UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MINERAÇÃO DE DADOS PARALELA E DISTRIBUÍDA BASEADA NO AMBIENTE WEKA

TRABALHO DE GRADUAÇÃO

Vinícius Garcia Pinto

Santa Maria, RS, Brasil

2010

MINERAÇÃO DE DADOS PARALELA E DISTRIBUÍDA BASEADA NO AMBIENTE WEKA

por

Vinícius Garcia Pinto

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Profª Drª Andrea Schwertner Charão

Trabalho de Graduação N. 316

Santa Maria, RS, Brasil

2010

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

MINERAÇÃO DE DADOS PARALELA E DISTRIBUÍDA BASEADA NO AMBIENTE WEKA

elaborado por
Vinícius Garcia Pinto

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Profª Drª Andrea Schwertner Charão
(Presidente/Orientador)

Profª Drª Deise de Brum Saccol (UFSM)

Marcelo Lopes Kroth (CPD/UFSM)

Santa Maria, 07 de Dezembro de 2010.

“Ninguém é tão grande que não possa aprender, nem tão pequeno que não possa ensinar.”

— PÍNDARO

AGRADECIMENTOS

Primeiramente agradeço a Deus e a minha família, aos meus pais por tudo o que eles representam pra mim e pelo que me ensinaram até hoje, a minha irmã pelo incentivo acadêmico e por estar sempre ao meu lado, ainda que virtualmente, muito obrigado por acreditarem em mim e por me aturarem em todos os momentos, em especial naquelas semanas de mau humor de fim de semestre.

Aos meus amigos que me acompanham nos mais diversos momentos, aos mais antigos, meus primos Bruna, Fernando e Bruno e minhas colegas de pré-escola Anelise e Suzan, aos que conheci um pouco depois, a Fernanda, o Mega e a Camila, muito obrigado pela parceria nas festas, nos estudos, nas conversas, nas viagens, enfim, obrigado por estarem ao meu lado.

Aos meus colegas de aula e de curso, o Adriano, a Lari e o Lucas, amigos e companheiros de muitos dos trabalhos de aula e das milhares de linhas de código programadas, testadas, debugadas e apresentadas. Ao pessoal do PET CC, o Rissetti, o Vielmo, o Fábio, o Cassales, o Fred, o Jr/Candia e o Ktok. Aos amigos que conheci nos outros PETs da UFSM, no Universitar e nas outras incontáveis comissões.

À professora Andrea pela atenção, pela paciência e pela orientação recebida no PET, nas disciplinas do curso e no TG. A professora Deise e ao Marcelo pela disponibilidade de participar da banca examinadora desse TG e pelas sugestões recebidas. Ao CPD/UFSM, LAD/PUCRS e LSC pelo apoio técnico recebido.

A todos os demais amigos, familiares, professores, colegas, conhecidos, enfim, a todos que de alguma forma colaboraram para este trabalho e por tornarem o curso mais humanamente possível e divertido. Muito Obrigado!

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

MINERAÇÃO DE DADOS PARALELA E DISTRIBUÍDA BASEADA NO AMBIENTE WEKA
Autor: Vinícius Garcia Pinto
Orientador: Profª Drª Andrea Schwertner Charão
Local e data da defesa: Santa Maria, 07 de Dezembro de 2010.

O desenvolvimento tecnológico tem permitido a geração e o registro de volumes de dados cada vez maiores. A Mineração de Dados, de uma maneira geral, consiste na aplicação de algoritmos sobre grandes bases de dados com a finalidade de extrair conhecimento útil não trivial dessas. Devido ao tamanho das bases de dados atuais e a complexidade das técnicas envolvidas, torna-se vantajoso o uso de soluções distribuídas para o processo de mineração de dados. O WEKA (Waikato Environment for Knowledge Analysis) é um ambiente de mineração de dados centralizado que tem sido usado como base para ferramentas de mineração de dados paralelas e distribuídas. O presente trabalho tem por objetivo explorar, através de um estudo de caso, a ferramenta de processamento paralelo e distribuído Grid WEKA. Nesse contexto, são identificadas as técnicas implementadas, a disponibilidade de paralelismo e distribuição e é feita a análise e discussão do desempenho da ferramenta utilizando diferentes técnicas de mineração de dados em diferentes configurações do ambiente distribuído.

Palavras-chave: Mineração de Dados; Mineração de Dados Distribuída; Mineração de Dados Paralela; WEKA.

ABSTRACT

Trabalho de Graduação
Undergraduate Program in Computer Science
Universidade Federal de Santa Maria

DISTRIBUTED AND PARALLEL DATA MINING WEKA BASED ENVIRONMENT

Author: Vinícius Garcia Pinto
Advisor: Profª Drª Andrea Schwertner Charão

Technological development have allowed the generation and recording of data volumes increasing. Data mining consists in apply algorithms over large data bases to extract them useful knowledge. The database size and the complexity of the techniques involved makes necessary use of distributed solutions in the data mining process. WEKA (Waikato Environment for Knowledge Analysis) is a centralized data mining environment that has been used as a basis to parallel and distributed data mining tools. The purpose of this work is explore, through a case study, Grid WEKA, a tool based on WEKA. In this context are identified included techniques, the available of parallelism and distribution and analyzed and discussed the performance using different data mining techniques in different aspects of the distributed environment.

Keywords: Data Mining, Distributed Data Mining, Parallel Data Mining, WEKA.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ARFF	Attribute-Relation File Format
ASCII	American Standard Code for Information Interchange
CSV	Comma-Separated Values
GPL	General Public License
GUI	Graphical User Interface
JDBC	Java DataBase Connectivity
JVM	Java Virtual Machine
KDD	Knowledge Discovery in Databases
NFS	Network File System
UFSM	Universidade Federal de Santa Maria
WEKA	Waikato Environment for Knowledge Analysis
WSRF	Web Services Resource Framework

SUMÁRIO

1 INTRODUÇÃO

No contexto da evolução histórica do gerenciamento de informações, os processos de informatização e automatização modificaram profundamente a maneira como as informações são identificadas, atualizadas e armazenadas. O uso de sistemas computacionais permitiu não apenas o aumento da produtividade, mas também o processamento das informações. Dessa forma, o armazenamento e a divulgação das informações foram acrescidos à identificação de relacionamentos e definições que possibilitaram a criação de interpretações, facilitando a compreensão do conteúdo armazenado.

Nas últimas décadas, o desenvolvimento da tecnologia no âmbito do processamento computacional, da confiabilidade e velocidade da comunicação, das ferramentas de armazenamento e dos sistemas gerenciadores de bases de dados têm permitido a geração e o registro de um número cada vez maior de informações. O grande volume armazenado torna necessário o uso de ferramentas que auxiliem a identificação de informações que estão ocultas, convertendo-as em conhecimento significativo.

A mineração de dados compreende um conjunto de técnicas e algoritmos que possibilitam a identificação de padrões, relações, regras e associações a partir dos dados brutos. Dessa forma, é possível extrair conhecimento relevante a partir de grandes volumes de dados armazenados (LAROSE, 2005). As técnicas e algoritmos utilizados possuem características multidisciplinares, sendo fundamentados em áreas como inteligência artificial, análise estatística e banco de dados (BERNARDI, 2010). No cenário atual, onde os volumes de dados são enormes e heterogêneos, os algoritmos são complexos e os repositórios de dados podem estar geograficamente dispersos, a mineração de dados realizada de forma distribuída e paralela têm se tornado a alternativa para viabilizar e acelerar o processo de mineração (FU, 2001).

O ambiente WEKA (University of Waikato, 2010a), desenvolvido em Java e licen-

ciado como software livre, é uma das mais conhecidas ferramentas para mineração de dados. Esta ferramenta, originalmente desenvolvida para mineração de dados de forma centralizada, tem sido utilizada como base para diversas outras soluções as quais buscam prover a execução do processo de mineração de dados, ou ao menos de parte dele, de forma paralela e distribuída (ZUO; KHOUSSAINOV; KUSHMERICK, 2004; CELIS; MUSICANT, 2002; TALIA; TRUNFIO; VERTA, 2005).

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho consiste em investigar uma solução para mineração de dados paralela e distribuída baseada no ambiente Weka, através de estudo das ferramentas que implementam distribuição e/ou paralelismo com o WEKA, da preparação de uma fonte de dados para uso na ferramenta escolhida e de estudos de casos de mineração de dados utilizando essa ferramenta.

1.1.2 Objetivos Específicos

- Conhecer ambientes que implementem distribuição/paralelismo com o WEKA
- Escolher um dos ambientes que implementem distribuição/paralelismo com o WEKA
- Preparar uma fonte de dados para uso com um desses ambientes
- Identificar as técnicas implementadas e a disponibilidade de paralelismo e distribuição em um dos ambientes
- Analisar o desempenho de um dos ambientes utilizando diferentes técnicas de mineração de dados em diferentes configurações do ambiente distribuído

1.2 Organização do Texto

Este trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta uma revisão da literatura acerca dos temas relacionados a este trabalho, tais como conceitos, técnicas e o processo de mineração de dados e a mineração de dados de forma distribuída. Além disso, são apresentados o ambiente para mineração de dados WEKA e algumas ferramentas para mineração de dados distribuída que utilizam o WEKA como base.

O Capítulo 3 aborda o ambiente computacional de mineração de dados, escolha da ferramenta, procedimentos de instalação e configuração da ferramenta selecionada, a preparação dos dados que contempla a descrição do formato de entrada dos dados, a fonte e os procedimentos para conversão dos dados, além de testes preliminares para identificar características do funcionamento da ferramenta.

No Capítulo 4, apresenta-se um estudo de caso da ferramenta escolhida para mineração de dados paralela e distribuída. São descritos os ambientes computacionais utilizados, os algoritmos aplicados, os conjuntos de dados utilizados e quatro casos de teste reunindo ambientes, algoritmos e conjuntos de dados. Também é apresentada uma seção de discussão sobre os resultados identificados durante a execução dos testes.

Por fim, no Capítulo 5, são apresentadas as conclusões deste trabalho. A última seção contém as referências bibliográficas utilizadas no desenvolvimento do trabalho.

2 FUNDAMENTAÇÃO

Neste capítulo são descritos alguns conceitos que formaram o embasamento teórico necessário para o desenvolvimento deste trabalho.

2.1 Mineração de Dados

A aquisição de informações, a conversão dessa em conhecimento, e a utilização desse para fins úteis são atividades presentes na sociedade desde o seu princípio. O conhecimento gerado é utilizado para controlar, otimizar, administrar, analisar, investigar, prever, negociar ou tomar decisões sobre um domínio ou área específica. Dessa forma, os indivíduos, instituições e organizações têm dado grande importância as informações e ao conhecimento que possuem.

O constante aprimoramento tecnológico dos dispositivos de processamento, armazenamento e comunicação têm permitido aumentar o volume de dados gerados e registrados. Ainda que uma quantidade enorme de informações esteja sendo armazenada, não se tem conseguido extrair conhecimento na mesma proporção: alguns registros são armazenados e nunca mais são acessados, outros são ocultados pela grande quantidade de informações. Dessa forma, são necessárias soluções que permitam facilitar o entendimento e a compreensão das informações registradas. Técnicas como a mineração de dados permitem extrair a essência da informação através da exploração do uso de abstrações de dados e da identificação de padrões e correlações significativos.

De acordo com Fayyad (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996), Mineração de Dados pode ser definida como um processo não trivial de identificação de padrões válidos, novos e potencialmente úteis implícitos em grandes volumes de dados. O processo de mineração de dados fornece um método automático ou semiautomático para descoberta de padrões de dados e regras relevantes (BERRY; LINOFF, 2004), ex-

cluindo a tendenciosidade e a limitação da análise baseada unicamente na intuição humana (BRAGA, 2005), através do uso de tecnologias de reconhecimento de padrões, da estatística, matemática e da inteligência artificial (BERNARDI, 2010). De acordo com a definição de Larose (LAROSE, 2005), a mineração de dados é responsável pela procura por padrões, análise de variáveis, aplicação de regras associativas e métodos estatísticos sobre um grande conjunto de informações, podendo ser organizada em seis categorias, sendo elas: tarefas de descrição, tarefas de estimativa, tarefas de predição, de classificação, de aglomeração e de associação.

As fontes de dados nas quais será aplicada a mineração de dados devem ser compatíveis com as técnicas a serem aplicadas, ou seja, estarem previamente preparadas e corretamente populadas visando garantir a integridade e acuracidade dos dados armazenados, contribuindo assim para o sucesso do processo de mineração de dados (BERNARDI, 2010).

2.1.1 Processo de Mineração de Dados

Alguns autores (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996) entendem que a mineração de dados é parte de um processo maior o qual é denominado KDD (*Knowledge Discovery in Databases* - Descoberta do Conhecimento em Bases de Dados). Nesse contexto, a mineração de dados é vista como uma etapa particular do processo KDD a qual consiste na aplicação de algoritmos específicos para a extração de padrões a partir do conjunto de dados.As etapas de preparação, seleção e limpeza dos dados, incorporação de conhecimento prévio e interpretação dos resultados da mineração consistiriam no restante do processo de descoberta de conhecimento em bases de dados.

Em (WILLIAMS; HUANG, 1996) o processo de descoberta de conhecimento em bases de dados é organizado em quatro estágios que encapsulam as etapas propostas em (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996), conforme pode ser visto na figura 2.1. O primeiro estágio proposto consiste em extrair da base de dados um conjunto de dados que será utilizado nos demais estágios do processo. No pré-processamento é feita a organização e tratamento dos dados, eliminando dados incompletos ou sem consistência. O terceiro estágio é a mineração de dados onde são escolhidas e aplicadas as técnicas de mineração de dados apropriadas ao problema em questão. No estágio de pós-processamento os resultados da etapa de mineração são interpretados e analisados.

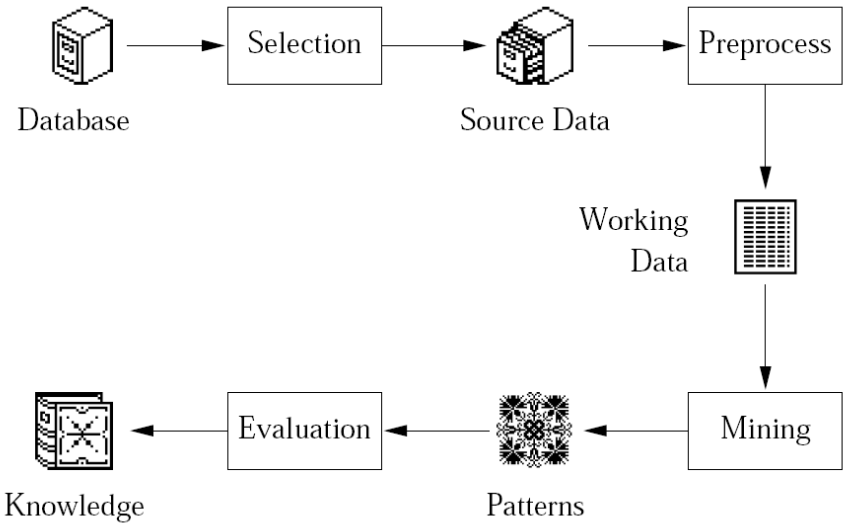

Figura 2.1: Organização Processo de Descoberta do Conhecimento em Bases de Dados proposta por (WILLIAMS; HUANG, 1996)

2.1.2 Técnicas de Mineração de Dados

Devido à diversidade das aplicações, determinadas técnicas são mais adequadas a um domínio de aplicação específico do que outras (FÉLIX, 2002).

As técnicas de mineração de dados mais conhecidas são:

- Agrupamento;
- Classificação;
- Associação;
- Regressão;

2.1.2.1 Agrupamento (Clustering)

A técnica de agrupamento, também identificada pelo termo em inglês *clustering*, consiste na identificação e na partição do conjunto inicial de dados heterogêneo em grupos ou em subconjuntos distintos. Não são especificadas classes pré-definidas. Os grupos ou subconjuntos são identificados por meio de análise de similaridade entre seus atributos. De um grupo heterogêneo são extraídos vários subgrupos mais homogêneos. Essa técnica geralmente é aplicada antes de alguma outra técnica de mineração, por exemplo a técnica de classificação (subseção 2.1.2.2), podendo ser utilizada como etapa de preparação de dados para a(s) técnica(s) a ser(em) aplicadas a seguir (BERNARDI, 2010;

CASTANHEIRA, 2008).

Essa técnica pode ser aplicada, por exemplo, para propósitos de auditoria segmentando o comportamento financeiro em categorias positivas ou suspeitas. Outra aplicação é o agrupamento de expressões genéticas, onde grandes quantidades de genes podem apresentar comportamento semelhante (LAROSE, 2005).

Os algoritmos *Cobweb*, *Expectation-Maximization*, *FarthestFirst* e *k-Means* são exemplos de algoritmos que implementam a técnica de agrupamento.

2.1.2.2 Classificação

A técnica de classificação consiste no mapeamento dos registros de um conjunto de dados de entrada em um número determinado de classes. Após o mapeamento, cada registro passa a pertencer a uma classe, dentre o conjunto de classes pré-definidas. Essa técnica busca encontrar correlações entre uma classe e os atributos do registro que está sendo analisado. Dessa forma, a técnica permite inferir a qual das classes um novo registro deve pertencer ou então fornecer o valor dos atributos para construir um novo registro, pois permite obter um modelo a partir de um conjunto de exemplos fornecido. A técnica de classificação geralmente é aplicada após a técnica de agrupamento ou *clustering* (subseção 2.1.2.1).

A classificação pode ser aplicada para diagnosticar se uma determinada doença está presente ou ainda na colocação de um novo aluno em uma classe particular já existente (LAROSE, 2005).

Os algoritmos de árvore de decisão, vizinho mais próximo, *NaiveBayes* e de redes neurais são exemplos de algoritmos que implementam a técnica de classificação.

2.1.2.3 Associação

A técnica de associação consiste em encontrar relações associativas entre os atributos dos registros presentes em um conjunto de dados, ou seja a presença de algum(ns) atributo(s) implica na presença de outro(s) no mesmo registro. A associação busca encontrar tendências que permitam entender os padrões de ocorrência dos atributos nos dados, permitindo inferir em valores percentuais quanto à ocorrência de um atributo está atrelada a ocorrência de um outro (CASTANHEIRA, 2008).

Essa técnica pode ser aplicada, por exemplo, para descobrir quais os itens de um supermercado são comprados em conjunto e quais nunca são comprados juntos (LAROSE,

2005).

Os algoritmos *Apriori*, *FP-Growth* e *FilteredAssociator* são exemplos de algoritmos que implementam a técnica de associação.

2.1.2.4 Regressão

A técnica de regressão consiste em classificar os registros de um conjunto de dados em valores contínuos, diferentemente da técnica de classificação (subseção 2.1.2.2) que utiliza valores discretos (CASTANHEIRA, 2008). A análise de regressão é o processo de determinar o quanto um atributo x está relacionado com um ou mais atributos x_1, x_2, ..., x_k (MAIMON; ROKACH, 2005). A regressão possibilita que seja feita a predição de um valor desconhecido y (variável dependente) com base nos valores de outros atributos (variáveis independentes). Uma aplicação dessa técnica é a predição do valor de um imóvel, que é o resultado de outras variáveis como a metragem quadrada do imóvel, o tamanho do lote, o número de cômodos, entre outros.

2.2 Mineração de Dados Distribuída

As ferramentas e técnicas tradicionais de mineração de dados são desenvolvidas para a execução centralizada em uma máquina, utilizando aplicações sequenciais sobre um conjunto de dados pequeno ou médio (WEGENER et al., 2009). A manipulação de grandes quantidades de dados, a heterogeneidade desses e a complexidade dos algoritmos fazem com que as técnicas tradicionais encontrem dificuldades ao serem aplicadas às fontes de dados atuais. Tamanho das bases de dados e custos de processamento são alguns pontos que elevam a complexidade das tarefas de mineração provocando grande consumo de memória e elevando exponencialmente o tempo de execução.

A utilização da mineração de dados de forma distribuída, onde os dados e os recursos podem estar distribuídos geograficamente, é uma maneira de resolver ou pelo menos minimizar estes problemas. Sistemas distribuídos de mineração de dados permitem o uso eficiente de múltiplos processadores e bases de dados, acelerando assim o processo de mineração de dados (PÉREZ et al., 2007).

Conforme visto em (WEGENER et al., 2009), há uma notável necessidade de mineração de dados distribuída e alto desempenho para acelerar os processos de mineração de dados assim como possibilitar aplicações com grandes volumes de dados. Segundo (FU,

2001), existem duas razões principais para que a mineração de dados de dados seja feita de forma distribuída, a primeira está relacionada ao custo de processamento de grandes conjuntos de dados enquanto a segunda refere-se à distribuição geográfica dos repositórios de dados.

Existem formas distintas para execução de mineração de dados de forma paralela. Uma delas é a utilização de paralelismo de algoritmo, ou seja, o algoritmo de mineração de dados é desenvolvido de forma que possa executar múltiplas operações simultaneamente. Isso permite o uso mais eficiente dos recursos de sistemas multiprocessados. Entretanto a paralelização de certas técnicas de mineração de dados não é trivial, por isso, são poucos os algoritmos paralelos de mineração de dados existentes. Outra abordagem é a utilização de paralelismo de tarefas, onde mais de uma etapa do processo de mineração de dados é colocada em execução simultaneamente. Dessa maneira pode-se executar dois algoritmos diferentes ao mesmo tempo, ou ainda, realizar em múltiplas execuções o teste de uma tarefa de mineração treinada previamente.

2.3 Ambiente de Mineração de Dados WEKA

O WEKA (Waikato Environment for Knowledge Analysis) (University of Waikato, 2010a) é um pacote de software que agrega diferentes algoritmos e técnicas para mineração de dados, sendo uma das ferramentas mais populares na área de mineração de dados. Foi inicialmente desenvolvido na University of Waikato, na Nova Zelândia, sendo atualmente distribuído como software livre com a licença GPL. A versão mais recente, conhecida como WEKA 3, é totalmente baseada em Java, sendo utilizada em diferentes áreas incluindo propósitos educacionais e de pesquisa.

O ambiente WEKA provê ferramentas para pré-processamento de dados, agrupamento ou *clustering*, classificação, regressão, visualização e regras de associação de dados. O WEKA permite que os algoritmos possam ser aplicados diretamente a determinado conjunto de dados através da interface gráfica de usuário (GUI) fornecida ou então invocados por meio de código Java, através da API disponibilizada. Os conjuntos de dados podem ser acessados diretamente de um banco de dados, utilizando JDBC, ou através arquivos no formato ARFF.

2.4 Ferramentas baseadas no ambiente WEKA

O ambiente WEKA, por ser desenvolvido em Java, facilita a disponibilidade de ferramentas de mineração de dados independentemente da plataforma do computador. Trata-se de um ambiente relativamente extensível, podendo ser visto como um conjunto de pacotes Java. Dessa forma, novas funcionalidades ou novos algoritmos e técnicas de mineração de dados podem ser adicionados ao ambiente (WITTEN et al., 1999).

Devido a essa abordagem utilizada no desenvolvimento do WEKA, existem diversas ferramentas para mineração de dados que são desenvolvidas com base no código do WEKA. Essas ferramentas fazem uso de alguns algoritmos disponibilizados no ambiente original e até mesmo de algumas partes da interface gráfica. As ferramentas WEKA4WS (2.4.1), WEKA Parallel (2.4.2) e Grid WEKA (2.4.3) são exemplos de extensões do WEKA que implementam algumas soluções para distribuição e/ou paralelismo possibilitando que a mineração de dados seja feita de forma distribuída.

2.4.1 WEKA4WS

O Weka4WS é um *framework* desenvolvido na University of Calabria, estendendo o WEKA (seção 2.3) para oferecer suporte à mineração de dados distribuída em ambientes de grades computacionais. Weka4WS possibilita a execução local ou remota de todos os algoritmos de mineração disponibilizados no ambiente WEKA. Cada algoritmo é disponibilizado como um *Web Service* utilizando a biblioteca Java WSRF do Globus Toolkit. O *middleware* Globus Toolkit fornece uma estrutura para grades computacionais, realizando o gerenciamento dos serviços disponíveis bem como oferecendo suporte a recursos de segurança. A execução remota das tarefas de mineração de dados em grade permite explorar a distribuição dos dados e incrementar o desempenho da aplicação (TALIA; TRUNFIO; VERTA, 2005).

2.4.2 WEKA Parallel

O Weka Parallel é uma modificação para o ambiente Weka, criada com o objetivo de permitir a execução paralela da técnica de validação cruzada de forma mais rápida, através da utilização de múltiplas máquinas. A técnica de validação cruzada é utilizada para avaliar o sucesso de outras técnicas de mineração de dados como agrupamento, classificação, regressão e associação. Além disso, é uma técnica inerentemente paralelizável pois

permite que os dados sejam quebrados em subgrupos, cada um a ser processado por uma máquina (CELIS; MUSICANT, 2002).

2.4.3 Grid WEKA

O Grid Weka é uma implementação modificada do WEKA, desenvolvido no departamento de Ciência da Computação da University College Dublin na Irlanda, para permitir a utilização distribuída, possibilitando a utilização dos recursos de vários computadores ao executar as funções (ZUO; KHOUSSAINOV; KUSHMERICK, 2004). As funções podem ser executadas de forma distribuída, explorando o paralelismo entre diferentes máquinas e reunindo os resultados parciais para formar um resultado final. O Grid Weka executa em uma grade *ad-hoc*, usando soluções que não tratam todos os aspectos característicos de um ambiente de grade, como gerenciamento de serviços disponíveis, composição e suporte a padrões de segurança.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas realizadas no desenvolvimento deste trabalho. A seção 3.1 apresenta o ambiente computacional utilizado nesta etapa do trabalho. A seção 3.2 aborda a seleção da ferramenta de mineração de dados utilizada no trabalho. A próxima seção (3.3) traz os procedimentos para instalação e configuração da ferramenta selecionada. Na seção 3.4 são abordados detalhes, tais como formato, fonte, formatação, conversão e partição dos dados utilizados no trabalho. Por fim, a seção 3.5 apresenta os testes preliminares realizados para verificar o funcionamento da ferramenta escolhida.

3.1 Ambiente Computacional

Ambientes computacionais com arquiteturas diversas, como um ambiente de *cluster* formado por máquinas monoprocessadas semelhantes ou um ambiente heterogêneo que combina uma arquitetura multiprocessada com distribuição, influem no comportamento das ferramentas de mineração de dados distribuída.

Nesta etapa do trabalho, utilizou-se uma plataforma do tipo Linux/Unix, onde o acesso aos recursos administrativos de cada máquina foi limitado. Não havia disponibilidade de *middleware* ou de infra-estruturas que implementem abstrações quanto a atividades como submissão e controle de aplicações, movimentação de dados, segurança e descoberta de recursos. O sistema de arquivos do tipo *Network File System (NFS)* estava disponível em algumas máquinas e o acesso remoto via protocolo de rede *SSH* foi possível em todos os equipamentos.

3.2 Seleção da ferramenta

Dentre as ferramentas estudadas, WEKA4WS, WEKA Parallel e Grid WEKA, o WEKA4WS é o único que disponibiliza a execução remota de todos os algoritmos dispo-

nibilizados pelo WEKA e de uma estrutura de grade computacional que é baseada sobre o *middleware* Globus Toolkit. No entanto, o *middleware* Globus Toolkit não está disponível nos equipamentos, o que inviabilizou a utilização desta ferramenta no ambiente computacional distribuído.

As ferramentas WEKA Parallel e Grid WEKA não fazem uso de um *middleware* específico e assumem que a organização dos nós é do tipo *ad-hoc*, ou seja, os nós são organizados para atender um propósito específico, neste caso, mineração de dados distribuída. Porém, o Grid WEKA disponibiliza um número maior de algoritmos para execução remota. Na ferramenta WEKA-Parallel apenas uma técnica de validação pode ser executada em paralelo. Dessa forma, foi selecionada a ferramenta Grid WEKA como ambiente de mineração de dados a ser utilizado.

3.3 Instalação e Configuração do Ambiente de Mineração de Dados

As subseções a seguir descrevem os procedimentos básicos para a configuração do ambiente Grid WEKA para execução remota e paralela. Serão descritas as características específicas para a configuração dos nós cliente e servidor(es).

3.3.0.1 Instalação e Requisitos

O Grid WEKA foi desenvolvido sobre a versão 3.4.2 do WEKA. Por ser escrito em linguagem de programação Java, faz-se necessária a presença de uma *"Java Virtual Machine"* (JVM) no ambiente onde o Grid WEKA será executado. A versão requerida para a JVM é a 1.4.2 ou superior. A versão instalada nos computadores é a 1.6.0.

3.3.0.2 Configuração do servidor

Para execução de um nó como servidor do Grid WEKA, é utilizada a classe *DistributedServer*, especificando a porta na qual o nó servidor irá receber as requisições do(s) cliente(s), através do comando *"java weka.core.DistributedServer <Porta>"*. É necessário que o *classpath* do Java contenha o caminho para o arquivo *weka.jar*, caso contrário a opção *"-classpath <caminho para weka.jar>"* deve ser informada quando o servidor for colocado para executar.

Em caso de máquinas multiprocessadas, vários nós servidores podem ser colocados em execução paralelamente, informando números de porta subsequentes, e cada nó em um diretório diferente para evitar conflitos no acesso aos dados.

3.3.0.3 Configuração do cliente

A execução de um nó cliente faz uso de um arquivo de configuração nomeado *.weka-parallel*, localizado no diretório padrão do usuário. A primeira linha do arquivo contém o número da porta inicial onde os servidores estão aguardando requisições, a segunda linha informa o *host* do(s) nó(s) servidor(es), a terceira linha contém o número de nós servidores em execução no *host* informado, a linha seguinte contém a quantidade máxima de memória do *host* em *megabytes*. A descrição de outros *host* servidores é feita de forma semelhante, informando respectivamente *host*, número de nós e quantidade máxima de memória. Um exemplo desse arquivo de configuração é apresentado na figura 3.1. A execução de forma remota deve ser selecionada através do parâmetro *"-a"*, caso contrário o Grid WEKA executa como uma versão tradicional do WEKA (centralizada). A utilização de vários servidores paralelos é habilitada através do parâmetro *"-C"* seguido pelo número máximo de servidores a serem utilizados. A figura 3.2 apresenta uma provável organização dos nós do Grid WEKA em acordo com a configuração mostrada na figura 3.1.

```
1 PORT=9001
2 maq01.inf.ufsm.br
3 2
4 4000
5 maq02.inf.ufsm.br
6 1
7 1000
8 maq03.inf.ufsm.br
9 1
10 1000
```

Figura 3.1: Um exemplo do arquivo de configuração *.weka-parallel* para um nó cliente do Grid WEKA

3.4 Preparação dos dados

3.4.1 Attribute-Relation File Format (ARFF)

O formato de arquivo ARFF é um arquivo do tipo texto com codificação ASCII que descreve uma lista de instâncias de um respectivo conjunto de atributos. É o formato padrão de entrada de dados do WEKA, e por consequência, das demais ferramentas nele baseadas. Os arquivos são compostos de duas seções: a primeira delas contém o cabeçalho (*header*) e a segunda contém os dados (*data*) propriamente ditos. As figuras 3.3 e 3.4

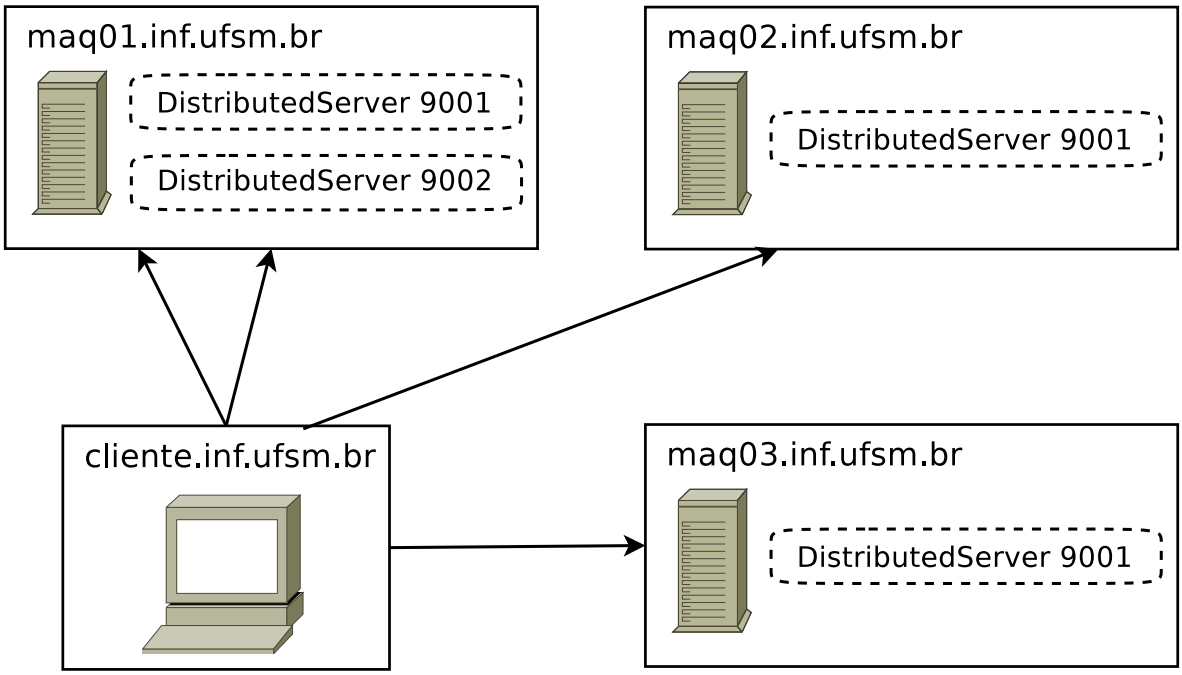

Figura 3.2: Organização dos nós do Grid WEKA

são exemplos das seções de *header* e *data* respectivamente.

No *header* a declaração *"@relation"* descreve o nome da relação, e é definida na primeira linha do arquivo. A declaração dos atributos *"@attribute"* contém o nome do atributo seguido do tipo de dado, que pode ser numérico, *string*, data ou uma lista prédefinida. Nesse caso os campos possíveis para esse atributo devem ser listados entre chaves e separados por vírgula.

```
 9  @relation car
10
11  @attribute buying {vhigh,high,med,low}
12  @attribute maint {vhigh,high,med,low}
13  @attribute doors NUMERIC
14  @attribute class {unacc,acc,good,vgood}
```

Figura 3.3: Seção *Header* de um arquivo ARFF

A seção de dados é iniciada pela declaração *"@data"*, e seguida pelos valores dos dados. Cada instância ocupa apenas uma linha. O s valores dos atributos são separados por vírgula e devem ser listados na mesma ordem em que foram declarados na seção *header*.

```
21 @data
22 vhigh,vhigh,2,unacc
23 vhigh,high,2,unacc
24 high,med,5,unacc
25 low,high,3,acc
26 low,high,3,vgood
```

Figura 3.4: Seção *Data* de um arquivo ARFF

3.4.2 Fonte de dados

Os dados utilizados neste trabalho são provenientes do sistema de informações acadêmicas da UFSM (SIE). Esses dados foram previamente selecionados pelo Centro de Processamento de Dados (CPD) da UFSM. O conjunto de dados fornecido para análise foi separado em seis grupos que contêm dados em formato *Comma-Separated Values* (CSV). As chaves primárias de cada grupo foram destacadas para que seja possível fazer o cruzamento entre os registros de grupos diferentes. As colunas que contêm identificadores pessoais, tais como números de documentos de identidade ou matricula foram cifradas com objetivo de preservar a confidencialidade das informações.

Os dados em formato CSV foram agrupados, reformatados, e posteriormente convertidos para o formato ARFF (descrito na subseção 3.4.1) a fim de que pudessem ser utilizados na ferramenta Grid WEKA. Para isso foram utilizadas classes disponibilizadas pela API do WEKA em sua versão mais recente (versão de desenvolvimento): 3.7.2 (University of Waikato, 2010b). Os arquivos fornecidos contêm, na primeira linha, os nomes dos atributos e nas demais os valores destes atributos.

3.4.2.1 Formatação dos arquivos de entrada

Para viabilizar o uso da API do WEKA para conversão de CSV para ARFF foi necessário corrigir a formatação de alguns arquivos da fonte de dados. A API requisita que: campos em branco sejam representados pelo símbolo "?", tabulações não estejam presentes, símbolos "%"e "'"não sejam utilizados, não existam linhas em branco e que todas as linhas de dados tenham o mesmo número de campos que a primeira linha (considerando que a primeira linha contém o nome dos campos).

Foram implementados, em linguagem Java, códigos que corrigem os arquivos, eliminando linhas em branco, juntando linhas quebradas, inserindo "?" nos campos vazios e substituindo os caracteres não permitidos. Os separadores do arquivo CSV (";") foram

substituídos por "¬". As figuras 3.5 e 3.6 apresentam partes dos códigos implementados para recuperação de linhas quebradas e para substituição dos caracteres. A figura 3.7 ilustra um exemplo de uma linha de um arquivo antes e após a correção.

```
if (s.split("¬").length < numColunas) { //Linha lida menor que o número de colunas
    while (entrada.ready()) {
        String nova = entrada.readLine();
        if (nova.trim().isEmpty()) { //Salta linhas vazias
            continue;
        }
        s = s + " " + nova; //tenta formar uma linha completa juntando a atual com a próxima
        if (s.startsWith("¬"))
            s = "?" + s;
        if (s.endsWith("¬"))
            s = s + "?";
        if (s.split("¬").length != numColunas) { //Se não conseguiu, descarta a linha
            if (s.split("¬").length < numColunas)
                continue;
            System.out.println("Não conseguiu recuperar: " + s);
            sair = true;
            break;
        }
        System.out.println("Linha Recuperada: " + s);
        sair = false;
        break;
    }
    if (sair)
        continue;
} else if (s.split("¬").length > numColunas) {
    System.out.println("Erro, Linha maior: " + s);
    continue;
}
```

Figura 3.5: Parte do código para recuperação de linhas quebradas

```
s = s.replaceAll("\"", "");
s = s.replaceAll("¬¬¬", "¬?¬?¬");
s = s.replaceAll("¬¬", "¬?¬");
s = s.replaceAll("%", " por cento");
s = s.replaceAll("\t", " "); //remove tab
s = s.replaceAll("[\\[\\]]", ""); //expressão regular que remove [ ]
s = s.replaceAll("'", "");

saida.write(s + "\n"); //escreve nova linha na saída
```

Figura 3.6: Parte do código para substituição dos caracteres

3.4.2.2 Conversão para ARFF

Concluídas as alterações na formatação dos arquivos CSV, foi possível convertê-los para ARFF utilizando as classes da API do WEKA 3.7.2. Foram utilizadas as classes *CSVLoader* e *ArffSaver* para manipular os arquivos de entrada e saída, respectivamente. A classe *Instances* foi utilizada para recuperar o conjunto de dados do CSV e inseri-los no ARFF. O código implementado para realizar a conversão é apresentado na figura 3.8.

3.4.2.3 *Partição dos arquivos ARFF*

Os arquivos completos são relativamente grandes, o que torna interessante dividí-los em arquivos menores contendo frações do arquivo original. Dessa forma, é possível utilizar um arquivo de entrada para uma etapa de mineração de dados e posteriormente utilizar outro maior. Buscando evitar concentração e perdas na variabilidade dos dados, a partição não foi feita separando os *n* primeiros registros, mas sim utilizando saltos entre uma leitura e outra. Ou seja, quando se aplica uma partição de 1/5 sobre o arquivo original, a cada cinco linhas do arquivo de entrada uma delas é copiada para o arquivo de saída. Os comentários e o cabeçalho do arquivo de entrada são copiados integralmente para o arquivo de saída. A figura 3.9 apresenta o código para partição dos arquivos ARFF. Um exemplo da partição (1/4) de um arquivo ARFF é apresentado na figura 3.10.

```
¬"14° SEMANA ACADÊMICA DA MEDICINA"¬"[31/08/2009]"¬"[04/09/2009]"¬"Extensão"¬171538¬"Participante"¬2¬"A"
¬"ASSOCIAÇÃO ENTRE RINITE, RESPIRAÇÃO BUCAL E BRUXISMO DO SONO


"¬"[10/08/2008]"¬"[24/08/2009]"¬"Pesquisa"¬171588¬"Participante"¬2¬"A"
```

(a) Antes da Correção

```
?¬14° SEMANA ACADÊMICA DA MEDICINA¬31/08/2009¬04/09/2009¬Extensão¬171538¬Participante¬2¬A
?¬ASSOCIAÇÃO ENTRE RINITE, RESPIRAÇÃO BUCAL E BRUXISMO DO SONO¬10/08/2008¬24/08/2009¬Pesquisa¬171588¬Participante¬2¬A
```

(b) Após a Correção

Figura 3.7: Exemplo de correção da formatação de uma linha

```
Instances dados = null;
CSVLoader cl = new CSVLoader(); //Abrir CSV
cl.setFieldSeparator("¬");
File fCsv = new File(this.getNomeCSV());
try {
    cl.setSource(fCsv);
    dados = cl.getDataSet(); //Recupera instâncias do CSV
} catch (IOException ex) {
    System.err.println("Erro ao abrir arquivo CSV.");
    Logger.getLogger(CsvArff.class.getName()).log(Level.SEVERE, null, ex);
}
ArffSaver asDestino = new ArffSaver(); //Abrir ARFF
File fArff = new File(this.getNomeARFF());
dados.setRelationName(fCsv.getName().substring(0, fCsv.getName().indexOf(".")));
asDestino.setInstances(dados); //Insere instâncias no ARFF
try {
    asDestino.setFile(fArff);
    asDestino.writeBatch();
} catch (IOException ex) {
    System.err.println("Erro ao abrir arquivo ARFF.");
    Logger.getLogger(CsvArff.class.getName()).log(Level.SEVERE, null, ex);
}
```

Figura 3.8: Código para conversão CSV em ARFF

```
while (entrada.ready()) { //Copia cabeçalho
    tmp = entrada.readLine();
    if (tmp.startsWith("%")){ // Comentários
        saida.write(tmp + "\n");
    }else if(tmp.contains("@data") ){
        saida.write(tmp + "\n");
        break; //Fim do cabeçalho
    }else{
        saida.write(tmp + "\n");
    }
}
//Parte o Arquivo
List<String> dados = new ArrayList<String>();
while (entrada.ready()) {
    dados.add(entrada.readLine() + "\n");
}
//Lê as linha de dados, gravando a cada n linhas uma linha na saída
for (int i = 0 ; i < dados.size() ; i+=n){
    saida.write(dados.get(i));
}
```

Figura 3.9: Código para partição de um ARFF

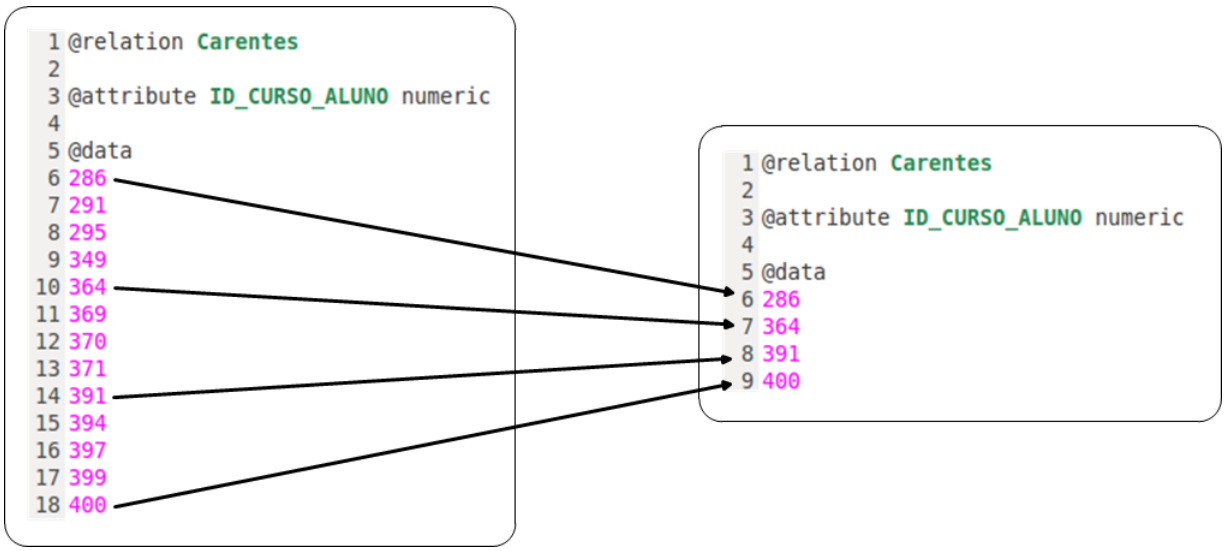

Figura 3.10: Exemplo da partição (1/4) de um ARFF

3.5 Testes preliminares

Após a configuração do Grid WEKA foram executados testes preliminares com objetivo de verificar o funcionamento da ferramenta. Utilizou-se o algoritmo de agrupamento *Simple K-means* sobre dois conjuntos de dados de amostra, um deles obtido do *UCI Machine Learning Repository* (University of California, Irvine, 2010) e outro do conjunto de exemplos fornecido com o WEKA. Após a execução, foi detectado que o algoritmo testado não executa de forma distribuída.

Com isso, buscou-se identificar quais tarefas de mineração são realizadas de maneira não centralizada pelo Grid WEKA. Através da análise do código fonte da classe *Dis-*

tributedServer (onde são tratadas as requisições feitas a um servidor do Grid WEKA) identificou-se que as tarefas previstas são: treinamento remoto do modelo, validação cruzada e teste de um modelo treinado anteriormente. Isso foi confirmado através da tentativa de execução do Grid WEKA sem o arquivo de configuração *.weka-parallel*. Nas tentativas envolvendo as tarefas citadas acima foi reportado erro devido à falta do arquivo de configuração, entretanto, as outras tarefas de mineração foram executadas normalmente, o que demonstrou que são executadas usando o código da versão centralizada do WEKA, não fazendo uso das funcionalidades do Grid WEKA.

Além disso, também identificou-se que a tarefa de validação cruzada é realizada juntamente com a tarefa de treinamento remoto do modelo. Quando é feito o treinamento do modelo de forma distribuída utilizando um servidor, a validação cruzada é feita paralelamente entre este servidor e o computador cliente que fez a requisição. Quando da utilização de dois ou mais servidores, o treinamento do modelo é feito remotamente em um desses servidores e a validação cruzada é feita paralelamente nos servidores e no cliente.

4 ESTUDO DE CASO

Neste capítulo é apresentado um estudo de caso da ferramenta Grid WEKA para mineração de dados paralela e distribuída. Este estudo de caso é composto por quatro casos de teste, subdivididos em duas partes, cada uma delas utilizando um dos algoritmos de classificação que serão apresentados na seção 4.2. A seção 4.1 apresenta as características dos ambientes computacionais utilizados. Os conjuntos de dados utilizados nos casos de teste são apresentados na seção 4.3.

4.1 Ambientes de Testes

Na realização dos testes foram utilizados dois ambientes computacionais distintos para a análise do comportamento da ferramenta Grid WEKA, através de variações no ambiente e nos conjuntos de dados utilizados. Os dois ambientes são descritos a seguir:

4.1.1 Ambiente 1

Este ambiente computacional é composto pelo *cluster* "Ombrófila" do Laboratório de Alto Desempenho da Pontifícia Universidade Católica do Rio Grande do Sul (LAD-PUCRS). É composto por computadores HP equipados com processador Intel Pentium III, com 1GHz e Intel Pentium IV com 1.5GHz, com 256Mb de memória RAM e sistema operacional Ubuntu Linux na versão 8.04.2. A comunicação entre os computadores é realizada por meio de rede *Fast-Ethernet*. O *cluster* é gerenciado pelo sistema de gerenciamento de *cluster* CRONO. Nesse ambiente um dos computadores equipado com processador Pentium IV foi utilizando para a execução do cliente do Grid WEKA, os demais foram utilizados como servidores.

4.1.2 Ambiente 2

Este ambiente computacional é composto pelo computador "Haroldo" com 8 núcleos e pelo computador "Itautec" com 1 núcleo de processamento. A máquina "Haroldo" é equipada com dois processadores Intel Xeon E335 com 2Ghz, cada um destes com quatro núcleos, e 4Gb de memória RAM. A máquina "Itautec" é equipada com um processador Intel Pentium IV com 3GHz, 512Mb de memória RAM e sistema operacional Arch Linux. A comunicação entre estes computadores é realizada por meio de rede *Fast-Ethernet*. Neste ambiente, o computador "Itautec" foi utilizado para execução do cliente do Grid WEKA e a máquina "Haroldo" para a execução dos servidores. Na máquina "Haroldo", por ser multiprocessada, foram colocadas em execução várias instâncias servidoras do Grid WEKA. Nos testes a seguir cada instância servidora do Grid WEKA nesta máquina será considerada como um servidor.

4.2 Algoritmos Utilizados

Foram selecionados dois algoritmos de classificação para esse estudo de caso da ferramenta Grid WEKA. A escolha desses algoritmos foi feita com base em testes na versão centralizada do WEKA observando o comportamento de alguns dos algoritmos de classificação disponíveis. Algoritmos de redes neurais como o *MultilayerPerceptron* rapidamente estouraram a memória disponível quando aplicados aos conjuntos de dados disponíveis (3.4.2). Os algoritmos de classificação Bayesiana (*NaiveBayes*) apresentaram dados pouco consistentes como resultado de saída da mineração.

Escolheu-se os algoritmos *J48*, um algoritmo do tipo árvore de decisão que é uma implementação *open source* do algoritmo C4.5, e o algoritmo *KStar* (CLEARY; TRIGG, 1995), um algoritmo baseado em exemplos que utiliza uma função de distância baseada na medida da entropia. Observou-se que o algoritmo *J48*, da mesma forma que outros algoritmos de árvore de decisão, consome maior tempo de execução na etapa de treinamento do modelo do que na etapa de teste do modelo treinado. Já o algoritmo *KStar* possui comportamento diferente, consumindo a maior parte do tempo de execução na etapa de teste do modelo treinado.

COD_CURSO
NOME_CURSO
NIVEL_CURSO
TIPO_CURSO
MODALIDADE_CURSO
CLASSIFICACAO_CURSO
NUM_VERSAO
CH_CURSO
NUM_PERIODOS
AREA_CURSO
TURNO_CURSO
ID_CURSO_ALUNO
SEXO
DT_NASCIMENTO
ANO_INGRESSO
PERIODO_INGRESSO
FORMA_INGRESSO
ANO_EVASAO
PERIODO_EVASAO
FORMA_EVASAO
DT_CONCLUSAO
DT_COLACAO
NACIONALIDADE
ESTADO_CIVIL
NATURALIDADE
UF
ETNIA
DEFICIENCIA
ID_1
ID_2
ID_CANDIDATO
ID_INSCRICAO
ID_PESSOA
TIPO_ESCOLA_ENS_MEDIO

(a) Alunos

ANO
CONCURSO
SITUACAO
COD_OPCAO
ESTADO_CIVIL
DEFICIENCIA
SEXO
NACIONALIDADE
ETNIA
NATURALIDADE
UF
COD_COTA
OPCAO_LINGUA
DT_NASCIMENTO
ID_1
ID_2
NUM_INSCRICAO
ORDEM_COM_COTA
ORDEM_SEM_COTA
ID_INSCRICAO
ID_CANDIDATO

(b) Candidatos

Figura 4.1: Atributos dos Conjuntos de Dados

4.3 Dados Utilizados

Dos dados descritos na seção 3.4.2, utilizaram-se os que são referentes a alunos e a candidatos. No conjunto referente a alunos utilizou-se como campo de classificação o atributo "CLASSIFICACAO_CURSO", que pode ser do tipo "Presencial"ou "EAD". A partir dos demais atributos presentes nos registros, os algoritmos treinam um modelo que permite identificar qual o provável valor do atributo "CLASSIFICACAO_CURSO" para cada um dos registros presentes. No conjunto referente a candidatos utilizou-se como campo de classificação o atributo "COD_COTA", que pode ser "A", "B", "C", "D"ou "E". Os atributos dos conjuntos de dados são apresentados na figura 4.1.

Para a execução de alguns dos testes, os arquivos que contêm os dados foram parti-

Tabela 4.1: Subconjuntos de Dados Gerados para Alunos

	Proporção em relação ao original	Nº de instâncias
Alunos	1	28519
Alunos5	1/5	5704
Alunos10	1/10	2852
Alunos30	1/30	951
Alunos70	1/70	408

Tabela 4.2: Subconjuntos de Dados Gerados para Candidatos

	Proporção em relação ao original	Nº de instâncias
Candidatos	1	94056
Candidatos10	1/10	9406
Candidatos70	1/70	1344
Candidatos140	1/140	672

cionados em diferentes proporções formando subconjuntos dos dados originais. Isso foi necessário pois a quantidade de memória necessária para a realização do treinamento é muito grande quando utilizados os conjuntos totais de dados. O particionado foi realizado conforme descrito em 3.4.2.3. As tabelas 4.1 e 4.2 descrevem os subconjuntos gerados e utilizados, a proporção que estes representam em relação aos arquivos originais e o número de instâncias.

4.4 Casos de Teste

Cada um dos testes realizados envolve um dos dois ambientes computacionais citados em 4.1, um dos conjuntos de dados descritos em 4.3 e um dos algoritmos citados em 4.2. O caso de teste I foi executado no ambiente 1 (4.1.1), utilizando o conjunto de dados Alunos e é dividido em duas partes: I.A utilizando o algoritmo *J48* e a parte I.B utilizando o algoritmo *KStar*. Os caso de teste II foi executado no ambiente 2 (4.1.2), utilizando o conjunto de dados Alunos e é composto por duas partes: II.A e II.B, que utilizam, respectivamente, os algoritmos *J48* e *KStar*. O caso de teste III foi executado no ambiente 2 (4.1.2), utilizando o conjunto de dados Candidatos e é composto por duas partes: III.A utilizando o algoritmo *J48* e III.B utilizando o algoritmo *KStar*. O caso de teste IV foi executado no ambiente 2 (4.1.2) e é composto pela comparação da execução da etapa de teste do modelo de classificação quando os arquivos estão ou não presentes no servidor, neste caso de teste são utilizados os modelos de classificação treinados nos casos III.A e III.B. A tabela 4.3 apresenta um resumo dos casos de teste realizados.

Tabela 4.3: Casos de Teste

Caso de Teste	Ambiente Computacional	Conjunto de Dados	Algoritmo de Classificação
I.A	Ambiente 1	Alunos	*J48*
I.B	Ambiente 1	Alunos	*KStar*
II.A	Ambiente 2	Alunos	*J48*
II.B	Ambiente 2	Alunos	*KStar*
III.A	Ambiente 2	Candidatos	*J48*
III.B	Ambiente 2	Candidatos	*KStar*
IV.A	Ambiente 2	Candidatos	*J48*
IV.B	Ambiente 2	Candidatos	*KStar*

4.4.1 Caso de Teste I.A

Este caso de teste foi realizado no ambiente 1 (4.1.1), utilizando o conjunto de dados referente a alunos e com o algoritmo de classificação *J48*.

4.4.1.1 Treinamento do Modelo de Classificação

Para o treinamento do modelo de classificação, utilizou-se o arquivo "Alunos70"com 408 instâncias, equivalente a 1/70 do conjunto de dados original, particionado conforme descrito em 3.4.2.3. A utilização de um subconjunto dos dados foi necessária devido à quantidade de memória que seria necessária para realização do treinamento no conjunto total de dados. Juntamente com o treinamento de um modelo, o Grid WEKA executa a validação cruzada sobre o modelo gerado.

A tabela 4.4 apresenta os tempos médios obtidos para a etapa de treinamento do modelo de classificação nas execuções realizadas localmente e utilizando um, dois ou três servidores remotos paralelamente para a validação cruzada. Os tempos apresentados na tabela 4.4 também estão representados graficamente na figura 4.2. A figura 4.3 mostra a árvore gerada pelo algoritmo *J48* durante o treinamento do modelo de classificação.

Tabela 4.4: Etapa de treinamento do modelo - Caso de Teste I.A

	Tempo (s)
Local	50,163s
Remoto (1)	59,632s
Remoto (2)	78,784s
Remoto (3)	93,085s

Após a execução do treinamento do modelo, foi observado que 406 das 408 instâncias do arquivo utilizado no treinamento foram classificadas corretamente utilizando o modelo gerado, o que equivale a uma taxa de acertos de 99,5%.

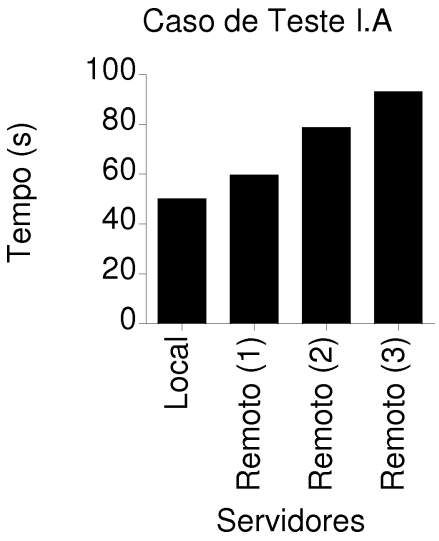

Figura 4.2: Tempos de Execução do Treinamento do Modelo - Caso de Teste I.A

```
J48 pruned tree
------------------

NUM_VERSAO <= 2006: Presencial (371.0/2.0)
NUM_VERSAO > 2006
|   TIPO_CURSO = Curso Presencial: Presencial (16.0)
|   TIPO_CURSO = Habilitacao: Presencial (4.0)
|   TIPO_CURSO = Curso EAD: EAD (13.0)
|   TIPO_CURSO = Curso Eventual: Presencial (1.0)
|   TIPO_CURSO = Curso EAD - REGESD: EAD (3.0)

Number of Leaves  :  6
Size of the tree :   8
```

Figura 4.3: Árvore gerada pelo algoritmo J48 durante a etapa de treinamento do modelo - Caso de Teste I.A

4.4.1.2 Testes do Modelo de Classificação

Na etapa de teste do modelo de classificação treinado anteriormente, utilizou-se como entrada o arquivo Alunos30 contendo 951 instâncias, equivalente a 1/30 do conjunto de dados total. A tabela 4.5 mostra os tempos de execução obtidos nesse cenário, tanto para a execução local quanto para a execução remota utilizando um, dois ou três servidores paralelamente. A figura 4.4 apresenta de forma gráfica os dados mostrados na tabela 4.5.

A taxa de acertos obtida neste teste utilizado o modelo gerado anteriormente foi de 99,57%, com 947 das 951 instâncias classificadas corretamente.

4.4.2 Caso de Teste I.B

Este caso de teste foi realizado no ambiente 1 (4.1.1), utilizando o conjunto de dados referente a alunos e com o algoritmo de classificação *KStar*.

Tabela 4.5: Etapa de teste do modelo - Caso de Teste I.A

	Tempo (s)
Local	54,534s
Remoto (1)	68,386s
Remoto (2)	70,272s
Remoto (3)	71,116s

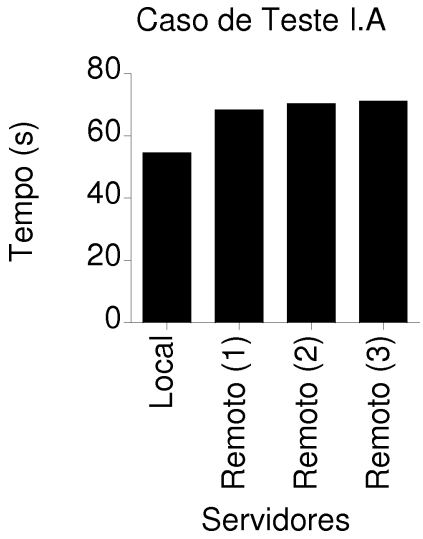

Figura 4.4: Tempos de Execução do Teste do Modelo - Caso de Teste I.A

4.4.2.1 Treinamento do Modelo de Classificação

Para o treinamento do modelo de classificação do algoritmo *KStar*, utilizou-se o arquivo Alunos70 com 408 instâncias, equivalente a 1/70 do conjunto de dados original.

Foram realizadas execuções da etapa de treinamento do modelo de classificação de forma local e de forma remota. Na forma remota utilizou-se um, dois ou três servidores para a realização da validação cruzada paralelamente. Os tempos médios obtidos nessa etapa são apresentados na tabela 4.6. Os tempos de execução obtidos também estão representados no gráfico da figura 4.5.

Tabela 4.6: Etapa de treinamento do modelo - Caso de Teste I.B

	Tempo (s)
Local	66.000s
Remoto (1)	97.393s
Remoto (2)	98.250s
Remoto (3)	111.177s

Após a execução do treinamento do modelo, foi observado que 391 das 408 instâncias do arquivo utilizado no treinamento foram classificadas corretamente utilizando o modelo

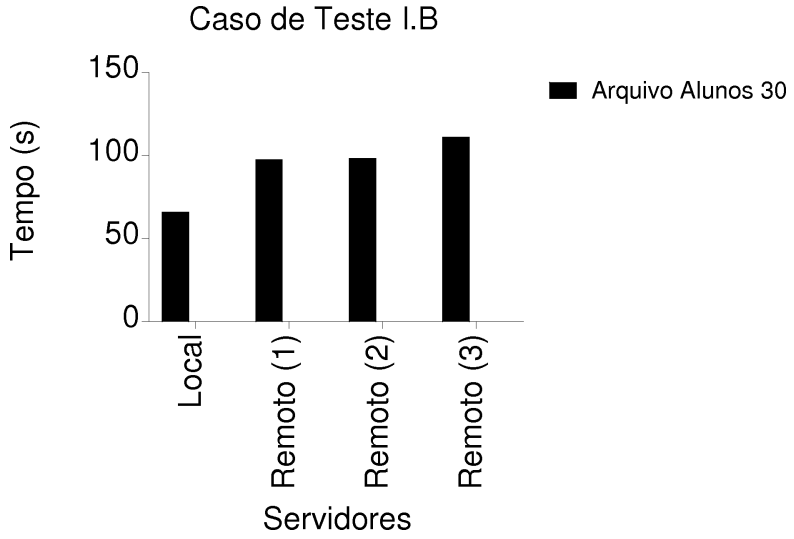

Figura 4.5: Tempos de Execução do Treinamento do Modelo - Caso de Teste I.B

gerado, o que equivale a uma taxa de acertos de 95,83%, 10 (2,45%) instâncias foram classificadas incorretamente e 7 (1.71%) não foram classificadas.

4.4.2.2 Testes do Modelo de Classificação

Na etapa de teste do modelo treinado anteriormente, utilizou-se como entradas dois arquivos. Um deles, Alunos30, contendo 951 instâncias, equivalente a 1/30 do conjunto de dados total. O outro arquivo utilizado, Alunos5, contém 5704 instâncias, equivalente a 1/5 do conjunto de dados total. Foram realizadas execuções locais e remotas utilizando um, dois ou três servidores remotos paralelamente. Os tempos de execução transcorridos para ambos os arquivos são apresentados na tabela 4.7 e no gráfico da figura 4.6.

Para o primeiro arquivo a taxa de acertos obtida neste teste utilizado o modelo gerado anteriormente foi de 98%, com 932 das 951 instâncias classificadas corretamente. No segundo arquivo, a taxa de acertos obtida pelo modelo foi de 98,1%, com 5596 das 5704 instâncias classificadas corretamente.

Tabela 4.7: Etapa de teste do modelo - Caso de Teste I.B

	Arquivo Alunos30	Arquivo Alunos5
Local	90,075s	250,190s
Remoto (1)	101,623s	248,898s
Remoto (2)	88,670s	178,275s
Remoto (3)	86,741s	145,576s

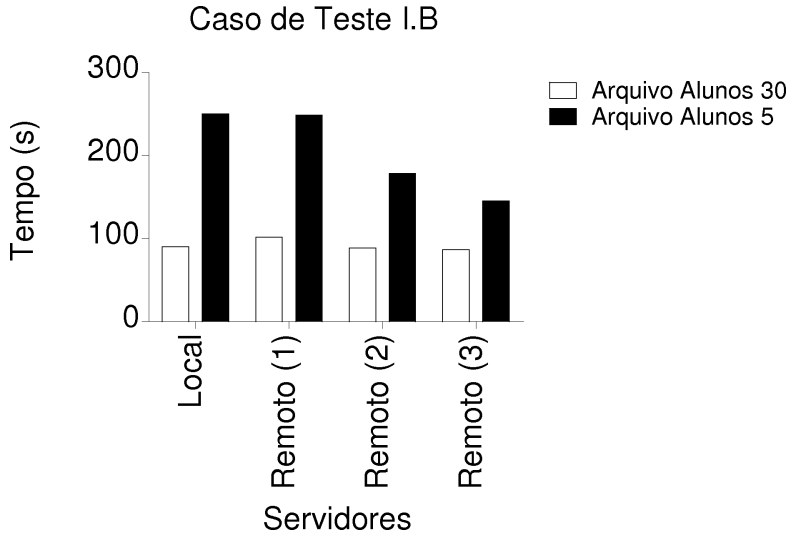

Figura 4.6: Tempos de Execução do Teste do Modelo - Caso de Teste I.B

4.4.3 Caso de Teste II.A

Este caso de teste foi realizado no ambiente 2 (4.1.2), utilizando o conjunto de dados referente a alunos e com o algoritmo de classificação *J48*.

4.4.3.1 Treinamento do Modelo de Classificação

Para a etapa de treinamento do modelo de classificação, utilizou-se o arquivo Alunos70 com 408 instâncias, equivalente a 1/70 do conjunto de dados original.

A tabela 4.8 apresenta os tempos médios obtidos para a etapa de treinamento do modelo de classificação nas execuções realizadas localmente e utilizando um, dois, três ou quatro servidores remotos paralelamente para a validação cruzada. Os tempos apresentados na tabela 4.8 também estão representados graficamente na figura 4.7.

Tabela 4.8: Etapa de treinamento do modelo - Caso de Teste II.A

	Tempo (s)
Local	17,399s
Remoto (1)	22,547s
Remoto (2)	28,266s
Remoto (3)	32,398s
Remoto (4)	33,667s

Após a execução do treinamento do modelo, o resultado da validação cruzada indicou que 406 (99,50%) das 408 instâncias do arquivo utilizado no treinamento foram classificadas corretamente e 2 (0,49%) foram classificadas incorretamente.

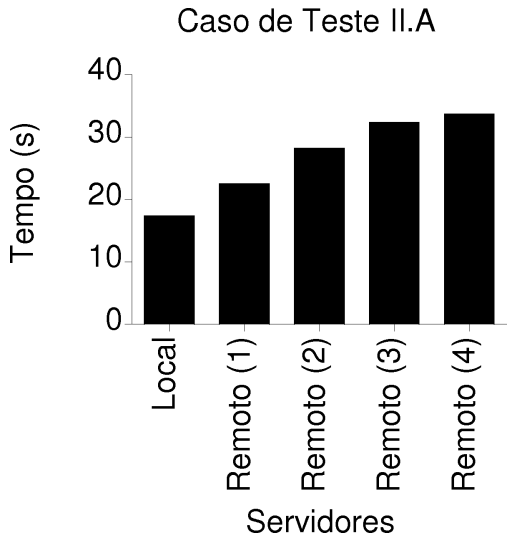

Figura 4.7: Tempos de Execução do Treinamento do Modelo - Caso de Teste II.A

4.4.3.2 Testes do Modelo de Classificação

Na execução da tarefa de teste do modelo treinado anteriormente, utilizou-se como entrada o arquivo Alunos contendo 28519 instâncias, equivalente ao conjunto de dados total. A tabela 4.9 mostra os tempos de execução obtidos nesse cenário, tanto para a execução local quanto para a execução remota utilizando um, dois, três, quatro, cinco ou seis servidores paralelamente. A figura 4.8 apresenta de forma gráfica os dados mostrados na tabela 4.9.

Tabela 4.9: Etapa de teste do modelo - Caso de Teste II.A

	Tempo (s)
Local	23,253s
Remoto (1)	74,773s
Remoto (2)	95,082s
Remoto (3)	100,293s
Remoto (4)	112,519s
Remoto (5)	133,825s
Remoto (6)	165,613s

A taxa de acertos obtida neste teste utilizado o modelo gerado anteriormente foi de 99,56%, com 28395 acertos dentre as 28519 instâncias classificadas, a taxa de classificações incorretas foi de 0,43%, equivalente a 124 instâncias.

4.4.4 Caso de Teste II.B

Este caso de teste foi realizado no ambiente 2 (4.1.2), utilizando o conjunto de dados referente a alunos e com o algoritmo de classificação *KStar*.

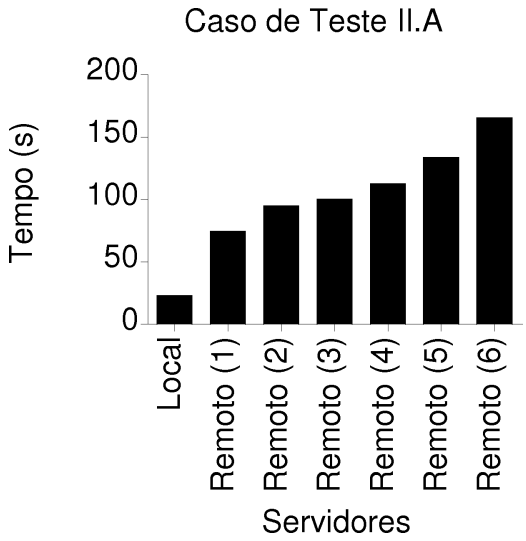

Figura 4.8: Tempos de Execução do Teste do Modelo - Caso de Teste II.A

4.4.4.1 Treinamento do Modelo de Classificação

Nesta etapa de treinamento do modelo de classificação utilizou-se o arquivo Alunos70, que possui com 408 instâncias, equivalente a 1/70 do conjunto de dados original.

Foram realizadas execuções de forma local e remota da etapa de treinamento do modelo de classificação. Na forma remota utilizou-se um, dois, três ou quatro servidores para a realização da validação cruzada paralelamente. Os tempos médios obtidos nessa etapa são apresentados na tabela 4.10. Os tempos de execução obtidos também estão representados no gráfico da figura 4.9.

Tabela 4.10: Etapa de treinamento do modelo - Caso de Teste II.B

	Tempo (s)
Local	23,692s
Remoto (1)	31,426s
Remoto (2)	34,415s
Remoto (3)	37,831s
Remoto (4)	39,315s

Após a execução do treinamento do modelo de classificação, o resultado da validação cruzada indicou que 391 (95,83%) das 408 instâncias do arquivo utilizado no treinamento foram classificadas corretamente e 10 (2,45%) foram classificadas incorretamente. Não foram classificadas 7 instâncias.

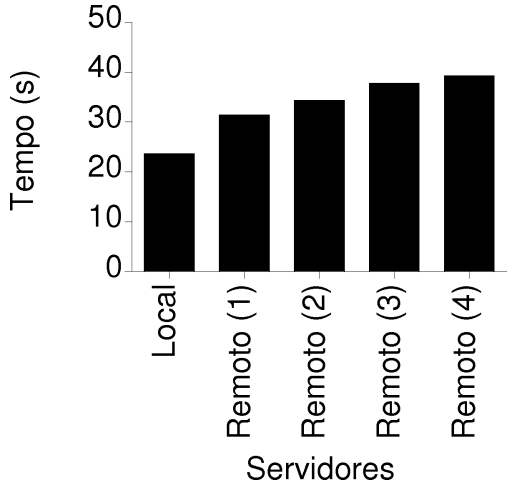

Figura 4.9: Tempos de Execução do Treinamento do Modelo - Caso de Teste II.B

4.4.4.2 Testes do Modelo de Classificação

Na etapa de teste do modelo treinado anteriormente, utilizou-se como entradas dois arquivos. Um dos arquivos, Alunos10, contendo 2852 instâncias, equivalente a 1/10 do conjunto de dados total. O outro arquivo, Alunos, contendo 28519 instâncias, equivalente ao conjunto de dados total. Foram realizadas execuções locais e remotas utilizando um, dois, quatro ou seis servidores remotos paralelamente. Os tempos de execução transcorridos para ambos os arquivos são apresentados na tabela 4.11 e no gráfico da figura 4.10.

Tabela 4.11: Etapa de teste do modelo - Caso de Teste II.B

	Arquivo Alunos10	Arquivo Alunos
Local	51,940s	294,217s
Remoto (1)	46,333s	191,213s
Remoto (2)	41,231s	154,265s
Remoto (4)	41,687s	138,718s
Remoto (6)	47,306s	123,210s

Para o primeiro arquivo a taxa de acertos obtida neste teste utilizado o modelo gerado anteriormente foi de 98,31%, com 2804 acertos dentre as 2852 instâncias classificadas, a taxa de erros foi de 1,6%, equivalente a 48 instâncias. No segundo arquivo, a taxa de acertos obtida pelo modelo foi de 98,02%, com 27957 acertos dentre as 28519 instâncias classificadas, a taxa de erros foi de 1,97%, equivalente a 562 instâncias.

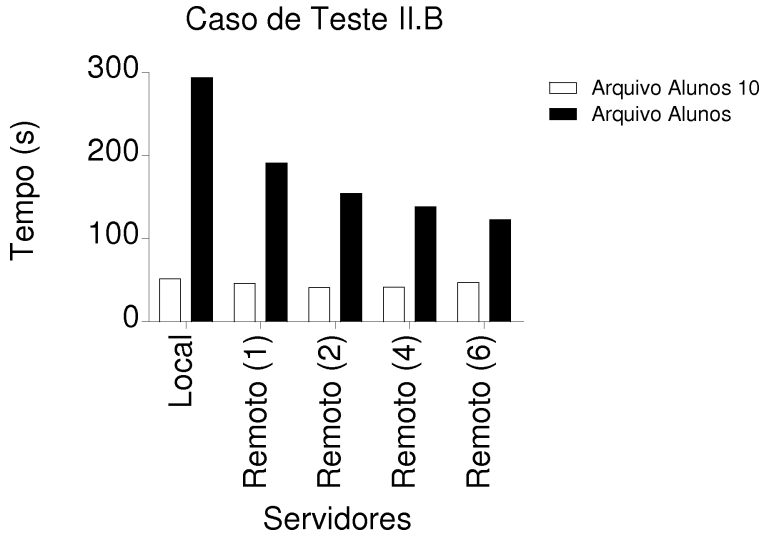

Figura 4.10: Tempos de Execução do Teste do Modelo - Caso de Teste II.B

4.4.5 Caso de Teste III.A

Este caso de teste foi realizado no ambiente 2 (4.1.2), utilizando o conjunto de dados referente a candidatos e com o algoritmo de classificação *J48*.

4.4.5.1 Treinamento do Modelo de Classificação

Nesta etapa da de treinamento do modelo de classificação utilizou-se o arquivo Candidatos140, com 672 instâncias, equivalente a 1/140 do conjunto de dados original.

A tabela 4.12 apresenta os tempos médios transcorridos na etapa de treinamento do modelo de classificação nas execuções realizadas localmente e utilizando um, dois, três ou quatro servidores remotos paralelamente para a validação cruzada. Os tempos apresentados na tabela 4.12 também estão representados graficamente na figura 4.11. A figura 4.12 mostra a árvore gerada pelo algoritmo *J48* durante o treinamento do modelo de classificação.

Tabela 4.12: Etapa de treinamento do modelo - Caso de Teste III.A

	Tempo (s)
Local	150,607s
Remoto (1)	144,992s
Remoto (2)	185,094s
Remoto (3)	206,710s
Remoto (4)	241,984s

Após a execução do treinamento do modelo, o resultado da validação cruzada indicou que 306 (82,70%) das 370 instâncias do arquivo utilizado no treinamento foram

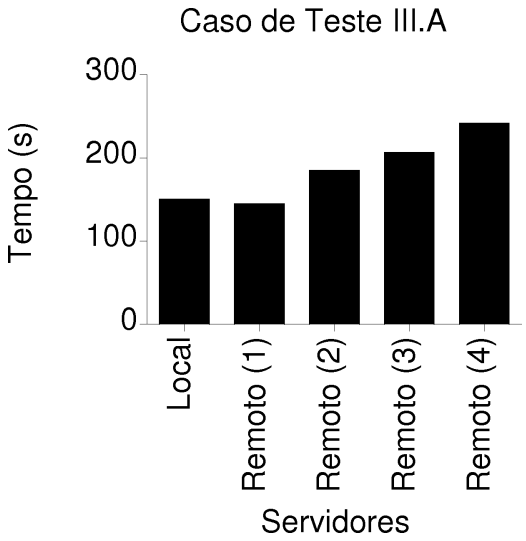

Figura 4.11: Tempos de Execução do Treinamento do Modelo - Caso de Teste III.A

classificadas corretamente e 64 (17,29%) foram classificadas incorretamente. Não foram classificadas 302 instâncias[1] .

4.4.5.2 Testes do Modelo de Classificação

Na execução da tarefa de teste do modelo de classificação treinado, utilizou-se como entrada o arquivo Candidatos contendo 94056 instâncias, equivalente ao conjunto de dados total. A tabela 4.13 mostra os tempos de execução obtidos nesse cenário, tanto para a execução local quanto para a execução remota utilizando um, dois, três, quatro, cinco ou seis servidores paralelamente. A figura 4.13 apresenta de forma gráfica os dados mostrados na tabela 4.13.

Tabela 4.13: Etapa de teste do modelo - Caso de Teste III.A

	Tempo (s)
Local	150,734s
Remoto (1)	257,547s
Remoto (2)	282,939s
Remoto (3)	301,844s
Remoto (4)	318,456s
Remoto (5)	362,139s
Remoto (6)	427,466s

A taxa de acertos obtida neste teste utilizado o modelo gerado anteriormente foi de 77,6%, com 39982 acertos dentre as 51518 instâncias classificadas, a taxa de classifica-

[1]As instâncias que não foram classificadas são aquelas em que o atributo utilizado para classificação (COD_COTA) está marcado como "?" (desconhecido).

```
ID_INSCRICAO <= 654123: C (101.0/5.0)
ID_INSCRICAO > 654123
|   ETNIA = 6.NAO DECLARADA: E (0.0)
|   ETNIA = 1.BRANCA: E (241.0/58.0)
|   ETNIA = 4.PARDA
|   |   ORDEM_COM_COTA <= 39
|   |   |   NUM_INSCRICAO <= 13889
|   |   |   |   SITUACAO = Candidato Habilitado em Aluno: A (2.0)
|   |   |   |   SITUACAO = Classificado: C (0.0)
|   |   |   |   SITUACAO = Eliminado - acertos abaixo do minimo: C (0.0)
|   |   |   |   SITUACAO = Eliminado - sem teste de aptidao: C (0.0)
|   |   |   |   SITUACAO = Eliminado - zerou acertos em disciplina: A (1.0)
|   |   |   |   SITUACAO = Nao atingiu Nota Minima para Redacao: C (0.0)
|   |   |   |   SITUACAO = Nao comparecimento Confirmacao Vaga: C (0.0)
|   |   |   |   SITUACAO = Nao Selecionado Fase1: C (1.0)
|   |   |   |   SITUACAO = Suplente: C (3.0)
|   |   |   |   SITUACAO = Ausente: A (1.0)
|   |   |   |   SITUACAO = Eliminado - zerou prova teste de aptidao: C (0.0)
|   |   |   |   SITUACAO = Vaga Confirmada: C (0.0)
|   |   |   |   SITUACAO = Confirmou vaga, fez matricula e cancelou: C (0.0)
|   |   |   |   SITUACAO = Confirmou vaga e nao fez Matricula: C (0.0)
|   |   |   NUM_INSCRICAO > 13889: E (2.0)
|   |   ORDEM_COM_COTA > 39: E (12.0)
|   ETNIA = 3.AMARELA: E (1.0)
|   ETNIA = Etnias: E (0.0)
|   ETNIA = 5.INDIGENA: E (0.0)
|   ETNIA = 2.PRETA: A (5.0/1.0)

Number of Leaves  :  23
Size of the tree :   28
```

Figura 4.12: Árvore gerada pelo algoritmo J48 durante a etapa de treinamento do modelo - Caso de Teste III.A

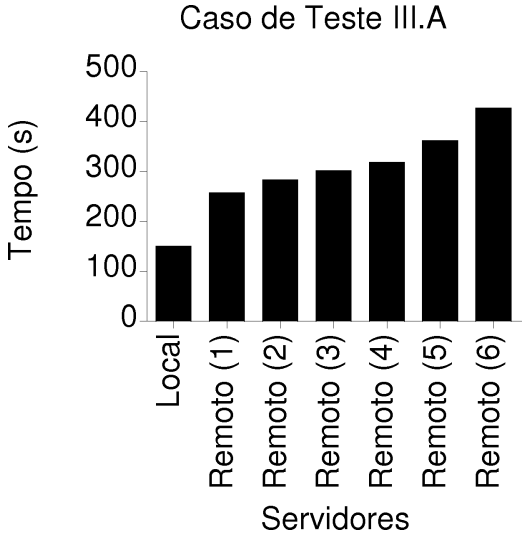

Figura 4.13: Tempos de Execução do Teste do Modelo - Caso de Teste III.A

ções incorretas foi de 22,3%, equivalente a 11536 instâncias. Não foram classificadas 42538 instâncias [1].

[1] As instâncias que não foram classificadas são aquelas em que o atributo utilizado para classificação (COD_COTA) está marcado como "?" (desconhecido).

4.4.6 Caso de Teste III.B

Este caso de teste foi realizado no ambiente 2 (4.1.2), utilizando o conjunto de dados referente a candidatos e com o algoritmo de classificação *KStar*.

4.4.6.1 Treinamento do Modelo de Classificação

Nesta etapa de treinamento da classificação utilizou-se o arquivo Candidatos70, que possui 1344 instâncias, equivalente a 1/70 do conjunto de dados original.

Foram realizadas execuções de forma local e remota da etapa de treinamento do modelo de classificação. Na forma remota utilizou-se um, dois, três ou quatro servidores para a realização da validação cruzada paralelamente. Os tempos médios obtidos nessa etapa são apresentados na tabela 4.14. Os tempos de execução obtidos também estão representados no gráfico da figura 4.14.

Tabela 4.14: Etapa de treinamento do modelo - Caso de Teste III.B

	Tempo (s)
Local	123,314s
Remoto (1)	172,158s
Remoto (2)	209,827s
Remoto (3)	229,663s
Remoto (4)	262,418s

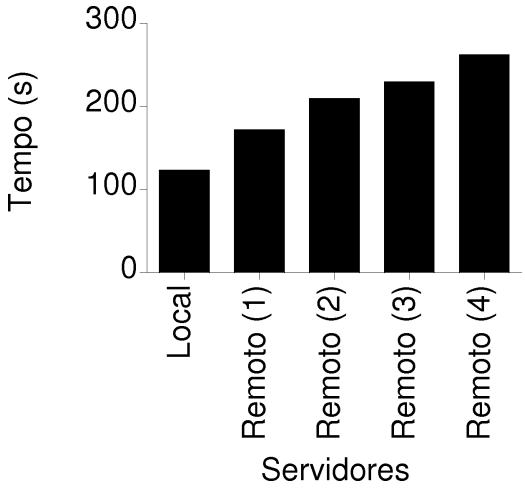

Figura 4.14: Tempos de Execução do Treinamento do Modelo - Caso de Teste III.B

Após a execução do treinamento do modelo, o resultado da validação cruzada indicou que 692 (94,02%) das 736 instâncias do arquivo utilizado no treinamento foram classificadas corretamente e 44 (5,97%) foram classificadas incorretamente. Não foram

classificadas 608 instâncias [1].

4.4.6.2 Testes do Modelo de Classificação

Na execução da tarefa de teste do modelo de classificação treinado, utilizou-se como entradas dois arquivos. Um dos arquivos, Candidatos10, possui 9406 instâncias, equivalente a 1/10 do conjunto de dados total. O outro arquivo, Candidatos, possui 94056 instâncias, equivalente ao conjunto de dados total. Foram realizadas execuções locais e remotas utilizando um, dois, quatro ou seis servidores remotos paralelamente. Os tempos de execução transcorridos para ambos os arquivos são apresentados na tabela 4.15 e no gráfico da figura 4.15.

Tabela 4.15: Etapa de teste do modelo - Caso de Teste III.B

	Arquivo Candidatos10	Arquivo Candidatos
Local	265,032s	1219,860s
Remoto (1)	234,379s	712,274s
Remoto (2)	216,463s	532,196s
Remoto (4)	210,657s	466,103s
Remoto (6)	205,457s	526,158s

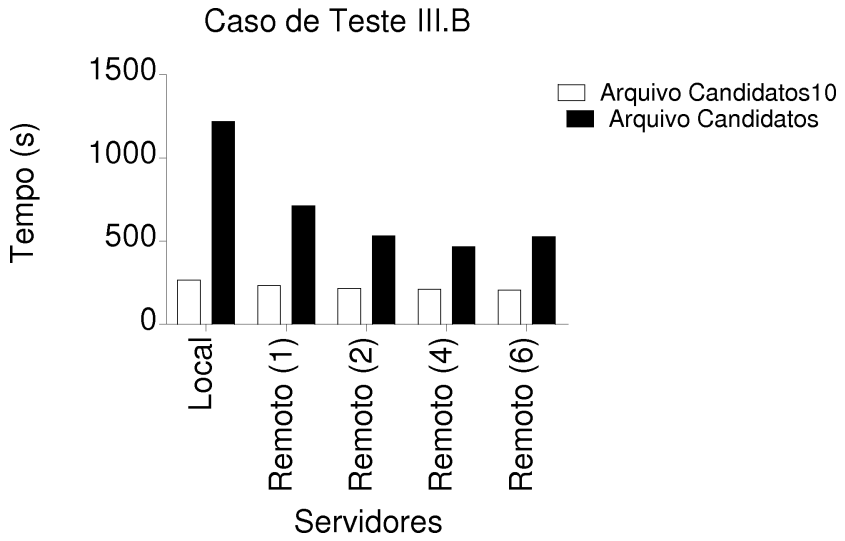

Figura 4.15: Tempos de Execução do Teste do Modelo - Caso de Teste III.B

Para o primeiro arquivo a taxa de acertos obtida neste teste utilizado o modelo de classificação gerado anteriormente foi de 95,39%, com 4905 acertos dentre as 5142 instâncias classificadas, a taxa de erros foi de 4,6%, equivalente a 237 instâncias. Não foram classificadas 4264 instâncias. No segundo arquivo, a taxa de acertos obtida pelo modelo foi de

[1] As instâncias que não foram classificadas são aquelas em que o atributo utilizado para classificação (COD_COTA) está marcado como "?" (desconhecido).

94,51%, com 48694 acertos dentre as 51518 instâncias classificadas, a taxa de erros foi de 5,48%, equivalente a 2824 instâncias. Não foram classificadas 42538 instâncias.

4.4.7 Caso de Teste IV.A

Este caso de teste foi realizado no ambiente 2 (4.1.2), utilizando o conjunto de dados referente a candidatos e com o algoritmo de classificação *J48*.

Durante etapa de teste de um modelo de classificação gerado, o Grid WEKA realiza o envio dos arquivos de dados para os servidores. Neste cenário de teste será comparada a execução da etapa de teste do modelo sem os arquivos nos servidores (Grid WEKA envia os arquivos) e com os arquivos no servidor (não é necessário o envio). O modelo de classificação utilizado foi gerado no Caso de Teste III.A (4.4.5).

Tabela 4.16: Etapa de teste do modelo - Caso de Teste IV.A

	Com Envio de Dados	Sem Envio de Dados
Remoto (2)	281,939s	189,707s
Remoto (3)	301,844s	192,423s
Remoto (5)	362,139s	197,945s

A tabela 4.16 apresenta os tempos transcorridos nas execuções remotas com e sem o envio de dados quando utilizados dois, três ou cinco servidores remotos paralelamente. A figura 4.16 mostra uma representação gráfica dos tempos apresentados na tabela 4.16.

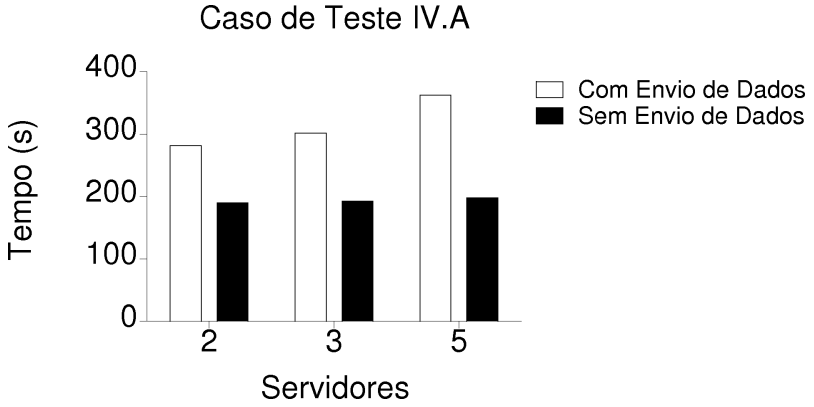

Figura 4.16: Tempos de Execução do Teste do Modelo - Caso de Teste IV.A

4.4.8 Caso de Teste IV.B

Este caso de teste foi realizado no ambiente 2 (4.1.2), utilizando o conjunto de dados referente a candidatos e com o algoritmo de classificação *KStar*.

Neste cenário de teste será comparada a execução da etapa de teste do modelo de classificação sem os arquivos nos servidores (Grid WEKA envia os arquivos) e com os arquivos no servidor (não é necessário o envio). O modelo de classificação utilizado foi gerado no Caso de Teste III.B (4.4.6).

A tabela 4.17 apresenta os tempos transcorridos nas execuções remotas com e sem o envio de dados quando utilizados dois, três, quatro, cinco ou seis servidores remotos paralelamente. A figura 4.17 mostra uma representação gráfica dos tempos apresentados na tabela 4.17.

Tabela 4.17: Etapa de teste do modelo - Caso de Teste IV.B

	Enviar Dados	Sem Enviar Dados
Remoto (2)	532,196s	424,779s
Remoto (3)	481,902s	372,511s
Remoto (4)	466,103s	339,001s
Remoto (5)	465,986s	314,856s
Remoto (6)	526,158s	314,396s

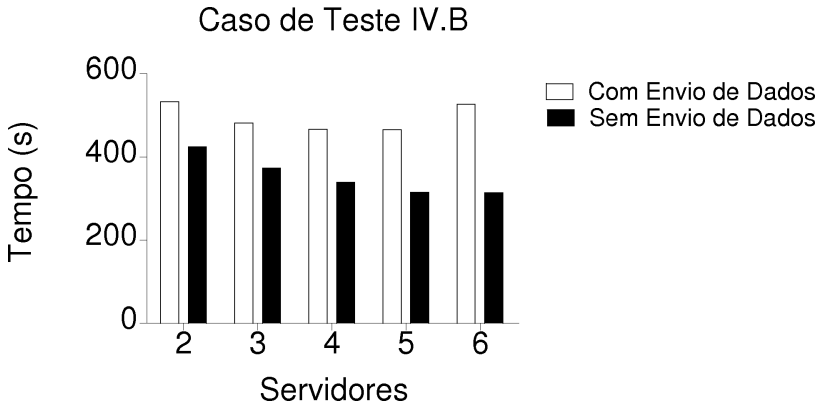

Figura 4.17: Tempos de Execução do Teste do Modelo - Caso de Teste IV.B

4.5 Discussão

A ferramenta Grid WEKA permite a execução distribuída de uma parte das tarefas de mineração de dados disponibilizadas pelo WEKA. Podem ser executados de forma remota o treinamento de um modelo de classificação e de forma remota e paralela a validação cruzada e o teste de um modelo de classificação. Todas essas tarefas são relacionadas à técnica de classificação. A ferramenta não teve o comportamento adequado quando executada em ambientes computacionais mais recentes. Em especial, o cliente do Grid WEKA apresentou travamentos sucessivos quando habilitada a opção de execu-

ção remota. A solução encontrada foi executar o cliente Grid WEKA em equipamentos mais antigos, conforme descrito no ambiente 2 (4.1.2). Entretanto isso inviabilizou testes em ambientes computacionais mais robustos e com conjuntos maiores de dados, devido à limitação dos recursos dos equipamentos mais antigos.

A análise dos resultados de desempenho verificados na etapa de treinamento do modelo de classificação nos casos de teste I.A, I.B, II.A, II.B, III.A e III.B mostra que a execução remota utilizando as funcionalidades do Grid WEKA não apresentou vantagens sobre a versão centralizada do WEKA em relação ao tempo transcorrido para a execução da tarefa de treinamento do modelo de classificação, tanto em relação ao algoritmo *J48* quanto ao algoritmo *KStar*.

Acredita-se que o custo adicionado pela distribuição das tarefas não é compensado pelo ganho obtido pelo uso de um computador remoto com capacidade de processamento semelhante para realização do treinamento. Além disso a tarefa de validação cruzada, que é realizada na mesma execução da tarefa de treinamento do modelo, representa uma parte pequena do tempo total transcorrido (tempo gasto no treinamento do modelo somado ao tempo gasto na validação cruzada) e portanto o uso de vários servidores em paralelo para realizar essa tarefa não contribui suficientemente para diminuição do tempo total.

Durante a etapa de teste dos modelos de classificação utilizando o algoritmo J48 também não foi identificada melhora no tempo de processamento quando são adicionados servidores remotos, o tempo total de execução da tarefa aumentou conforme aumentava o número de servidores remotos em utilização. Entretanto, quando utilizou-se o algoritmo *KStar* o tempo de execução da tarefa de teste do modelo diminui conforme o aumento do número de servidores remotos utilizados paralelamente, o que neste caso, torna a execução desta etapa utilizando o Grid WEKA mais rápida do que utilizando a versão centralizada do WEKA.

Nos casos de teste IV.a e IV.B observou-se que a presença nos servidores dos arquivos utilizados para o teste influi consideravelmente na diminuição do tempo de execução da tarefa de teste do modelo. Porém, os arquivos de teste são enviados pelo Grid WEKA na primeira execução, assim os arquivos somente estarão presentes nos servidores a partir da segunda execução. Não é possível enviar os arquivos manualmente para os diretórios dos servidores porque o Grid WEKA não faz a busca pelo mesmo nome do arquivo original submetido pelo cliente. Na primeira execução o Grid WEKA salva o arquivo nos servi-

dores com um nome gerado, a partir da segunda execução a busca é feita com base nesse nome atribuído pelo Grid WEKA na primeira execução.

5 CONCLUSÃO

Neste trabalho apresentou-se um estudo de caso de uma ferramenta para mineração de dados paralela e distribuída baseada no ambiente de mineração de dados WEKA. Foram analisadas ferramentas que implementem distribuição e paralelismo baseadas no ambiente WEKA e foi selecionada a ferramenta Grid WEKA. Foram preparados conjuntos de dados provenientes do sistema de informações acadêmicas da UFSM para serem utilizados nos casos de teste com a ferramenta Grid WEKA. Através dos casos de teste realizados foram identificadas em quais tarefas de mineração de dados a ferramenta implementa paralelismo e distribuição, bem como analisado o desempenho da ferramenta comparando a execução distribuída e a centralizada. Os casos de teste foram realizados em dois ambientes computacionais distintos, utilizando dois algoritmos de classificação aplicados a dois dos conjuntos de dados preparados.

Identificou-se que a ferramenta Grid WEKA permite a execução distribuída e paralela de tarefas de treinamento e de teste relacionadas a técnica de classificação. Verificou-se que o desempenho da tarefa de treinamento do modelo de classificação quando executada de forma distribuída foi inferior ao da execução centralizada. Quanto à tarefa de teste do modelo de classificação, o desempenho foi superior ao da versão centralizada quando utilizado o algoritmo *KStar* e inferior quando utilizado o algoritmo *J48*.

A ferramenta para mineração de dados paralela e distribuída Grid WEKA é baseada na versão 3.4 do WEKA centralizado, entretanto este está na versão 3.7. Dessa forma, acredita-se que o desenvolvimento da ferramenta esteja estagnado. Além disso, a ferramenta comportou-se de forma instável quando utilizados ambientes computacionais atuais.

Acredita-se que resultados melhores quanto ao desempenho de uma ferramenta de mineração de dados paralela e distribuída baseada no WEKA seriam obtidos caso além

das tarefas de testes fossem paralelizados os algoritmos de mineração de dados, visto que as etapas de treinamento tendem a ser tão ou mais custosas que as etapas de teste.

REFERÊNCIAS

BERNARDI, E. F. F. **Uma Arquitetura para Suporte à Mineração de Dados Paralela e Distribuída em Ambientes de Computação de Alto Desempenho**. 2010. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul.

BERRY, M.; LINOFF, G. **Data Mining Techniques**: for marketing, sales, and customer relationship management. Indianapolis: [s.n.], 2004.

BRAGA, L. P. V. **INTRODUÇÃO A MINERAÇAO DE DADOS**. Rio de Janeiro: [s.n.], 2005.

CASTANHEIRA, L. G. **Aplicação de Técnicas de Mineração de Dados em Problemas de Classificação de Padrões**. 2008. Dissertação (Mestrado) — Universidade Federal de Minas Gerais.

CELIS, S.; MUSICANT, D. R. **Weka-Parallel**: machine learning in parallel. Northfield: Department of Mathematics and Computer Science. Carleton College, 2002.

CLEARY, J. G.; TRIGG, L. E. K*: an instance-based learner using an entropic distance measure. In: IN PROCEEDINGS OF THE 12TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 1995. **Anais...** Morgan Kaufmann, 1995. p.108–114.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From Data Mining to Knowledge Discovery in Databases. **AI Magazine**, [S.l.], v.17, p.37–54, 1996.

FÉLIX, L. C. M. **Data mining**: torturando a los datos hasta que confiesen. Disponível em: http://www.uoc.edu/web/esp/art/uoc/molina1102/molina1102.html. Acesso em: setembro de 2010.

FU, Y. Distributed Data Mining: an overview. **IEEE Technical Committee on Distributed Processing newsletter**, [S.l.], 2001.

LAROSE, D. T. **Discovering Knowledge in Data**: an introduction to data mining. Hoboken: [s.n.], 2005.

MAIMON, O.; ROKACH, L. **Data Mining and Knowledge Discovery Handbook**. Secaucus, NJ, USA: [s.n.], 2005.

PÉREZ, M. S.; SÁNCHEZ, A.; ROBLES, V.; HERRERO, P.; PEÑA, J. M. Design and implementation of a data mining grid-aware architecture. **Future Generations Computer Systems**, Amsterdam, The Netherlands, The Netherlands, v.23, n.1, p.42–47, 2007.

TALIA, D.; TRUNFIO, P.; VERTA, O. Weka4WS: a wsrf-enabled weka toolkit for distributed data mining on grids. In: EUROPEAN CONFERENCE ON PRINCIPLES AND PRACTICE OF KNOWLEDGE DISCOVERY IN DATABASES (PKDD 2005, 9., 2005. **Proceedings...** Springer-Verlag, 2005. p.309–320.

University of California, Irvine. **UCI Machine Learning Repository**. Disponível em: http://archive.ics.uci.edu/ml/. Acesso em: outubro de 2010.

University of Waikato. **WEKA 3 - Data Mining with Open Source Machine Learning Software in Java**. Disponível em: http://www.cs.waikato.ac.nz/ ml/weka/. Acesso em: agosto de 2010.

University of Waikato. **WEKA 3.7.2 (Development version)**. Disponível em: http://prdownloads.sourceforge.net/weka/weka-3-7-2.zip. Acesso em: outubro de 2010.

WEGENER, D.; MOCK, M.; ADRANALE, D.; WROBEL, S. Toolkit-Based High-Performance Data Mining of Large Data on MapReduce Clusters. **Data Mining Workshops, International Conference on**, Los Alamitos, CA, USA, v.0, p.296–301, 2009.

WILLIAMS, G. J.; HUANG, Z. **Modelling the KDD Process**. 1996.

WITTEN, I. H.; FRANK, E.; TRIGG, L.; HALL, M.; HOLMES, G.; CUNNINGHAM, S. J. **Weka**: practical machine learning tools and techniques with java implementations. 1999.

ZUO, X.; KHOUSSAINOV, R.; KUSHMERICK, N. Grid-enabled Weka: a toolkit for machine learning on the grid. **ERCIM News**, [S.l.], v.59, 2004.