UNIVERSIDADE ESTADUAL DE PONTA GROSSA
MESTRADO EM COMPUTAÇÃO APLICADA

CRISTIAN COSMOSKI RANGEL DE ABREU

COMPUTAÇÃO PARALELA PARA REDUZIR O TEMPO DE RESPOSTA DA MINERAÇÃO DE DADOS AGRÍCOLAS

PONTA GROSSA
2013

CRISTIAN COSMOSKI RANGEL DE ABREU

COMPUTAÇÃO PARALELA PARA REDUZIR O TEMPO DE RESPOSTA DA MINERAÇÃO DE DADOS AGRÍCOLAS

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada, curso de Mestrado em Computação Aplicada da Universidade Estadual de Ponta Grossa, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Orientação: Dr. Luciano José Senger

PONTA GROSSA
2013

Ficha Catalográfica
Elaborada pelo Setor de Tratamento da Informação BICEN/UEPG

Abreu, Cristian Cosmoski Rangel de
A162 Computação paralela para reduzir o tempo de resposta da mineração de dados agrícolas/ Cristian Cosmoski Rangel de Abreu. Ponta Grossa, 2013.
65f.

Dissertação (Mestrado em Computação Aplicada - Área de Concentração: Agricultura), Universidade Estadual de Ponta Grossa.
Orientador: Prof. Dr. Senger Luciano José.

1.Computação paralela. 2.Mineração de dados. 3.Peer to -peer. 4.Tipos de coberturas florestais. I.Luciano José, Senger. II. Universidade Estadual de Ponta Grossa. Mestrado em Computação Aplicada. III. T.

CDD: 004.3

TERMO DE APROVAÇÃO

Cristian Cosmoski Rangel de Abreu

"COMPUTAÇÃO PARALELA PARA REDUZIR O TEMPO DE RESPOSTA DA MINERAÇÃO DE DADOS AGRÍCOLAS".

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre no Programa de Pós-Graduação em Computação Aplicada da Universidade Estadual de Ponta Grossa, pela seguinte banca examinadora:

Orientador: *Dr. - Luciano José Senger*
UEPG

Drª. - Maria Salete Marcon Gomes Vaz
UEPG

Dr. - Lourival Aparecido de Góis
UTFPR

Ponta Grossa, 30 de abril de 2013.

Dedico: Aos meus pais João e Silvia e ao meu grande amor Camila.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pois é ele quem nos guia, dá força e saúde para seguir na longa jornada de nossas vidas.

Aos meus pais, João Fernando e Silvia Regina, que sempre apoiaram, auxiliaram e deram força para conseguir chegar ao fim de mais uma fase da minha vida.

A minha noiva e meu grande amor, Camila Almeida Martins, que sempre me apoiou, incentivou, deu força para não desistir deste objetivo e sendo compreensiva nos momentos quando me ausentei.

A todos os meus familiares que compreenderam a ausência, apoiaram e estiveram presentes neste período.

Ao meu sócio e grande amigo, Alison Weber, que me auxiliou muito durante este período e aos demais amigos que sempre estiveram ao meu lado.

Ao meu orientador, Professor Dr. Luciano José Senger, pela amizade e contribuição de informações que sem duvida ajudaram muito a construção deste trabalho, e aos demais colegas de trabalho do LCAD - Laboratório de Computação de Alto Desempenho.

A Universidade Estadual de Ponta Grossa, pela oportunidade e estrutura cedida para os estudos e a Capes pela bolsa de estudos.

E por fim, a todos aqueles que não foram supracitados, mas que estiveram presentes e contribuíram para este trabalho.

RESUMO

O objetivo deste trabalho foi investigar a utilização da computação paralela para reduzir o tempo de resposta da mineração de dados na agricultura. Para esse fim, uma ferramenta, chamada de *Fast Weka* foi definida e implementada. Essa ferramenta permite executar algoritmos de mineração de dados e explorar o paralelismo em computadores multi-núcleos com o uso de *threads* e em sistemas distribuídos empregando redes *peer-to-peer*. A exploração do paralelismo ocorre por meio do paralelismo de dados inerente ao processo de validação cruzada (*folds*). A ferramenta foi avaliada por meio de experimentos de mineração de dados utilizando algoritmos de redes neurais artificiais aplicados em um conjunto de dados de tipos de coberturas florestais. A computação multi-*thread* e a computação em redes *peer-to-peer* permitiram reduzir o tempo de resposta das atividades de mineração de dados. Os melhores resultados foram obtidos quando empregados um número múltiplo de *threads* ou pares em relação ao número de *folds* da validação cruzada. Observou-se uma eficiência de 87% quando utilizadas 4 *threads* para 24 *folds* e 86% de eficiência, também, com 24 *folds* utilizando redes *peer-to-peer* com 11 pares.

Palavras-chave: Computação Paralela, Mineração de Dados, *Peer-to-Peer*, Tipos de Coberturas Florestais

ABSTRACT

The objective of this study was investigate the use of parallel computing to reduce the response time of data mining in agriculture. For this purpose, a tool, called Fast Weka been defined and implemented. This tool allows running data mining algorithms and explore parallelism in multi-core computers with the use of threads and in distributed systems employing peer-to-peer networks. The exploration of parallelism occurs through the data parallelism inherent to the process of cross-validation (folds). The tool was evaluated through experiments using artificial neural networks data mining algorithms applied to a data set of forest cover types. The multi-thread computing and computing on peer-to-peer networks allowed to reduce the response time of data mining activities. The best results were achieved when employed a multiple number of threads or pairs in the number of folds of cross validation. It was observed an efficiency of 87% when used 4 threads to 24 folds and 86% efficiency also in peer-to-peer networks using 24 folds with 11 pairs.

Keywords: Parallel Computing, Data Mining, Peer-to-Peer, Forest Cover Types

LISTA DE SIGLAS

AG	Algoritmos Genéticos
DEM	*Digital Elevation Model*
GIS	*Geographic Information System*
GLP	Licença Pública Geral
KDD	Knowledge Discovery in Database
LCAD	Laboratório de Computação de Alto Desempenho
MD	Mineração de Dados
MPI	*Message Passing Interface*
MPMD	Multiple Program Multiple Data
PVM	*Parallel Virtual Machine*
RN	Redes Neurais
RNA	Rede Neural Artificial
SPMD	*Single Process, Multiple Data*
UCP	Unidade Central de Processamento
USFS	*United States Forest Service*
USGS	*United States Geological Survey*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, avanços tecnológicos ocorridos na área agronômica e computacional trouxeram melhorias nos processos de obtenção e armazenamento de dados agrícolas e ambientais devido ao uso de sensores remotos, satélites, automação agrícola e sistemas de gerenciamento de cultivos. Com esse avanço, surgiu a necessidade de explorar dados com o objetivo de extrair conhecimento relevante por meio de padrões que contribuam de forma estratégica e rápida na tomada de decisão no campo (GUIMARÃES, 2005).

Pesquisas na área agrícola tem sido conduzidas afim de observar tendências, como sistemas de classificação de frutos, previsão de informações climáticas, classificação de solo, análise de tipos de vegetação, detecção de doenças e pragas. Nessas pesquisas, é comum a utilização de técnicas e ferramentas de mineração de dados (MUCHERINO; PAPAJORGJI; PARDALOS, 2009). A mineração de dados ou MD vem também sendo amplamente utilizada em diversas áreas do conhecimento como: astronomia, geologia, medicina, a qual consiste da análise, exploração e extração de informações de conjuntos de dados.

A MD tem como característica o uso de técnicas e ferramentas computacionais afim de explorar bases de dados com o objetivo de encontrar padrões ou regras para construir conhecimento. Análise de agrupamentos, algoritmos de árvores de decisão e redes neurais artificiais são exemplos de técnicas empregadas em tarefas de MD (WITTEN; FRANK; HALL, 2011).

O tempo de processamento das tarefas de MD é proporcional ao tamanho da base a ser explorada, também sendo influenciada pelo tipo de algoritmo de MD a ser aplicado. Assim, é importante utilizar técnicas que permitam diminuir o tempo de resposta das atividades com baixo custo, que pode ser realizado por meio da computação paralela (RAUBER; RUNGER, 2010).

Trabalhos tem demonstrado que a MD na agricultura necessita frequentemente de um tempo de processamento elevado (BLACKARD; DEAN, 1999; CHAGAS *et al.*, 2009; PESSOA, 2009). Observa-se que a computação paralela tem se tornado uma alternativa viável para reduzir esse tempo de resposta, haja visto o desenvolvimento da indústria de microprocessadores com arquitetura paralela (multi-núcleo) e a tendência da utilização da computação paralela em trabalhos correlatos (GUIMARÃES, 2005; SOUZA *et al.*, 2011; DAMIATI; SOUZA; SENGER, 2012).

A computação paralela ou processamento paralelo tem como foco principal melhorar o desempenho de aplicações computacionais por meio da utilização de computado-

res multi-núcleos, multi-processados e multi-computadores, buscando uma melhor relação custo/beneficio (SENGER, 2004; WILKINSON; ALLEN, 2004; HENNESSY; PATTERSON, 2012).

Atualmente, com a consolidação dos processadores multinúcleos, pode-se reduzir o tempo de execução utilizando apenas um computador, por meio dos diversos núcleos executando atividades de modo concorrente (STALLINGS, 2010). Outra opção é utilizar computadores disponíveis que estejam ociosos, interconectados por meio de uma rede *Ethernet*, ou na própria Internet, para distribuir as atividades a serem executadas, essa abordagem por ser obtida por meio de redes P2P (SENGER; SOUZA; FOLTRAN, 2011).

O objetivo principal deste trabalho foi investigar a utilização da computação paralela afim de melhorar o tempo de resposta das atividades de mineração de dados agrícolas. Os objetivos específicos são:

- Investigar estratégias de paralelização da mineração de dados que permitam reduzir o tempo da resposta, sem redução de qualidade nos resultados obtidos;
- Definir e implementar modelo de computação paralela para tarefas de mineração de dados que permita explorar o paralelismo em processadores multi-núcleo e grades computacionais;
- Avaliar o ganho que pode ser obtido na mineração de dados agrícolas, identificar parâmetros relacionados com a validação cruzada e com as arquiteturas paralelas de execução, que produzem o melhor desempenho, expresso pela eficiência e fator de aceleração (*speed-up*).

Este trabalho está organizado em cinco capítulos e dois apêndices. O Capítulo 2 apresenta a revisão da literatura, no qual estão relacionados os conceitos de mineração de dados, computação paralela e trabalhos correlatos. O Capítulo 3 descreve os materiais adotados com detalhes do *framework* P2PComp, a ferramenta Weka, a descrição da base de dados e as definições referentes a RNA. O método de trabalho adotado também é apresentado nesse capítulo.

No Capítulo 4 estão presentes os resultados, no qual é apresentado o *software* desenvolvido *Fast Weka*, as análises de tempo de execução e do classificador. O Capítulo 5 apresenta as conclusões. E por fim estão presentes dois apêndices. No Apêndice A estão presentes os resultados das execuções e no Apêndice B uma pequena descrição do uso da ferramenta.

2 REVISÃO DA LITERATURA

2.1 Considerações Iniciais

Este capítulo apresenta conceitos e os trabalhos correlatos a esta dissertação. Assim, a Seção 2.2 apresenta os conceitos da área de computação paralela, enfatizando computadores multi-núcleos e a utilização de sistemas computacionais distribuídos como ambiente de execução. A Seção 2.3 apresenta os conceitos de mineração de dados. A Seção 2.4 é dedicada os trabalhos correlatos na área mineração de dados na agricultura e mineração de dados em paralelo.

2.2 Computação Paralela

Computação paralela ou processamento paralelo constitui-se na exploração de eventos computacionais concorrentes, através de unidades de processamento que cooperam e comunicam-se (ALMASI; GOTTLIEB, 1994). O objetivo principal é melhorar o desempenho de aplicações computacionais que tem desempenho não satisfatório em arquiteturas sequenciais (SENGER, 2004).

A exploração de eventos concorrentes pode ser realizada em diferentes níveis: O primeiro nível está mais próximo ao hardware que pode ser encontrado nas unidades que compõem a UCP - Unidade Central de Processamento. Essa técnica de execução paralela tem sido empregada em arquiteturas de hardware que têm múltiplas unidades funcionais e empregam técnicas de *pipeline* para execução de eventos concorrentes. Também pode ser encontrado através de ambientes que permitem a paralelização automática, onde o compilador é responsável por paralelizar um programa através do código sequencial. No nível intermediário, o paralelismo pode ser atingido por meio de procedimentos ou sub-rotinas de programas paralelos que são atribuídos aos elementos de processamento e executados concorrentemente, por meio do qual, geralmente, é necessário o suporte explícito de linguagens de programação. No nível mais alto, a tarefa a ser realizada é dividida em *threads* ou processos, que são executados concorrentemente. Assim, uma aplicação paralela é um conjunto de *threads* ou processos que interagem e cooperam entre si para realizar uma determinada tarefa (ALMASI; GOTTLIEB, 1994).

Mesmo sendo possível aumentar o desempenho de processos computacionais utilizando paralelismo nos níveis um e dois, a solução frequentemente empregada é a utilização do terceiro nível de implementação (ALMASI; GOTTLIEB, 1994). O paralelismo explorado em nível de *threads* ou processos é atrativo pelo fato que pode ser adotado em máquinas

de memória compartilhada (*threads*) ou distribuída (processos), que confere portabilidade ao código paralelo.

O desenvolvimento de aplicações paralelas pode ser seguido por dois modelos principais: paralelismo funcional e paralelismo de dados. Paralelismo funcional é aquele no qual os processos ou *threads* da aplicação realizam diferentes funções, cada um responsável por uma etapa do trabalho. O paralelismo funcional é referenciado como MPMD - *Multiple Program Multiple Data*. O paralelismo de dados é um modelo mais simples se comparado ao paralelismo funcional, no qual os processos possuem a mesma função, trabalhando sobre dados diferentes. O paralelismo de dados é também conhecido como SPMD - *Simple Program Multiple Data* (QUINN, 1994).

A escolha e utilização de um determinado modelo está diretamente relacionado ao problema a ser solucionado. Além do problema, a escolha também é influenciada pelo hardware utilizado. Características como modos de execução, arquitetura das UCP, organização da memória, heterogeneidade dos recursos e desempenho da rede afetam diretamente o desenvolvimento de aplicações paralelas.

O desenvolvimento de *software* em paralelo necessita de mecanismos para definir quais tarefas serão executadas concorrentemente, formas de ativação e finalização dessas tarefas e métodos de trocas de mensagens (SENGER; SOUZA; FOLTRAN, 2011). Para Almasi e Gottlieb (1994), métodos e ferramentas para construção de programas em paralelo podem ser divididos em três grupos: ambientes de paralelização automática, linguagens de programação concorrentes e extensões paralelas para linguagens de programação imperativas.

Ambientes de paralelização automática tentam encontrar o paralelismo existente em um código fonte, os quais são escritos, geralmente, em uma linguagem de programação imperativa, para que de maneira automática seja gerado um programa paralelo. A vantagem dessa abordagem é a utilização de códigos sequenciais reduzindo o trabalho do programador.

As linguagens de programação concorrentes permitem o desenvolvimento de programas paralelos por meio da sintaxe da linguagem. A linguagem *Occam*, baseada no paradigma CSP é um exemplo, que permite o desenvolvimento de aplicações paralelas em máquinas com memória distribuída.

As extensões paralelas para linguagens de programação imperativas são bibliotecas que possibilitam a criação, comunicação e sincronismo entre processos, normalmente são implementadas por meio de bibliotecas de passagem de mensagem, por tornar possí-

vel a computação paralela em sistemas de memória distribuída, como por exemplo: PVM - *Parallel Virtual Machine* (BEGUELIN *et al.*, 1994), MPI - *Message Passing Interface* (BURNS; DAOUD; VAIGL, 1994).

O PVM é composto por um conjunto de bibliotecas e componentes de *software* que possibilitam o desenvolvimento de sistemas paralelos (BEGUELIN *et al.*, 1994; SUNDERAM *et al.*, 1994). Inicialmente, foi desenvolvido para ser utilizado sobre plataformas Unix, posteriormente sendo adaptado para ser utilizado em diversas plataformas como Linux e Windows (BEGUELIN *et al.*, 1994).

O MPI é um conjunto de rotinas e procedimentos de comunicação para ambientes de passagem de mensagens. O MPI tem como objetivo principal permitir que aplicações paralelas sejam independentes de plataformas. O MPI também oferece a possibilidade de executar, de modo transparente, em sistemas heterogêneos (BURNS; DAOUD; VAIGL, 1994).

Dentro do padrão MPI existem diversos procedimentos para troca de mensagens, suporte a grupos de processos e criação de tipos de mensagens. Os métodos de trocas de mensagens podem ser otimizados de acordo com a arquitetura utilizada. Exemplos de linguagens de programação contempladas pelo MPI são: C , Fortran e Java.

Outra forma de paralelismo pode ser encontrada através do uso de linguagem de alto nível como C++ e Java, que implementam em sua estrutura o uso de *threads*. Tendo em vista as operações de entrada/saída e uso de recursos que podem ocorrer dentro de uma aplicação, o uso de multi-*threads*, mesmo quando utilizado em computadores mono-núcleo, pode trazer ganhos significativos (RAUBER; RUNGER, 2010; COULOURIS *et al.*, 1988).

Por meio da evolução do *hardware*, processadores multi-núcleos tornaram-se a tecnologia corrente em sistemas computacionais (HENNESSY; PATTERSON, 2012). Assim, o uso de multi-*threads* vem como uma solução eficiente para implementar o paralelismo, sendo que o número de *threads* independe da quantidade de núcleos (PENHA; CORRêA; MARTINS, 2002).

Threads são apresentadas como processos leves, porém da mesma forma que processos são partes de um *software*, executando um conjunto de instruções com um baixo custo computacional para serem criadas e destruídas, e a comunicação entre as *threads* ocorre por meio de variáveis compartilhadas (PENHA; CORRêA; MARTINS, 2002). Entretanto, essa forma de paralelismo é limitada, pois, à medida que cresce o número de *threads* no mesmo computador, crescem também problemas de concorrência durante o acesso à memória principal e a outros recursos, que levam à degradação do desempenho.

O uso de *threads* está frequentemente restrito ao computador na qual estão sendo executadas e o número de unidades de processamento por computador é um fator limitante. Assim, outras técnicas de paralelismo que permitam a comunicação entre computadores através da rede podem melhorar o desempenho, como sistemas distribuídos.

Dentre as tecnologias existentes em sistemas distribuídos, o modelo de computação par-a-par (P2P) tem se destacado por oferecer uma alternativa para construção de programas com um eficiente uso de recursos, permitindo um crescimento em escala e capacidade de auto-organização (HAUSWIRTH; SCHMIDT, 2005; AMORETTI, 2006; RANJAN; HARWOOD; BUYYA, 2006).

O modelo P2P possibilita a criação de sistemas distribuídos por meio de um grupo de computadores que cooperam entre si denominados de *pares*. O meio mais frequentemente utilizado para realizar a comunicação entre os pares é a Internet. O P2P implementa uma rede sobreposta, onde cada par possui uma identificação única independente da localização real na rede física. Assim, a rede virtual permite que pares possam ser localizados e troquem informações, mesmo em redes internas à *firewalls* (SENGER; SOUZA; FOLTRAN, 2011). Dentre os trabalhos que empregam o modelo P2P para execução de aplicações paralelas, destacam-se o *framework* P2PComp (SENGER; SOUZA; FOLTRAN, 2011) e o Boinc (ANDERSON, 2004).

O *framework P2PComp* segue a filosofia pura de redes P2P, pois os pares da rede executam as mesmas funções como: execução da aplicação, compartilhamento de recursos e organização da rede. Para o seu desenvolvimento foi utilizado o padrão P2P JXTA (VERSTRYNGE, 2008; GRADECKI, 2002) que define protocolos que permitem o crescimento em escala e suporte à diferentes sistemas operacionais e arquiteturas (SENGER; SOUZA; FOLTRAN, 2011). O P2PComp foi adotado com sucesso em trabalhos de computação aplicada a agricultura (SERCKUMECKA, 2012; SCHMITKE, 2012).

Na busca de verificar os diferentes ganhos obtidos com a computação paralela, tem-se a medida de desempenho fator de aceleração chamada *speedup*. O *speedup* determina o ganho que pode ser obtido por meio do processamento paralelo em relação ao tempo de execução do melhor algoritmo sequencial. O mesmo é calculado conforme a Equação 1, onde ts representa o tempo de execução do algoritmo sequencial, tp o tempo de execução da mesma atividade em paralelo e n que representa o número de elementos de processamento envolvidos. O resultado do *speedup* pode ser dividido em linear, sublinear e superlinear (WILKINSON; ALLEN, 2004).

Speedup linear é quando o resultado de S(n) é igual ao número de elementos

de processamento utilizados, representando que todo o código sequencial foi paralelizado por igual, *speedup* sublinear é observado quando o valor de S(n) é menor que o número de elementos de processamento, isto ocorre devido a não paralelização de todo o código sequencial, *Lei de Amdahl*, por fim, o *speedup* superlinear é atingido quando o resultado de S(n) é maior que o número de elementos de processamento utilizados.

$$S(n) = \frac{ts}{tp} = \frac{tempo\ de\ execução\ serial}{tempo\ de\ execução\ em\ paralelo} \tag{1}$$

A eficiência é outra medida utilizada para verificar os ganhos da computação paralela e determina o quanto os processadores então sendo empregados para o processamento paralelo. Ela é calculada conforme Equação 2, na qual S(n) representa o *speedup* obtido e n corresponde ao número de elementos de processamento utilizados para obter este *speedup* (SENGER, 2004).

$$E(n) = \frac{S(n)}{n} = \frac{speedup}{número\ de\ elementos\ de\ processamento} \tag{2}$$

2.3 Mineração de Dados

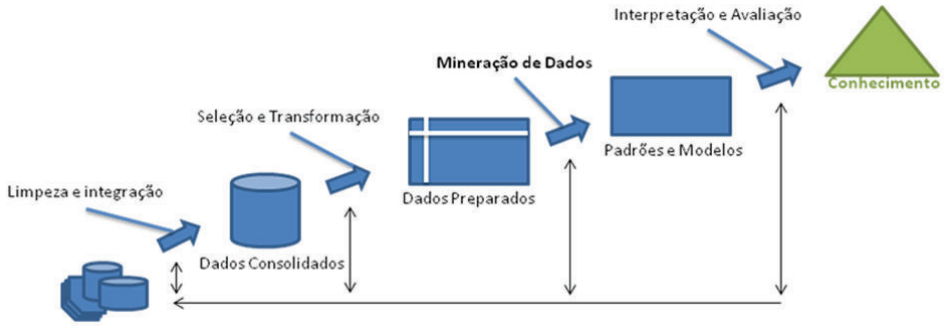
A mineração de dados (MD) começou a ser utilizada no início da década de 80, com o surgimento da necessidade de analisar os dados computacionais armazenados. Nesta época, a MD consistia em apenas extrair informações de grandes conjuntos de dados de forma ágil e automatizada (WITTEN; FRANK; HALL, 2011).

Com a evolução da MD, atualmente ela consiste na análise, exploração e extração de informações relevantes de dentro de um conjunto de dados, gerando conhecimento por meio de padrões e tendências observadas. A MD é uma atividade ou processo do KDD (*Knowledge Discovery in Databases*), descoberta de conhecimento em banco de dados, que é mais amplo envolvendo várias etapas para se chegar à descoberta de conhecimento (WITTEN; FRANK; HALL, 2011). O KDD é dividido em algumas etapas, conforme observado na Figura 1:

1. O primeiro processo para a descoberta de conhecimento é realizado por meio de uma limpeza, onde são eliminados os ruídos e dados inconsistentes do conjunto de dados a ser processado;

2. Após a limpeza dos dados, estes dados oriundos de diferentes fontes são combinados em um único repositório;

Figura 1: Etapas do processo de Descoberta de Conhecimento em Bases de Dados



Fonte: O autor

3. O processo de transformação e seleção serve para determinar e aprimorar as informações que serão computadas no processo. Por meio desta etapa são selecionados os atributos relevantes do conjunto de dados e é realizada a transformação dos dados em um formato de arquivo apropriado para aplicar os algoritmos de mineração;

4. A mineração de dados é a aplicação das técnicas inteligentes para extração dos padrões de interesse;

5. Realizada a MD, a avaliação ou pós-processamento identifica os padrões interessantes, de acordo com os critérios estabelecidos pelo usuário;

6. Finalizando, com a visualização dos resultados utilizando técnicas de representação de conhecimento para poder apresentar ao usuário o conhecimento observado.

A MD é composta por diversas tarefas que consistem na especificação do que desejamos buscar nas bases de dados, dentre elas são destacadas as tarefas de: classificação, agrupamento, associação e regressão. Cada tarefa da MD faz o uso de algoritmos para explorar os dados obtendo diferentes resultados (FAYYAD *et al.*, 1996).

Dentre as tarefas supracitadas, a classificação é uma das tarefas mais utilizadas para minerar dados. Um classificador é um modelo que foi criado através de um processo chamado de treinamento, o qual permite classificar as linhas de um conjunto de dados dentre diversas classes pré-determinadas. Cada linha do conjunto de dados é chamada de exemplo ou amostra. Após ser construído o modelo ele é submetido a uma etapa de verificação ou validação, onde as regras criadas por ele são testadas sobre outro conjunto de dados independente. Assim, uma das formas de mensurar o desempenho do modelo é medindo a porcentagem de exemplos do conjunto de dados de testes que o modelo consegue classificar de forma satisfatória (FAYYAD *et al.*, 1996).

Dentre as técnicas de validação, a validação cruzada, *cross-validation*, é uma técnica destinada a avaliar a capacidade de generalização de um modelo a partir de um conjunto de dados, sendo aplicada a problemas cujo objetivo é a predição, buscando avaliar o desempenho do modelo estudo em prática. A validação cruzada baseá-se no conceito de particionamento da base de dados, *folds*, em subconjuntos mutuamente exclusivos. Existem diversos modos de realizar o particionamento dos dados, dentre as mais utilizadas estão: método *holdout*, *k-folds* e *leave-one-out* (KOHAVI, 1995).

O método *holdout* consiste em particionar todo o conjunto de dados em dois subconjuntos, sendo destinado um subconjunto para treinamento e outro para validação. Estes subconjuntos podem ser divididos em tamanhos iguais ou não. A porcentagem mais comumente utilizada consiste em utilizar 2/3, 66%, do conjunto de dados para treinamento e o restante, 1/3 ou 33% do conjunto para teste. Esta técnica normalmente é aplicada a grandes bases de dados devido ao menor custo computacional (KOHAVI, 1995).

A técnica *k-folds* consiste em particionar o conjunto total de dados em k *folds*, partições, de mesmo tamanho. Dos quais k - 1 *folds* são utilizados para treinamento do modelo e o *fold* restante é utilizado como conjunto de validação. O processo é repetido k vezes, até que cada partição seja utilizada uma vez com o conjunto de teste. Após finalizar o processo, as medidas de desempenho são calculadas pela média dos resultados obtidos em cada *fold*, obtendo-se assim uma estimativa da qualidade do modelo, sendo mais confiável que método citado anteriormente, porém demandando de maior processamento (KOHAVI, 1995; FAYYAD *et al.*, 1996).

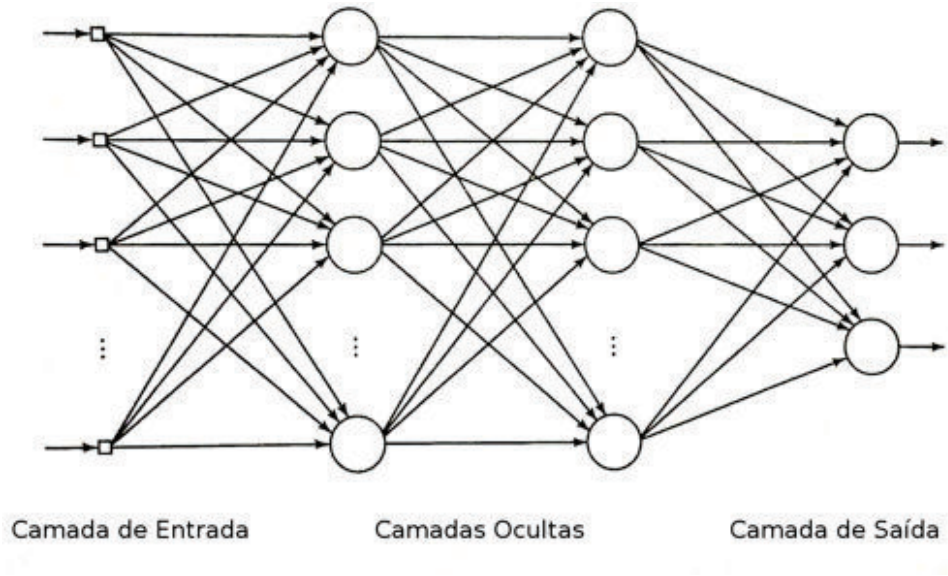
O método *leave-one-out* é um tipo especifico da utilização do método *k-folds*, no qual o total de partições k é igual ao total de instancias da base de dados. Assim são realizadas k cálculos de erro, um para cada dado. Este método apresenta uma análise mais completa sobre a variação do modelo, tendo um alto custo computacional e é mais indicado para poucas instâncias de dados (KOHAVI, 1995).

Existem diversos algoritmos de aprendizagem que podem ser aplicados a tarefa de classificação, dentre os quais temos aqueles inspirados no modelo de Redes Neurais Artificiais (RNAs). RNAs também conhecida na literatura como Redes Neurais (RN), são definidas de diferentes formas por diferentes autores. Apesar de não existir um único modo de definir RNAs, as redes neurais são definidas como um método para solucionar problemas, partindo de um sistema que possui circuitos que conectam neurônios com a finalidade de simular o cérebro humano, assim, errando, aprendendo e fazendo descobertas (MUCHERINO; PAPAJORGJI; PARDALOS, 2009; HAYKIN, 1998; KUMAR *et al.*, 2009; WITTEN;

FRANK; HALL, 2011)

O funcionamento de uma rede neural ocorre do seguinte modo, os neurônios funcionam como somadores, tendo os circuitos que os conectam contendo pesos sinápticos para cada conexão. O conhecimento da RNA é armazenado nos pesos sinápticos da rede (HAYKIN, 1998). Normalmente a RNA é representada por um grafo orientado, como pode ser visto na Figura 2.

Figura 2: Arquitetura de uma Rede Neural Artificial



Fonte: O autor

Braga, Ludermir e Carvalho (2007) afirma que a propriedades mais relevante de uma RNA é a capacidade de aprender por meio de instâncias e realizar inferências sobre o que aprendeu, dessa maneira podendo melhorar o seu desempenho a cada nova iteração. O aprendizado em redes neurais pode ocorrer de diversas formas. Uma forma delas é o aprendizado supervisionado, o qual é apresentado um conjunto de padrões de entradas a rede e suas respectivas saídas. Então, para cada entrada indica-se quanto houve de erro para a instância calculada. O erro encontrado é utilizado pela RNA para que seja feito os ajustes dos pesos sinápticos.

Para o funcionamento deste método é necessário que haja um conhecimento prévio do comportamento que se deseja encontrar na rede. Existem vários algoritmos de aprendizado supervisionado que diferem entre si pela forma como é ajustado o peso sináptico, um deles é o algoritmo *backprogation* (BRAGA; LUDERMIR; CARVALHO, 2007).

Durante o treinamento de uma RNA utilizando o algoritmo de correção de erro *backpropagation*, a RN opera em duas etapas. Primeiro, uma instância do conjunto de dados é apresentado à camada de entrada da RN, e são realizados os cálculos por meio

da rede, camada por camada, até que a instância de entrada chegue à camada de saída. Na segunda etapa, a saída obtida é comparada à saída real. Se esta não estiver correta, o erro é calculado. Assim, o erro é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das camadas internas vão sendo alterados conforme o erro é retornado as camadas de entrada (HAYKIN, 1998).

As RNAs podem realizar o tratamento de informações lineares e não lineares, aprendendo através de um conjunto de exemplos da base de dados e tenta encontrar o melhor desempenho mesmo na presença de ruídos, dados incompletos ou de uma grande complexidade (HAYKIN, 1998).

Trabalhos são encontrados na literatura utilizando técnicas de MD para exploração de dados da agricultura e ambientais. O tempo de processamento destas atividades de MD pode ser muitas vezes significativo, demandando de um alto custo computacional. Assim, na subseção seguinte estão descritos trabalhos correlatos utilizando MD na agricultura e também técnicas para explorar o paralelismo em tarefas de MD.

2.4 Trabalhos Correlatos

El-Telbany, Warda e El-Borahy (2006) desenvolveram um trabalho utilizando MD no qual o objetivo foi desenvolver modelos para classificação de doenças do arroz egípcio. Um dos algoritmos de aprendizagem utilizado foi a RNA. A RNA foi construída e treinada utilizando uma configuração de 52 entradas, 33 neurônios na camada oculta, 5 saídas, taxa de aprendizagem de 0,3, momento de 0,2 e 500 iterações. O modelo obtido para a previsão de doenças de arroz atingiu um índice de acerto de 96,4% para o conjunto de dados de teste.

Em um trabalho similar de aplicação de RNAs, Blackard e Dean (1999) realizaram a comparação entre RNA e análise discriminante para criação de classificadores para tipos de coberturas florestais a partir de variáveis cartográficas. O RNA construída utilizou as configurações de 54 entradas, 120 neurônios na camada oculta, 7 classes de tipos de coberturas florestais, com uma taxa de aprendizagem de 0.05, taxa de momento de 0.5 e 1000 interações. Para obter estas configurações para a RNA, foram realizados 56 análises diferentes demandando cerca de 56 horas para cada análise. Após a comparação das técnicas, as RNAs obtiveram uma maior precisão, chegando a 70,58%.

Chagas *et al.* (2009) utilizaram duas técnicas de MD afim de construir e avaliar a eficiência de classificadores para níveis de degradação de pastagens no município de

Viçosa, MG. Quando utilizadas RNAs, eles observaram que para encontrar os parâmetros ideais da RNA foi necessário alto custo computacional e elevado tempo de execução. Os resultados apresentados demonstram que RNA obteve uma maior eficiência que outros métodos e chegando a uma correlação de 83,3%.

Pessoa (2009) utiliza técnicas de MD afim de realizar previsões climáticas em variáveis como temperatura e precipitação. Ele utiliza a teoria dos conjuntos aproximativos, afim de reduzir as variáveis para criação do modelo de previsão climática, tendo em vista o elevado poder computacional necessário para criação de modelos climáticos. Após a redução das variáveis é aplicado técnicas de RNA para criação do modelo. Foram utilizados 18 anos de dados afim de predizer variáveis para os próximos 3 anos à obtenção dos mesmo. Obtendo uma melhor precisão para temperatura e menor para precipitação.

Os trabalhos supracitados não empregam computação paralela para reduzir o tempo de resposta. Abaixo são apresentados alguns trabalhos que empregam a computação paralela para redução do tempo de execução.

GUIMARÃES (2005) observou que quando o processo de aprendizado de um modelo por meio de algoritmos de aprendizagem é iniciado, este com um grande conjunto de dados, ou com um alto número de repetições, demanda de um alto custo computacional e tempo de execução. Por meio destas observações, Guimarães desenvolveu um aplicativo para distribuir o processamento da construção de seu modelo, por meio do qual o processamento poderia ser realizado por vários computadores, utilizando o algoritmo de aprendizagem Algoritmos Genéticos (AG). Este aplicativo obteve bons resultados conseguindo reduzir seu tempo de execução estimado para construção do modelo de 1450 horas para 84 horas. Apesar de reduzir o tempo de resposta, a aplicação compreende algoritmos genéticos e não outros algoritmos de aprendizado de máquina.

Souza *et al.* (2011) aplicaram uma ferramenta de MD em paralelo para construção de um modelo de classificação utilizando RNA para produção de soja, afim de observar a relação existente entre os atributos químicos do solo e a produção. Utilizando apenas um computador, considerando 4 núcleos reais e 4 simulados, eles reduziram o tempo de processamento de 280 para 80 segundos. Esses resultados demostram que a utilização de técnicas de computação paralela pode melhorar o tempo de resposta da mineração de dados.

Pimenta *et al.* (2009) desenvolveram e avaliaram uma ferramenta de mineração de dados em paralelo baseada nas ferramentas *OurGrid* e Weka. O *OurGrid* cria um ambiente de grades computacionais para que a ferramenta Weka seja executada em modo

paralelo. O software proposto obteve um redução de tempo de 49 minutos para 17 minutos quanto utilizados dez computadores de mesma configuração. A ferramenta não permite explorar conjuntamente o paralelismo em nível de *threads*.

Damiati, Souza e Senger (2012) avaliaram o desempenho de uma ferramenta de mineração de dados em paralelo denominada de GridWeka 2. Tal ferramenta implementa o paralelismo utilizando comunicação TCP, por meio do qual o cliente solicita uma atividade de MD a ser processada e os servidores processam esta atividade em paralelo. O algoritmo de RNA foi aplicado para classificação de índices pluviométricos. Os resultados observados por eles demonstram uma redução de quase três vezes em relação ao tempo de execução sequencial, quando utilizados 6 computadores. A comunicação diretamente em TCP, apesar de ser uma das primitivas mais eficientes de comunicação em redes, em relação à latência e vazão, não é totalmente adequada em relação ao crescimento em escala, pois computadores protegidos em redes virtuais e por *firewall* podem não estar acessíveis.

Talia, Trunfio e Verta (2008) desenvolveram e avaliaram o *software* Weka4WS, o qual implementa o paralelismo baseado no *framework Web Services Resource Framework* e permite a execução remota de algoritmos de MD disponíveis no Weka. Seus resultados demonstram que de modo sequencial a execução das atividades demandou 181,37 minutos contra 40,7 minutos quando executados de modo paralelo, utilizando seis computadores, obtendo um *speedup* de 4,45.

Zuo (2004) desenvolveu uma ferramenta denominada de *Weka Parallel* para realizar mineração de dados em paralelo, baseando o paralelismo em nível de validação cruzada utilizando e comunicação TCP. A ferramenta é executada somente por meio de linhas de comando, e suas configurações são definidas por arquivos. Xin Zuo observou um *speedup* de 1,95 quando utilizados quatro computadores aplicados a um conjunto de testes de 60 registros.

2.5 Considerações Finais

Observando os diferentes trabalhos na área de mineração de dados que indicam o tempo de execução como um fator limitador das atividades de MD, considerando o tamanho das bases de dados e os algoritmos aplicados e os diferentes trabalhos abordando o paralelismo na mineração de dados, pode-se observar a necessidade do uso da computação paralela para as atividades de MD.

Assim, no próximo capítulo serão abordadas as ferramentas utilizadas para o

desenvolvimento do *software Fast Weka*, a descrição da base de dados e a definição do método de trabalho adotado.

3 MATERIAIS E MÉTODOS

3.1 Considerações Iniciais

Este capítulo apresenta os materiais e métodos relacionados aos objetivos desta dissertação. A Seção 3.2 apresenta o *framework* P2PComp utilizado neste trabalho juntamente com o software Weka, descrito na Seção 3.3, para o desenvolvimento da ferramenta *Fast Weka*. A Seção 3.4 apresenta a base de dados de tipos de coberturas florestais. A Seção 3.5 apresenta a seleção realizada nos dados e as configurações da RNA utilizada para a mineração de dados. Por fim a Seção 3.6 traz quais serão os métodos utilizados para realização dos experimentos visando demonstrar o desempenho da ferramenta *Fast Weka*.

3.2 Framework P2PComp

Foi utilizado neste trabalho o *framework* P2PComp[1], agrupamento de classes e algoritmos desenvolvidos em uma linguagem de programação utilizadas para fornecer funcionalidades genéricas ao desenvolvedor, desenvolvido pelos pesquisadores: Senger, Souza e Foltran na Universidade Estadual de Ponta Grossa - UEPG, Brasil.

O *framework* P2PComp foi implementado como um conjunto de classes escritas em Java que necessita da biblioteca JXSE[2] como núcleo para criação da rede P2P. O JXSE é uma implementação do padrão JXTA escrito em java. O *framework* segue o modelo P2P puro, no qual todos os pares tem a mesma funcionalidade (SENGER; SOUZA; FOLTRAN, 2011).

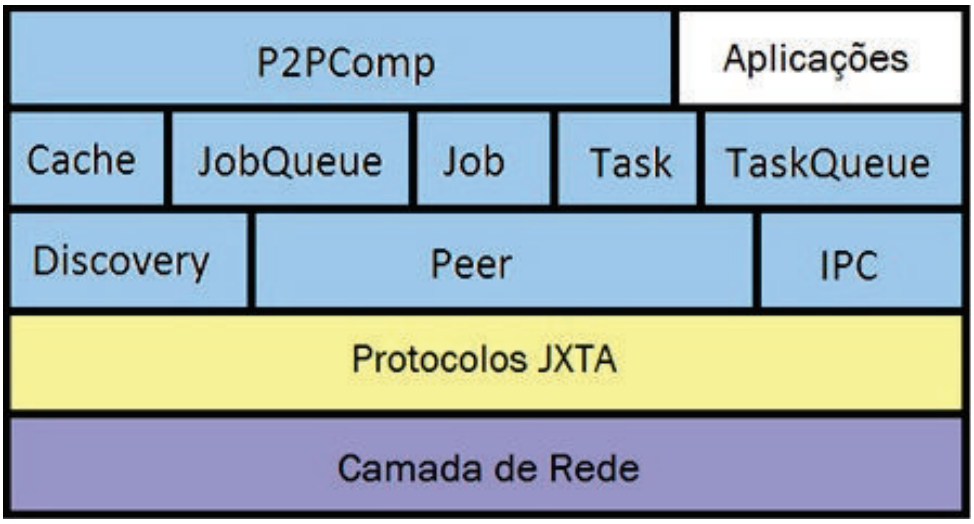
Na Figura 3, é possível visualizar a estrutura interna básica de funcionamento do *framework*. Em azul estão presentes as classes principais do *framework* implementadas em java, em branco estão representadas as aplicações desenvolvidas sobre a plataforma P2PComp, em amarelo a biblioteca JXSE e por fim em lilás a camada de rede que está em baixo nível, transparente ao desenvolvedor.

O P2PComp define que aplicações paralelas são compostas por tarefas que executam o mesmo código fonte, conforme Código Fonte 1, seguindo o padrão SPMD, no qual as tarefas podem trocar mensagens entre si. Este formato utilizado para criação de

[1]P2PComp: SENGER, L. J. P2PComp. jul. 2011. Disponível em: <http://www.ljsenger.net/downloads.html>

[2]JXSE: The Java Implementation of the JXTA Protocols. fev. 2012. Disponível em: <http://jxse.kenai.com/>

Figura 3: Principais Componentes P2PComp



Adaptado de: Senger, Souza e Foltran (2011)

aplicações paralelas é similar ao utilizado pelo MPI (BURNS; DAOUD; VAIGL, 1994) e PVM (BEGUELIN *et al.*, 1994).

Código Fonte 1: Simples código P2PComp com troca de mensagens

```
1  public class HelloWorld extends Task {
2      public void run() {
3          if(getRank() == 0) { //Tarefa Mestre
4              int size = getSize();
5              int i = 1;
6              while(i < size) {
7                  IPCMessage msg = recv(-1, -1);
8                  i++;
9              }
10             Vector results = new Vector();
11             results.add(i);
12             sendResults(results);
13         }
14         else { //Tarefa Trabalhadora
15             Vector message = new Vector();
16             message.add(new String("Hello world from "+ getRank()));
17             send(0, 1, message);
18         }
19     }
20 }
```

Na linha 3, Código Fonte 1, é visualizado o teste condicional onde é determinado que caso o *rank* da tarefa seja igual a 0 (zero), esta tarefa será responsável por ser a tarefa

mestre da aplicação, código mestre. Se o *rank* for diferente de 0 (zero) esta tarefa vai executar o código trabalhador. O *rank* das tarefas é distribuído de maneira aleatória pelo P2PComp, o *rank* 0 (zero) é o mais importante por ficar responsável pelo gerenciamento, distribuindo as atividades e enviando os resultados ao par proprietário.

Dentro do *framework* P2PComp, existem os pares: Rendezvous/Relay que tem a função de armazenar as informações de anúncios e informações de localização, como tabelas de roteamento, o par proprietário que é responsável por criar a tarefa que será distribuída e processada pelos demais pares da rede, o par controlador, ou *rank* 0 e os pares trabalhadores. Estes pares podem estar distribuídos em um mesmo computador ou em computadores diferentes.

3.3 Weka

Neste trabalho foi utilizada a ferramenta Weka[3] versão: 3.6.4, desenvolvida pelo grupo de pesquisa em aprendizagem de máquina da Universidade de Waikato, Nova Zelândia. A ferramenta é escrita em Java, código aberto, testado em multi-plataformas (Linux, Windows e Machintosh) e distribuído sob os termos da Licença Pública Geral - GNU GLP (HALL *et al.*, 2009).

A ferramenta fornece um ambiente uniforme e a implementação de um conjunto de algoritmos de aprendizado de máquina para tarefas de mineração de dados. Projetada para que o usuário possa aplicar rapidamente os métodos existentes em novos conjuntos de dados de forma flexível. A Weka implementa algoritmos para pré-processamento de dados, pós-processamento de dados, classificação, regressão, agrupamento, regras de associação e visualização (WITTEN; FRANK; HALL, 2011).

Para utilizar os diferentes algoritmos de MD disponíveis no Weka, é preciso uniformizar os dados de entrada que devem estar na forma de uma única tabela relacional, arquivo ARFF, que pode ser obtido através da transformação de conjunto de dados a partir de um arquivo, gerado por uma consulta do banco de dados ou transformado manualmente pelo usuário (BOUCKAERT *et al.*, 2010).

A partir da ferramenta Weka, foi desenvolvida a aplicação denominada *Fast Weka*, que implementa os algoritmos de aprendizagem de máquina do Weka de modo concorrente (execução multi-*threads*) e também de modo P2P (*framework* P2PComp), para reduzir o tempo de resposta das tarefas de MD.

[3]Machine Learning Group at the University of Waikato. Weka 3: Data Mining Software in Java. mar. 2012. Disponível em: <http://www.cs.waikato.ac.nz/ml/weka/>

3.4 Tipos de Coberturas Florestais

Para os experimentos realizados neste trabalho, foi utilizado o conjunto de dados de tipos de coberturas florestais[4], composta por 581.012 registros e 55 atributos conforme descrito na Tabela 1.

Tabela 1: Atributos

Atributo	Tipo de Dado	Medição	Descrição
Elevação	Quantitativo	Metros	Elevação em metros
Aspecto	Quantitativo	Azimute	Azimute a partir do norte
Declive	Quantitativo	Graus	Inclinação em graus
Incidência Solar 9h	Quantitativo	0 à 255	Incidência solar às 9:00 no solstício de verão
Incidência Solar 12h	Quantitativo	0 à 255	Incidência solar às 12:00 no solstício de verão
Incidência Solar 15h	Quantitativo	0 à 255	Incidência solar às 15:00 no solstício de verão
Distância Horizontal para Água	Quantitativo	Metros	Distância Horizontal ao mais próximo recurso de Água
Distância Vertical para Água	Quantitativo	Metros	Distância Vertical ao mais próximo recurso de Água
Distância para Estrada	Quantitativo	Metros	Distância para mais próximo estrada
Distância ponto de ignição	Quantitativo	Metros	Distância ao mais próximo ponto de incêndio, histórico
Área selvagem	Binário	0 e 1	Quatro colunas, um para cada área
Tipo de Solo	Binário	0 e 1	Quarenta colunas, tipos de solo
Tipo de Cobertura Florestal	Quantitativo	1 à 7	Sete tipos de coberturas florestais

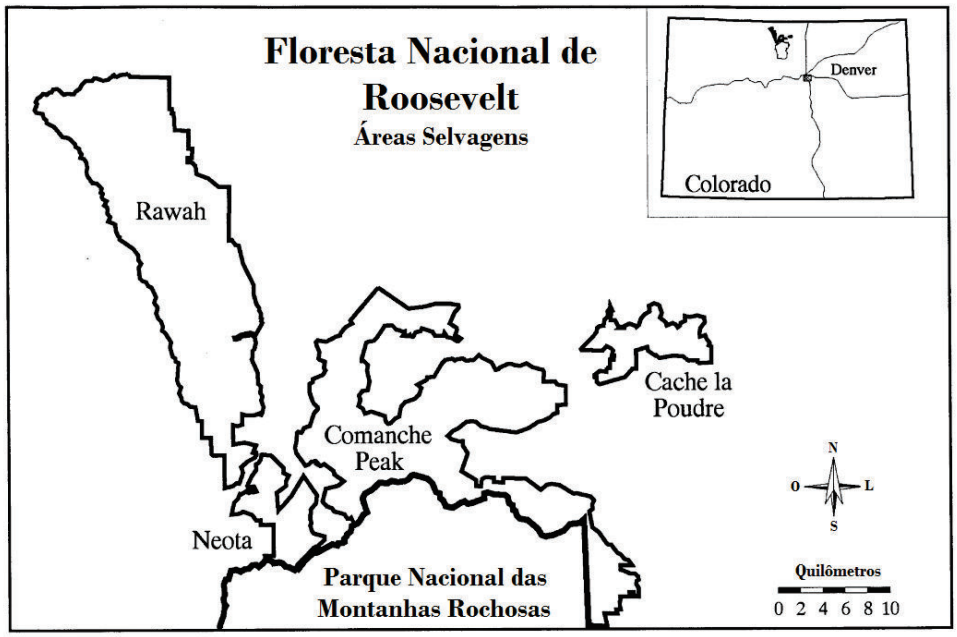
Fonte: O autor

Estes dados são constituídos pelas áreas: Rawah (29.628 hectares), Comanche Peak (27.389 hectares), Neota (3.904 hectares) e Cache la Poudre (3.817 hectares), localizadas a aproximadamente 113km a noroeste da cidade de Denver - Colorado, Estados Unidos, pertencentes a Floresta Nacional Roosevelt, conforme Figura 4. Estas áreas florestais foram selecionadas por Blackard e Dean (1999) devido as mesmas terem sofrido relativamente poucas alterações, pouca influência humana e por representar uma composição de tipos de coberturas florestais resultante do processo evolutivo ecológico natural (BLACKARD; DEAN, 1999).

As variáveis foram alcançadas a partir de dados espaciais digitais obtidos da

[4]Base de Dados: BLACKARD, J. A. Covertype Data Set. fev. 2012. Disponível em: <http://archive.ics.uci.edu/ml/datasets/Covertype>

Figura 4: Área Correspondente aos Dados



Adaptado de: Blackard e Dean (1999)

USGS - *United States Geological Survey* (Pesquisa Geológica dos Estados Unidos), e do USFS - *United States Forest Service* (Serviço Florestal dos Estados Unidos).

A elevação foi obtida a partir dos dados do DEM - Digital Elevation Model (Modelo Digital de Elevação) do USGS, com base em células matriciais (*Raster/Bitmap*) de 30x30 metros, abrangendo 52.291 hectares, 522.910.000 metros quadrados. Estes dados foram utilizados como padrão para os demais dados restantes, de modo que cada amostra representa uma célula matricial dos dados do USGS DEM (BLACKARD; DEAN, 1999).

Aspecto, declive, e os outros três atributos relativos a incidência de luz solar, incidência solar as 9, 12 e 15 horas, foram construídos a partir do DEM usando análise de superfície padrão baseada em GIS - *Geographic Information System* (Sistemas de Informação Geográfica)[5].

Os dados da distância horizontal até o mais próximo recurso de águas superficial e a distância horizontal até a rodovia mais próxima foram alcançados por meio da aplicação de análises de distância euclidiana dos dados hidrográficos e os dados de rodovias do USGS. A distância horizontal até o ponto mais próximo de ignição de incêndio, histórico, foi determinado a partir de análises da distância euclidiana dos pontos de ignição de incêndio do USFS, utilizando os pontos de ignição de incêndio florestal que ocorreram nos últimos 20 anos, 1975 - 1995. A distância vertical mais próxima dos recurso de águas

[5]Environmental Systems Research Institute. dez. 2012. Disponível em: <http://www.esri.com/>

superficiais foram calculadas utilizando a combinação do DEM, dos dados hidrológicos, e de um programa de análise espacial simples (BLACKARD; DEAN, 1999).

Os quatro atributos de área selvagem, considerando um atributo para cada região, referentes a classificação da área como selvagem ou não selvagem, foram obtidos a partir da USFS. Estas variáveis foram tratadas como valores binários, no qual o valor '0' representa ausência e o valor '1' representa presença de uma área selvagem. Os outros quarenta atributos referentes ao tipo de solo também foram tratados como binários, totalizando quatro áreas selvagens e quarenta variáveis de tipos de solos, formando um conjunto total de 54 variáveis independentes, disponíveis para cada modelo (BLACKARD; DEAN, 1999).

Os sete tipos de coberturas florestais utilizadas nesse estudo, que correspondem ao atributo meta são: classe 1 - Pinheiro/Coníferas - *Picea engelmannii* - *Abies lasiocarpa* composta por duas espécies, classe 2 - Pinheiro - *Pinus contorta*, classe 3 - Pinheiro - *Pinus ponderosa*, classe 4 - Algodão Americano/Salgueiro - *Populus angustifolia* - *Populus deltoides* - *Salix bebbiana* - *Salix amygdaloides* composta por quatro espécies, classe 5 - Caducifólia - *Populus tremuloides*, classe 6 - Coníferas - *Pseudotsuga menziesii* e classe 7 - Pinheiro/Coníferas - *Pinus aristata* - *Picea engelmannii* - *Abies lasiocarpa* composta por três espécies. Conforme observado por Blackard e Dean (1999), tais classes de tipos de coberturas florestais foram selecionadas por representarem as espécies dominantes principais encontradas nas quatro regiões. Alguns outros tipos de cobertura florestal estão presentes na região, porém em menor escala e portanto foram ignoradas no estudo.

3.5 Seleção dos dados e RNA

Para esta dissertação, inicialmente todas as observações de tipos de coberturas florestais foram agrupadas em um único conjunto de dados. Em seguida, foram extraídos do mesmo sete subconjuntos de dados, os quais correspondem aos sete tipos de coberturas florestais. Para realizar a segmentação do conjunto dos dados foi utilizado o software conhecido como R[6]. O R é definido como um ambiente e linguagem de programação estatístico e gráfico, a partir dele é possível analisar e manipular dados e construir desde programas simples até programas mais complexos (R, 2010).

Após a divisão da base de dados, foi iniciado o processo de seleção dos dados entre cada um dos sete tipos de coberturas florestais. A partir da análise das amostras de dados, foi observado que apenas dois tipos de coberturas florestais representam cerca de 85,22% de todos os dados, classe 1 e classe 2. Assim, para criar um modelo mais homogêneo, foi

[6]R Foundation for Statistical Computing. mai. 2012. Disponível em: <http://www.Rproject.org>

determinado que o conjunto de dados de teste e validação seria construído com o mesmo número de observações para cada um dos tipos de cobertura florestal.

A partir da classe com menor representatividade na base de dados, classe 4 - Algodão Americano/Salgueiro com 2.747 observações, sendo todos os dados selecionados, foram extraídas das outras seis classes de tipos de cobertura florestal 2.747 observações, selecionadas aleatoriamente com o auxilio do software R, totalizando o conjunto de teste com 19.229 observações, conforme a Tabela 2.

Tabela 2: Observações Selecionadas

Tipo de Cobertura Florestal	Total de Observações	Observações Selecionadas	Porcentagem por Tipo de Cobertura
Classe 1	211.840	2747	1,29%
Classe 2	283.301	2747	0,97%
Classe 3	35.754	2747	7,68%
Classe 4	2.747	2747	100%
Classe 5	9.493	2747	28,94%
Classe 6	17.367	2747	15,82%
Classe 7	20.510	2747	13,39%
Total	**581.012**	**19.229**	**3,31%**

Fonte: O autor

Após a seleção dos dados, foi aplicado um algoritmo de aprendizagem de máquina conhecido como RNA de multi-camadas (*Multi LayerPerceptron*) para uma classificação supervisionada, afim de criar um modelo para poder determinar as observações dentre cada uma das sete classes de tipos de coberturas florestais (FAYYAD *et al.*, 1996).

Segundo Haykin (1998), modelos de RNA requerem vários parâmetros arquitetônicos e de treinamento a serem selecionados antes da análise. O número ideal de camadas ocultas (*Hidden Layers*) e o número de neurônios por camada oculta não são inicialmente conhecidos para um conjunto de dados específicos, e devem ser determinados empiricamente.

Blackard e Dean (1999) utilizaram somente uma camada oculta, para essa camada oculta eles realizaram uma analise do erro médio quadrático em experimentos com: 6, 12, 18, 24, 30, 60, 90, 120, 150, 180, 210, 240, 270 e 300 neurônios, sendo verificado que a melhor escolha para essa base de dados seriam 120 neurônios, nessa única camada oculta.

Além dos parâmetros que definem a arquitetura da rede, parâmetros de formação também são necessários para inicializar o algoritmo de aprendizagem (HAYKIN, 1998). O algoritmo de correção de erro utilizado foi o *Backpropagation*. O mesmo requer dois parâmetros de inicialização, denominados taxa de aprendizado (*Learning Rate*) e taxa de

momento (*Momenthum Rate*). Não é possível saber inicialmente os valores ideais para esses parâmetros dado um conjunto específico de dados, assim foram utilizadas as configurações observadas por Blackard e Dean (1999) que obtiveram um melhor resultado em seu trabalho, que são 0.05 para a taxa de aprendizado e 0.5 para taxa de momento.

Uma vez que a arquitetura de rede e os parâmetros de treinamento são selecionados, uma RNA é treinada de forma iterativa. Cada iteração representa uma passagem completa através de um conjunto de dados de treinamento, uma época, também denominado *Training Time* (HAYKIN, 1998). O número de épocas também não sendo conhecido inicialmente, foi determinado a utilização de 1000 épocas, conforme utilizado por Blackard e Dean (1999).

Para realizar a validação do modelo foi utilizado a técnica de validação cruzada que consiste em dividir os dados em N partes, *folds*, utilizando duas configurações de *folds*, 10 *folds*, que segundo Bouckaert *et al.* (2010) é utilizado como padrão na validação cruzada e 24 *folds* por ser múltiplo do número de núcleos presentes para realização do experimento, utilizados para demonstrar tanto os resultados na mineração quanto para computação paralela.

Após definidos todos os parâmetros para a execução da RNA, foi utilizado o *software* desenvolvido neste projeto, *Fast Weka*, para obtenção dos resultados das execuções dos classificadores sobre diferentes configurações de *folds* e também diferente número de núcleos e pares.

3.6 Análise dos Tempos de Execução

Realizada a seleção dos dados, definições referentes a arquitetura, parâmetros da RNA e treinamento, partiu-se para as análises com referência direta ao objetivo desde trabalho. Os experimentos realizados para obtenção dos tempos de execução de cada teste foram realizados utilizando seis computadores homogêneos, do LCAD - Laboratório de Computação de Alto Desempenho[7], com a seguinte configuração:

- Processador Intel core I7 - 2600 com quatro núcleos de 3,4 Ghz e 4 núcleos virtuais, com tecnologia *Turbo Boost* habilitada[8];

[7]LCAD - Laboratório de Computação de Alto Desempenho, UEPG - Universidade Estadual de Ponta Grossa

[8]Tecnologia que aumenta o desempenho do processador dinamicamente. nov. 2012. Disponível em: <http://www.intel.com.br/content/www/br/pt/architecture-and-technology/turbo-boost/turbo-boost-technology.html>

- Memória RAM de 4,0 GB DDR3 com barramento 1333Mhz;
- Disco rígido de 500 GB 7200 rpm;
- Sistema Operacional Linux Slackware 13.37, *kernel* 2.6.37.6.

Esses experimentos foram organizados em etapas, sendo determinado que todas as execuções seriam realizadas ao menos cinco vezes. Inicialmente, foram realizadas as execuções da mineração de dados utilizando 10 e 24 *folds*, com a versão padrão do Weka 3.6.4, pois o algoritmo é executado de modo sequencial, e esses resultados serviram como amostras de comparação para verificar os ganhos obtidos com a ferramenta *Fast Weka*.

Após a execução sequencial, partiu-se então para a análise das técnicas de paralelismo, *Fast Weka*. Na primeira etapa de execução foi utilizado modo multi-*threads*, visando utilizar o paralelismo em nível de máquina, cada computador. Tendo em vista que hoje há diversas tecnologias que implementam múltiplos núcleos no mesmo processador (CHAI; GAO; PANDA, 2007), como os processadores: Core 2 Duo, Core 2 Quad, I3, I5, I7, fabricados Intel[9], Phenom II, Phenom X3, Phenom X4, Athlon 2 X2, Athlon X2 fabricados pela AMD[10].

A execução no modo multi-*threads* foi realizada utilizando 10 e 24 *folds*, sendo que para essas duas configurações foi utilizado diferentes configurações de núcleos, visando utilizar os núcleos reais e virtuais dos computadores disponíveis. As execuções foram realizadas utilizando 2, 4, 6 e 8 *threads*.

Na segunda etapa, a execução dos experimentos realizados de modo P2P (P2PComp) também foram realizados utilizando 10 e 24 *folds*, com diferentes números de pares na rede. As execuções foram realizadas utilizando 3, 5, 7, 9 e 11 pares quando executado com 10 *folds* e 3, 5 ,7, 9 e 13 pares quando executado com 24 *folds*, considerando apenas núcleos reais[11]. Para a distribuição dos pares na rede foi determinado que seriam distribuídos todos os pares ao maior número de computadores possível, utilizando o menor número de pares por computador.

Os experimentos realizados estão apresentados na Tabela 3, na qual estão relacionadas as configurações de *folds*, número de núcleos ou pares por experimento e o total

[9]Intel Processadores. dez. 2012. Disponível em: <http://ark.intel.com/pt-br#desktopprocessors>

[10]Processadores AMD para PCs desktop. dez. 2012. Disponível em: <http://www.amd.com/br/products/desktop/processors/Pages/desktop-processors.aspx>

[11]Núcleo real é aquele que existe fisicamente, núcleo virtual é aquele que é simulado a partir de um núcleo físico, denominados de núcleos lógicos, como implementado pela tecnologia *Hyper-Threading*.

de observações realizadas por tipo de execução: sequencial, modo multi-*threads* e modo P2P.

Tabela 3: Experimentos Realizados

Número de *Folds*	Sequencial	Modo multi-*threads*	Modo P2P
10	1	2,4,6,8	3,5,7,9,11
20	1	2,4,6,8	3,5,7,9,13
Total	**10**	**40**	**50**

Fonte: O autor

Após obtidos todos os tempos de execução dos teste supracitados, foram aplicadas diferentes análises para determinar os ganhos obtidos. Análises de *speedup* e eficiência, e analises estatísticas descritivas para determinar se existe diferença significativa nos tempos obtidos, todas através do software R. Inicialmente foi aplicado o teste de Shapiro-Wilk[12], para verificar a normalidade dos dados. O resultado principal obtido com esse teste é o valor p. Caso o valor p seja maior que 0,05, o teste indica que os dados são distribuídos de acordo com a curva normal (MAROCO, 2011).

As medidas descritivas média, variância e desvio padrão foram calculados para cada grupo de amostras. Os testes T, ANOVA e teste estendido de Tukey foram adotados para determinar a ocorrência de diferenças entre os grupos, considerando um intervalo de confiança de 95% (SCHEFLER, 1988).

[12]Teste de Shapiro-Wilk. out. 2012. Disponível em: <http://www.portalaction.com.br/content/64-teste-de-shapiro-wilk>

4 RESULTADOS E DISCUSSÕES

4.1 Considerações Iniciais

Este capítulo apresenta os resultados obtidos através deste trabalho afim de melhorar o desempenho das atividades de mineração de dados agrícolas. A Seção 4.2 apresenta a ferramenta de mineração de dados desenvolvida nesta pesquisa denominada de *Fast Weka*, utilizando técnicas de programação concorrente (*threads*) e também redes P2P (P2PComp). A Seção 4.3 mostra os diferentes tempos de execução, *speedup* e eficiência obtidos através da ferramenta, em diferentes configurações de teste. A Seção 4.4 apresenta os resultados da mineração de dados agrícolas utilizando a técnica de redes neurais artificiais.

4.2 *Fast Weka*

Para a construção da ferramenta de mineração de dados em paralelo, *Fast Weka*, definiu-se que o paralelismo seria implementado em nível de validação, *folds*, tendo em vista que as técnicas de validação cruzada são mais confiáveis para a realização da validação de dados na tarefa de mineração.

Inicialmente foi desenvolvida a versão de modo multi-*threads* da ferramenta. A classe principal modificada para que esta funcionalidade fosse implementada foi a classe: *ClassifierPanel*, que pertence ao pacote: *weka.gui.explorer*. Foi adicionada também a classe: *RunLocal*. Por meio da classe *ClassifierPanel* é obtido o número de *folds* que o usuário deseja utilizar, e a partir deste número são criados os N classificadores.

Após criados os N classificadores, *folds*, são criadas as N *threads* paralelas de acordo com o número de processos paralelos concorrentes ao computador, determinados pelo usuário ao iniciar a ferramenta, conforme Apêndice B, instâncias da classe *RunLocal*, assim a ferramenta instancia as N *threads* e distribuí por demanda para estas instâncias os *folds* para serem processados.

Finalizado o desenvolvimento da ferramenta no modo multi-*threads*, foi iniciado o desenvolvimento no modo P2P, através do *framework* P2PComp. Conforme supracitado, por meio da classe *ClassifierPanel* são divididos os *folds* e classificadores e na tela inicial do sistema são determinados os números de pares da rede. Pada a realização da mineração de modo P2P, foram adicionadas as classes: *Dados*, *StartClientPeer*, *DataProcess* e *RunParallel*.

A classe *StartClientPeer* recebe uma instância da classe *Dados* instanciada pelo *ClassifierPanel* com todas as informações necessárias para a execução da mineração. Por meio da *StartClientPeer*, o par proprietário da tarefa conecta-se a rede P2P, cria um novo *Job* (aplicação) e envia as tarefas que compõem a aplicação ao número de pares determinados. Para conectar na rede são utilizadas as configurações definidas pelo usuário na tela inicial do programa, Apêndice B.

A tarefa enviada aos pares da rede é apresentada conforme Código Fonte 2.

Código Fonte 2: Classe principal do paralelismo com P2PComp.

```
1  public class RunParallel extends Task {
2      private int response = 3, request = 7, finish = 5, send = 9;
3      @Override
4      public void run() {
5          if(this.getRank() == 0) {
6              Dados dados = (Dados) this.getObject(0);
7              Classifier[] classifiers = dados.getClassifiers();
8              Instances[] trainParallel = dados.getTrainParallel();
9              int numFolds = classifiers.length;
10             int numPeers = (dados.getNumPeers() - 1);
11             int foldsRecebidos = 0, peersFinalizados = 0, foldsEnviados =
                   0;
12             while(true) {
13                 IPCMessage ret = this.recv(-1, -1);
14                 if(ret.getTag() == request) {
15                     if(foldsEnviados < numFolds) {
16                         try {
17                             send(ret.getRank(), send, new DataProcess(
                                   foldsEnviados, Classifier.makeCopy(
                                   classifiers[foldsEnviados]), trainParallel[
                                   foldsEnviados]) );
18                             foldsEnviados++;
19                         } catch (Exception ex) {
20                         }
21                     }
22                     else {
23                         send(ret.getRank(), finish, null);
24                         peersFinalizados++;
25                     }
26                 }
27                 else if(ret.getTag() == response) {
28                     DataProcess result = (DataProcess) ret.getPayLoad();
29                     try {
```

```
30                          classifiers[result.getNumFold()] = Classifier.
                                makeCopy(result.getClassifiers());
31                          foldsRecebidos++;
32                      } catch (Exception ex) {
33                      }
34                  }
35                  if((peersFinalizados == numPeers) && (foldsRecebidos ==
                        numFolds) ) {
36                      break;
37                  }
38              }
39              sendResults(dados);
40          }
41          else {
42              while(true) {
43                  send(0, request, null);
44                  IPCMessage ret = this.recv(-1, -1);
45                  if(ret.getTag() == send) {
46                      DataProcess result = (DataProcess) ret.getPayLoad();
47                      try {
48                          result.getClassifiers().buildClassifier(result.
                                getTrainParallel());
49                          send(0, response, result);
50                      } catch (Exception ex) {
51                      }
52                  }
53                  else if(ret.getTag() == finish) {
54                      break;
55                  }
56              }
57          }
58      }
59  }
```

A classe *RunParallel* apresentada acima é responsável pelo processamento no modo P2P. Na linha cinco é realizada a verificação para determinar se o par é o *rank* 0 ou não. Definiu-se que o *Rank* 0 será responsável por realizar todo o controle de troca de mensagens entre os pares que estão executando a aplicação. Ele recebe uma instância da classe *Dados*, recupera deste mesmo objeto as instâncias dos classificadores e instâncias de treinamento e começa a realizar a distribuição para os pares processarem esta aplicação.

A distribuição é realizada a todos os pares que solicitaram alguma atividade

para processamento, conforme a linha 43, um par qualquer da rede solicita ao *rank* 0 informações para serem processadas. Estas informações são enviadas aos *ranks* solicitantes conforme apresentado na linha 17. Os *ranks* recebem as informações e iniciam o processo de classificação conforme linha 48.

Terminada a execução dos *folds*, os pares retornam os classificadores processados ao *rank* 0. Após todos *folds* serem processados o *rank* 0 envia os resultados do processamento ao par proprietário do *job*, observado na linha 39. E a tarefa de mineração em paralelo chega ao fim.

4.2.1 Alterações P2PComp

É importante apresentar as alterações realizadas no *framework* P2PComp que aumentaram seu desempenho, reduzindo taxas de comunicação, processamento e uso de memória.

Uma das alterações realizadas para reduzir o uso de memória do *framework* foi a remoção de classes que realizavam a conexão entre classes para troca de mensagens, pois estas classes intermediárias não possuíam um papel significativo e podem ser removidas. Algumas outras classes também não utilizadas na dissertação, remanescentes de versões anteriores do *framework*, foram descartadas.

As classes utilizadas para recebimento de mensagens foram reescritas. Anteriormente, o projeto era composto por duas classes para recebimento de mensagens, *IPC-Manager* e *Peer* (SENGER; SOUZA; FOLTRAN, 2011). As duas classes realizam a mesma atividade, recebimento de mensagem, porém recebendo mensagens de tipos diferentes. Foi observado que não há necessidade dessas duas classes e que apenas a classe *Peer* pode realizar esta tarefa, verificando apenas o tipo de mensagem recebida e removendo assim a classe *ICPManager* e suas dependentes.

Para o envio de mensagens as classes: *Messenger* e *ICPMessenger* (SENGER; SOUZA; FOLTRAN, 2011) haviam sido implementadas segundo padrão *Singleton*, do padrão de projeto *Design Patterns*. Esse padrão de projeto determina que apenas uma instância da classe que utiliza o padrão seja instanciada. A utilização desse padrão permite reduzir muito o consumo de memória, mas pode aumentar o processamento para realizar a sincronização das variáveis e métodos (PAPAJORGJI; PARDALOS, 2009). Desse modo, as classes poderiam enviar apenas uma mensagem por vez.

Com a remoção do padrão *Singleton* das classes de comunicação, as classes foram

implementadas estendendo a classe *Thread*, para que as mensagem fossem enviadas de forma simultâneas, aumentando o processamento concorrente e reduzindo significativamente o tempo de comunicação entre processos, tendo em vista que diversas mensagens podem ser enviadas concomitantemente.

A próxima subseção apresenta os ganhos obtidos com as duas implementações de paralelismo, comparando-as com a execução sequencial.

4.3 Avaliação de Desempenho da Ferramenta *Fast Weka*

A seguir são apresentados os resultados obtidos com as duas configurações de *folds*, 10 e 24, executados de modo concorrente (multi-*thread*) e modo P2P (P2PComp), comparados com o modo sequencial. As estatísticas descritivas média e desvio-padrão são apresentadas, e os valores amostrados estão apresentados no Apêndice A.

4.3.1 *Fast Weka* modo Multi-*threads*

Por meio da diferença do tempo médio de execução dos grupos 1 e 2 observados na Tabela 4, modo sequencial e modo multi-*threads*, 2 *threads*, observa-se que o tempo médio de execução foi reduzido de 301,92 para 157,21 minutos. A diferença de 6,25 minutos, a qual não permitiu obter uma eficiência igual a um, podendo ser justificada pela tecnologia implementada nos processados utilizados para o teste (*Turbo Boost*, conforme citado no Capítulo 3).

Tabela 4: Resumo dos resultados modo multi-*threads* - 10 *Folds*

Grupo	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5
Experimento	Sequencial	2-*Threads*	4-*Threads*	6-*Threads*	8-*Threads*
Média	301,92	157,21	98,21	81,44	90,63
Desvio Padrão	1,05	1,08	0,47	1,05	0,25
SpeedUp		1,92	3,07	3,71	3,33
Eficiência		0,96	0,77	0,62	0,42

Fonte: O autor

Comparando os resultados dos grupos 2 e 3, foi observado uma redução na eficiência, isto ocorreu devido ao fator da tecnologia, já citado, e também pelo fato da implementação do paralelismo ser realizado em nível de *folds*. Como neste experimento foram usados 10 *folds*, o grupo 2, no qual são utilizadas 2 *threads* são realizados 5 ciclos (número de *folds* dividido pelo número de *threads*) de processamento completos, sempre com as duas *threads* executando dois *folds*. Entretanto, no grupo 3 como são utilizadas

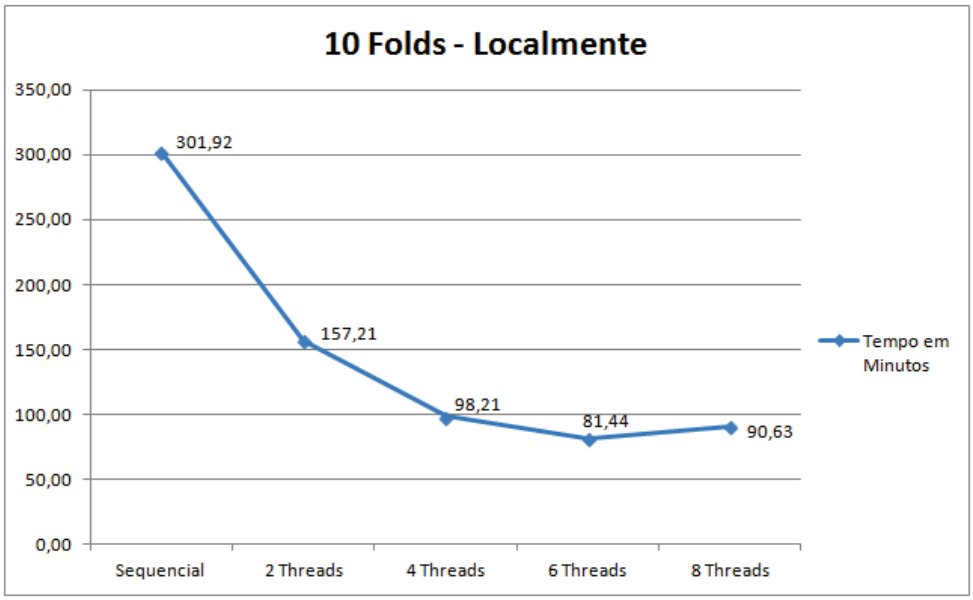
4 *threads*, sendo realizados 2 ciclos completos utilizando as 4 *threads*, totalizando 8 *folds* processados, e um ciclo extra utilizando apenas duas *threads* para completar a atividade, reduz-se assim a eficiência.

Nos grupos 4 e 5, os resultados obtidos demonstram uma redução da eficiência ainda mais significativa que nos outros grupos. Isso ocorre devido aos fatores já supracitados (tecnologia e método de implementação), e também pelo fato de que os computadores em questão possuem quatro núcleos reais e simulam oito núcleos, assim a eficiência não consegue ser mantida. Esse fato também foi observado por Souza *et al.* (2011) e Damiati, Souza e Senger (2012).

Mesmo a aplicação contemplando os núcleos virtuais, grupos 4 e 5 é possível visualizar que existe uma redução de tempo em relação ao grupo 3. Mesmo o grupo 4 implementando um número menor de *threads* em relação ao grupo 5, o grupo 4 foi mais rápido devido ao número de ciclos de execução ser o mesmo, dois ciclos, porém quando são usadas 8 *threads* a concorrência por recursos no computador é maior que quando usadas 6 *threads*.

Na Figura 5, é possível observar a curva de redução do tempo de execução utilizando 10 *folds*. Por meio das análises estáticas aplicadas sobre estes tempos, foi observado que todos os grupos apresentam diferença significativa entre eles, e o melhor resultado é obtido com 6 *threads*.

Figura 5: Tempo de execução modo multi-*threads*, 10 *Folds*



Fonte: O autor

Por meio dos resultados obtidos nos testes de modo multi-*threads* com 24 *folds*

(Tabela 5), pode-se observar o ganho obtido. Análises foram realizadas em conjunto com os experimentos realizados com 10 *folds*.

Observando a tabela dos resultados, a diferença entre os tempos médios de execução dos grupo 1 e grupo 2 são significativos, atingindo novamente uma eficiência próxima a um. A diferença entre os grupos 2 e 3 é menor que a diferença dos mesmo grupos utilizando 10 *folds*, isso ocorre devido a esses experimentos estarem sendo executados utilizando um número múltiplo de *folds* em relação ao número de núcleos, assim, sofrendo influência apenas da tecnologia do processador.

Com os grupos 4 e 5, novamente a redução na eficiência é muito maior devido as *threads* contemplarem os núcleos virtuais dos computadores, porém diferente dos resultados com 10 *folds*, ocorrendo uma redução do tempo de execução entre o grupo 4 e grupo 5, devido ao uso de uma quantidade de *folds* múltipla do número de *threads*.

Tabela 5: Resumo dos resultados modo multi-*threads* - 24 *Folds*

Grupo	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5
Experimento	Sequencial	2-*Threads*	4-*Threads*	6-*Threads*	8-*Threads*
Média	763,06	408,76	218,09	203,79	192,78
Desvio Padrão	3,94	1,73	0,44	3,51	1,67
SpeedUp		1,87	3,50	3,74	3,96
Eficiência		0,93	0,87	0,62	0,49

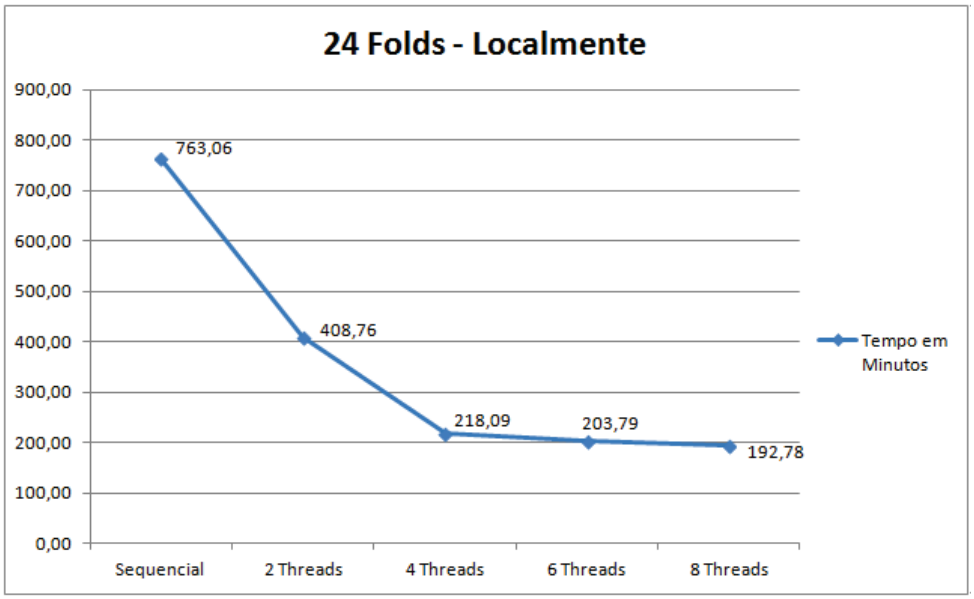
Fonte: O autor

A Figura 6 apresenta a curva de redução do tempo de execução utilizando 24 *folds*. Por meio das análises estatísticas aplicadas sobre estes tempos, foi observado que todos apresentam diferença significativa entre eles. Assim, é possível observar nas Figuras 5 e 6, que utilizando as técnicas de multi-*threads*, foi obtido um desempenho satisfatório, reduzindo-se o tempo da mineração de dados. A eficiência foi reduzida na medida que aumentam-se o número de *threads*.

Quando utilizados os núcleos virtuais dos computadores, mesmo tendo uma redução de eficiência, é possível reduzir os tempos de resposta. Observa-se que é mais indicado para a mineração de dados de modo multi-*threads* o uso de um número múltiplo de *threads* em relação ao número de *folds*.

Tendo em vista que o número de núcleos reais dos computadores utilizados é 4 e quando utilizado 6 e 8 *threads* este número se mantêm, o cálculo da eficiência para 6 e 8 *threads* poderia ser realizado utilizando o valor 4 para o número de elementos de processamento. Pode-se assim obter uma eficiência melhor do que apresentado anteriormente

Figura 6: Tempo de execução modo multi-*threads*, 24 *Folds*



Fonte: O autor

devido aos cálculos serem realizado com os valores 6 e 8.

4.3.2 *Fast Weka* modo P2P

Utilizando o modo P2P foram observados diversos resultados obtidos por meio dos experimentos que avaliaram o seu desempenho. Conforme citado no Capítulo 3, para os experimentos utilizando o modo P2P, a distribuição dos pares por computador foi realizada utilizando o maior número possível de computadores com o menor número de pares por computador.

Esta distribuição foi determinada sabendo da tecnologia existente nos computadores utilizados, *Turbo Boost*, e por meio dos experimentos realizados. Na Tabela 6 são apresentados os resultados da análise utilizando 5 pares de modo P2P, sendo o grupo 2 utilizando os pares distribuídos entre vários computadores, e o grupo 3, os cinco pares utilizando o mesmo computador, utilizando 5 pares devido a um dos pares assumir o *Rank* 0 que realizar somente a distribuição dos *folds*.

Observando os resultados, foi verificada uma redução de 3,66 minutos quando utilizados pares de modo distribuído em diferentes computadores, mesmo ocorrendo um aumento significativo da comunicação. O tamanho da base de dados é igual a 2,38 MB e é distribuída a todos os pares da rede, aumentando a comunicação entre processos. Esse resultado pode ser atribuído ao fator da tecnologia dos processadores e a menor concorrência por recursos internos do computador, memória principal e disco.

Tabela 6: Resumo da comparação do Tempo de Execução com P2P - 10 *Folds*

5 Distribuídos X 5 Pares Localmente			
Grupo	Grupo 1	Grupo 2	Grupo 3
Experimento	Sequencial	Distribuído	Localmente
Média	301,92	91,22	94,88
Desvio Padrão	1,05	1,10	1,01
SpeedUp		3,31	3,18
Eficiência		0,66	0,64

Fonte: O autor

Os valores indicam uma redução de aproximadamente 3,85% do tempo total quando os pares estão distribuídos. Esta diferença foi comprovada como significativa através da análise estatística teste T de Student, por meio da qual foi rejeitada a hipótese nula, a qual indica a semelhança entre as médias. Assim, observa-se melhores resultados quando os pares estão distribuídos em diferentes computadores.

Para a distribuição utilizando 10 *folds* no modo P2P, foi observada uma melhoria de desempenho com relação ao uso do modo multi-*threads* (quando usados os mesmo números de aplicações concorrentes realizando o processamento dos *folds*). Os resultados são apresentados na Tabela 7.

Quando comparados os grupos 1 e 2 (3 pares), verifica-se uma redução de mais da metade do tempo de execução, 301,92 para 146,07 minutos. A eficiência deste resultado é igual 0,69. Isso deve-se ao fato de que ao utilizar o modo p2p, adiciona-se um par a mais na rede para ficar responsável pelo *rank* 0, porém existem efetivamente dois pares processando as atividades de mineração. Além do *rank* 0, existe também o par proprietário da tarefa de mineração de dados, porém o mesmo não foi considerado para o cálculo da eficiência.

De maneira geral, a medida que aumenta-se o número de pares, aumenta a eficiência. O *rank* 0, que não realiza processamento, começa a representar uma porcentagem cada vez menor para o calculo da eficiência. Considerando 3 pares, o *rank* 0 representa 33% do calculo da eficiência, já considerando 5 pares o mesmo representa 20%, assim, na medida que aumentam os pares, a eficiência aumenta. Esta melhoria ocorre até certo ponto, pois a comunicação entre processos aumenta na medida que são adicionados pares. É importante ressaltar que esta particularidade ocorre devido ao *rank* 0 não processar nenhuma atividade e participar do cálculo de eficiência, e com o aumento do número de pares a eficiência diminui devido as taxas de comunicação. Se desconsiderado o *rank* 0, a eficiência observada inicia elevada e na medida que aumentam o número de pares a

eficiência deve diminuir devido à comunicação entre processos da rede.

Tabela 7: Resumo dos resultados modo P2P - 10 *Folds*

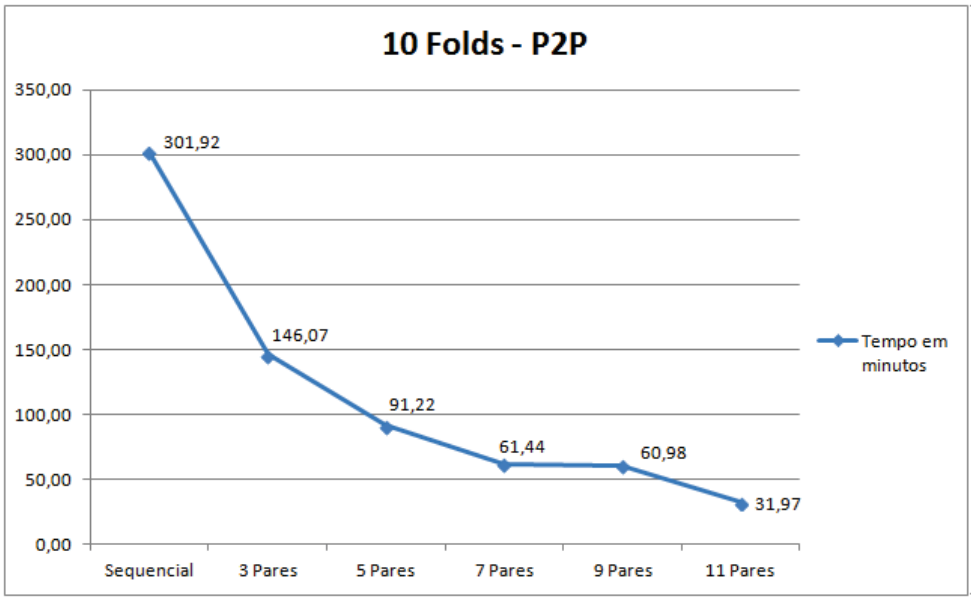
Grupo	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5	Grupo 6
Experimento	Sequencial	3-Pares	5-Pares	7-Pares	9-Pares	11-Pares
Média	301,92	146,07	91,22	61,44	60,98	31,97
Desvio Padrão	1,05	0,88	1,10	0,69	0,75	0,82
SpeedUp		2,07	3,31	4,91	4,95	9,44
Eficiência		0,69	0,66	0,70	0,55	0,86

Fonte: O autor

Deixando de lado a eficiência e observando os tempos médios de execução dos grupos, de modo geral, na medida que aumentam os pares, o tempo de resposta de execução diminui, porém entre os grupos 4 e 5, a diferença é de apenas 0,46 minutos e após análises estatísticas, pode-se dizer que estes valores não representam diferenças significativas, sendo considerados iguais. Isso ocorre devido ao fato do paralelismo estar relacionado ao número de *folds*.

Quando utilizados 7 pares, 6 *folds* são processados paralelamente, e na segunda rodada de processamento 4 *folds*, totalizando 2 ciclos para finalizar a tarefa. Ao utilizar 9 pares, 8 *folds* serão processados paralelamente, e na segunda rodada de processamento 2 *folds*, também totalizando 2 ciclos para realizar a atividade. Portanto, quando o número de ciclos é igual, o tempo é significativamente igual, quando as tarefas de MD são executadas de modo P2P utilizando apenas os núcleos reais.

Figura 7: Tempo de execução modo P2P, 10 *Folds*



Fonte: O autor

A partir da Figura 7, é possível observar tais considerações. Observando a redução de tempo de execução, de modo sequencial, o processamento de 10 *folds* da atividade demandou 301,92 minutos ou aproximadamente 5 horas e 2 minutos e utilizando 11 pares, com 10 processadores executando a atividade de mineração, esse tempo foi reduzido para aproximadamente 32 minutos.

Na Tabela 8, pode-se observar essa tendência, em relação ao processamento paralelo utilizando o modo P2P para as atividades de mineração de dados.

Tabela 8: Resumo dos resultados modo P2P - 24 *Folds*

Grupo	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5	Grupo 6
Experimento	Sequencial	3-Pares	5-Pares	7-Pares	9-Pares	13-Pares
Média	763,06	378,10	192,83	132,24	100,53	72,70
Desvio Padrão	3,94	3,88	2,87	3,71	1,72	1,23
SpeedUp		2,02	3,96	5,77	7,59	10,50
Eficiência		0,67	0,79	0,82	0,84	0,81

Fonte: O autor

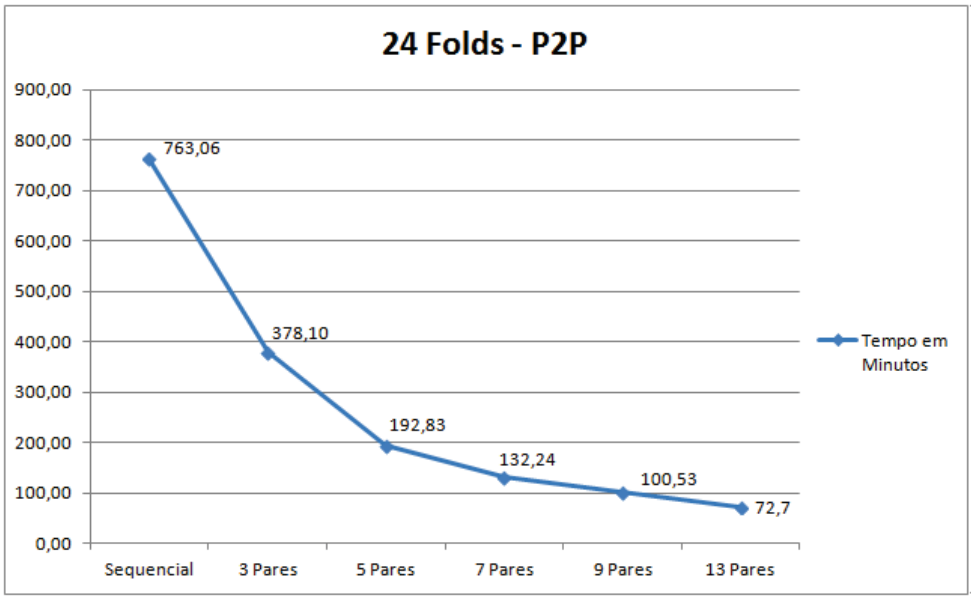
Os valores da eficiência são crescentes até o grupo 5 (9 pares). A eficiência aumenta do grupo 2 para o 3, e reduzindo este aumento até chegar a eficiência de 0,84 para o grupo 5, a partir da qual começa a ser reduzida novamente, devido ao *rank* 0 influenciar cada vez menos na eficiência, conforme ja citado, e as taxas de comunicação simultâneas aumentarem sua influência no tempo de resposta.

Este aumento do tempo de execução ocorre devido as taxas de comunicação de dados simultâneas. Foram observadas como um fator relevante, tendo em vista que através dos experimentos foi possível observar que com o aumento do número de pares na rede, as classes JXTA começam a perder desempenho. Este problema foi observado durante a execução de todos os experimentos em modo P2P desta dissertação.

Por meio da Figura 8, é possível observar a redução do tempo de execução das atividades. A partir dela é observado que a curva de tempo é mais uniforme se comparado a de 10 *folds*, devido ao fato de que sempre foi utilizado um número de pares múltiplo do número de *folds*.

Concluídas as análises dos tempos de execução das atividades de modo multi-*threads* e P2P. Na próxima subseção serão apresentados os resultados obtidos com relação as atividades de mineração de dados aplicados a base de dados de tipos de coberturas florestais.

Figura 8: Tempo de execução modo P2P, 24 *Folds*



Fonte: O autor

4.4 Mineração na Agricultura

Utilizando as técnicas de mineração de dados, RNA e o método de validação cruzada, aplicados ao conjunto de dados de tipos de coberturas florestais foram obtidos os resultados presentes na Tabela 9, expressos pelos índices de acerto, erro de classificação e índice Kappa.

Tabela 9: Precisão do Classificador

Resultado	10 *Folds*	24 *Folds*
Índice de Acerto	15833 - 82.33%	15921 - 82.79%
Índice de Erro	3396 - 17.66%	3308 - 17.20%
Índice Kappa	0.794	0.7993

Fonte: O autor

Observando os percentuais de acerto na utilização das duas diferentes configuração de *folds*, não se verifica uma grande diferença entre os resultados. Com 24 *folds* o classificador acertou 88 instancias a mais que utilizando 10 *folds*, que corresponde cerca de 0,45% a mais de precisão.

Blackard e Dean (1999), utilizando os mesmo parâmetros da RNA, obtiveram uma precisão de 70,58% com o modelo, porém não pode-se afirmar que os resultados obtidos neste trabalho são melhores que os observador por Blackard e Dean, devido aos fatores: seleção de dados e método de validação empregado.

Além do percentual de acerto, foi observado o coeficiente estatístico denominado índice Kappa ou Estatística K, definido como uma medida de concordância em escalas nominais. Neste contexto de classificação, verificando os altos valores do índice Kappa, podemos verificar o elevado nível de concordância entre a classificação do modelo e a classificação de referência, ou seja, o quão os dois estão de acordo quanto à classificação, sendo observado um maior valor para 24 *folds*.

Observando a Tabela 10, a área ROC, que retrata o desempenho de um classificador sem levar em conta os custos de distribuição ou de classe de erro, os resultados são expressivos, tendo em vista que todos os valores para área ROC estão superiores a 0.9.

Tabela 10: Detalhes da Precisão por Classe

Tipo de Cobertura	Área Roc		Precisão	
Florestal	10 *Folds*	24 *Folds*	10 *Folds*	24 *Folds*
Classe 1	0.941	0.942	0.749	0.761
Classe 2	0.924	0.925	0.687	0.704
Classe 3	0.963	0.965	0.805	0.798
Classe 4	0.995	0.996	0.927	0.926
Classe 5	0.987	0.985	0.876	0.861
Classe 6	0.973	0.974	0.776	0.798
Classe 7	0.994	0.994	0.927	0.929
Média	**0.968**	**0.969**	**0.821**	**0.825**

Fonte: O autor

Verificando a precisão obtida pelo modelo para cada classe na tabela 10, pode ser observado um elevado índice de acerto para todas as classes existentes no estudo. Verifica-se um aumento na precisão média quando utilizado a configuração de 24 *folds*, ocorrendo um aumento de precisão nas classes: 1, 2, 6 e 7, e uma redução para as classes 3, 4 e 5, registrando um aumento de 0.002 de precisão.

5 CONCLUSÕES

Devido ao continuo crescimento dos conjuntos de dados a serem processados e o elevado tempo de processamento para construção de modelos utilizando mineração de dados, o objetivo deste trabalho foi reduzir o tempo de resposta das atividades de mineração de dados aplicados a base de dados agrícolas.

Para atingir esse objetivo, a ferramenta *Fast Weka* foi desenvolvida e foram avaliados as técnicas de multi-*threading* e de execução em paralelo utilizando rede P2P. A utilização da ferramenta *Fast Weka* em modo de execução multi-*threads* permitiu obter uma redução de tempo satisfatória, sendo considerado como ideal o uso de aplicações concorrentes múltiplas do número de *folds* selecionados para a execução, sendo relevante a utilização de núcleos virtuais em arquiteturas com suporte a está tecnologia.

Por meio da ferramenta *Fast Weka*, em modo P2P, também ocorreu uma redução significativa de tempo, sendo ideal o uso de pares múltiplas do número de *folds* mais um, pois deve ser considerado o *rank* 0.

Na execução dos experimentos, foi observada a eficiência do uso do *framework* P2PComp para construção de aplicações de alto desempenho, tendo em vista a fácil implementação e a utilização do padrão SPMD.

O modelo de previsão de tipos de coberturas florestais criado utilizando as técnicas de RNA de validação cruzada apresentou um bom nível de confiabilidade atingindo uma confiabilidade de 82.79%.

É possível concluir também que a ferramenta desenvolvida é muito relevante tendo em vista que com o passar dos anos as bases de dados tende a aumentar seu volume de dados.

Assim, fica comprovado por meio dos resultados obtidos a viabilidade da utilização da ferramenta *Fast Weka* para melhorar o desempenho das atividades de mineração de dados agrícolas, bem como para a aplicação em outras áreas do conhecimento.

5.1 Trabalhos Futuros

Considerando os resultados supracitados outros trabalhos podem ser desenvolvidos. Abaixo são apresentados alguns trabalhos futuros que poderão ser desenvolvidos.

- Alterar o funcionamento do *framework* P2PComp para que o *rank* 0, também possa

executar atividades, aumentando a eficiência do *framework*.

- Estudar e analisar a ferramenta JXTA para melhorar o desempenho da troca de mensagens.
- Utilizar as mesmas configurações de RNA e base de dados utilizando a validação *leave-one-out* para construir outro classificador.
- Aplicar a ferramenta *Fast Weka* a diferentes bases de dados e diferentes técnicas de mineração de dados.

REFERÊNCIAS

ALMASI, G.; GOTTLIEB, A. *Highly Parallel Computing*. [S.l.]: Benjamin - Cummings Publishing Company Inc., 1994.

AMORETTI, M. *Peer-to-peer based grid architectures*. Tese (Doutorado) — Università Degli Studi Di Parma, Janeiro 2006.

ANDERSON, D. P. Boinc: A system for public-resource computing and storage. In: *GRID 04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004. p. 4 – 10.

BEGUELIN, A. *et al. PVM: Parallel Virtual Machine: User's Guide and tutorial for Networked Parallel Computing*. [S.l.]: MIT Press, 1994.

BLACKARD, J. A.; DEAN, D. J. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, v. 24, p. 131–151, 1999.

BOUCKAERT, R. R. *et al. WEKA Manual for Version 3-6-4*. Hamilton, New Zealand, Dezembro 2010.

BRAGA, A. P.; LUDERMIR, T. B.; CARVALHO, A. C. P. de L. F. *Redes Neurais Artificiais: Teoria e Aplicações*. Rio de Janeiro, RJ: LTC, 2007.

BURNS, G.; DAOUD, R.; VAIGL, J. Lam: An open cluster environment for mpi. *In Proceedings of Supercomputing Symposium*, p. 379–386, 1994.

CHAGAS, C. S. *et al.* Utilização de redes neurais artificiais na classificação de níveis de degradação em pastagens. *Revista Brasileira de Engenharia Agrícola e Ambiental*, v. 13, n. 3, p. 319 – 327, 2009.

CHAI, L.; GAO, Q.; PANDA, D. K. Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system. In: *Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007*. [S.l.: s.n.], 2007. p. 471 – 478.

COULOURIS, G. *et al. Distributed Systems Concepts and Design*. 5. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1988.

DAMIATI, F. V. F.; SOUZA, M. A. de; SENGER, L. J. Avaliação de desempenho computacional da mineração de dados pluviométricos com a utilização de computação paralela. In: MARINGá, U. E. de (Ed.). *2 Encontro Anual de Iniciação Tecnológica e Inovação*. [S.l.]: ANAIS DO 21º EAIC, 2012.

EL-TELBANY, M.; WARDA, M.; EL-BORAHY, M. Mining the classification rules for egyptian rice diseases. *The International Arab Journal of Information Technology*, v. 3, p. 303 – 306, 2006.

FAYYAD, U. M. *et al. Advances in Knowledge Discovery and Data Mining*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996.

GRADECKI, J. D. *Mastering Jxta: Building Java Peer-to-Peer Applications*. 1st. ed. New York, NY, USA: John Wiley & Sons, Inc., 2002.

GUIMARÃES, A. M. *Aplicação de Computação Evolucionária na Mineração de Dados Físico-Químicos da Água e do Solo*. Tese (Doutorado) — Universidade Estadual Paulista Júlio de Mesquita Filho, Botucatu, São Paulo, 2005.

HALL, M. *et al. The WEKA Data Mining Software*. 11. ed. [S.l.], 2009.

HAUSWIRTH, M.; SCHMIDT, R. An overlay network for resource discovery in grids. In: *Proceedings of Sixteenth Workshop on Database and Expert Systems Applications, 2005*. [S.l.: s.n.], 2005. p. 343–348.

HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN 0132733501.

HENNESSY, J.; PATTERSON, D. *Computer architecture: A Quantitative Approach*. [S.l.]: Elsevier Science & Technology Books, 2012. 856 p. (The Morgan Kaufmann Series in Computer Architecture and Design Series).

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995. (IJCAI'95), p. 1137–1143.

KUMAR, V. *et al. Introdução ao Data Mining - Mineração de Dados*. Rio de Janeiro: Ciência Moderna Ltda, 2009. ISBN 9788573937619.

MAROCO, J. *Análise Estatística com o SPSS Statistics*. 5. ed. [S.l.]: ReportNumber, Lda, 2011.

MUCHERINO, A.; PAPAJORGJI, P. J.; PARDALOS, P. M. *Data Mining in Agriculture*. 1. ed. [S.l.]: Springer Publishing Company, 2009.

PAPAJORGJI, P. J.; PARDALOS, P. M. *Software Engineering Techniques Applied to Agricultural Systems: An Object-Oriented and UML Approach*. Berlin, Heidelberg: Springer-Verlag, 2009. 264 p.

PENHA, D. O.; CORRêA, J. B. T.; MARTINS, C. A. P. S. Análise comparativa do uso de multi-thread e openmp aplicados a operações de convolução de imagem. In: *3º Workshop em Sistemas Computacionais de Alto Desempenho*. Vitória - ES: [s.n.], 2002.

PESSOA, A. S. A. *Mineração de Dados Meteorológicos Pela Teoria dos Conjuntos Aproximativos na Previsão de Clima por Redes Neurais Artificiais*. Dissertação (Mestrado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, 2009.

PIMENTA, A. *et al.* Weka - g: Mineração de dados paralela em grades computacionais. *Revista de Sistemas de Informação da FSMA*, v. 4, p. 2 – 9, 2009.

QUINN, M. J. *Parallel Computing: Theory and Practice*. [S.l.]: MCGraw Hill, 1994.

R, D. C. T. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2010. 1731 p.

RANJAN, R.; HARWOOD, A.; BUYYA, R. *A Study on Peer-to-Peer Based Discovery of Grid Resource Information*. [S.l.], Dezembro 2006.

RAUBER, T.; RUNGER, G. *Parallel Programming For Multicore and Cluster Systems*. [S.l.]: Springer, 2010.

SCHEFLER, W. C. *Statistics: Concepts and Applications*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1988.

SCHMITKE, L. R. *Preenchimento de falhas de séries de dados climáticos utilizando redes P2P*. Dissertação (Mestrado) — Universidade Estadual de Ponta Grossa - UEPG, Ponta Grossa, Paraná, 2012.

SENGER, L. J. *Escalonamento de processos: uma abordagem dinâmica e incremental para a exploração de características de aplicações paralelas.* 236 p. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, São Carlos, 2004.

SENGER, L. J.; SOUZA, M. A.; FOLTRAN, D. C. J. Towards a peer-to-peer framework for parallel and distributed computing. In: *Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on*. [S.l.: s.n.], 2011.

SERCKUMECKA, A. *Avaliação do Uso de Computação Paralela Utilizando uma Rede P2P na Simulação de Dados Climáticos: Velocidade e Direção do Vento*. Dissertação (Mestrado) — Universidade Estadual de Ponta Grossa - UEPG, Ponta Grossa, Paraná, 2012.

SOUZA, M. A. *et al.* Considerações sobre a utilização de técnicas de computação paralela para melhorar o tempo de resposta da mineração de dados agrícolas. In: . [S.l.]: VIII Congresso Brasileiro de Agroinformática-SBIAgro, 2011.

STALLINGS, W. *Computer Organization and Architecture: Designing for Performance*. 8. ed. [S.l.]: Prentice Hall, 2010.

SUNDERAM, V. S. *et al.* The pvm concurrent computing system: Evolution, experiences and trends. *Parallel Computing*, v. 20, p. 531–545, 1994.

TALIA, D.; TRUNFIO, P.; VERTA, O. The weka4ws framework for distributed data mining in service-oriented grids. *Concurrency and Computation: Practice & Experience*, v. 20, n. 16, p. 1933 – 1951, nov 2008.

VERSTRYNGE, J. *Practical JXTA: Cracking the P2P puzzle*. [S.l.]: Lulu, 2008.

WILKINSON, B.; ALLEN, M. *Parallel Programming: Techniques and Applications using networked workstations and parallel computers*. Upper Saddle River: Pearson/Prentice Hall, 2004.

WITTEN, I. H.; FRANK, E.; HALL, M. A. *Data Mining: Practical Machine Learning Tools and Techniques*. 3. ed. Amsterdam, MA: Morgan Kaufmann Publishers Inc., 2011. Paperback.

ZUO, X. *High Level Support for Distributed Computation in Weka*. Dissertação (Mestrado) — University College Dublin, Dublin, Irlanda, 2004.

APÊNDICE A – RESULTADOS DOS TEMPOS DE EXECUÇÃO

Este apêndice apresenta os resultados obtidos por meio dos experimentos realizados nesta dissertação. Os resultados acompanham suas estatísticas descritivas, como os valores de média, variância e desvio padrão. Assim como os resultados das inferências estatísticas empregadas, como análise de normalidade (teste de Shapiro-Wilk), análise estatística da variância, ANOVA, e os testes de comparação de médias T e Tukey. Descritos em detalhes em Schefler (1988).

A.1 Tempos de Execução Sequencial

Abaixo são apresentados os resultados obtidos por meio das execuções sequenciais, utilizados como parâmetros de controle, para comparação com os resultados obtidos com as duas técnicas de computação paralela. Na Tabela 11, estão presentes os resultados das cinco repetições do experimento de modo sequencial, utilizando 10 *folds*.

Tabela 11: Tempo Sequencial - 10 *Folds*

Experimento	Tempo Total
1	303,43
2	301,42
3	302,28
4	300,62
5	301,83
Média	**301,92**
Variância	**1,10**
Desvio Padrão	**1,05**
P-Value	**0,9732**

Fonte: O autor

A Tabela 12, também de modo sequencial, porem utilizando 24 *folds*.

Tabela 12: Tempo Sequencial - 24 *Folds*

Experimento	Tempo Total
1	761,17
2	760,88
3	759,56
4	764,13
5	769,47
Média	**763,06**
Variância	**15,54**
Desvio Padrão	**3,94**
P-Value	**0,2267**

Fonte: O autor

A.2 Tempos de Execução em Paralelo

Os resultados das execuções da computação paralela obtendo o tempo de execução, *speedup* e eficiência, estão divididos em duas seções. A próxima subseção apresenta os resultados obtidos com o modo multi-*threads*, e a subseção seguinte os resultados com redes P2P (modo P2P).

A.2.1 Modo Multi-*threads*

Os experimentos realizados utilizando o modo multi-*threads*, apresentam os ganhos obtidos com a utilização de diferentes números de *threads*, visando utilizar os núcleos reais e também os núcleos virtuais dos computadores, conforme descritos na Tabela 13 e Tabela 16.

Tabela 13: Tempo de Execução Modo Multi-*threads* - 10 *Folds*

Grupo	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5
Experimento	Sequencial	2-*Threads*	4-*Threads*	6-*Threads*	8-*Threads*
1	303,43	158,52	97,98	82,97	90,22
2	301,42	157,03	98,75	81,28	90,70
3	302,28	157,12	97,85	81,43	90,87
4	300,62	155,60	97,78	80,02	90,77
5	301,83	157,77	98,70	81,48	90,62
Média	**301,92**	**157,21**	**98,21**	**81,44**	**90,63**
Variância	**1,10**	**1,16**	**0,22**	**1,10**	**0,06**
Desvio Padrão	**1,05**	**1,08**	**0,47**	**1,05**	**0,25**
P-Value	**0,9732**	**0,8315**	**0,0862**	**0,5331**	**0,2938**
SpeedUp		**1,92**	**3,07**	**3,71**	**3,33**
Eficiência		**0,96**	**0,77**	**0,62**	**0,42**

Fonte: O autor

Tabela 14: Tabela ANOVA para resultados Modo Multi-*threads* - 10 *Folds*

Fonte de Variação	**Soma dos quadrados**	**G.L.**	**Variância Estimada**	**Razão F**	**Valor crítico de F**
Entre os grupos	169764,9160	4	42441,2290	58301,9508	2,8660
Dentro dos grupos	14,5591	20	0,7279		
Totais	169779,4751	24			

Fonte: O autor

Tabela 15: Comparação entre Grupos, Teste Estendido de Tukey Modo Multi-*threads* - 10 *Folds*

Valor de q igual a 4,23 e Valor de Tukey igual a 1,6140										
Grupo	1x2	1x3	1x4	1x5	2x3	2x4	2x5	3x4	3x5	4x5
Valor	144,71	203,70	220,48	211,28	58,99	75,77	66,57	16,78	7,58	9,20
Resultado	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$

Fonte: O autor

Tabela 16: Tempo de Execução Modo Multi-*threads* - 24 *Folds*

Grupo	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5
Experimento	Sequencial	2-*Threads*	4-*Threads*	6-*Threads*	8-*Threads*
1	761,17	411,42	217,58	206,92	193,67
2	760,88	407,58	218,73	206,70	193,42
3	759,65	409,17	217,83	199,03	189,88
4	764,13	408,68	218,25	205,12	192,88
5	769,47	406,93	218,03	201,20	194,03
Média	**763,06**	**408,76**	**218,09**	**203,79**	**192,78**
Variância	**15,54**	**2,99**	**0,19**	**12,33**	**2,79**
Desvio Padrão	**3,94**	**1,73**	**0,44**	**3,51**	**1,67**
P-Value	**0,2267**	**0,7084**	**0,9236**	**0,2838**	**0,0525**
SpeedUp		**1,87**	**3,50**	**3,74**	**3,96**
Eficiência		**0,93**	**0,87**	**0,62**	**0,49**

Fonte: O autor

Tabela 17: Tabela ANOVA para resultados Modo Multi-*threads* - 24 *Folds*

Fonte de Variação	**Soma dos quadrados**	**G.L.**	**Variância Estimada**	**Razão F**	**Valor crítico de F**
Entre os grupos	1186507,7105	4	296626,9276	43826,0929	2,8660
Dentro dos grupos	135,3654	20	6,7682		
Totais	1186643,0759	24			

Fonte: O autor

Tabela 18: Comparação entre Grupos, Teste Estendido de Tukey Modo Multi-*threads* - 24 *Folds*

Valor de q igual a 4,23 e Valor de Tukey igual a 4,9214										
Grupo	1x2	1x3	1x4	1x5	2x3	2x4	2x5	3x4	3x5	4x5
Valor	354,30	544,97	559,27	570,28	190,67	204,96	215,98	14,29	25,31	11,02
Resultado	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$

Fonte: O autor

A.2.2 Modo P2P

Os experimentos realizados utilizando redes P2P (modo P2P), apresentam os ganhos obtidos com diferentes números de pares, conforme descritos na Tabela 19, Tabela 21 e Tabela 24.

Tabela 19: Comparação do Tempo de Execução Modo P2P - 10 *Folds*

5 Pares Distribuídos X 5 Pares Localmente			
Grupo	Grupo 1	Grupo 2	Grupo 3
Experimento	Sequencial	Distribuído	Localmente
1	303,43	90,03	95,73
2	301,42	90,43	94,53
3	302,28	92,88	94,28
4	300,62	91,45	93,72
5	301,83	91,30	96,12
Média	**301,92**	**91,22**	**94,88**
Variância	**1,10**	**1,21**	**1,02**
Desvio Padrão	**1,05**	**1,10**	**1,01**
P-Value	**0,9732**	**0,6916**	**0,5658**
SpeedUp		**3,31**	**3,18**
Eficiência		**0,66**	**0,64**

Fonte: O autor

Tabela 20: Teste T para 5 Pares - Distribuído X Local - 10 *Folds*

8 graus de liberdade e Valor crítico igual a 2,306	
Variância Combinada	1,1175
Erro Padrão	0,6685
Valor de T	-5,4692
Teste de Hipótese	Rejeita H_0

Fonte: O autor

Tabela 21: Tempo de Execução Modo P2P - 10 *Folds*

Grupo	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5	Grupo 6
Experimento	Sequencial	3-Pares	5-Pares	7-Pares	9-Pares	11-Pares
1	303,43	146,10	90,03	62,13	60,32	31,33
2	301,42	146,42	90,43	62,25	60,60	33,15
3	302,28	147,33	92,88	61,00	61,62	31,15
4	300,62	145,35	91,45	60,83	61,95	32,42
5	301,83	145,15	91,30	60,97	60,43	31,80
Média	**301,92**	**146,07**	**91,22**	**61,44**	**60,98**	**31,97**
Variância	**1,10**	**0,77**	**1,21**	**0,48**	**0,56**	**0,67**
Desvio Padrão	**1,05**	**0,88**	**1,10**	**0,69**	**0,75**	**0,82**
P-Value	**0,9732**	**0,7196**	**0,6916**	**0,0556**	**0,1685**	**0,6455**
SpeedUp		**2,07**	**3,31**	**4,91**	**4,95**	**9,44**
Eficiência		**0,69**	**0,66**	**0,70**	**0,55**	**0,86**

Fonte: O autor

Tabela 22: Tabela ANOVA para resultados Modo P2P - 10 *Folds*

Fonte de Variação	**Soma dos quadrados**	**G.L.**	**Variância Estimada**	**Razão F**	**Valor crítico de F**
Entre os grupos	245736,6471	5	49147,3294	61539,9335	2,6206
Dentro dos grupos	19,1670	24	0,7986		
Totais	245755,8141	29			

Fonte: O autor

Tabela 23: Comparação entre Grupos, Teste Estendido de Tukey Modo P2P - 10 *Folds*

Valor de q igual a 4,37 e Valor de Tukey igual a 1,7464															
Grupo	1x2	1x3	1x4	1x5	1x6	2x3	2x4	2x5	2x6	3x4	3x5	3x6	4x5	4x6	5x6
Valor	155,85	210,70	240,48	240,93	269,95	54,85	84,63	85,09	114,10	29,78	30,24	59,25	0,45	29,47	29,01
Resultado	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$=$	$\neq$	$\neq$

Fonte: O autor

Tabela 24: Tempo de Execução Modo P2P - 24 *Folds*

Grupo	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5	Grupo 6
Experimento	Sequencial	3-Pares	5-Pares	7-Pares	9-Pares	13-Pares
1	761,17	372,78	190,18	136,65	98,12	73,52
2	760,88	381,33	193,13	134,58	101,73	72,28
3	759,65	376,37	189,92	132,65	102,52	72,07
4	764,13	382,37	194,17	127,17	99,75	71,27
5	769,47	377,63	196,77	130,17	100,52	74,37
Média	**763,06**	**378,10**	**192,83**	**132,24**	**100,53**	**72,70**
Variância	**15,54**	**15,04**	**8,22**	**13,79**	**2,96**	**1,52**
Desvio Padrão	**3,94**	**3,88**	**2,87**	**3,71**	**1,72**	**1,23**
P-Value	**0,2267**	**0,7547**	**0,5610**	**0,9587**	**0,9325**	**0,7774**
SpeedUp		**2,02**	**3,96**	**5,77**	**7,59**	**10,50**
Eficiência		**0,67**	**0,79**	**0,82**	**0,84**	**0,81**

Fonte: O autor

Tabela 25: Tabela ANOVA para resultados Modo P2P - 24 *Folds*

Fonte de Variação	**Soma dos quadrados**	**G.L.**	**Variância Estimada**	**Razão F**	**Valor crítico de F**
Entre os grupos	1736550,1602	5	347310,0320	36525,2023	2,6206
Dentro dos grupos	228,2106	24	9,5087		
Totais	1736778,3708	29			

Fonte: O autor

Tabela 26: Comparação entre Grupos, Teste Estendido de Tukey Modo P2P - 24 *Folds*

Valor de q igual a 4,37 e Valor de Tukey igual a 6,0264															
Grupo	1x2	1x3	1x4	1x5	1x6	2x3	2x4	2x5	2x6	3x4	3x5	3x6	4x5	4x6	5x6
Valor	384,96	570,23	630,82	662,53	690,36	185,26	245,85	277,57	305,40	60,59	92,31	120,13	31,72	59,54	27,83
Resultado	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$

Fonte: O autor

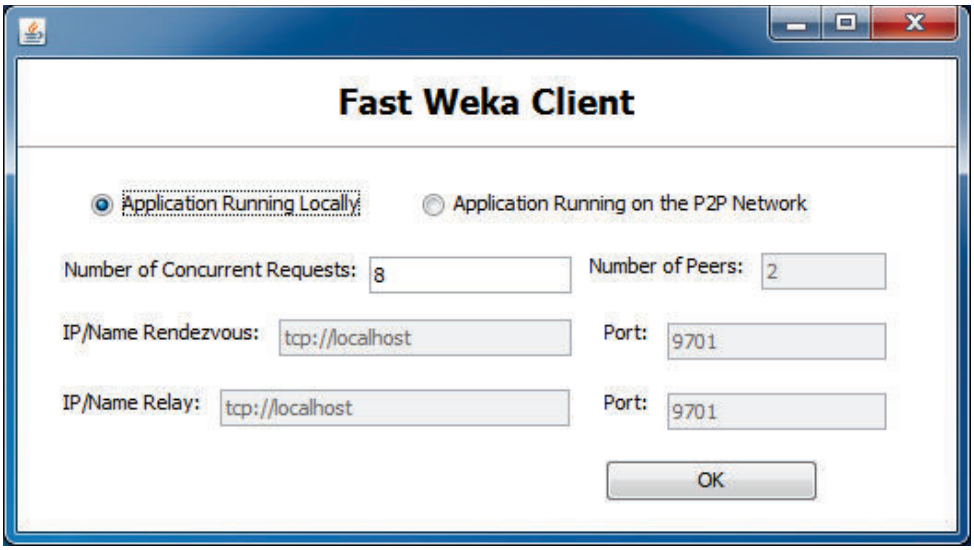
APÊNDICE B – *FAST WEKA*

B.1 Ferramenta *Fast Weka*

A ferramenta *Fast Weka* tem também como objetivo ser de fácil utilização a todos aqueles que necessitam de uma ferramenta de mineração de dados, ou que já utilizam a ferramenta Weka, mas que tem suas tarefas de mineração demandando de um alto tempo de execução e buscam melhor desempenho.

Assim, como principal e única mudança visual realizada na ferramenta Weka, ocorreu a criação de uma tela inicial de configuração, onde são definidas as configurações de execução concorrente ou paralela, conforme a Figura 9.

Figura 9: Tela Inicial *Fast Weka*



Fonte: O autor

Está tela é carregada toda vez que o programa é iniciado com a execução do arquivo fastweka.jar, ou por meio da linha de comando:

```
java -cp fastweka.jar weka.gui.GUIChooser
```

Por padrão, a opção *Application Running Locally* está pré-selecionada, que é utilizada quando deseja-se uma execução concorrente na maquina em questão (modo multi-*threads*), o campo *Number of Concurrent Requests* inicializa com o número máximo de núcleos do computador, considerando os núcleos reais e virtuais, podendo receber um valor entre um e o número máximo de núcleos do computador, o qual vai delimitar o número de execuções concorrente (*Threads*).

A segunda opção disponível é *Application Running on the P2P Network*, quando deseja-se realizar uma execução de modo paralelo com redes P2P, com está opção se-

lecionada o campo citado acima é desabilitado e são habilitados os campos: *IP/Name Rendezvous*, *IP/Name Relay*, os dois campos *Port* e também o campo *Number of Peers*.

O campo *Number of Peers* é responsável por definir o número de pares da rede P2P, os quais devem estar disponíveis para a execução das tarefas de mineração de dados, o mesmo aceita valores entre 2 até N. Os outros campos supracitados, identificam os endereços de IP/Nome e Porta, do *Rendezvous* e *Relay* que são responsáveis pela concentração dos anúncios e informações de roteamento, respectivamente, podendo os dois estarem definidos no mesmo endereço.

Para iniciar a aplicação em modo *Rendezvous/Relay* deve-se utilizar o fastweka.jar por meio da linha de comando:

```
java -cp fastweka.jar weka.p2p.RunRendezvous
```

Com os pares da rede P2P, também deve ser usado o fastweka.jar, porém com a linha de comando:

```
java -cp fastweka.jar weka.p2p.StartPeer
```

Para que os pares encontrem o endereço do *Rendezvous* e *Relay*, e necessário que existam os arquivos: relay.txt e seeds.txt, no mesmo diretório onde encontra-se o arquivo fastweka.jar, contendo a informação do IP/Nome e Porta (tcp://IP/Nome:Porta), onde estão localizados os mesmo, exemplo:

```
tcp://localhost:9701
```