto a problemas binários quanto a problemas de múltiplas classes.*

Abstract. *The storage facilitated by new technologies has made unfeasible the use of traditional techniques in obtaining knowledge in databases. This paper demonstrates the mathematical modeling and the development of the Adaboost meta-classifier and boosting algorithm for the classification task in databases. This task consists of the assignment of already defined classes to objects or instances, since the version used of this algorithm serves both binary problems and problems of multiple classes.*

1. Introdução

Novos métodos e abordagens são necessários, uma vez que as técnicas tradicionais são inviáveis no processamento do volume crescente de dados que são armazenados a todo momento. O *data mining*, uma das etapas da descoberta de conhecimento em bases de dados (do inglês, *Knowledge Discovery in Databases*, KDD), surge como solução ao unir algoritmos avançados a estas técnicas tradicionais e extrair padrões de bases de dados.

A partir do que se busca extrair nas bases de dados diversas tarefas do *data mining* podem ser aplicadas, como a análise de associação, o agrupamento, a detecção de anomalias e a classificação. A tarefa de classificação resume-se na atribuição de uma

classe a um objeto ou instância. Quanto mais instâncias forem classificadas corretamente, melhor é o classificador gerado. Contudo, não é possível gerar um classificador eficiente em todas as situações (DZEROSKI; ZENKO, 2004, tradução nossa). A estratégia de metaclassificação parte do princípio de que não é necessário desenvolver um algoritmo classificador para se melhorar o resultado da classificação, e sim, unir vários classificadores já existentes, denominados classificadores-base, para se alcançar este objetivo (HAN; KAMBER; PEI, 2012, tradução nossa).

A metaclassificação possui duas abordagens no treinamento dos seus classificadores-base: *bagging*, que faz o treinamento dos classificadores de forma paralela e *boosting*, que faz o treinamento de forma sequencial. Sendo *boosting* conhecido e estudado para se desenvolver algoritmos de alta performance (FERREIRA; FIGUEIREDO, 2012, tradução nossa).

A principal característica dos algoritmos de *boosting* é a de gerar um classificador com alta capacidade de predição a partir de classificadores-base pouco melhores que a decisão aleatória (POLIKAR, 2012, tradução nossa). O algoritmo metaclassificador e de *boosting* com maior destaque é o Adaboost, que foi nomeado na *International Conference on Data Mining*, ICDM, conferência internacional do Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE), em 2008, um dos dez algoritmos mais populares em *data mining* (ZHOU; YU, 2009, tradução nossa).

Algoritmos como o Adaboost normalmente são implementados em ferramentas de *data mining*. Estas geralmente são proprietárias de alto custo, entretanto, a Shell Orion Data Mining Engine é uma ferramenta de *data mining* gratuita e multiplataforma que é desenvolvida pelo Grupo de Pesquisa em Inteligência Computacional Aplicada do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense.

Neste artigo é apresentada a implementação do algoritmo Adaboost no módulo de classificação da Shell Orion Data Mining Engine.

2. Adaboost

O algoritmo Adaboost é um algoritmo metaclassificador de *boosting* adaptável, daí surge o acrônimo Adaboost: *Adaptive Boosting*. Sua capacidade de adaptação provem da atribuição de pesos às instâncias mais difíceis de ser classificadas (FREUND; SCHAPIRE, 1995, tradução nossa).

Estes pesos são atribuídos durante as rodadas de treinamento do Adaboost, que é feito da seguinte maneira: Em uma distribuição de dados, cada instância rotulada d, representada por $(x_i, y_1), (x_i, y_2), \ldots, (x_d, y_d)$, possui uma instância x_i que é rotulada por uma classe y_i e todas as instâncias possuem valor inicial de $1/n$. Durante as t rodadas necessárias para gerar K classificadores, as instâncias têm seus pesos atualizados de acordo com o resultado das suas classificações: se classificada corretamente seu peso é aumentado e se não é classificada corretamente seu peso é diminuído. Este peso serve para medir o quão uma instância é difícil de classificar, na geração das instâncias do algoritmo seguinte e também no cálculo do peso do algoritmo, uma vez que alguns algoritmos são mais eficientes do que outros na classificação de certas instâncias. Ao fim do treinamento é feita a combinação que gerará o classificador final a partir voto majoritário considerando os pesos dos classificadores-base treinados (HAN; KAMBER; PEI, 2012 tradução nossa).

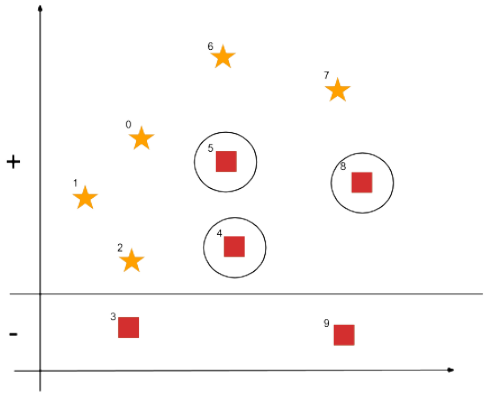
3. Adaboost na Shell Orion

A implementação do algoritmo Adaboost como módulo na Shell Orion iniciou-se primeiramente com o desenvolvimento dos diagramas de caso de uso, de sequência e de atividades. Em seguida, foi feita a modelagem matemática do algoritmo a fim de esclarecer as etapas de seu funcionamento.

Para a modelagem matemática foi considerado um conjunto de treinamento hipotético com 10 instâncias representadas na figura 1, extraído de uma base de dados binária, com as instâncias positivas simbolizadas por estrelas e as instâncias negativas simbolizadas por quadrados. A primeira etapa executada no Adaboost é a inicialização dos pesos, em que todas as instâncias receberão o mesmo pesos de 1 dividido pelo número de instâncias, neste caso 1 dividido por 10.

Em seguida, utilizando-se de um algoritmo-base que faça a predição das classes de forma linear e que o classificador obtido deste algoritmo acerte 7 das 10 instâncias, têm-se a linha divisória da figura 1 que separa na parte superior as instâncias classificadas como positivas e na parte inferior as classificadas como negativas.

Figura 1. Primeira rodada de treinamento do Adaboost



Os quadrados circulados evidenciam as três instâncias negativas erroneamente classificadas como positivas. A partir disso, o erro de treinamento ϵ_1 pode ser calculado pela soma dos pesos das instâncias que foram classificadas incorretamente, sendo ele, neste caso, a soma dos pesos destas instâncias.

$$\epsilon_1 = \sum_{i=0}^{9} w_i^{1-1} . I(M_1(x_i) \neq y_i) = \left(\frac{1}{10} + \frac{1}{10} + \frac{1}{10} \right) = 0{,}3000$$

O erro de treinamento foi diferente de 0 e menor que 0,5. Caso fosse igual a 0 significaria que o classificador foi perfeito, o que não ocorreu e, portanto, não seria necessário mais rodadas, e se fosse maior que 0,5 ultrapassaria o limite inferior à decisão aleatória, tornando-o inválido para ser usado como base do Adaboost. Dado ϵ_1 estar entre essas condições, pode-se fazer o cálculo do peso do classificador, que será armazenado para ser usado na geração do classificador final ao término das rodadas do Adaboost.

$$\alpha_1 = \ln\left(\frac{1-\epsilon_1}{\epsilon_1} \right) = \ln\left(\frac{1-0{,}3000}{0{,}3000} \right) \approx 0{,}8473$$

Em seguida, o valor do erro de treinamento é também utilizado para realizar a atualização dos pesos das instâncias, sendo aumentado os pesos das instâncias que foram

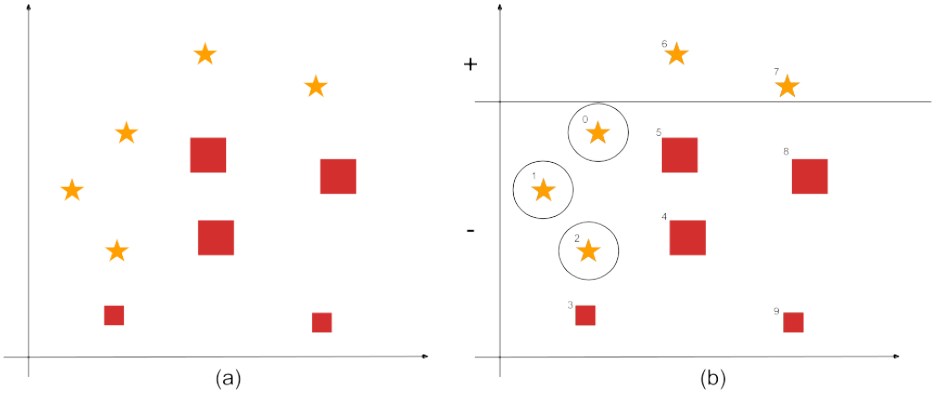
classificadas incorretamente. Após as atualizações dos pesos das instâncias, faz-se a normalização dos pesos, que é feita pela divisão do peso de cada membro pela soma de todos os pesos dos membros do conjunto. A figura 2 apresenta os pesos anteriores, que foram os pesos atribuídos ao criar o vetor de probabilidades; os pesos após a atualização gerada pelas classificações incorretas; e os pesos depois da normalização.

Figura 2. Distribuição atualizada e normalizada na 1ª rodada

índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
pesos anteriores (w^0)	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
pesos atualizados	0,1	0,1	0,1	0,1	0,2333	0,2333	0,1	0,1	0,2333	0,1
pesos normalizados (w^1)	0,0714	0,0714	0,0714	0,0714	0,1667	0,1667	0,0714	0,0714	0,1667	0,0714

Ao término da normalização, segue-se para a segunda rodada, porém com um adendo: as instâncias classificadas incorretamente na rodada anterior têm maior relevância nesta rodada. Esta relevância dos pesos é aplicada pela adição do vetor de probabilidades normalizado junto com o conjunto de treinamento diretamente no código do algoritmo que gerará o classificador ou, em algoritmos que não aceitem os pesos, é feita pela reamostragem considerando os pesos.

Figura 3. Segunda rodada de treinamento do Adaboost



No gráfico (a) da figura 3 nota-se os quadrados referentes às instâncias classificadas de forma incorreta representados em tamanho maior para indicar sua relevância. No gráfico (b) tem-se a predição do segundo classificador influenciada pela predição do classificador anterior. Nota-se três círculos neste mesmo gráfico que demonstram o erro do classificador, sendo então, possível fazer o cálculo do erro de treinamento do segundo classificador-base.

$$\epsilon_2 = \sum_{i=0}^{9} w_i^1 . I\left(M_2(x_i) \neq y_i\right) = (0{,}0714 + 0{,}0714 + 0{,}0714) = 0{,}2142$$

Conseguintemente, faz-se o cálculo do peso do classificador.

$$\alpha_2 = \ln\left(\frac{1-\epsilon_2}{\epsilon_2}\right) = \ln\left(\frac{1-0{,}2142}{0{,}2142}\right) \approx 1{,}2998$$

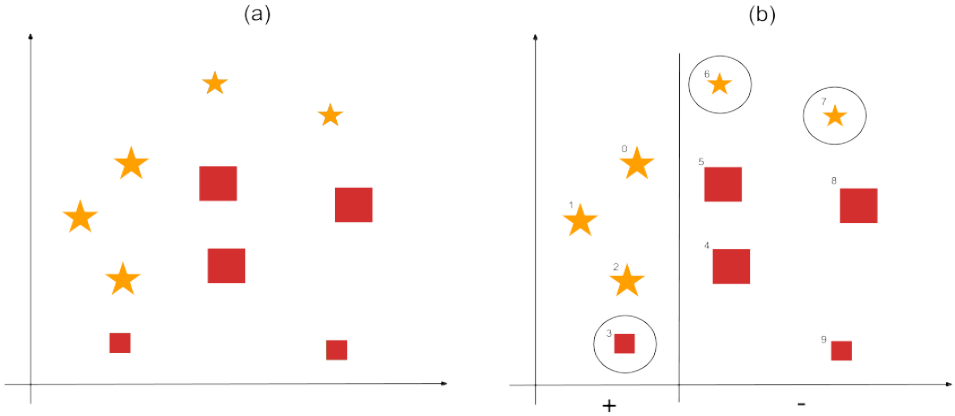
Em seguida, atualiza-se o peso das instâncias classificadas incorretamente e a normalização dos pesos, demonstrada na figura 4.

Figura 4. Distribuição atualizada e normalizada na 2ª rodada

índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
pesos anteriores (w^0)	0,0714	0,0714	0,0714	0,0714	0,1667	0,1667	0,0714	0,0714	0,1667	0,0714
pesos atualizados	0,2619	0,2619	0,2619	0,0714	0,1667	0,1667	0,0714	0,0714	0,1667	0,0714
pesos normalizados (w^1)	0,1667	0,1667	0,1667	0,0454	0,1061	0,1061	0,0454	0,0454	0,1061	0,0454

Isto feito, pode-se passar para a terceira rodada do Adaboost. Da mesma forma que ocorreu na rodada anterior, atribui-se maior importância às instâncias de maior peso, sendo que os pesos das duas rodadas anteriores influenciam nesta rodada.

Figura 5. Terceira rodada de treinamento do Adaboost



Na figura 5 tem-se o resultado da classificação nesta rodada. O classificador errou também três instâncias, portanto, seu erro de treinamento é:

$$\epsilon_3 = \sum_{i=0}^{9} w_i^2 . I\left(M_3(x_i) \neq y_i\right) = (0{,}0454 + 0{,}0454 + 0{,}0454) = 0{,}1362$$

Dado o erro de treinamento, o peso do classificador é obtido por:

$$\alpha_3 = \ln\left(\frac{1-\epsilon_3}{\epsilon_3}\right) = \ln\left(\frac{1-0{,}1362}{0{,}1362}\right) \approx 1{,}8472$$

Depois do cálculo do peso do classificador, as instâncias são atualizadas. Finda a atualização, segue-se com a normalização dos pesos (figura 6).

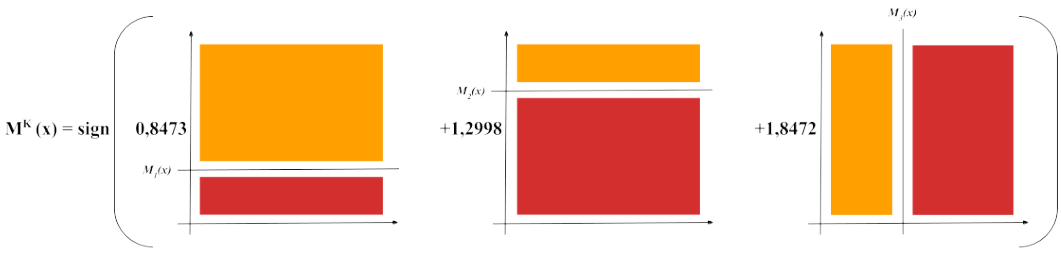
Figura 6. Distribuição atualizada e normalizada na 3ª rodada

índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
pesos anteriores (w^0)	0,1667	0,1667	0,1667	0,0454	0,1061	0,1061	0,0454	0,0454	0,1061	0,0454
pesos atualizados	0,1667	0,1667	0,1667	0,2879	0,1061	0,1061	0,2879	0,2879	0,1061	0,0454
pesos normalizados (w^1)	0,0965	0,0965	0,0965	0,1667	0,0614	0,0614	0,1667	0,1667	0,0614	0,0263

É possível notar que este último classificador foi induzido a concentrar-se justamente nas instâncias em que os classificadores anteriores falharam, o que pode ser justificado pela maior importância dada a elas. Também é possível notar que sua predição se torna complementar às predições dos outros dois classificadores.

Com essa função complementar entre os classificadores gerados, a combinação feita no Adaboost pode predizer corretamente todas as instâncias deste conjunto de treinamento. A figura 7 representa esta combinação dos classificadores, tendo sido os pesos α_i de cada classificador $M_i(x)$ introduzidos ao lado de cada gráfico.

Figura 7 – Predição do conjunto dos classificadores



O resultado da classificação torna-se o sinal da classe (+ e -) considerando o peso de cada membro e sua resposta. Por exemplo, a instância de índice 8 foi classificada como positiva (+1) pelo primeiro classificador, então adiciona-se para esta classe o peso de 0,8473 deste classificador; o segundo classificador classificou-a como negativa (-1), atribuindo para a classe 1,2998; por fim, o terceiro classificador também classificou-a como negativa, acrescentando o peso de 1,8472, o que resulta na equação:

$$\begin{aligned} & sign\left(M_1(x)*(+1) \ + \ M_2(x)*(-1) \ + \ M_3(x)*(-1)\right) \\ & = \ 0{,}8473-1{,}2998-1{,}8472 \\ & = \ sign(-2{,}2997) \\ & = \ -1 \end{aligned}$$

O resultado da combinação foi o sinal obtido da classe -1, que é a classe correta por ser a classe real da instância.

Como foi mencionado ao início da modelagem, o caso supracitado foi aplicado em uma base de dados binária. Bases de dados com múltiplas classes já são consideradas no algoritmo que foi apresentado. Modificações ocorrem apenas na geração do classificador final, onde leva-se em conta, ao invés do sinal do voto da maioria, a classe com o maior voto, tornando-se necessário computar o voto da maioria para cada classe.

3.1. Implementação

Para a implementação do Adaboost foi utilizado o ambiente de desenvolvimento integrado Nebeans na versão 8.1, com a linguagem de programação Java utilizando-se do kit de desenvolvimento Java na versão 1.8.0_101. O Sistema Gerenciados de Bases de Dados no qual as bases de dados foram aplicadas foi o MySQL.

Antes da implementação do Adaboost fez-se a seleção e implementação do algoritmo-base. O algoritmo selecionado foi o 1-Rule, também denominado *Decision Stump*, de Holte (1993, tradução nossa), por ele ser comumente utilizado na literatura como base do Adaboost, tendo inclusive os autores originais do Adaboost feito experimentos utilizando o 1-Rule como algoritmo-base do mesmo (FREUND; SCHAPIRE, 1996, tradução nossa) e obras mais recentes, como Schapire e Freund (2012, tradução nossa) e Markoski et al (2015, tradução nossa) também referenciam o algoritmo como um possível algoritmo-base do Adaboost.

3.2. Resultados Obtidos

A primeira base de dados utilizada é formada 203 instâncias usadas para identificar pacientes que possuem doença renal crônica ou não. Na base preprocessada 79 possuíam doença renal crônica e 124 não possuíam.

A segunda base de dados, com múltiplas classes, conta com 210 instâncias e 8 atributos, incluindo com o atributo de classe e diferencia sementes de trigo em três variedades: *Canadian*, *Kama* e *Rosa.*

Os resultados obtidos compreendem a análise das medidas de qualidade empregadas no algoritmo e a análise do desempenho pela avaliação da acurácia do classificador gerado em uma base de dados binária e outra de múltiplas classes. Também foi efetuada a análise do tempo gasto para a execução do Adaboost e a comparação dos tempos de processamento obtidos na Shell Orion com a ferramenta Weka.

3.2.1 Identificação binária das classes pelo Adaboost

Para o primeiro teste realizado foram informados os parâmetros *holdout* em 66% e executado com o número de iterações em 10, a fim de avaliar a taxa de acertos do Adaboost na base de dados aplicada.

O número de iterações foi definido em 10, pois, não necessariamente um maior número de rodadas aumentará a capacidade de previsão do classificador, dado que, conforme Zhou (2012, tradução nossa) informar um número muito grande de rodadas tende a gerar *overfitting* no Adaboost e complexidade desnecessária e com um número muito pequeno não traria os benefícios obtidos pelo aprendizado sucessivo que ocorre nas rodadas.

Tabela 1. Resumo das medidas de qualidade para *ckd* e *notckd*

Métrica	*ckd*	*notckd*
Proporção de Falso Positivos	0,0435	0,0000
Proporção de Falso Negativos	0,0000	0,0435
Sensibilidade	1,0000	0,9787
Especificidade	0,9787	1,0000
Precisão	0,9565	1,0000
F-Score	0,9999	0,9787

A tabela 1 apresenta as medidas de qualidade individuais para cada classe, sendo *ckd* a classe que representa a presença de doença renal crônica e *notckd* a ausência de doença renal crônica.

Para *ckd*, a Proporção de Falso Positivos, avaliada em 0,0435, mostra um pequeno erro ao considerar a classe correta e a Proporção de Falso Negativos evidencia que classificador não considerou nenhum caso da classe *notckd* como *ckd*, sendo perfeito na identificação. A sensibilidade, que informa que a porcentagem de Verdadeiro Positivos que realmente são positivos, alcança o valor 1, que é o valor máximo possível. De forma semelhante, a especificidade informa um valor de 0,9787 para a porcentagem de Verdadeiro Negativos que realmente são negativos. O F-Score, que pode ser considerado a média ponderada entre a sensibilidade e a especificidade (BOROVICKA et al, 2012, tradução nossa), obteve 0,9999 de 1 possíveis.

Para *notckd, os* valores de Proporção de Falso Positivos, em 0, e Proporção de Falso Negativos próximo a zero revelam uma proporção mínima de identificação incorreta das classes. A sensibilidade e o f-score aproximaram-se do valor máximo de 1, valor esse que foi obtido pela especificidade e a precisão, indicando uma boa capacidade de predição na classe.

Os resultados independentes da classe analisada foram resumidos na tabela 2.

Tabela 2. Medidas de qualidade para *Chronic Kidney Disease*

Métrica	Valor
Taxa de erro	1,4493%
Acurácia	98,5507%
Area Under Curve (AUC)	0,9894
Estatística Kappa	0,9851

Os resultados da tabela 2, denotam uma acurácia próxima de 100%, o que informa uma ótima capacidade de generalização para o conjunto de testes e consequentemente um erro mínimo. A *Area Under Curve*, que é a capacidade do classificador de evitar a falsa classificação foi calculada em 0,9894 e a estatística *Kappa,* obteve uma concordância entre o que foi predito e observado quase perfeita (0,9851). A partir destes resultados, somados aos resultados individuais, identifica-se a qualidade do classificador gerado para a base de dados binária.

3.2.2 Indetificação de múltiplas classes pelo Adaboost

Os testes aplicados a base de dados com múltiplas classes foram semelhantes aos aplicados na base de dados binária. O parâmetro *holdout* foi definido em 66% e o número de iterações foi informado em 10.

As medidas obtidas para cada classes foram disponibilizadas na tabela 3, a fim de demonstrar o desempenho de cada uma com relação as outras.

Tabela 3. Medidas por classe do Adaboost na segunda base de dados

Classe	Acurácia (%)	PFP	PFN	Sensibilidade	Especificidade	Precisão	F-Score	AUC
Canadian	91,4286%	0,1852	0,0233	0,9565	0,8936	0,8148	0,9563	0,9251
Kama	90,1408%	0,0667	0,1071	0,7000	0,9804	0,9333	0,7008	0,8402
Rosa	98,4615%	0,0345	0,0000	1,0000	0,9730	0,9655	1,0000	0,9865

A acurácia de todas as três classes foram superiores a 90%, sendo a semente *Rosa* a mais bem avaliada nas medidas de qualidade.

As medidas gerais para o classificador obtido foram reunidas na tabela 4.

Tabela 4. Medidas gerais para a base de dados multiclasse

Métrica	Valor
Taxa de erro Média	6,6563%
Acurácia Média	93,3447%
Sensibilidade M	0,8855
Sensibilidade μ	0,9014
Precisão M	0,9046
Precisão μ	0,9014
F-Score M	0,8855
F-Score μ	0,9014
Estatística Kappa	0,8986

Tanto os valores micro *(*μ), que são as médias geradas dos valores obtidos por classe, quanto os valores macro (M), que são médias calculadas de valores gerais, foram satisfatórios na base de dados, sendo todos os valores superiores a 0,88 e considerando o maior valor alcançável o valor 1. As acurácias por classe de 91,4286% para *Canadian*, 90,1408% para *Kama* e 98,4615% para *Rosa*, além a acurácia média de 93,3447%, demostram a eficiência do Adaboost na predição da base de dados para múltiplas classes.

3.2.3 Tempos de processamento do algoritmo Adaboost

A avaliação do tempo de processamento do Adaboost, foi feita pela análise do tempo de treinamento do algoritmo na Shell Orion Data Mining Engine e a análise do tempo de treinamento na ferramenta *Weka.*

Weka é uma ferramenta de *data mining* desenvolvida na linguagem de programação Java (o que permite que a ferramenta seja multiplataforma) e é frequentemente utilizada em pesquisas da área. Sua distribuição é feita sob a licença *General Public License* (GNU) e sua manutenção é feita pela Universidade de *Wakaito,* na Nova Zelândia (WITTEN; FRANK; HALL, 2011, tradução nossa).

A análise na ferramenta *Weka* levou em conta a adequação dos parâmetros de forma que eles fossem os mesmos que foram aplicados na Shell Orion, e considerou-se para a etapa de avaliação, diferentes quantidades de instâncias, que foram replicadas de uma base de dados com 470 instâncias. A estratégia de replicação dos dados foi feita pelo crescimento geométrico, que é definido pela equação:

$$C_k = N.2^{k-1}$$

A tabela 5 apresenta o resultado processo de replicação dos dados na coluna *Quantidade de Instâncias* já com os tempos de execução obtidos das duas ferramentas.

Tabela 5. Tempo de processamento do Adaboost na *Weka*

Quantidade de instâncias	Shell Orion	*Weka*
470	00 m: 00 s. 137 ms	00 m: 00 s. 020 ms
940	00 m: 00 s. 242 ms	00 m: 00 s. 030 ms
1880	00 m: 00 s. 382 ms	00 m: 00 s. 040 ms
3770	00 m: 00 s. 876 ms	00 m: 00 s. 080 ms
7520	00 m: 02 s. 239 ms	00 m: 00 s. 140 ms
15040	00 m: 07 s. 145 ms	00 m: 00 s. 140 ms
30080	00 m: 30 s. 288 ms	00 m: 00 s. 310 ms
60160	02 m: 58 s. 325 ms	00 m: 00 s. 570 ms
120320	15 m: 57 s. 353 ms	00 m: 01 s. 353 ms

Os tempos obtidos foram analisados por meio da ferramenta IBM *Statistical Package for Socil Sciences* (SPSS).

Para provar a hipótese de que os tempos distribuem normalmente, aplicou-se o teste de *Shapiro-Wilk*, que resultou em um valor de $p > 0.05$, demonstrando que os tempos não distribuem normalmente. Conseguintemente, fez-se a normalização dos dados pela conversão logarítmica dos tempos e aplicou-se novamente o teste de *Shapiro-Wilk*, resultando agora em distribuições dentro da normalidade.

Em seguida, com o teste F de Levene, proposto em 1960 por Levene, buscou-se avaliar a homogeneidade das variâncias obtidas. Este teste resultou em p = 0,270, sendo possível a hipótese nula e a homogeneidade existente entre as variâncias. Dada a possibilidade da hipótese nula, aplicou-se o teste *t* de *Student* com o objetivo de comparar as médias obtidas do logaritmo natural dos tempos. O valor *p* definido em $p < 0,001$, inferior a $p = 0,05$, demostra que as diferenças entre as médias foi significativa e que a hipótese nula é inválida, não havendo igualdade entre as médias do logaritmo natural dos tempos da Shell Orion e da *Weka.*

4. Considerações finais

Este artigo apresentou o algoritmo Adaboost que utiliza a abordagem de metaclassificação com o propósito de gerar um classificador com alta capacidade de predição. Este algoritmo foi implementado como um módulo na tarefa de classificação

com a finalidade de ampliar as funcionalidades da ferramenta Shell Orion Data Mining Engine.

Ao analisar os resultados obtidos, pode-se considerar a aplicabilidade da versão implementada do Adaboost na tarefa de classificação, tanto para problemas binários quanto para problemas de múltiplas classes, dado este algoritmo ter produzido resultados satisfatórios nas medidas de qualidade aplicadas. Conclui-se portanto, que o Adaboost foi implementado corretamente no módulo de classificação da Shell Orion.

Referências

BOROVICKA, Tomas et al. Selecting representative data sets.In: **Advances in Data Mining Knowledge Discovery and Applications.** Rijeka: InTech, 2012. p. 43-70.

DZEROSKI, Saso; BERNARD, Zenko. Is Combining Classifiers Better than Selecting the Best One? **Machine Learning**, Ljubljana, v. 54, n. 3, p. 255–273, abr. 2004. Disponível em: <http://kt.ijs.si/bernard/papers/04-dze_zen-mlj04.pdf>. Acesso em: 21 mar. 2016.

FERREIRA, Artur J.; FIGUEIREDO, Mário A. T. Ensemble learning. In: **Ensemble machine learning**. New York: Springer, 2012. p. 1-34.

FREUND, Yoav; SCHAPIRE, Robert E. A desicion-theoretic generalization of on-line learning and an application to boosting. In: **Computational learning theory**. Berlin: Springer Berlin Heidelberg, 1995. p. 23-37. Disponível em <http://dx.doi.org/10.1006/jcss.1997.1504>. Acesso em: 10 mai. 2016.

FREUND, Yoav; SCHAPIRE, Robert E. Experiments with a New Boosting Algorithm. In: **International Conference on Machine Learning**. 1996, Murray Hill: AT&T Laboratories, 1996. p.148–156. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.6252>. Acesso em 3 out. 2016.

HAN, Jiawei; KAMBER, Micheline; PEI, Jian. **Data mining:** Concepts and techniques. 3.ed. Massachusetts: Morgan kaufmann, 2012.

HOLTE, Robert C. Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. **Machine Learning** v. 11, n. 1, p. 63–91, 1993.

MARKOSKI, Branko et al. Application of AdaBoost algorithm in basketball player detection. **Acta Polytechnica Hungarica** v. 12, n. 1, p. 189–207, 2015.

POLIKAR, Robi. Ensemble learning. In: **Ensemble machine learning**. New York: Springer, 2012. p. 1-34.

SCHAPIRE, Robert E.; FREUND, Yoav. **Boosting:** Foundations and algorithms. London: MIT press, 2012.

ZHOU, Zhi-Hua; YU, Yang. Adaboost. In: **The Top Ten Algorithms in Data Mining.** Boca Raton: CRC Press, 2009. p. 127-147.