UNIVERSIDADE FEDERAL DE CAMPINA GRANDE - UFCG
CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO - DSC
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

SELEÇÃO DE MODELOS DE CLASSIFICAÇÃO ATRAVÉS DE HEURÍSTICAS

David Moises Barreto dos Santos

(Mestrando)

Marcus Sampaio

(Orientador)

CAMPINA GRANDE, JULHO DE 2005

David Moises Barreto dos Santos

Seleção de Modelos de Classificação Através de Heurísticas

Dissertação submetida ao Curso de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande, como requisito parcial para a obtenção de grau de Mestre em Informática.

Área de concentração: Ciência da Computação
Linha: Sistemas de Informação e Banco de Dados

Marcus Sampaio
Orientador

Campina Grande, Julho de 2005

Ficha catalográfica

SANTOS, David Moises Barreto dos
S237S

Seleção de modelos de classificação através de heurísticas

Dissertação (Mestrado) - Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Junho de 2005.

86 p.

Orientador: Marcus Costa Sampaio

Palavras Chave:

1. Bancos de Dados
2. Mineração de Dados
3. Processo de Mineração de Dados
4. Heurísticas

CDU - 681.3.07B

"SELEÇÃO DE MODELOS DE CLASSIFICAÇÃO ATRAVÉS DE HEURÍSTICAS"

DAVID MOISES BARRETO DOS SANTOS

DISSERTAÇÃO APROVADA EM 19.07.2005

PROF. MARCUS COSTA SAMPAIO, Dr.
Orientador

PROF. HERMAN MARTINS GOMES, Dr.
Examinador

PROF. DUNCAN DUBUGRAS ALCOBA RUIZ, Dr.
Examinador

CAMPINA GRANDE – PB

"Este é o meu mandamento: amai-vos uns aos outros,
como eu vos amei"

Jesus Cristo (Jo 15,12)

Agradecimentos

Em primeiro lugar, agradeço a Deus por ter guiado meus passo ao longo desta caminhada e por me dar saúde para realizar este trabalho.

Agradecimentos especiais aos meus pais David e Alzira, e à Dinha, Agnaldo e Gustavinho. O amor e apoio de vocês foram indispensáveis. Muito obrigado!

A Mirna pelo carinho e atenção.

Ao meu orientador, Marcus Sampaio, pela confiança, competência e discussões enriquecedoras.

A Simone Branco, que me incentivou desde o início da minha carreira acadêmica.

A todos os colegas de apartamento com os quais convivi durante esta etapa de minha vida, em especial, a Robson pela amizade.

Ao pessoal do Flamingo pelos momentos de confraternização e companheirismo. Especialmente a Daniel, Roger, Verônica, Valnir, Aislene e Larissinha.

A Rolando, Rosalva, Renan, Guga e Milena pelo acolhimento fraternal.

Às funcionárias da COPIN, Aninha e Vera, sempre prestativas.

Agradeço a todos os professores e funcionários do Departamento de Sistemas e Computação da UFCG.

Aos colegas do DSC, pelos valorosos conhecimentos compartilhados durante nossa convivência.

A todos que de alguma forma contribuíram para realização deste trabalho.

Lista de Tabelas

Lista de Figuras

Sumário

Resumo

O processo de indução automática de modelos de classificação é composto de três etapas principais, pela ordem: amostragem, fragmentação e classificação. Devido à diversidade de técnicas que podem ser utilizadas em cada uma destas etapas, os resultados são muito instáveis — não existe a melhor técnica de amostragem, ou de fragmentação, etc. Para resolver o problema da indução do melhor modelo de classificação dentro de um espaço de busca, uma alternativa é arranjar todas as técnicas de amostragem, fragmentação e classificação disponíveis e buscar exaustivamente o melhor modelo de classificação. Entretanto, isto implicará em um alto custo computacional. Neste trabalho, descrevemos como foi possível descobrir heurísticas que podem ajudar a reduzir substancialmente o espaço de busca do melhor modelo de classificação. Também, discutimos o projeto e a implementação de um "framework" baseado nas heurísticas descobertas, objetivando guiar o minerador a escolher um modelo de classificação que satisfaça seus requisitos de qualidade, traduzidos em acurácia do modelo de classificação induzido, a um custo de processamento aceitável.

Abstract

The automated induction process of classification models is composed of the following steps: sampling, splitting and classification. Due to the diversity of techniques that may be used in each one of these steps, the results are much unstable — there is not the best sampling technique or the best splitting technique and so on. In order to solve the problem of inducing the best classification model within a search space, one way may arrange all available techniques for sampling, splitting and classification to search exhaustively the best classification model. However, this will imply in a high computational cost. In this work, we describe how to discover heuristics that help to reduce this computational cost. Also, we discuss the design and implementation of a framework that was developed based on the heuristics discovered in order to guide the miner in choosing a classification model which satisfies his or her quality requirements translated in the accuracy of the classification model induced with acceptable computational cost.

Capítulo 1

Introdução

1.1 Contexto da pesquisa

Nos últimos anos tem havido um amplo crescimento da capacidade de se gerar e coletar dados. As consultas rotineiras dos usuários estão muito longe de esgotar todo o potencial de informação para a tomada de decisão (Fayyad *et al*., 1996). Para explorar a informação 'escondida' nos bancos de dados têm surgido novas tecnologias capazes de extrair automaticamente conhecimento de grandes massas de dados (Han & Micheline, 2001). Mineração de Dados (MD)[1] é uma dessas novas tecnologias para a transformação de dados em conhecimento.

A principal meta da MD é descobrir *padrões* — (modelos de) conhecimento — de fácil assimilação, não-triviais e compreensíveis, de grandes quantidades de dados (Fayyad *et al*, 1996). Algoritmos de MD induzem modelos formais de conhecimento. Neste trabalho, tratamos somente de modelos de conhecimento do tipo *classificação*. Um modelo de classificação mapeia um objeto de negócio em uma *classe* — um valor de um *atributo de classificação*. Regras de classificação da forma `se <condição> então <atributo-de-classificação> = <classe>` e árvores de decisão são exemplos de modelos de classificação (Han & Micheline, 2001; Weiss & Indurkhya, 1998; Witten & Frank, 1999).

Para induzir conhecimento é necessário que o minerador[2] siga um processo, isto é, uma série de etapas igualmente importantes. O processo (de MD) tem como entrada um banco de dados (em estado 'bruto') e como saída o conhecimento (modelo de classificação) induzido. Processos de MD é o tema desta dissertação. Na próxima seção, definimos o processo de MD, no contexto do trabalho.

---

[1] O nome Mineração de Dados é uma metáfora da atividade de mineração: a 'mina' é a imensa massa de dados e o conhecimento é o 'minério' extraído.

[2] Agente da mineração

1.2 O processo de mineração de dados

O processo de MD — ilustrado na Figura 1.1 — em toda a sua abrangência, é composto das seguintes etapas: Preparação de dados, Seleção de amostra de dados, Fragmentação de amostra, Indução de conhecimento (modelo de classificação) e Análise e Assimilação do conhecimento induzido (Brumen *et al*, 2001; Han & Micheline, 2001; Pyle, 1999; Cabena *et al*, 1997; Kohavi, 1996; Fayyad *et al*, 1996; Witten & Frank, 1999). Para cada uma destas etapas, inúmeras técnicas podem ser aplicadas.

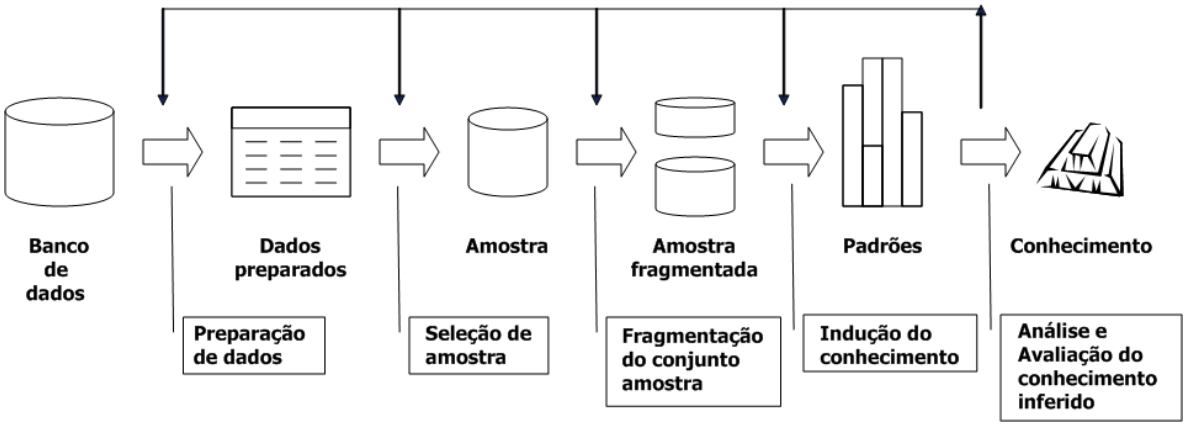


Figura 1.1: Processo de Mineração de Dados

Neste trabalho, adotamos um processo de MD simplificado, isto é, que exclui a etapa de preparação de dados — consideramos ainda a etapa de Análise e Assimilação como intrínseca à etapa de indução do conhecimento, como discutiremos mais tarde. A razão da simplificação é que as técnicas de preparação de dados se encontram muito desenvolvidas, não despertando maiores polêmicas. Nossa visão de processo de MD compreende então somente três etapas: amostragem, fragmentação e indução. Também, com relação a técnicas de indução, levamos em conta somente algoritmos classificadores, ou algoritmos que induzem modelos de classificação. Com isto, ainda cobrimos uma imensa gama de aplicações de MD, todas de caráter classificatório, como perfil de clientes para concessão de empréstimos bancários, probabilidade (maior, menor) de que algum tipo de evento venha a ocorrer, etc.

Formalmente, uma instância do processo de MD[3], no contexto desta dissertação, é uma tripla <técnica de amostragem, técnica de fragmentação, algoritmo de classificação>. A análise e assimilação do conhecimento induzido é tratada à parte do processo, uma vez que não se trata aqui de aplicação de técnicas específicas, mas de abordagens "ad hoc" de se aferir a qualidade do modelo de classificação induzido.

Na seqüência, comentamos as etapas de amostragem, fragmentação e indução do processo de MD, na nossa visão simplificada. Também, fazemos considerações sobre a análise e assimilação do modelo de classificação induzido por um processo de MD.

1.2.1 Seleção de amostra

Para explicar a seleção de amostras e as outras etapas do processo de MD, fazemos uso de um caso fictício de uma empresa de crediário. As instâncias do banco de dados são mostradas na Tabela 1.1, onde se vê informações de quatorze clientes. Cada linha da tabela representa um cliente, ou um caso de mineração.

Idade	Escolaridade	Casa própria	Telefone fixo	Pagador
jovem	1º grau	sim	não	mau
jovem	1º grau	sim	sim	mau
maduro	1º grau	sim	não	bom
idoso	2º grau	sim	não	bom
idoso	3º grau	não	não	bom
idoso	3º grau	não	sim	mau
maduro	3º grau	não	sim	bom
jovem	2º grau	sim	não	mau
jovem	3º grau	não	não	bom
idoso	2º grau	não	não	bom
jovem	2º grau	não	sim	bom
maduro	2º grau	sim	sim	bom
maduro	1º grau	não	não	bom
idoso	3º grau	sim	sim	mau

Tabela 1.1: Banco de dados de uma empresa de crediário

De acordo com os valores respectivos dos atributos `Idade`, `Escolaridade`, `Casa própria` e `Telefone fixo`, um cliente pode ser considerado um bom ou mau pagador — `Pagador` é o atributo de classificação, enquanto que `bom` e `mau` são

[3] Sempre que não houver prejuízo à clareza, confundiremos 'uma instância do processo de MD' com 'o processo de MD'.

as classes. O desafio é identificar tendências de comportamento de clientes maus pagadores que ocorrem com uma maior freqüência. Assim, no futuro, a empresa poderá discernir pessoas potencialmente más pagadoras e, conseqüentemente, não aceitar as suas solicitações de crédito.

A extração de amostras (Brumen *et al*., 2001; Han & Micheline, 2001; Pyle, 1999) é apropriada quando a quantidade de dados disponíveis é muito grande. Optar por não utilizar amostras nestes casos pode provocar um alto custo computacional ao processo de MD. Portanto, a vantagem de utilizar técnicas de amostragem é a diminuição do custo do processo de MD. A Tabela 1.1 pode ser uma amostra de um universo muito maior de dados.

As técnicas de amostragem usadas nos nossos experimentos são `Adaptive Incremental Framework (AIF)` (Brumen *et al*., 2001) e `Convergência` (`Conv`) (Pyle, 1999). Uma questão que fica por enquanto sem resposta é: qual a melhor técnica de amostragem para a mineração de um certo banco de dados?

1.2.2 Fragmentação de amostra

Uma vez que uma amostra tenha sido escolhida, o próximo passo é fragmentá-la em dois subconjuntos distintos, denominados conjunto de treinamento e conjunto de teste (Kohavi, 1996; Witten & Frank, 1999). *Conjuntos de treinamento* são entradas para os algoritmos de classificação — classificadores — induzir modelos de classificação, enquanto que *conjuntos de teste* são utilizados para avaliar os modelos de classificação induzidos.

Uma forma de avaliar um modelo de classificação é através de medidas estatísticas como exatidão, ou *acurácia* (Witten & Frank, 1999). A acurácia de treinamento (teste) é a proporção das instâncias do conjunto de treinamento (teste) classificadas corretamente por um modelo de classificação. A acurácia de treinamento é valiosa para se ter uma descrição aproximada dos dados minerados. Por exemplo, supondo que a acurácia de treinamento seja 100%, então o modelo de classificação é uma síntese fiel dos dados minerados; isto muito provavelmente é um conhecimento novo sobre os dados disponíveis, desconhecidos em sua essência.

A acurácia de teste é um indicador de confiabilidade de um modelo de classificação para a classificação automática de novos casos de mineração – predição

(Witten & Frank, 1999). A conclusão é que a acurácia de teste[4] é ainda mais importante que a acurácia de treinamento.

Retornando ao nosso exemplo, a Tabela 1.1 foi fragmentada — utilizando uma técnica de fragmentação chamada `Holdout` (Kohavi, 1996) — em conjunto de treinamento (Tabela 1.2) e conjunto de teste (Tabela 1.3).

Idade	Escolaridade	Casa própria	Telefone fixo	Pagador
jovem	1º grau	sim	não	mau
Jovem	1º grau	sim	sim	mau
idoso	2º grau	sim	não	bom
idoso	3º grau	não	não	bom
maduro	3º grau	não	sim	bom
jovem	3º grau	não	não	bom
idoso	2º grau	não	não	bom
jovem	2º grau	não	sim	bom
maduro	2º grau	sim	sim	bom
idoso	3º grau	sim	sim	mau

Tabela 1.2: Conjunto de treinamento

Idade	Escolaridade	Casa própria	Telefone fixo	Pagador
maduro	1º grau	sim	não	bom
idoso	3º grau	não	sim	mau
jovem	2º grau	sim	não	mau
maduro	1º grau	não	não	bom

Tabela 1.3: Conjunto de teste

Segundo a técnica `Holdout`, as instâncias do conjunto de treinamento foram obtidas através da extração aleatória de ⅔ dos dados da Tabela 1.1, enquanto que o conjunto de teste foi formado pelos casos restantes — ⅓ das 14 instâncias. Entretanto, o conjunto de treinamento (teste) pode não ser representativo; por exemplo, uma classe presente no conjunto de teste, pode não o estar no conjunto de treinamento. Uma forma de se ter conjuntos de treinamento e teste representativos é repetir o processo de fragmentação diversas vezes, por exemplo, dez vezes ou iterações. A acurácia então é calculada pela média aritmética das acurácias obtidas em cada iteração. A acurácia 'pessimista' ('otimista') é a média menos (mais) o desvio padrão.

Entre as técnicas de fragmentação que empregamos em nossos experimentos destacamos, além de `Holdout`, `BootStrap (BS)` e `Cross Validation (CV)` (Kohavi, 1996). Encerramos esta subseção deixando uma pergunta em aberto: qual a

[4] No restante do documento, a acurácia de teste será referida apenas por 'acurácia'.

melhor técnica de fragmentação a empregar em um problema específico de mineração? Esta é uma das preocupações dominantes, ao longo da dissertação.

1.2.3 Indução de modelos de classificação

Na etapa de Indução de modelos de classificação (Fayyad *et al*., 1996; Witten & Frank, 1999) é aplicado um algoritmo do tipo classificador, no intuito de induzir conhecimento a partir do conjunto de treinamento, segundo modelos formais de classificação. A Figura 1.2 mostra a árvore de decisão induzida com o algoritmo `J48` (Witten & Frank, 1999), para o conjunto de treinamento da Tabela 1.2.

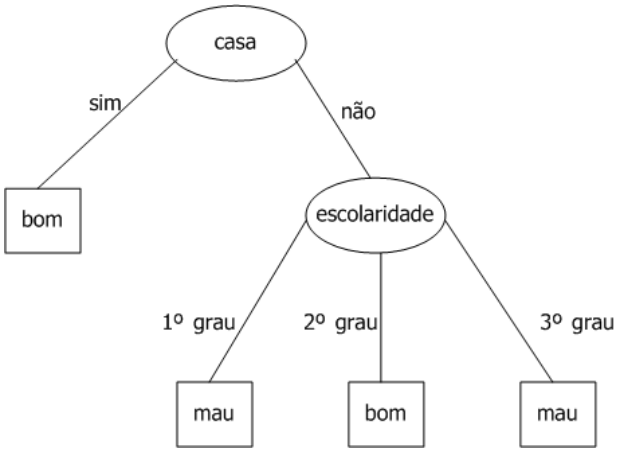


Figura 1.2: Modelo de classificação induzido sob a forma de árvore de decisão

Outros exemplos de classificadores são `Prism`, `Id3`, `OneR`, `NNge`, `Decision Stump (DS)`, `IBk`, `IB1`, e `Naïve Bayes (NB)`. Como nas duas seções anteriores, a questão 'qual é o melhor classificador para um problema específico de mineração?' deve ser convenientemente respondida.

Análise e Assimilação de modelos de classificação

A análise e assimilação de um modelo de classificação implica em aferir sua qualidade, no sentido de ser aproveitável ou não em situações práticas. Para isto, a medição da acurácia é indispensável: a acurácia medida pode ser considerada boa ou aceitável, ou então ruim ou inaceitável; neste último caso, o modelo de classificação concernente deve ser descartado.

Um outro critério de aferição da qualidade de um modelo de classificação é a medida de sua concisão (Ikizler & Güvenir, 2001; Silberschatz & Tuzhilin, 1996).

Considere uma árvore de decisão: se ela é pouco concisa, isto é, com muitos ramos e muito profunda, ela provavelmente é muito difícil de ser interpretada ou assimilada, devendo ser descartada. Até o presente, os algoritmos classificadores são muitas vezes verborrágicos.

Acurácia e concisão são dois critérios ortogonais de análise e assimilação de modelos de classificação. Ressaltamos que a pesquisa em concisão de modelos de conhecimento ainda é muito incipiente, e o tema está fora do escopo desta dissertação.

Voltemos ao exemplo dos clientes de um banco. A acurácia da árvore de decisão da Figura 1.2 é de 25%. Ela é certamente baixa, devido provavelmente aos poucos casos existentes tanto no conjunto de treinamento quanto no conjunto de teste. De qualquer modo, a qualidade de uma acurácia é relativa, dependendo das exigências de um particular minerador, para seu problema específico de mineração. Mais explicitamente, uma acurácia de 90%, por exemplo, pode ser suficiente para um minerador, porém pode não o ser para outro.

Neste trabalho, a aferição das acurácias dos modelos de classificação induzidos em nossos experimentos será muito importante — o outro critério é o tempo de resposta dos processos de mineração para induzir modelos de classificação — para as decisões que serão tomadas e devidamente explicadas, relativas a processos de MD.

1.3 O problema da instabilidade de processos de MD

Dado que existem diversas técnicas de amostragem, fragmentação e de indução de conhecimento, temos, portanto, para um mesmo problema de mineração, inúmeros processos de mineração possíveis — mais precisamente, inúmeras instâncias do processo podem ser escolhidas. A conseqüência de tudo isto é que, para um mesmo banco de dados a minerar, pode ser induzido um grande número de modelos de classificação, um para cada instância escolhida do processo de MD.

A Figura 1.3 exibe um modelo de classificação sob a forma de regras de classificação, obtido com o classificador `Prism` (Witten & Frank, 1999), do conjunto de treinamento da Tabela 1.2. Comparando este modelo de regras de classificação e o modelo de árvore de decisão da Figura 1.2, percebe-se uma diferença notável de semântica e sintaxe entre os conhecimentos induzidos. Outra forma de dizer é que os dois processos de MD, o que induziu as regras de classificação da Figura 1.2 e o que

induziu a árvore de decisão da Figura 1.3, produzem resultados altamente instáveis entre si.

IF casa = sim *THEN* pagador = bom
IF idade = maduro *THEN* pagador = bom
IF idade = idoso *AND* telefone = sim *THEN* pagador = bom
IF escolaridade = 1 *THEN* pagador = mau
IF casa = nao *AND* idade = idoso *AND* telefone = nao *THEN* pagador = mau

Figura 1.3: Conhecimento induzido sob a forma de regras de classificação

Medindo-se a acurácia de ambos os modelos, pode-se escolher um dentre estes dois modelos. Mas o que dizer de inúmeros outros modelos que também podem ser induzidos, com as diferentes triplas <amostragem,fragmentação, classificação>? Outrossim, deve ser lembrado que o tempo de execução das triplas, ou dos processos de MD, é outro critério a ser considerado na escolha de um modelo de classificação.

As questões desta forma postas são: qual o melhor modelo de classificação, dentre as diferentes instâncias do processo? Qual a instância do processo de MD que induziu o melhor modelo? Estes questionamentos caracterizam o *problema da instabilidade* dos processos de MD.

A rigor, usamos o termo *instabilidade* por simplificação no sentido de que, dado um conjunto de arquivos de dados a minerar e um conjunto de processos de MD, um processo P deste conjunto poderá ser o melhor (maior acurácia) para um arquivo A_1, porém P será o terceiro melhor para A_2, décimo melhor para A_3, quinto melhor para A_4, etc. Sendo assim, à medida que muda o arquivo de dados, o desempenho de um processo de MD qualquer pode variar bastante em relação à acurácia.

Para ilustrar ainda mais o problema da instabilidade dos processos de MD, considere a indução de um modelo de classificação para o banco de dados splice (Blake & Merz, 2002), contendo seqüências de DNA. As técnicas de amostragem são AIF e Conv; as de fragmentação, BS e CV; e os classificadores J48 e Prism. Arranjamos estas técnicas de quatro formas como estão apresentadas na Tabela 1.4.

Cada um dos três primeiros processos de MD induz uma árvore de decisão distinta, e cada árvore de decisão é diferente sintaticamente e semanticamente do modelo de classificação induzido pelo último processo de MD. O problema é agravado

pelo fato de que há muitas outras técnicas de amostragem, fragmentação e classificação. Em síntese, um minerador não tem como saber a priori qual é o melhor processo de MD amostragem-fragmentação-classificação para os dados a minerar.

Técnica de amostragem	Técnica de fragmentação	Classificador	Modelo de classificação induzido
Conv	CV	J48	árvore de decisão 1
AIF	CV	J48	árvore de decisão 2
Conv	BS	J48	árvore de decisão 3
Conv	CV	Prism	regras de classificação

Tabela 1.4: Diferentes modelos de classificação para o mesmo arquivo de dados

Para resolver o problema da instabilidade dos processos de MD, uma alternativa seria arranjar todas as técnicas de amostragem, fragmentação e classificação disponíveis, e buscar exaustivamente o melhor modelo de classificação. Entretanto, isto implica obviamente em um alto custo computacional.

Nesta dissertação, propomos uma solução para o problema da instabilidade dos processos de MD focada em heurísticas.

1.4 Objetivos

O objetivo central deste trabalho é ajudar o minerador a contornar o problema da instabilidade dos processos de MD, da seguinte forma: dado um arquivo de dados a minerar, um sistema de apoio ao minerador o ajudará a encontrar um modelo de classificação que poderá resolver seu problema, tudo dentro de custos computacionais aceitáveis pelo minerador. Com a finalidade de cumprir esta meta, estabelecemos os seguintes dois grandes objetivos específicos:

- Descobrir heurísticas que reduzam o tempo de execução dos processos de MD sem comprometer a qualidade (acurácia) do modelo de classificação induzido. As heurísticas devem levar em conta características dos bancos de dados assim como a variedade de técnicas existentes para amostragem, fragmentação e classificação. A melhor acurácia obtida por uma busca de

modelos de classificação apoiada em heurísticas deve ser similar à obtida por uma busca exaustiva de modelos de classificação;

- Desenvolver um "framework", baseado nas heurísticas descobertas, que ajude o minerador a encontrar, o mais cedo possível, um modelo de classificação com uma acurácia que lhe seja aceitável;
    - ⇒ Elaborar e implementar um algoritmo que automatize a execução de processos de MD;
    - ⇒ Este algoritmo deve permitir que o minerador interaja de modo a influenciar no resultado final fazendo interferências ao longo da sua execução de duas formas: (i) intervindo na escolha de heurísticas e (ii) acompanhando o processo de MD a fim de interrompê-lo assim que uma acurácia aceitável for obtida. Tal interação pode reduzir ainda mais o tempo de obtenção de um modelo de classificação com uma 'boa' acurácia;

1.5 Contribuições

As principais contribuições deste trabalho são:

- Descoberta de heurísticas que ajudem o minerador a limitar o escopo de busca de processos de mineração;
- Desenvolvimento de um "framework", apoiado pelas heurísticas, para encontrar, de maneira eficiente, um modelo de classificação que satisfaça as exigências de qualidade do minerador;

1.6 Estrutura da dissertação

O restante desta dissertação está estruturado em cinco capítulos.

Discutimos minuciosamente nosso processo de descoberta de heurísticas no Capítulo 2.

O Capítulo 3 é dedicado ao "framework".

A avaliação experimental do "framework" é detalhada no Capítulo 4, na qual comparamos os resultados obtidos pela busca exaustiva de processos de mineração e pela busca baseada nas heurísticas descobertas. Tal avaliação visa a mostrar a eficácia das heurísticas.

A comparação de nossa abordagem com outros trabalhos relevantes sobre o problema da instabilidade dos processos de mineração é o tema do Capítulo 5.

Finalmente, concluímos o trabalho e traçamos direções para sua continuação no Capítulo 6.

Capítulo 2

O Processo de Descoberta de Heurísticas

Neste capítulo, apresentamos o nosso processo de descoberta de heurísticas e as heurísticas assim descobertas, visando a ajudar na solução do *problema da instabilidade dos processos de MD*, discutido no Capítulo 1. O propósito das heurísticas é explicado como segue: dado um arquivo de dados[5] a minerar, (1) que processos de MD podem ser previamente descartados, porque dificilmente levariam a um modelo de classificação com a acurácia mínima desejada pelo minerador; e (2) que processos de mineração são os melhores candidatos a induzir modelos de classificação com qualidade aceitável. Para os processos de MD, utilizamos três técnicas de amostragem: `Convergência` (`Conv`), `Adaptive Incremental Framework` (`AIF`) (Pyle, 1999; Brumen *et al*., 2001) e a pseudo-técnica de amostragem `Not Sample (NS)`, que é o próprio arquivo de dados; três técnicas de fragmentação (utilizando dez iterações): `Bootstrap (BS)`, `Holdout (HO)` e `Cross Validation (CV)` (Kohavi, 1996); e nove classificadores: `Prism`, `Id3`, `J48`, `OneR`, `NNge`, `Decision Stump (DS)`, `IBk`, `IB1`, e `Naïve Bayes (NB)` (Witten & Frank, 1999). Os 81 processos de MD (3 técnicas de amostragem x 3 técnicas de fragmentação x 9 classificadores) foram aplicados a 95 arquivos de dados — a grande maioria extraída do repositório UCI (Blake & Merz, 2002) — de uma grande diversidade de domínios, obtendo-se assim, 7.695 (81x95) modelos de classificação. O essencial desses modelos de classificação veio a constituir uma base de conhecimento, construída assim:

- para cada arquivo de dados, informações sobre os desempenhos (acurácias dos 81 modelos de classificação e tempos de execução dos 81 processos de MD respectivos) foram ordenadas separadamente, da seguinte forma: decrescentemente por acurácia — ranking das acurácias — e crescentemente por tempo de execução — ranking do tempo. É importante assinalar que

[5] No documento, as expressões 'arquivo de dados' e 'banco de dados' se confundem.

consideramos somente valores pessimistas para as acurácias, isto é, seu valor médio das fragmentações com dez iterações menos o desvio padrão. Esta decisão se deve ao fato de que o valor de uma acurácia pode variar muito em cada iteração, daí nossa justificativa da escolha pelo pior caso (acurácia menos o desvio padrão);

- cada arquivo de dados foi caracterizado por valores respectivos de três meta-atributos: `número de atributos`, `número de classes` e `número de instâncias`. Escolhemos estes três pela simplicidade, e por serem bastante utilizados. Veremos posteriormente que nossa escolha de meta-atributos ficou validada pelo fato de que, apenas com os três escolhidos, foi possível descobrir heurísticas interessantes.

A base de conhecimento assim obtida foi analisada de duas formas: por meio de meta-mineração (Subseção 2.1) e por análise do custo de processamento dos classificadores (Subseção 2.2). O processo, em linhas gerais, e suas duas variantes estão ilustrados na Figura 2.1.

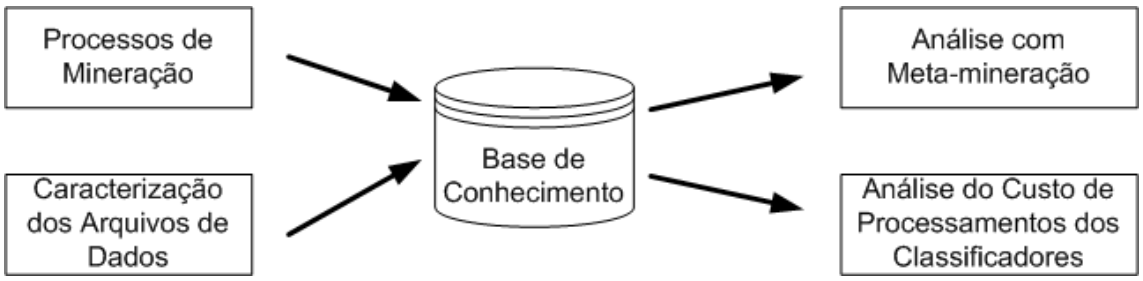


Figura 2.1: Etapa inicial para descoberta de heurísticas

2.1 Meta-mineração da base de conhecimento

A análise com meta-mineração (Figura 2.2) consistiu em utilizar classificadores para induzir, a partir da base de conhecimento, classificações sobre acurácias e tempos de execução — heurísticas de acurácia e heurísticas de tempo, respectivamente.

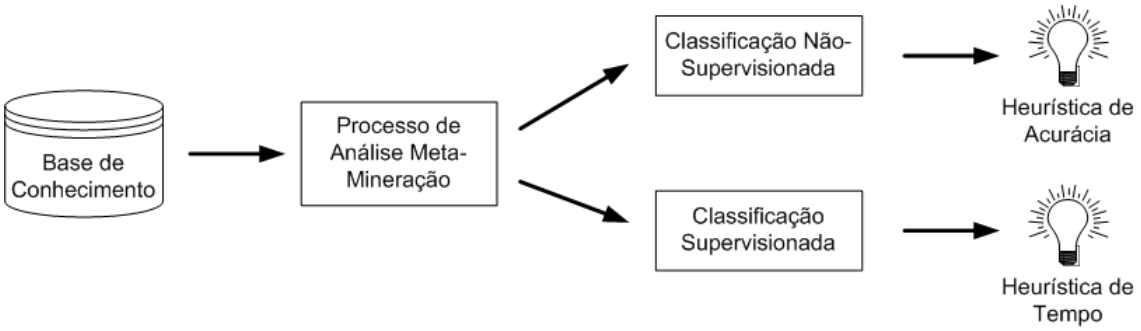


Figura 2.2: Análise com meta-mineração

As heurísticas de acurácia foram induzidas através de classificação não-supervisionada enquanto que as heurísticas de tempo o foram através de classificação supervisionada. Na classificação supervisionada, o atributo de classificação e seus valores possíveis (classes) são definidos previamente — este é o tipo de classificação discutido até então. No que diz respeito à classificação não-supervisionada, o atributo de classificação é induzido pelo classificador.

No intuito de facilitar as classificações não-supervisionada e supervisionada, a base de conhecimento foi estruturada em diversos *arquivos* (de dados) *intermediários*, assim respectivamente:

- Heurísticas de acurácia. Para cada processo de MD, e para cada um dos meta-atributos `número de atributos` (`NA`) e `número de classes` (`NC`), criamos um arquivo intermediário com dois atributos, `NA` (`NC`) e `posição` do processo de MD no ranking das acurácias — isso dá um total de 162 arquivos (81 processos de MD x 2 meta-atributos). Note que, cada instância deste arquivo intermediário representa um arquivo de dados dos 95 contidos na base de conhecimento; logo, cada arquivo intermediário assim construído tem 95 instâncias. A finalidade da meta-mineração com classificação não-supervisionada é classificar processos de MD como relativamente bom ou relativamente ruim no que concerne às acurácias induzidas por eles: aqueles processos de MD com desempenho ruim para determinadas faixas de valores de `NA` (`NC`) são então afastados do espaço de busca para *novos* arquivos de dados — isto é, arquivos de dados para minerar que não sejam nenhum dos 95 arquivos de dados da base de conhecimento — com `NA` (`NC`) dentro das respectivas faixas de valores;

- Heurísticas de tempo. Criamos apenas um arquivo intermediário, com dois atributos: `NI` (número de instâncias de arquivo de dados) e `Tempo_total` (a soma dos tempos de execução de todos os processos de MD, para cada arquivo de dados — é também o atributo para a classificação supervisionada cujas classes são `tempo_execucao > 60 min` e `tempo_execucao ≤ 60 min`). Deve estar claro então que a cardinalidade deste arquivo intermediário é 95 instâncias (a quantidade de arquivos de dados minerados na experimentação). O objetivo da meta-mineração com classificação supervisionada é chegar ao conceito de arquivo de dados 'grande', no sentido de avaliar se amostragem propriamente dita é fundamental ou não. Note que, dentre `NA`, `NC` e `NI`, `NI` é o que contribui mais fortemente para o tempo de execução de um processo de MD.

Nas duas subseções seguintes, detalhamos os processos de meta-mineração não-supervisionada e supervisionada da base de conhecimento, respectivamente.

2.1.1 Meta-mineração com classificação não-supervisionada

Para a classificação não-supervisionada, aplicamos a técnica de "clustering" `Simple k-Means` (Witten & Frank, 1999) aos 162 arquivos intermediários (ver Heurísticas de acurácia, seção anterior). Com esta técnica, mineradores especificam, através do parâmetro `k`, em quantas classes — "clusters" — as instâncias de um arquivo devem ser agrupadas, por critérios induzidos pelo próprio algoritmo, baseado na noção de função de similaridade.

Para os nossos experimentos, escolhemos `k=4` porque este valor conduziu, por tentativa e erro, a bons resultados — o principal critério utilizado para determinar o valor de `k` foi o quão compacto era cada "cluster" no sentido de maximizar o número de pontos dentro de um "cluster" e minimizar a faixa de valores de `NA` (`NC`) abrangida pelos "clusters". Como exemplo de classificação em "clusters", veja a Figura 2.3 que ilustra os quatro "clusters" induzidos para o arquivo correspondendo ao processo `AIF-HO-Prism` e ao meta-atributo `NC`. Mais precisamente: dado um dos 95 pontos no gráfico, ele representa um dos 95 arquivos da base de conhecimento, que tem um certo número de classes (valor da abscissa `NC`), e a posição do processo de MD `AIF-HO-Prism` aplicado ao arquivo, no ranking das acurácias dos 81 processos de MD para o respectivo arquivo (valor da ordenada `posição`).

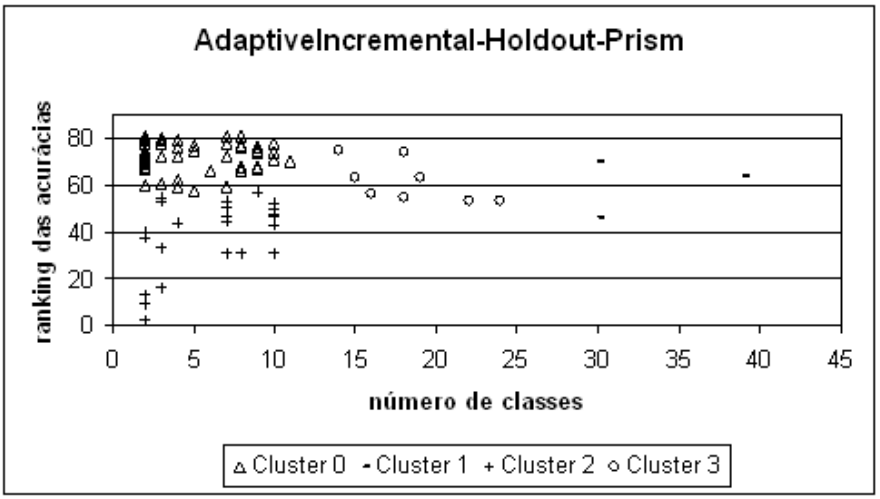


Figura 2.3: "Clusters" para o processo `AIF-HO-Prism` e o meta-atributo `NC`

Um conhecido problema com algoritmos de "clustering" é que eles deixam ao minerador a interpretação dos "clusters" induzidos. Considere, por exemplo, o algoritmo `Simple 4-Means`: os "clusters" induzidos são nomeados `{0,1,2,3}` como pode ser visto na Figura 2.3, e isto é tudo. A pergunta que deve ser feita em seguida é: o que significa o "cluster" `0(1)(2)(3)`? Para respondê-la, precisamos achar uma *definição* formal para cada "cluster", e saber interpretá-la. Obtivemos estas definições com classificação supervisionada, da seguinte maneira: a cada um dos 162 arquivos intermediários foi adicionado o atributo de classificação `cluster`, com valores $\in$ {0, 1, 2, 3}; assim, cada arquivo intermediário foi minerado com um algoritmo de classificação supervisionada, e seu respectivo modelo de conhecimento induzido foi usado como definição. Mais precisamente, os processos de MD utilizados para tal tarefa foram dois, da forma `Not Sample – Sem fragmentação – Um, dentre dois classificadores`. Não faz sentido amostragem (ou `Not Sample`) visto que os arquivos são muito pequenos (95 instâncias). Ainda como conseqüência de os arquivos serem muito pequenos, eles não foram fragmentados — ou cada arquivo intermediário inteiro é o próprio conjunto de treinamento. Finalmente, os classificadores aproveitados foram `Naïve Bayes` e `NNge`, pois, dentre todos os classificadores, eles foram os únicos em que, nas definições das classes, os atributos `NA`(`NC`) e `Posição` aparecem explicitamente, como convém para uma boa interpretação dos "clusters". Também, eles induziram modelos de classificação completos, ou seja, em que aparecem regras de classificação para os quatro "clusters".

Para dar um contra-exemplo, considere as únicas duas regras de classificação induzidas pelo classificador `OneR`, para os "clusters" 2 e 0 da Figura 2.3:

- `se posição<54 então "cluster"=2,` e
- `se posição>=54 então "cluster"=0.`

Este modelo de classificação não é útil por dois motivos: (1) é incompleto, pois, nada afirma sobre os "clusters" 1 e 3, e (2) quanto aos "clusters" 0 e 2, as regras não fazem referência a valores de `NC`. Definições interessantes de "clusters" podem ser visualizadas na Tabela 2.1.

Cluster	Definições
0	class 0 IF: 2<=NC<=11 ^ 58<=posicao<=81
1	class 1 IF: 30<=NC<=39 ^ 46<=posicao<=70
2	class 2 IF: 2<=NC<=10 ^ 2<=posicao<=57
3	class 3 IF: 14<=NC<=24 ^ 53<=posicao<=75

Tabela 2.1: Definições dos "clusters" da Figura 2.3

A questão que se põe agora é: como decidir se um valor do atributo `posição` é bom ou ruim? A finalidade é estabelecer se um processo de MD tem desempenho bom ou ruim a depender de sua posição no ranking das acurácias (valor do atributo `posição`). A resposta foi baseada em tentativa e erro. Inicialmente, estabelecemos arbitrariamente que um processo de MD é bom para um arquivo de dados quando `posição <= 15`; caso contrário (`posição > 15`), o processo é ruim. Entretanto, com este limiar de 15, para cada processo, o número de arquivo de dados cobertos pela faixa com desempenho bom (`posição <= 15`) é sempre muito baixo, ou seja, não existe um processo de MD que se destaque acentuadamente dos demais. Observando a primeira linha da Tabela 2.2, nota-se que a faixa com desempenho correspondendo a `posição <= 15` indica que cada processo de MD cobre, em média, apenas 18,52% das 95 instâncias.

Tendo em vista este baixo valor, procuramos então um limiar maior, pois, desta forma, encontrar-se-á mais facilmente processos de MD que se destaquem comparativamente. Variando o limiar a partir de 15 com incrementos de 5, encontramos que o limiar 40 é o primeiro que pode ser considerado significativo. Isto é mostrado na Tabela 2.2: para cada processo de MD, a faixa com desempenho correspondendo a `posição <= 40` cobriu em média aproximadamente 50% dos 95 dos arquivos de dados experimentados. O número 40 passou a ser o limiar de qualidade, isto é, posições

no ranking das acurácias abaixo ou igual a 40 são consideradas boas, enquanto que as posições além de 40 são ruins.

Tomando então como base o limiar de qualidade 40, processos de MD podem ser rejeitados (ou podados) para uma determinada faixa de valores de `NA` (`NC`) a depender de seus respectivos desempenhos em relação à acurácia. Esta é a idéia básica do algoritmo de poda de processos de MD que apresentamos a seguir.

Limiar	Desempenho de Processos de MD (% de arquivos de dados cobertos)	
	Bom	Ruim
15	18,52	81,48
20	24,69	75,31
25	30,86	69,14
30	37,04	62,96
35	43,21	56,79
40	49,38	50,62
45	55,56	44,44
50	61,73	38,27
55	67,89	32,11

Tabela 2.2: Simulação de limiares

Algoritmo de Poda de Processos de MD

Para um processo de MD, seja um "cluster" `C`, com `n` pontos (arquivos de dados), definido para um intervalo de `NC` (`NA`), e para uma faixa de posições no ranking das acurácias: qual é o mínimo percentual de pontos do "cluster" com `posição <= 40` para que o processo de MD não seja podado? O problema se complica porque muitas vezes, como veremos, mais de um "cluster" deve ser analisado ao mesmo tempo.

Para resolver o problema da poda de processos de MD, desenvolvemos um algoritmo genérico chamado de `Mining Process Prunning` (Figura 2.4), com os seguintes parâmetros de entrada:

- conjunto de "clusters": são os "clusters" induzidos, e sua quantidade pode variar de acordo o número de arquivos de dados (quantidade de instâncias do arquivo de dados intermediário). Em nossos experimentos, trabalhamos com quatro "clusters", como foi discutido no início da Seção 2.1.1;
- limiar de qualidade: discutido no final da Seção 2.1.1, pode ser modificado a depender do número de processos de MD sendo avaliados. Relembrando, nos

nossos experimentos, o limiar de qualidade é 40, levando em conta os 81 processos de MD usados;

- suporte mínimo: isto é, a quantidade mínima de arquivos de dados (pontos no gráfico da Figura 2.3) que devem ser cobertos pelos "clusters" na faixa de NA (NC) em análise para que se tenha uma validade estatística. Nos nossos experimentos, julgamos que 10% dos 95 arquivos experimentados é o suporte mínimo necessário.

```
void MiningProcessPruning(Clusters[],limiar,suporte)

Repita
      Selecione o "cluster" com maior faixa de NA(NC) em Clusters e
      armazene na coleção  Clusters_em_analise;
      Selecione todos os "clusters" que estão na mesma faixa do
      primeiro "cluster" selecionado, com maior faixa de NA(NC), e
      adicione na coleção  Clusters_em_analise;

      Se (numero de pontos de Clusters_em_analise > suporte)
            Contabilize quantos pontos de Clusters_em_analise tem
            posição > limiar e atribua a desempenho_ruim;
            Contabilize quantos pontos de Clusters_em_analise tem
            posição ≤ limiar e atribua a desempenho_bom;

            Se (desempenho_ruim>desempenho_bom)
                  Descarte o processo P para a faixa de NA(NC) coberta
                  pelos "clusters" da coleção Clusters_em_analise;
            fim_se
      fim_se

      Remova todos os "clusters" selecionados até então da coleção
      Clusters;
      Remova os "clusters" de Clusters_em_analise;

até a coleção Clusters não ter mais elementos;

fim_MiningProcessPruning
```

Figura 2.4: Algoritmo para poda de processos de MD do espaço de busca

Para melhor compreender o algoritmo, considere que "clusters" *candidatos* são todos os "clusters" induzidos para um processo de MD P e o atributo NA (NC), como ilustrado na Figura 2.5(a). Primeiramente, o "cluster" com a faixa mais larga de NA (NC) é identificado pelo algoritmo — "cluster" *majoritário* — e, em seguida, são selecionados os "clusters" que têm faixas de NA (NC) sobrepostas a ele, ou seja, mesma faixa — "clusters" *sobrepostos*.

Para que dois "clusters" possam ser considerados sobrepostos, é necessário que a faixa de valores de NA (NC) em comum entre os dois "clusters" seja muito maior do que

a faixa não comum do "cluster" sobreposto em relação ao majoritário. De uma maneira formal, sejam CA e CB os conjuntos-faixas de valores de NA (NC) dos "clusters" A e B, respectivamente. Supondo que A seja majoritário — |CA|>|CB| —, então B é considerado como "cluster" sobreposto a A se |CA∩CB| >> |CB−CA|. Experimentalmente, observamos que se pelo menos 75% de uma faixa do "cluster" candidato é coberta pelo "cluster" majoritário, então aquele é considerado um "cluster" sobreposto. A Figura 2.5(b) é um exemplo de "clusters" sobrepostos, onde o "cluster" majoritário é destacado e os "clusters" sobrepostos a ele estão dentro da faixa f1−f2 de NA (NC).

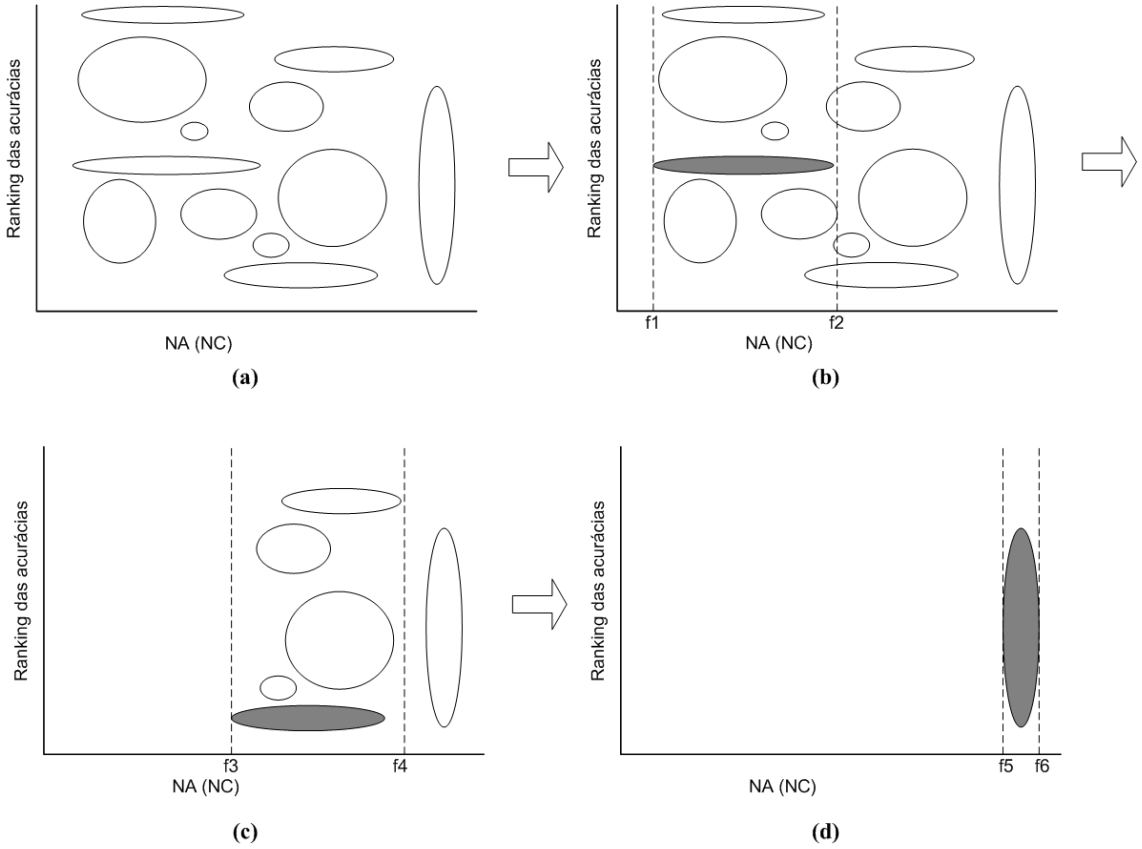


Figura 2.5: "Clusters" para um processo de MD P

Encontrados os "clusters" sobrepostos ao "cluster" majoritário, o próximo passo é contar quantos foram os arquivos de dados da união do "cluster" majoritário com os "clusters" sobrepostos, para os quais P obteve posição ≤ limiar_de_qualidade (posição > limiar_de_qualidade). A decisão de podar ou não o processo de MD P para a faixa de NA (NC) abrangida pela união do "cluster" majoritário com os "clusters" sobrepostos é por maioria simples: se a maioria

de pontos da união está em posições acima do limiar de qualidade então P é podado do espaço de busca para novos arquivos de dados, caso contrário P não é podado. Em síntese, se o processo P for (não for) podado, então não é (é) indicado para arquivos a ser minerados com a faixa de valores de NA (NC) dentro da faixa de valores de NA (NC) da união dos "clusters".

O algoritmo é iterativo, com a iteração terminando quando não houver mais "clusters" majoritários. As Figura 2.5(c) e Figura 2.5(d) ilustram respectivamente as uniões de "clusters" majoritários e sobrepostos para as segunda e terceira iterações.

Seguem-se alguns resultados de nossos experimentos com o algoritmo.

Exemplo 1

Primeiro, seguiremos o exemplo que vinha sendo usado. Para uma melhor visualização, condensamos a Figura 2.3 e Tabela 2.1 na Tabela 2.3, adicionando mais informações sobre a aplicação do algoritmo Mining Process Pruning aos "clusters" induzidos para o processo de MD AIF-HO-Prism considerando NC — os "clusters" majoritários estão discriminados pela marcação '*' na coluna Cluster enquanto os "clusters" sobrepostos podem ser identificados pela coluna Faixa de valores sobreposta; para cada definição, as sub-colunas bom e ruim informam o número de arquivos de dados para os quais o processo AIF-HO-Prism indica desempenho bom e ruim (em relação à acurácia), respectivamente.

Para os "clusters" indicados na Tabela 2.3, analisamos três faixas de valores de NC. A primeira faixa envolve os "clusters" 0 e 2. Já as duas faixas restantes abarcam os "clusters" 3 e 1, respectivamente, porém ambas são descartas por o suporte mínimo ser baixo, isto é, suporte < 10.

Para a primeira faixa, que tem o "cluster" 0 como majoritário, contabilizamos o número de pontos (arquivos) com posição > 40 e posição ≤ 40: o número com desempenho ruim (posição > 40) é maior do que o número com desempenho bom (posição ≤ 40).

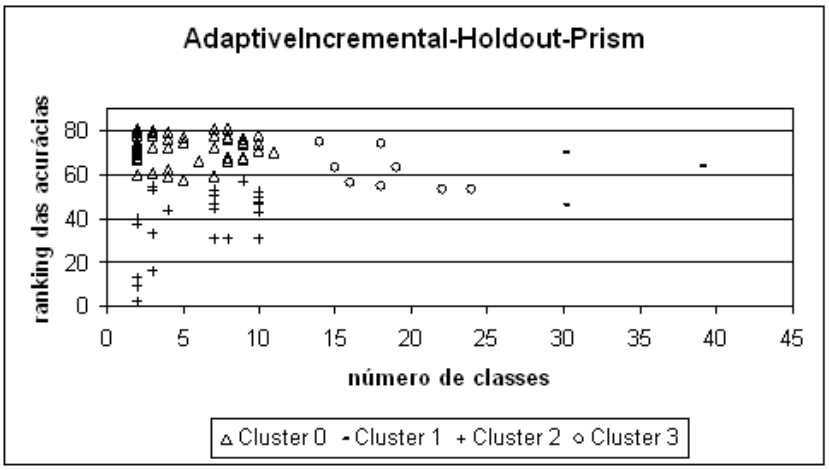
AIF−HO−Prism (NC)				
Cluster	**Definições**	**Suporte**		**Faixas de valores sobrepostas**
		Bom	**Ruim**	
0*	class 0 IF: 2<=NC<=11 ^ 58<=posicao<=81	0	57	2
1*	class 1 IF: 30<=NC<=39 ^ 46<=posicao<=70	0	3	-
2	class 2 IF: 2<=NC<=10 ^ 2<=posicao<=57	10	15	0
3*	class 3 IF: 14<=NC<=24 ^ 53<=posicao<=75	0	8	-
Gráfico				
				
Heurística				
O processo AIF−HO−Prism deve ser descartado do espaço de busca quando NC≤11.				

Tabela 2.3: Heurística para o processo de MD AIF−HO−Prism considerando NC

Concluindo, o leitor pode perceber que chega-se à seguinte heurística: *o processo de MD AIF−HO−Prism deve ser descartado do espaço de busca de processos de MD quando o número de classes NC do arquivo a ser minerado estiver na faixa NC≤11* (note que a condição não ficou 2≤NC≤11 uma vez que não existe arquivo de dados com menos de duas classes).

Exemplo 2

Neste exemplo, abordamos o arquivo intermediário correspondendo ao processo de MD AIF−BS−DS e o atributo NA. Os quatro "clusters" induzidos e suas respectivas definições estão indicados na Tabela 2.4.

Considerando o desvio padrão, obtivemos as seguintes faixas de NA para cada "cluster" ao analisar o modelo de classificação induzido: "cluster" 0 = 7-22, "cluster" 1 = 7-25, "cluster" 2 = 3-15, e "cluster" 3 = 34-55.

Seguindo o algoritmo `Mining Process Prunning`, o primeiro "cluster" majoritário selecionado é o 3, porém o mesmo não tem nenhum "cluster" sobreposto — observe que o gráfico contido na tabela facilita de certa forma verificar se um "cluster" é ou não sobreposto a outro. Então, passamos para a segunda iteração na qual o "cluster" 1 é selecionado como majoritário e os outros dois restantes, 0 e 2, como sobrepostos a ele. Desta forma, duas são as faixas de `NA` que serão avaliadas: 7-25 e 34-55. Ambas as faixas de `NA` a serem avaliadas têm suporte aceitável, com valores acima de dez arquivos de dados.

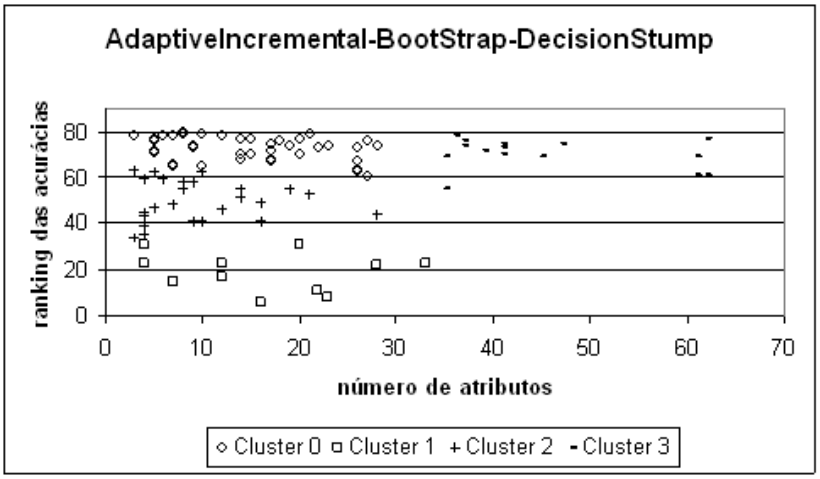
AIF-BS-DS (NA)				
Cluster	**Definições**	**Suporte**		**Faixas de valores sobrepostas**
		Bom	**Ruim**	
0	NA: Mean=14,94 StandardDev = 7,61 posicao: Mean = 72,79 StandardDev = 5,0	0	40	1
1*	NA: Mean = 16,43 StandardDev = 9,2981 posicao: Mean = 18,93 StandardDev = 8,1052	11	0	0,2
2	NA: Mean = 9,66 StandardDev = 6,2063 posicao: Mean = 49,83 StandardDev = 8,5075	0	25	1
3*	NA: Mean = 44,72 StandardDev = 10,6181 posicao: Mean = 70,12 StandardDev = 6,3959	0	16	-
Gráfico				
			
Heurística				
O processo de MD `AIF-BS-DS` deve ser descartado do espaço de busca quando `3≤NA≤25` ou `34≤NA≤55`.				

Tabela 2.4: Heurística para o processo de MD `AIF-BS-DS` considerando `NA`

Avaliando o processo `AIF-BS-DS` quanto ao desempenho em relação à acurácia, temos para a faixa 3-25 (resultante da união das faixas dos "clusters" 0,1 e 2), 65 instâncias com desempenho ruim e 11 com desempenho bom. Então: conclui-se que

o processo de MD AIF−BS−DS deve ser descartado do espaço de busca quando 3≤NA≤25. Analisando o "cluster" 3 da faixa 34-55, nota-se que o desempenho é predominantemente ruim, findando, portanto, que o processo de MD AIF−BS−DS deve ser descartado do espaço de busca quando 34≤NA≤55. Unificando as conclusões, chegamos à heurística descrita na Tabela 2.4.

Exemplo 3

Para o processo de MD AIF−HO−DS e o atributo NC obtivemos os "clusters" cujas definições estão apresentadas na Tabela 2.5. Aplicando o algoritmo Mining Process Pruning a estes "clusters", três é o número de faixas de NC selecionadas para análise: (1) faixa 8-22 envolvendo o "cluster" 0, (2) faixa 24-39 coberta pelo apenas "cluster" 1, e (3) faixa 2-10 abrangendo os "clusters" 2 e 3, em que este último é o majoritário.

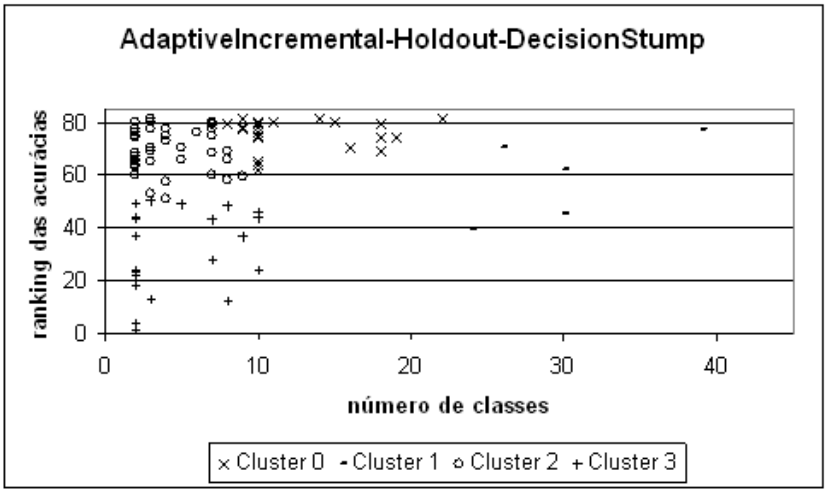
AIF−HO−DS (NC)				
Cluster	**Definições**	**Suporte**		**Faixas de valores sobrepostas**
		Bom	**Ruim**	
0*	class 0 IF: 9<=NC<=22 ^ 62<=posicao<=81 class 0 IF: NC=8 ^ posicao=79	0	24	-
1*	class 1 IF: 24<=NC<=39 ^ 39<=posicao<=77	1	4	-
2	class 2 IF: 2<=NC<=7 ^ 51<=posicao<=81 class 2 IF: NC=8 ^ 58<=posicao<=69 class 2 IF: NC=9 ^ posicao=59	0	42	3
3*	class 3 IF: 2<=NC<=10 ^ 1<=posicao<=50	14	10	2
Gráfico				
			
Heurística				
O processo de MD AIF−HO−DS deve ser descartado do espaço de busca quando NC≤22				

Tabela 2.5: Heurística para o processo de MD AIF−HO−DS considerando NC

Dentre as faixas selecionadas, a única que não dispõe de suporte suficiente é a faixa 24-39, pois o "cluster" 1 detém apenas 5 instâncias do arquivo de dados intermediário em foco.

Quanto ao desempenho, a faixa de valores abrangida pelos "clusters" 2 e 3 é ruim na grande maioria — a soma dos pontos com desempenho ruim para os dois "clusters" é maior que a com desempenho bom. Por fim, para a faixa 8-22, o desempenho ruim é predomina.

No "cluster" 3, a regra de decisão cobre 24 instâncias que estão dispostas entre desempenho bom e ruim, porém aquele é prevalecente — para contabilizar quantas instâncias pertencem a cada caso, consultamos a base de conhecimento. Por fim, para a faixa 8-2 o desempenho ruim predomina.

Desta forma, após a avaliação, inferimos que: *o processo de MD* `AIF-HO-DS` *deve ser descartado do espaço de busca quando* `NC≤22`.

Exemplo 4

Os "clusters" da Tabela 2.6 foram induzidos a partir do arquivo intermediário contendo informações do processo de MD `NS-BS-IBk` e do atributo `NC`. Com o algoritmo de poda de processos de MD, chegamos a três faixas de `NA` a serem avaliadas: faixa 22-39 cingida pelo "cluster" 1, faixa 7-19 coberta pelo "cluster" 0, e faixa 2-10 dos "clusters" 2 e 3.

Todas estas faixas têm o suporte mínimo requerido e para todas elas o processo `NS-BS-IBk` apresenta um desempenho predominantemente bom. Por exemplo, para a faixa 2-10, o referido processo apresentou desempenho bom em 46 casos e ruim, em apenas 5 casos. Desta forma, para este processo de MD, não chegamos a nenhuma heurística.

Usando a mineração não-supervisionada, descobrimos ao todo 80 heurísticas de acurácia, que estão listadas no Apêndice A. Maiores detalhes sobre as mesmas poderão ser obtidos pelo Apêndice B (disponível via CD anexo) através de tabelas como as Tabela 2.3, Tabela 2.4, Tabela 2.5 e Tabela 2.6, que contêm as informações necessárias para chegar a cada uma destas heurísticas.

NS-BS-IBk (NC)				
Cluster	Definições	Suporte		Faixas de valores sobrepostas
		Bom	Ruim	
0*	class 0 IF : 7.0<=NC<=19.0 ^ 1.0<=posicao<=5.0 (29) class 0 IF : 8.0<=NC<=16.0 ^ 6.0<=posicao<=12.0 (8)	37	0	-
1*	class 1 IF : 22.0<=NC<=39.0 ^ 1.0<=posicao<=20.0 (6)	6	0	-
2	class 2 IF : 2.0<=NC<=5.0 ^ 1.0<=posicao<=5.0 (12) class 2 IF : 2.0<=NC<=7.0 ^ 6.0<=posicao<=22.0 (25)	37	0	3
3*	class 3 IF : 2.0<=NC<=10.0 ^ 26.0<=posicao<=75.0 (15)	9	5	2
Gráfico				

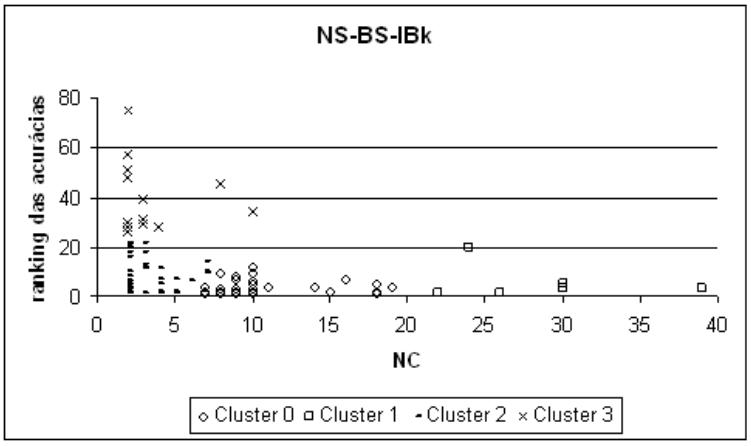


Tabela 2.6: Heurística para o processo de MD `NS-BS-IBk` considerando `NA`

2.1.2 Meta-mineração com classificação supervisionada

Com a classificação não-supervisionada utilizamos o critério acurácia para avaliar processos de MD no intuito de afastá-los do espaço de busca quando os mesmos apresentam desempenho ruim em determinadas faixas de valores de `NA` ou `NC`. Já com a classificação supervisionada, o critério em foco é tempo de execução com o fim de descartar processos de MD que aplicam técnicas de amostragem para arquivos de dados pequenos ou não as aplicam para arquivos de dados grandes. Para isto, teremos que conceituar arquivos grandes e pequenos.

O objetivo de técnicas de amostragem é reduzir o tempo de execução de processos de MD através da extração de amostras de grandes conjuntos de dados. Obviamente, se um arquivo de dados contém poucas instâncias, não é necessária amostragem. A questão é: um arquivo de dados para mineração com 3000 instâncias, por exemplo, é grande ou pequeno? Para responder a esta pergunta, utilizamos como

critério a relação entre soma do tempo de execução de todos os processos de MD para um arquivo de dados, e o número de instâncias do arquivo.

Dos 95 arquivos de dados usados na experimentação, para 20 deles o tempo total para executar todos os 81 processos de MD foi mais de uma hora; para 69, menos de trinta minutos; finalmente, 6 despenderam um tempo entre 30 e 60 minutos. Diante destes resultados, julgamos razoável estabelecer o limiar 60 minutos para decidir se um arquivo a minerar é grande ou pequeno. Formalizando, seja uma função `tempo_execucao(Arquivo,{`P_n`, n = 1, ..., 81})`, que retorna o tempo de execução, em minutos, de todos os processos de MD P_n aplicados ao arquivo de dados `Arquivo`: se o valor retornado for maior que 60 então `Arquivo` é grande, senão `Arquivo` é pequeno. Para os experimentos, usamos um computador com processador `Pentium III` 900 MHz e 512 MB de memória RAM para rodar estes experimentos.

Desta forma, o próximo passo é mapear o tempo de execução total para um arquivo de dados em função do número de instâncias a fim de responder o seguinte questionamento: a partir de quantas instâncias um arquivo de dados é classificado como grande ou pequeno? Então, criamos o arquivo intermediário, descrito no início Seção 2.1, que contém os atributos: `NI` (número de instâncias dos arquivos experimentados) e `Tempo_total` (soma dos tempos de execução dos 81 processos de MD) — atributo de classificação. Os valores deste atributo foram discretizados de acordo o limiar estabelecido de 60 minutos; portanto, suas respectivas classes são $\leq$`60` e `>60` que representam, respectivamente, um tempo de execução total menor ou igual a uma hora e maior do que uma hora.

Para minerar o arquivo intermediário, não empregamos amostragem e fragmentação, pois o mesmo tem poucas instâncias (apenas 95). Finalmente mineramo-lo e, dentre todos os classificadores, o resultado mais significativo, ilustrado na Figura 2.6, foi obtido pelo classificador `J48`. Como contra-exemplo observe o modelo de classificação induzido pelo classificador `NNge` na Figura 2.7: as regras de classificação são muitas e muito detalhistas ao contrário de `J48`, que conseguiu sintetizar as informações do arquivo intermediário em apenas duas regras.

```
NI <= 2567: <=60 (71.0/1.0)
NI > 2567: >60 (24.0/3.0)
```

Figura 2.6: Modelo induzido pela classificação supervisionada via `J48`

```
class <=60 IF : 144.0<=number_instances<=1728.0 (64)
class <=60 IF : 2048.0<=number_instances<=2567.0 (6)
class <=60 IF : 3922.0<=number_instances<=3949.0 (2)
class <=60 IF : number_instances=13996.0 (1)
class >60 IF : number_instances=1797.0 (1)
class >60 IF : 2644.0<=number_instances<=3772.0 (6)
class >60 IF : 4000.0<=number_instances<=8760.0 (13)
class >60 IF : 20000.0<=number_instances<=28056.0 (2)
```

Figura 2.7: Modelo induzido pela classificação supervisionada via `NNge`

Relembrando a função `tempo_execucao(Arquivo, {`P_n`, n = 1, ..., 81})`, mapeamo-la em função no número de instâncias, baseados no modelo de classificação da Figura 2.6: se o valor retornado for maior que 60 (`NI > 2567`) então `Arquivo` é grande; senão (`NI ≤ 2567`), `Arquivo` é pequeno. Assim, chegamos as seguintes heurísticas de tempo:

H81 O uso de amostras deve ser dispensado, quando o arquivo de dados é pequeno (`NI<=2567`);

H82 O uso de amostras deve ser empregado, quando o arquivo de dados é grande (`NI>2567`).

2.2 Análise do Custo de Processamento dos Classificadores

Com a meta-mineração descobrimos padrões (heurísticas) interessantes a respeito de processos de MD, levando em conta os critérios acurácia e tempo de execução. Com a Análise do Custo de Processamento dos Classificadores pretendemos dar continuidade à descoberta de heurísticas de tempo.

Para tanto, consultamos a base de conhecimento no intuito de identificar qual classificador contribui para que um processo de MD `<amostragem, fragmentação,classificação>` seja mais dispendioso que outros em relação ao tempo de execução — não analisamos o tempo focando técnicas de amostragem e fragmentação porque estas são rápidas; a complexidade em termos de tempo de processamento se detém nos algoritmos de indução.

O objetivo é descartar apenas *o classificador mais dispendioso*, e, não, os dois ou três mais dispendiosos, por exemplo. Isto porque o descarte de um classificador representa nove processos de MD fora do escopo de busca, o que já pode comprometer a acurácia devido ao problema da instabilidade — lembre que nesta análise temos como critério tempo de execução e, não, acurácia. Portanto, a eliminação de mais de um classificador poderia significar um grande aumento na possibilidade de se ter uma perda considerável na qualidade da acurácia.

Então, contabilizamos a quantidade de arquivos de dados em que cada classificador aparece entre as `n` últimas posições no ranking do tempo, onde $n \in \{1,2,3,4,5,6,7,8,9,10\}$. Em outras palavras, verificamos em quantos arquivos de dados cada classificador está entre os `n` processos de MD mais onerosos em termos de tempo de execução. Os resultados desta contagem estão na Tabela 2.7.

Classificadores	**Percentagem de arquivos de dados em que o classificador está entre os `n` processos de MD mais dispendiosos em termos de tempo de execução**									
	10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
DS	26,32	25,26	23,16	22,11	18,95	17,89	16,84	14,74	13,68	11,58
NNge	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
J48	1,05	1,05	1,05	1,05	1,05	1,05	1,05	0,00	0,00	0,00
Id3	24,21	22,11	21,05	21,05	17,89	16,84	13,68	10,53	9,47	8,42
IB1	92,63	91,58	86,32	85,26	82,11	71,58	65,26	61,05	18,95	1,05
IBk	94,74	94,74	94,74	93,68	90,53	90,53	87,37	74,74	73,68	71,58
NB	1,05	1,05	1,05	0,00	0,00	0,00	0,00	0,00	0,00	0,00
OneR	1,05	1,05	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Prism	44,21	40,00	31,58	26,32	24,21	18,95	7,37	4,21	2,11	2,11

Tabela 2.7: Avaliação do tempo de execução despendido pelos classificadores

Para citar um exemplo da tabela, podemos dizer que o classificador `Prism` está entre os cinco processos de MD mais onerosos em 18% dos 95 arquivos de dados contidos na base de conhecimento. Analisando-a minuciosamente, vemos que os classificadores `IBk` e `IB1` têm os valores mais altos, por exemplo, estão entre os dez processos mais dispendiosos em 94,74% e 92,63% dos 95 arquivos de dados, respectivamente. Entretanto, à medida que o valor de `n` diminui, a diferença aumenta:

IBk passa a ter valores notavelmente maiores. Desta forma, concluímos que `IBk` é o classificador mais dispendioso, dentre os nove, findando então na seguinte heurística:

H83 O classificador `IBk` deve ser descartado do espaço de busca, quando o arquivo de dados é grande (NI>2567).

2.3 Considerações finais

Neste capítulo, apresentamos as heurísticas descobertas concernentes aos critérios acurácia e tempo de execução bem como o processo que seguimos para descobri-las.

As 80 heurísticas de acurácia descobertas cobriram 43 processos de MD. Dentre estes processos, 4 fazem referência somente ao atributo `NA`, enquanto 3 heurísticas referem-se exclusivamente ao atributo `NC`. Logo, os 36 processos restantes, levaram em conta tanto o atributo `NA` quanto o atributo `NC`. Fica claro então que, para 37 (81 – 43) processos de MD nenhuma heurística de acurácia foi descoberta. Lembramos que, dado um arquivo de dados a minerar, é necessário que o mesmo satisfaça as condições impostas relativas aos atributos `NA` e `NC` para que os processos de MD cobertos pelas heurísticas de acurácia sejam podados do escopo de busca.

No que concerne às heurísticas de tempo, os 9 processos de MD relativos ao classificador `IBk` foram rejeitados; ou seja, definitivamente, o classificador `IBk` não se revelou um bom classificador quanto ao tempo de execução, na comparação com os demais classificadores. Também, 27 processos de MD devem ser descartados se o arquivo para mineração for grande; se o arquivo for pequeno, 54 processos de MD, ou o dobro, devem ser rejeitados.

Quando à eficácia das heurísticas descobertas, ela só pode ser avaliada unicamente por experimentação; isto é da própria natureza de qualquer heurística. Então, para validar nossas heurísticas, construímos uma ferramenta que permite comparar um modelo de classificação obtido de uma busca exaustiva dentre todos os processos de MD possíveis, e um outro modelo de classificação dentre certos processos de MD com a busca sendo guiada unicamente pelas heurísticas. O projeto e a implementação da ferramenta é o tema do capítulo seguinte. Finalmente, os testes com a ferramenta são analisados no Capítulo 4.

Capítulo 3

Um "framework" baseado em heurísticas para mineração de dados

Neste capítulo, discutimos a análise e projeto de um "framework" que tem como objetivo geral propiciar um ambiente interativo que facilite a tarefa do minerador de encontrar rapidamente um modelo de classificação com acurácia aceitável para seu problema.

"Um framework provê uma solução para uma família de problemas semelhantes, usando um conjunto de classes e interfaces que mostra como decompor a família de problemas. Como objetos dessas classes colaboram para cumprir suas responsabilidades, o conjunto de classes deve ser flexível e extensível para permitir a construção de várias aplicações com pouco esforço, especificando apenas as particularidades de cada aplicação" (Sauvé, 2005). Tendo em vista essas características, o uso da abordagem de "framework" oferece inúmeras vantagens como redução do tempo de desenvolvimento e menos manutenção, entre outras (Landin & Niklasson, 1995). Daí nossa justificativa para usar tal abordagem.

Na Seção 3.1 discorremos sobre os requisitos funcionais e não-funcionais do "framework", e, na Seção 3.2, apresentamos sua arquitetura.

3.1 Requisitos

Os principais requisitos funcionais (RF) da ferramenta são:

RF1 Escolher o arquivo de dados a ser minerado: um arquivo de dados classificado — isto é, com um atributo de classificação — deve ser indicado pelo minerador à ferramenta;

RF2 Escolher técnicas de amostragem: o minerador deve poder determinar quais técnicas de amostragem utilizar no processo de MD, baseado em sua experiência com processos de mineração;

RF3 Escolher técnicas de fragmentação: o minerador deve poder escolher as técnicas de fragmentação, de conformidade com a experiência adquirida com outros problemas de mineração;

RF4 Escolher classificadores: o minerador deve poder selecionar os classificadores de acordo o modelo de classificação desejado (regras de decisão, árvore de decisão, etc) e/ou conheça classificadores que produzam resultados comparativamente melhores;

RF5 Definir os critérios de qualidade do modelo de classificação a ser induzido: deve ser permitido ao minerador ponderar entre os critérios acurácia e tempo de execução antes de executar os processos de MD para a obtenção de modelos de classificação. Se optar pela melhor acurácia possível dentre as técnicas disponíveis, implicará em um tempo de execução alto, visto que o melhor modelo de classificação será obtido via o emprego exaustivo de todos os processos de mineração possíveis. Todavia, o minerador pode optar ainda por uma acurácia que não seja necessariamente a melhor, mas que lhe seja plenamente aceitável. Neste caso, o tempo de execução poderá ser (muito) menor quando comparado à busca exaustiva;

RF6 Obter um modelo de classificação de acordo com o RF5: ao executar o "framework", o minerador deverá obter um modelo de classificação de acordo com seus critérios de acurácia e/ou tempo de execução.

Os requisitos não-funcionais do software são:

RNF1 Interface gráfica: o *"*framework*"* deve ser dotado de uma interface gráfica no intuito de facilitar a interação com o minerador;

RNF2 Extensibilidade: Possibilitar a adição fácil de novas técnicas ao software, sejam elas de amostragem, fragmentação, classificação ou ainda algoritmos que automatizam a execução de processos de MD;

RNF3 Base de conhecimento heurístico: as heurísticas descobertas devem ajudar na redução do espaço de busca de processos de mineração, no intuito de induzir modelos de classificação com acurácia e tempo de execução aceitáveis.

3.2 Arquitetura

A arquitetura do *"framework"* é composta por três camadas: persistência, aplicação e apresentação.

A camada de persistência mantém as informações acessadas pela aplicação, que são as classes de objetos que representam a base de conhecimento heurístico e os algoritmos de amostragem, fragmentação e indução envolvidos.

A camada de aplicação diz respeito à lógica envolvida ao escolher, dentre vários processos de MD, aquele que induz um modelo de classificação de acordo os critérios de qualidade desejados pelo minerador.

Por fim, a camada de apresentação é responsável por dispor de forma gráfica a interação com a camada de aplicação.

Nas sub-seções seguintes, detalhamos cada uma das camadas da arquitetura descrita.

3.2.1 Camada de persistência

A estrutura de classes que tratam as técnicas de amostragem, fragmentação e classificação, está ilustrada na Figura 3.1 através de um diagrama de classes, em notação UML (Larman, 1998). Descrevemos analiticamente cada classe do diagrama da Figura 3.1 nas Figura 3.2, Figura 3.3, Figura 3.4 e Figura 3.5, usando como notação código-fonte JAVA. Para permitir uma maior compreensão do leitor, em todas as figuras há um comentário sucinto para cada atributo e assinatura de método.

Observando a Figura 3.1, note que uma nova técnica, seja de amostragem, fragmentação ou indução, pode ser facilmente adicionada — requisito RNF2 —, bastando apenas que estenda a respectiva classe abstrata, que tem como função fatorar o código comum entre os respectivos tipos de técnicas. Esta facilidade é expressada no diagrama pelas classes '?'.

A classe `MiningProcess` (Figura 3.2) exprime um processo de MD composto pela tripla de interfaces (técnicas) <`SamplingTechnique`, `SplittingTechnique`, `ClassificationTechnique`>.

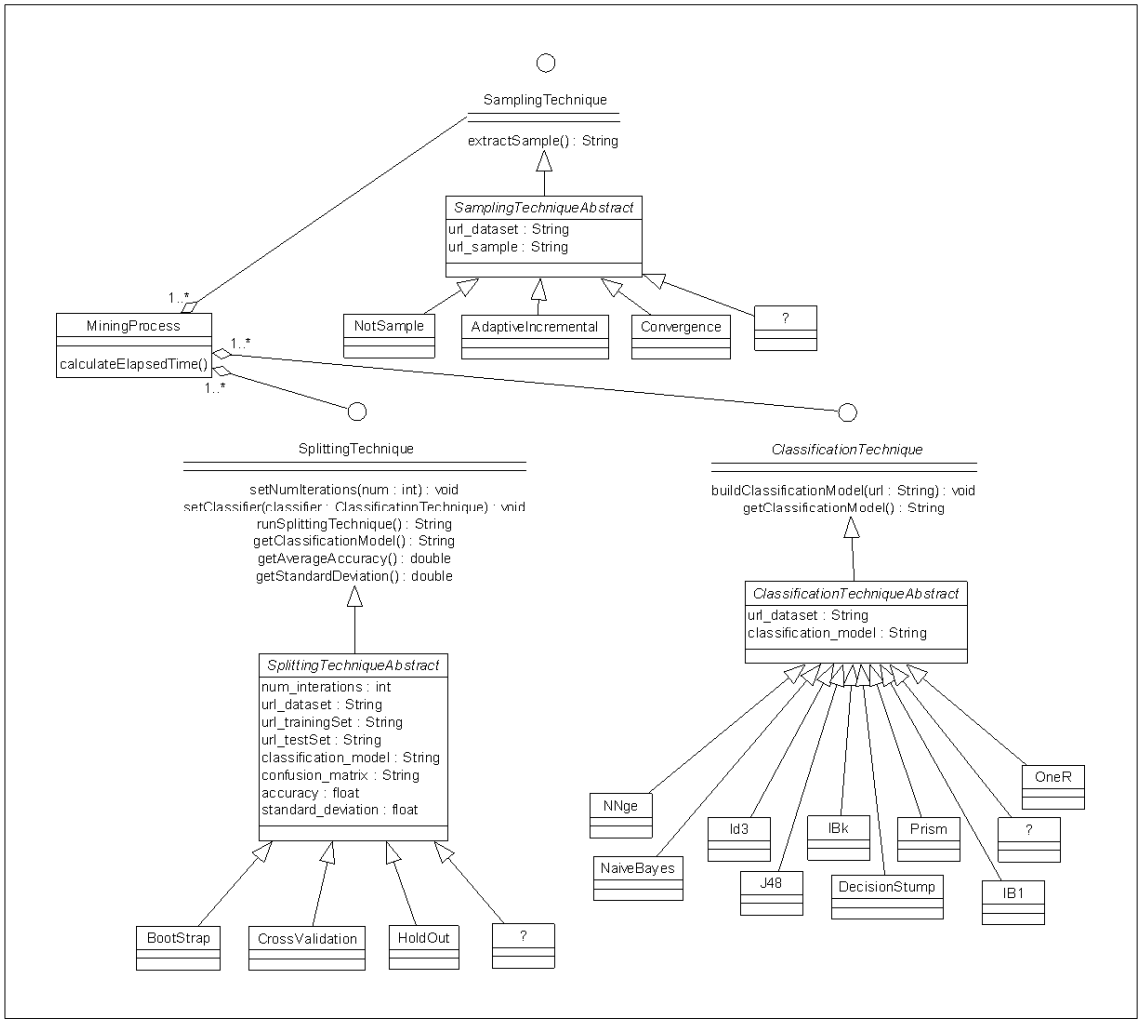


Figura 3.1: Diagrama de classes das técnicas para as etapas de amostragem, fragmentação e classificação

```
class MiningProcess {
  double elapsed_time;  /* tempo despendido na execução do processo */
  SamplingTechnique samplingTechnique;
       /* técnica de amostragem que compõe o processo */
  SplittingTechnique splittingTechnique;
       /* técnica de fragmentação que compõe o processo */
  ClassificationTechnique classificationTechnique;
       /* técnica de classificação que compõe o processo */

  void calculateElapsedTime();
       /* calcula o tempo gasto para executar o processo de MD composto pela tripla
       <SamplingTechnique, SplittingTechnique, ClassificationTechnique> */
}
```

Figura 3.2: Descrição JAVA da classe `MiningProcess`

A descrição das classes pertinentes às técnicas de amostragem está na Figura 3.3. É através da interface `SamplingTechnique` que o minerador pode escolher a(s) técnica(s) de amostragem a serem empregados nos processos de mineração de seu arquivo de dados. Ao implementar a interface `SamplingTechnique`, a classe

abstrata `SamplingTechniqueAbstract` fatora o código comum entre as técnicas de amostragem. E o método abstrato `extractSample()`, implementado nas classes `AdaptiveIncremental`, `Convergence` e `Not Sample`, é polimórfico, ou seja, tem a mesma assinatura, mas implementações diferentes, de acordo a lógica da respectiva técnica. Tanto estas classes quanto as classes relativas às técnicas de fragmentação foram reusadas de (Toebe, 2002).

```
interface SamplingTechnique {
  String extractSample();       /* retorna a url da amostra extraída do arquivo de dados */
}

abstract class SamplingTechniqueAbstract implements SamplingTechnique{
  String url_dataset;    /*url do arquivo de dados a ser minerado */
  String url_sample;     /*url da amostra extraída*/

  abstract String extractSample();
       /* retorna a url da amostra extraída do arquivo de dados. A classe que estender este método, deve
       implementá-lo segundo a lógica de um algoritmo de amostragem */
}

class NotSample extends SamplingTechniqueAbstract {
  String extractSample();       /* extrai a amostra segundo a técnica Not Sample */
}

class Convergence extends SamplingTechniqueAbstract  {
  String extractSample();       /* extrai a amostra segundo a técnica Convergence */
}

class AdaptiveIncremental extends SamplingTechniqueAbstract {
  String extractSample();
       /* extrai a amostra segundo a técnica Adaptive Incremental Framework */
}
```

Figura 3.3: Descrição JAVA das classes referentes às técnicas de amostragem

Como acontece com as técnicas de amostragem, para as técnicas de fragmentação também existe uma interface (`SplittingTechnique`) implementada por uma classe abstrata (`SplittingTechniqueAbstract`) que tem um método polimórfico (`runSplittingTechnique()`). Tal método é redefinido pelas classes `Boostrap`, `CrossValidation` e `Holdout` de acordo a respectiva lógica de cada técnica, como pode ser visto na Figura 3.4. É através da interface `SplittingTechnique` que o minerador pode escolher a(s) técnica(s) de fragmentação a serem empregadas nos processos de mineração de seu arquivo de dados.

```
interface SplittingTechnique {
  void setNumIterations(int num);
       /* define quantas iterações a técnica de fragmentação fará */
  void setClassificationTechnique(ClassificationTechnique classifier);
       /* especifica qual classificador será utilizado para induzir conhecimento do arquivo de dados */
  void runSplittingTechnique();
       /* põe em ação a técnica de fragmentação. Primeiramente, o arquivo é fragmentado de acordo
       a lógica da técnica de fragmentação implementada. Em seguida, com o conjunto de treinamento,
       um modelo de classificação é induzido utilizando o classificador especificado enquanto que, com
       o conjunto de teste, o modelo de classificação é avaliado. Todo este processo é repetido n vezes,
       onde n é o número de iterações definido. */
  String getClassificationModel();
       /* obtém o modelo de classificação induzido*/
  double getAverageAccuracy();
       /* informa a acurácia média obtida das n iterações realizadas*/
  double getStandardDeviation();
       /* retorna o desvio padrão da média das acurácias*/
}

abstract class SplittingTechniqueAbstract
       implements SplittingTechnique{
  int num_iterations;                /*número de iterações que a técnica é aplicada*/
  String url_dataset;                /*url do arquivo de dados a minerar */
  String url_trainingSet;            /*url do conjunto de treinamento */
  String url_testSet;                /*url do conjunto de teste */
  String classification_model;       /*modelo de classificação induzido*/
  double accuracy;                   /*acurácia média do conjunto de teste*/
  double standard_deviation;         /*desvio padrão da acurácia média*/
  ClassificationTechnique classifier;
       /* classificador usado para induzir conhecimento*/

  void setNumIterations(int num);
  void setClassificationTechnique(ClassificationTechnique classifier);
  void runSplittingTechnique();
  String getClassificationModel();
  double getAverageAccuracy();
  double getStandardDeviation();
  abstract void runSplittingTechnique();
       /* põe em ação a técnica de fragmentação. A classe que estender este método, deve
       implementa-lo segundo a lógica de um algoritmo de amostragem */
}

class Holdout extends SplittingTechniqueAbstract  {
  void runSplittingTechnique();
       /* implementação da etapa de fragmentação segundo a técnica Holdout */
}

class Bootstrap extends SplittingTechniqueAbstract {
  void runSplittingTechnique();
       /* implementação da etapa de fragmentação segundo a técnica Bootstrap */
}

class CrossValidation extends SplittingTechniqueAbstract {
  void runSplittingTechnique();
       /* implementação da etapa de fragmentação segundo a técnica Cross Validation */
}
```

Figura 3.4: Descrição JAVA das classes referentes às técnicas de fragmentação

```
interface ClassificationTechnique {
  void buildClassificationModel(String url);
       /* induz o modelo de classificação do arquivo de dados apontado por uma url */
  String getClassificationModel();
       /* obtém o modelo de classificação induzido*/
}

abstract class ClassificationTechniqueAbstract
       implements ClassificationTechnique {
  String url_dataset;            /*url do arquivo de dados (conjunto-treinamento) a ser minerado */
  String classification_model;   /*modelo de classificação induzido*/

  String getClassificationModel();
  abstract void buildClassificationModel(String url);
  /* induz um modelo de classificação do arquivo de dados apontado por uma url. A classe que estender este
       método, deve implementá-lo segundo a lógica de um algoritmo de classificação */
}

class J48 extends ClassificationTechniqueAbstract {
  void buildClassificationModel(String url);
  /* induz um modelo de classificação a partir do arquivo de dados indicado, segundo o classificador J48 */
}

class DecisionStump extends ClassificationTechniqueAbstract {
  void buildClassificationModel(String url);
  /* induz um modelo de classificação a partir do arquivo de dados indicado, segundo o classificador Decision
       Stump */
}

class IB1 extends ClassificationTechniqueAbstract {
  void buildClassificationModel(String url);
  /* induz um modelo de classificação a partir do arquivo de dados indicado, segundo o classificador IB1 */
}

class IBk extends ClassificationTechniqueAbstract {
  void buildClassificationModel(String url);
  /* induz um modelo de classificação a partir do arquivo de dados indicado, segundo o classificador IBk */
}

class NaiveBayes extends ClassificationTechniqueAbstract {
  void buildClassificationModel(String url);
  /* induz um modelo de classificação a partir do arquivo de dados indicado, segundo o classificador Naive
       Bayes */
}

class OneR extends ClassificationTechniqueAbstract {
  void buildClassificationModel(String url);
  /* induz um modelo de classificação a partir do arquivo de dados indicado, segundo o classificador OneR*/
}

class Prism extends ClassificationTechniqueAbstract {
  void buildClassificationModel(String url);
  /* induz um modelo de classificação a partir do arquivo de dados indicado, segundo o classificador Prism */
}

class NNge extends ClassificationTechniqueAbstract {
  void buildClassificationModel(String url);
  /* induz um modelo de classificação a partir do arquivo de dados indicado, segundo o classificador NNge*/
}

class Id3 extends ClassificationTechniqueAbstract {
  void buildClassificationModel(String url);
  /* induz um modelo de classificação a partir do arquivo de dados indicado, segundo o classificador Id3 */
}
```

Figura 3.5: Descrição JAVA das classes referentes aos classificadores

Por fim, nove classes redefinem o método abstrato `buildClassificationModel()` da classe abstrata `ClassificationTechniqueAbstract` (Figura 3.5), que por sua vez implementa a interface `ClassificationTechnique`. Estas classes foram reusadas da biblioteca WEKA (Witten & Frank, 1999).

A base de conhecimento heurístico representa as heurísticas descobertas, que estão todas armazenadas (estruturadas) em um arquivo XML. Cada heurística é mapeada para a classe `Heuristic`, ou seja, um objeto desta classe representa uma heurística. A especificação desta classe assim como a descrição para cada atributo e assinatura de método está na Figura 3.6, em código JAVA.

```
class Heuristic {
  String name;
       /*nome da heurística*/
  int heuristic_type;
       /*tipo da heurística: acurácia (0) ou tempo (1) */
  Collection dependencias;
       /* se alguma heurística desta lista estiver sido aplicada, então a heurística representada pelo
       objeto da classe não poderá ser usada */
  String sampling_technique;
       /* aponta a técnica de amostragem coberta pela heurística*/
  String splitting_technique;
       /* aponta a técnica de fragmentação coberta pela heurística*/
  String classification_technique;
       /* aponta o classificador coberto pela heurística*/
  String feature;
       /* indica qual meta-atributo está sendo avaliado*/
  String operator;
       /* indica o operador que comparará o valor do meta-atributo com o(s) valor(es) definido(s) pela
       heurística */
  String value1;
       /* especifica o valor de comparação da condição */
  String value2;
       /* especifica um segundo valor, caso seja uma faixa de valores */
  String action;
       /* estabelece a decisão que será tomada com a(s) técnica(s) definida(s) */


  String getHeuristic();
       /* retorna todas as especificações da heurística no formato XML */
}
```

Figura 3.6: Descrição JAVA da classe `Heuristic`

Para ilustrar, a Figura 3.7(a) é a saída do método `getHeuristic()` para o objeto da classe `Heuristic` que instancia a heurística `H6`, enquanto que a Figura 3.7(b) é a saída para o objeto que instancia a heurística `H83`.

Observando a Figura 3.7(a), percebemos que trata-se da heurística de acurácia H6 porque os atributos NAME e TYPE discriminam, respectivamente, o nome da heurística e o tipo (heurística de acurácia ou tempo). Portanto, da mesma forma, é possível identificar que a heurística da Figura 3.7(b) é uma heurística de tempo denominada H83.

A "tag" DEPENDS, contida no contexto da heurística H6, determina que se a heurística H5 já tiver sido aplicado, então aquela não o será. Diferentemente de H6, H83 não contém nenhuma dependência.

(a)

```
<HEURISTIC NAME="H6" TYPE="ACCURACY">
      <DEPENDS>H5</DEPENDS>
      <ALGORITHMS TYPE="TRIPLE">
            <SAMPLING>AdaptiveIncremental</SAMPLING>
            <SPLITTING>Holdout</SPLITTING>
            <INDUCER>DecisionStump</INDUCER>
      </ALGORITHMS>
      <CONDITION>
            <FEATURE>class</FEATURE>
            <OPERATOR>between</OPERATOR>
            <VALUE1>2</VALUE1>
            <VALUE2>22</VALUE2>
      </CONDITION>
      <ACTION>reject</ACTION>
</HEURISTIC>
```

(b)

```
<HEURISTIC NAME="H83" TYPE="TIME">
      <ALGORITHMS TYPE="SINGLE">
            <INDUCER>IBk</INDUCER>
      </ALGORITHMS>
      <CONDITION>
            <FEATURE>instance</FEATURE>
            <OPERATOR>greaterequals</OPERATOR>
            <VALUE1>2568</VALUE1>
      </CONDITION>
      <ACTION>reject</ACTION>
</HEURISTIC>
```

Figura 3.7: Trecho do arquivo XML que define as heurísticas

Através do conjunto de "tags" abrangida por ALGORITHMS, é especificado o escopo da heurística, isto é, qual(is) técnica(s) é(são) coberta(s) pela mesma. Na Figura 3.7(a), nota-se que é coberto um processo de MD ou uma tripla de técnicas (atributo TYPE="TRIPLE"), a qual é AIF-HO-DS. Já na Figura 3.7(b), a heurística refere-se à apenas uma única técnica (atributo TYPE="SINGLE"): o classificador IB1.

Uma vez definido o escopo, agora é estabelecida, através das "tags" abarcadas por `CONDITION`, qual a condição, para um certo meta-atributo (`NA`, `NC` ou `NI`), que deve ser satisfeita a fim de aplicar a ação determinada ("tag" `ACTION`). O meta-atributo em foco é apontado pela "tag" `FEATURE`. Já a "tag" `OPERATOR` indica o operador usado para fazer a comparação com o valor definido em `VALUE1`, ou faixa de valores definida por `VALUE1` e `VALUE2`. Na Figura 3.7(a), a condição especificada é traduzida por $2 \leq NC \leq 22$, enquanto que, na Figura 3.7(b), a condição é $NI \geq 2568$.

Finalmente, ambas as heurísticas indicam que se a condição for satisfeita, a(s) técnica(s) especificada(s) deve(m) ser rejeitada(s) do espaço de busca, já que o valor da "tag" `ACTION` é `reject`.

É importante ressaltar que a estrutura do arquivo XML é bastante flexível no sentido de especificar heurísticas para um algoritmo ou um par deles ou ainda um processo de MD, além de permitir que as condições e as ações sejam facilmente configuráveis. Por exemplo, é possível acrescentar sem dificuldades um novo meta-atributo, bastando apenas que o mesmo seja especificado na "tag" `FEATURE`. O arquivo XML, contendo todas as heurísticas descobertas, está no Apêndice C.

Em suma, esta seção descreveu todas as classes acessadas pela camada de aplicação, que discutimos a seguir.

3.2.2 Camada de aplicação

A camada de aplicação, basicamente, compreende a lógica do "framework", ou seja, dado um arquivo de dados a minerar e um conjunto de processos de mineração, é preciso escolher processos que possam induzir um modelo de classificação de acordo os critérios de qualidade estabelecidos pelo minerador — requisito RF5 discutido na Seção 3.1. Portanto, oferecemos dois algoritmos para busca de modelos de classificação (requisito RF6): um que faz investigação exaustiva, denominado `Mining Process Naïve Executer` — proposto por (Toebe, 2002) —, e outro, que os investiga baseado nas heurísticas descobertas, chamado `Mining Process Fast Executer`.

A classe abstrata `MiningProcessesExecuterAbstract`, que implementa a interface `MiningProcessesExecuter`, é o âmago da estrutura das classes da camada de aplicação (Figura 3.8), pois, detém o método abstrato `startSearch()`, responsável por automatizar a execução de processos de MD. Este

método também tem por função propiciar o poliformismo: as classes `MiningProcessesNaiveExecuter` e `MiningProcessesFastExecuter`, ao estender a classe abstrata `MiningProcessesExecuterAbstract`, redefinem tal método, implementando os seus devidos algoritmos. Detalhes a respeito dos atributos e métodos destas classes estão na Figura 3.9.

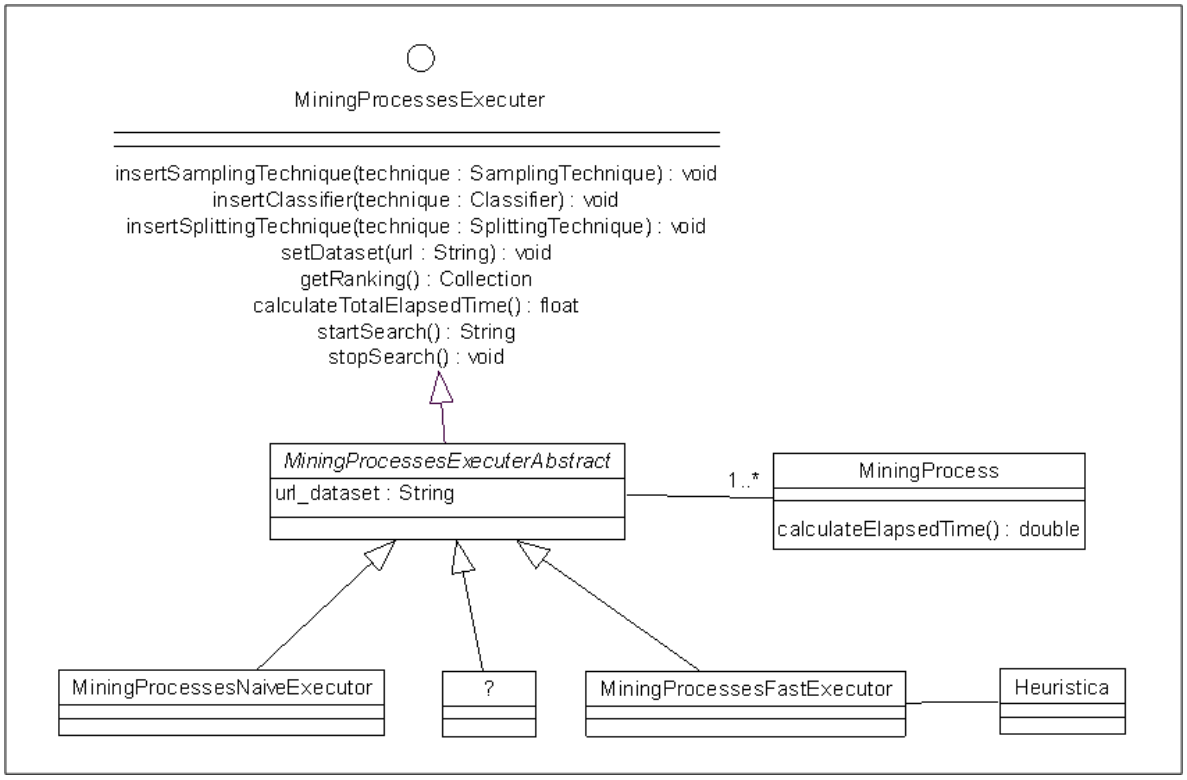


Figura 3.8: Diagrama de classes da camada de negócio

Ainda destacamos que os requisitos RF1, RF2, RF3 e RF4 são atendidos, respectivamente, através da implementação dos métodos `setDataset()`, `insertSamplingTechnique()`, `insertSplittingTechnique()` e `insertClassificationTechnique()`, onde no primeiro é designado o arquivo de dados a minerar e nos demais é possível escolher as técnicas de amostragem, fragmentação e classificação que serão aplicadas.

```
interface MiningProcessesExecuter{
  void insertSamplingTechnique(SamplingTechinque technique);
       /* insere uma técnica de amostragem ao espaço de busca */
  void insertSplittingTechnique(SplittingTechinque technique);
       /* insere uma técnica de fragmentação ao espaço de busca */
  void insertClassificationTechnique(
       ClassificationTechnique classifier);
       /* insere um classificador ao espaço de busca */
  void setDataset(String url);
       /* indica a URL do arquivo de dados a minerar*/
  Collection getRanking();
       /*provê uma coleção de objetos da classe MiningProcess, que representa a tripla de técnicas
       <amostragem, fragmentação,classificação>, ordenada decrescentemente de acordo a acurácia
       obtida por cada um dos processos */
  double calculateTotalElapseTime();
       /* calcula o tempo total gasto para executar todos os processos de MD */
  void startSearch();
       /* inicia a execução de processos de MD */
  void stopSearch();
       /* interrompe a execução de processos de MD*/
}

abstract class MiningProcessesExecuterAbstract
       implements MiningProcessesExecuter {
  String url_dataset;
       /*url do arquivo de dados a minerar */
  Collection miningProcesses;
       /* processos de MD que compõem o espaço de busca */

  abstract void startSearch();
       /*  inicia a execução dos processos de MD. A classe que estender este método, deve
       implementá-lo segundo a lógica de um algoritmo que automatiza a execução de processos de
       MD */
}

class MiningProcessesNaiveExecuter
       extends MiningProcessesExecuterAbstract {
  void startSearch();
       /* inicia a execução dos processos de MD de acordo o algoritmo Mining Processes Naive
       Executer */
}

class MiningProcessesFastExecuter
       extends MiningProcessesExecuterAbstract {
  Collection heuristics;
       /*conjunto de heurísticas usadas para selecionar processos de MD para execução */

  void startSearch();
       /* inicia a execução dos processos de MD de acordo o algoritmo Mining Processes Fast Executer
       */
}
```

Figura 3.9: Descrição JAVA das classes referentes aos algoritmos de execução de processos de MD

Já o requisito RNF3 é satisfeito por meio do relacionamento entre a classe `MiningProcessesFastExecutor` e `Heuristic`, pois o algoritmo `Mining`

`Processes Fast Executor`, implementado pelo método `startSearch()` da primeira classe, utiliza as heurísticas descobertas, que são mapeadas pela segunda classe. Pormenores do algoritmo `Mining Processes Fast Executer` são discutidos a seguir.

O algoritmo Mining Process Fast Executer

O algoritmo `Mining Process Fast Executer`, que está ilustrado em pseudocódigo na Figura 3.10, automatiza a execução de processos de MD levando em conta as etapas de amostragem, fragmentação e classificação, além de usar as heurísticas descobertas para inferir um modelo de classificação aceitável pelo minerador em um tempo menor, se comparado ao outro algoritmo, que faz busca exaustiva.

```
String startSearch (arquivo de dados A, inteiro tipo_heuristicas)
      Coleção Processos;
      enquanto (Minerador não interrompe processo)
            Processos = minerar(A,tipo_heuristicas);
      fim_enquanto

      retorna o melhor modelo de classificação induzido da Coleção
      Processos, em forma de String;
end Mining

Coleção minerar(arquivo de dados A, inteiro tipo_heuristica)
      Selecione todas as heurísticas de acordo o tipo escolhido pelo
      minerador (variável tipo_heuristica), e armazene-as na Coleção
      Heurísticas;
      Selecione todos os processos possíveis inicialmente, e armazene-
      os na Coleção Processos;

      Remova os processos descartados pelas heurísticas (Coleção
      Heurísticas) da Coleção Processos;
      Para cada processo da Coleção Processos
            Extraia uma amostra de A;
            Fragmente a amostra;
            Para cada par A.treinamento-A.teste
                  Induza um modelo de classificação do conjunto
                  A.treinamento;
                  Compute a acurácia do modelo de classificação usando
                  o cojunto A.teste;
            fim_para

            Compute a média m e o desvio padrão dv das acurácias;
            Selecione um dos modelos de classificação e atribua a ele
            a acurácia m-dv;
            Armazene o processo na Coleção Resultado;
            Notifique o valor da acurácia ao minerador;
      fim_para
      Retorne a Coleção Resultado;
fim_FastMiningProcessExecuter
```

Figura 3.10: Algoritmo `Fast Mining Processes Executer`

A entrada do algoritmo é o arquivo de dados a ser minerado e o tipo de heurística (acurácia/tempo) que será aplicado para podar processos de MD do espaço de busca. O algoritmo pode ser interrompido em qualquer momento de sua execução, caso o minerador encontre uma acurácia que lhe agrade.

Note que o minerador tem uma participação ativa e importante na mineração do arquivo: a escolha de qual tipo de heurística usar e o acompanhamento dos resultados. Estes dois aspectos constituem o módulo de interação "framework"-minerador que apresentamos a seguir.

Interação "framework"-minerador

Entendemos como interação o tipo de comunicação pela qual o minerador pode interferir no resultado final da mineração de um arquivo de dados. Neste sentido, disponibilizamos ao minerador duas formas de interação: (i) possibilidade de escolha do tipo de heurísticas (de acurácia e/ou tempo) a ser usado e (ii) acompanhamento da execução dos processos de MD.

O minerador poderá decidir quais tipos de heurísticas aplicar, conforme sua conveniência. Assim, ele poderá fazer escolhas de acordo os recursos disponíveis (tempo e hardware). Há três cenários possíveis:

1. o minerador deseja a melhor acurácia para um modelo de classificação independentemente de tempo de execução: este cenário se aplica a uma investigação exaustiva, que usa todos os processos de MD possíveis, representados na Figura 3.11 pelo conjunto `E`. Portanto, neste cenário não são usadas heurísticas;
2. o minerador anseia por um modelo de classificação de acurácia aceitável, mas está um pouco preocupado com o tempo: neste cenário é empregada a busca de modelos de classificação apoiada em heurísticas de acurácia, que pode descartar alguns processos de MD, executando portanto os processos de MD do subconjunto `HA` que está contido no conjunto `E`;
3. o minerador quer um modelo de classificação de acurácia aceitável no tempo mais rápido possível: neste caso, são utilizadas as heurísticas de acurácia e tempo ao buscar modelos de classificação, então são aplicados os processos de MD do subconjunto $HA \cap HT$.

Note que não colocamos um cenário no qual só sejam selecionadas as heurísticas de tempo, pois se o minerador está demasiadamente preocupado com o tempo, logo ele também selecionará as heurísticas de acurácia.

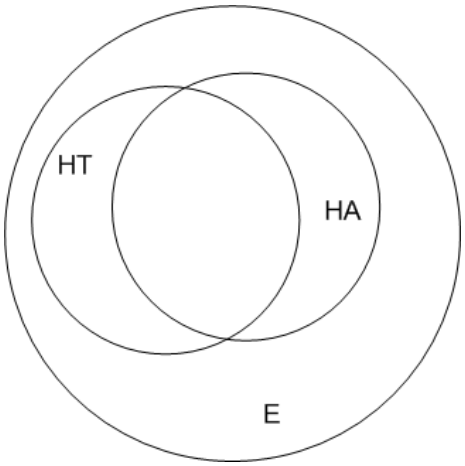


Figura 3.11: Cenários de seleção de heurísticas

A segunda forma de interação, o acompanhamento da execução de processos de MD dá-se da seguinte forma: à medida que a acurácia de cada processo de MD é computada, o valor da mesma é informada ao minerador, e, se em algum momento este valor lhe é aceitável, então ele poderá interromper o processo imediatamente, limitando assim, o tempo de execução que seria despendido ao aplicar todos os processos de MD previstos inicialmente.

Outro ponto positivo desta interação é que o minerador obtém o resultado parcial enquanto o processo está em execução, desta forma, ele tem um maior controle de como está o andamento do processo.

3.2.3 Camada de apresentação

A camada de apresentação é constituída por uma interface gráfica a fim de facilitar o uso da ferramenta por parte do minerador (requisito RNF1). Para apresentá-la, simulamos a mineração de um arquivo de dados sendo acompanhada por um minerador.

A tela inicial da ferramenta, a aba `PreProcess` — reusada de (Witten & Frank, 1999) —, pode ser vista na Figura 3.12. Nela, o minerador escolhe o arquivo de dados a ser minerado, e tem a possibilidade de ver algumas informações a respeito do mesmo como número de instâncias, número de atributos, distribuição das classes, dentre outras.

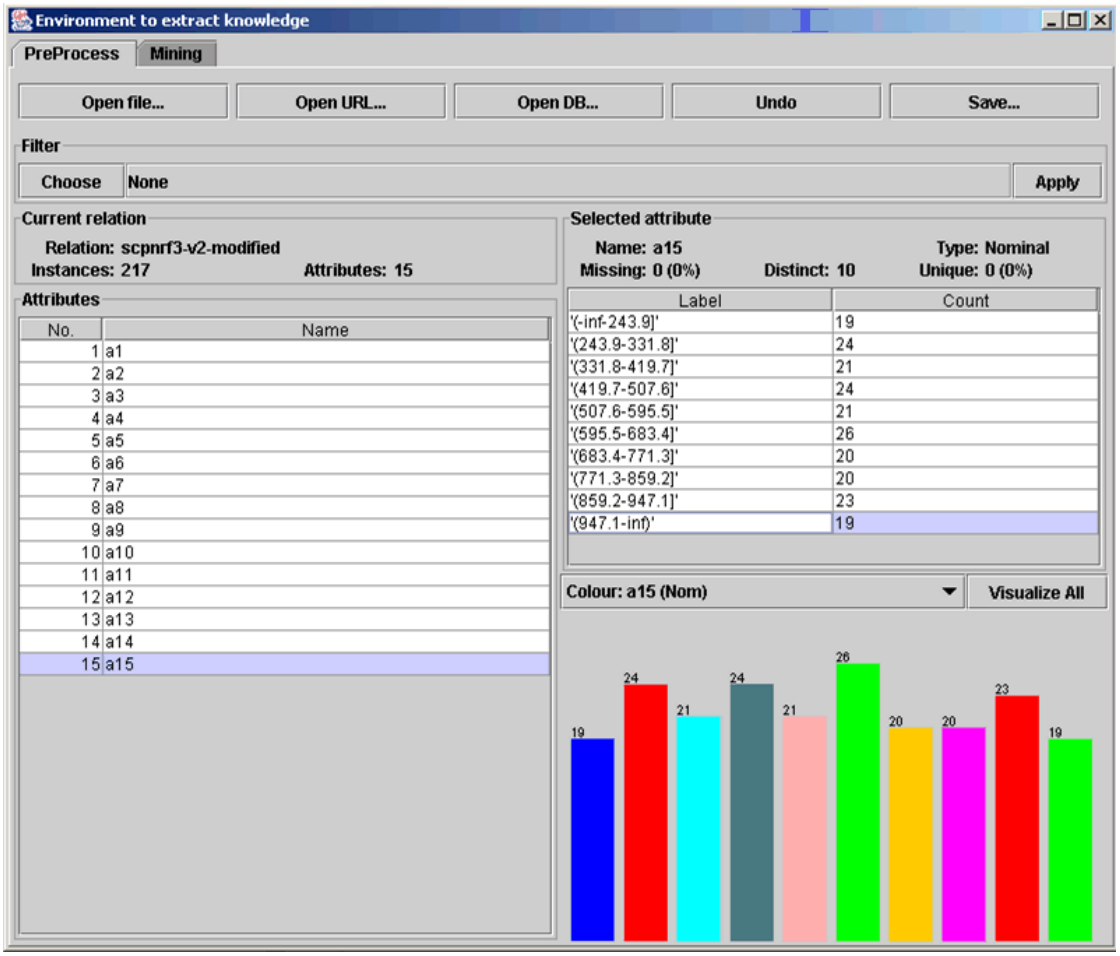


Figura 3.12: Interface gráfica para escolha do arquivo de dados

Em seguida, o minerador passa para a aba principal, denominada `Mining` (Figura 3.13), que gerencia a execução de processos de MD. Do lado esquerdo, o minerador seleciona o atributo de classificação, o algoritmo que automatiza a execução de processos de MD, as técnicas de amostragem, fragmentação e classificação, e, por fim, dá início à busca de modelos de classificação. Os resultados são exibidos do lado direito.

Ao iniciar a busca, o minerador é interrogado sobre qual tipo de heurística ele deseja empregar. No exemplo da Figura 3.14, foi optado por ambos os tipos de heurísticas.

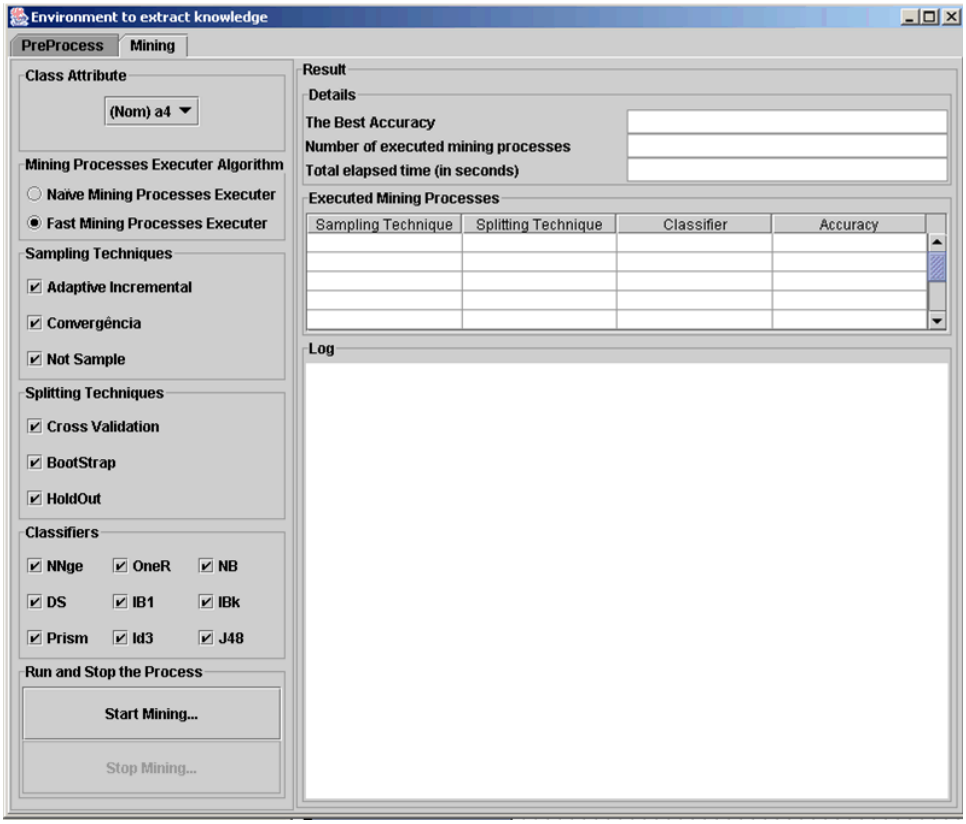


Figura 3.13: Interface gráfica para minerar o arquivo de dados selecionado

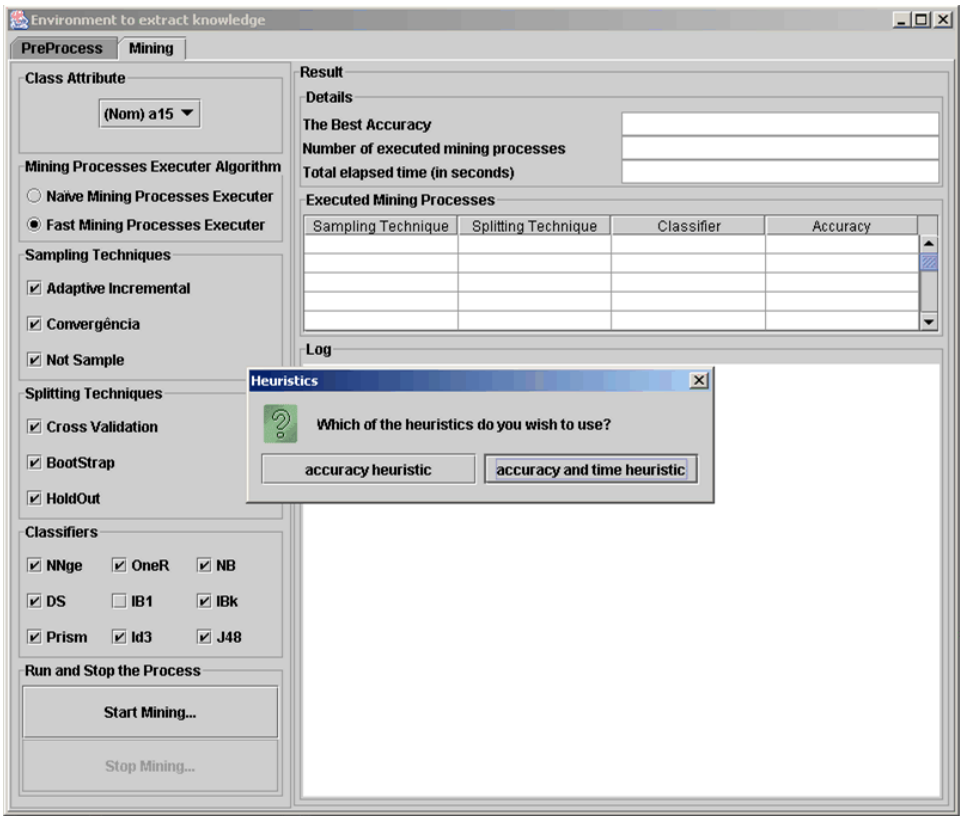


Figura 3.14: Interface gráfica para escolha do tipo de heurística a ser empregado

Consideraremos que a acurácia mínima desejada é de 95%. Logo, na Figura 3.15, observamos que as acurácias induzidas pelos quatro primeiros processos de MD não satisfaz ao minerador. Observe que eles são exibidos através de uma tabela que lista os processos já executados e suas respectivas acurácias, ordenadas de forma decrescente.

Tendo em vista as acurácias baixas induzidas até o presente momento, então, o minerador não faz nada, e o processamento continua. Mais processos são executados como é mostrado na Figura 3.16, porém ainda continua uma acurácia abaixo do esperado.

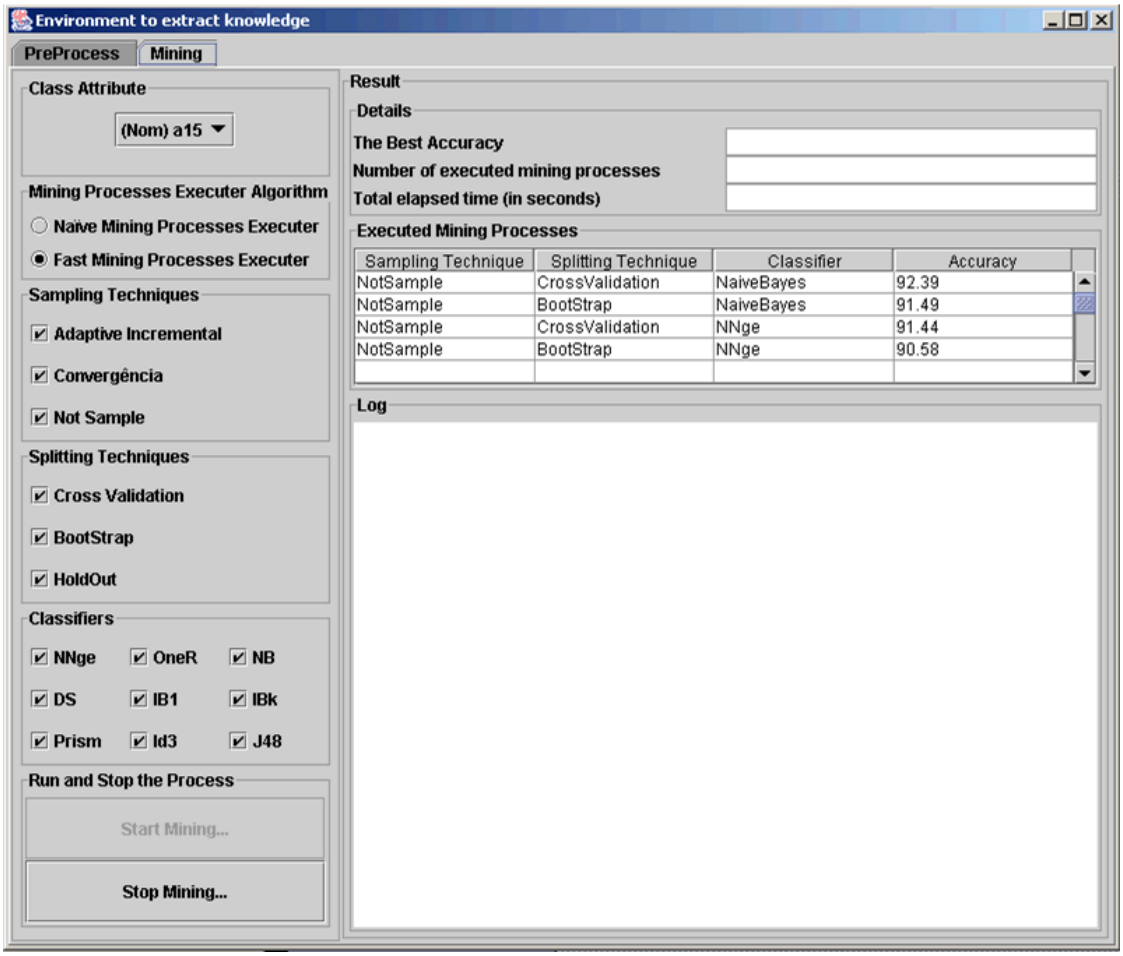


Figura 3.15: Exemplo 1 de interface gráfica com resultado parcial da mineração

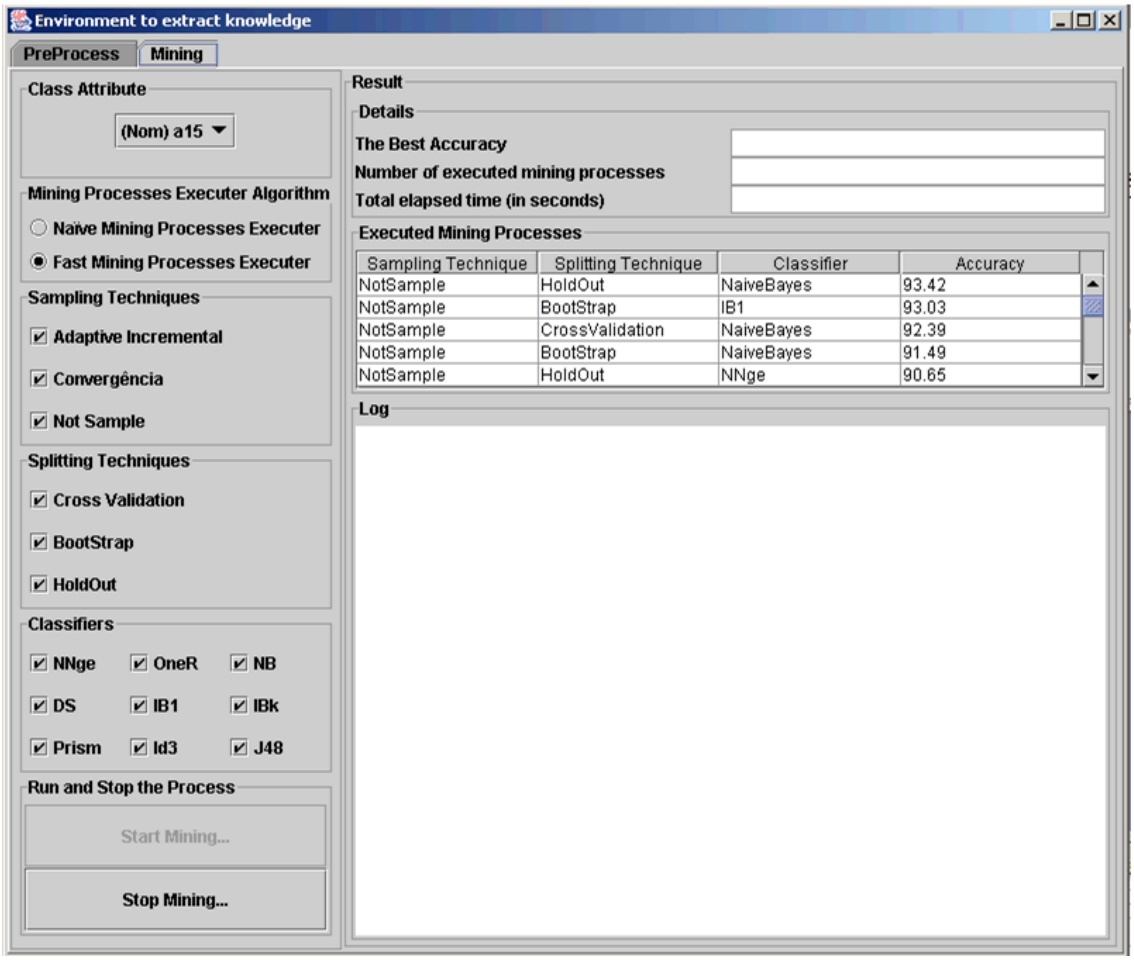


Figura 3.16: Exemplo 2 de interface gráfica com resultado parcial da mineração

Finalmente, como é visível na Figura 3.17, o processo de MD `NS-CV-J48` consegue a acurácia mínima deseja, e a execução do algoritmo é então interrompida pelo minerador. Os resultados são mostrados do lado direito da tela que gerencia a mineração dos dados. No topo, o minerador é notificado da melhor acurácia, do número de processos de MD executados e do tempo de execução total gasto, em segundos, para executar estes processos. O modelo de classificação induzido por cada processo de MD pode ser visto através de um clique de "mouse" em cima do processo desejado. O modelo de classificação com maior acurácia pode ser visualizado na Figura 3.17, enquanto que o modelo com a segunda maior acurácia, na Figura 3.18.

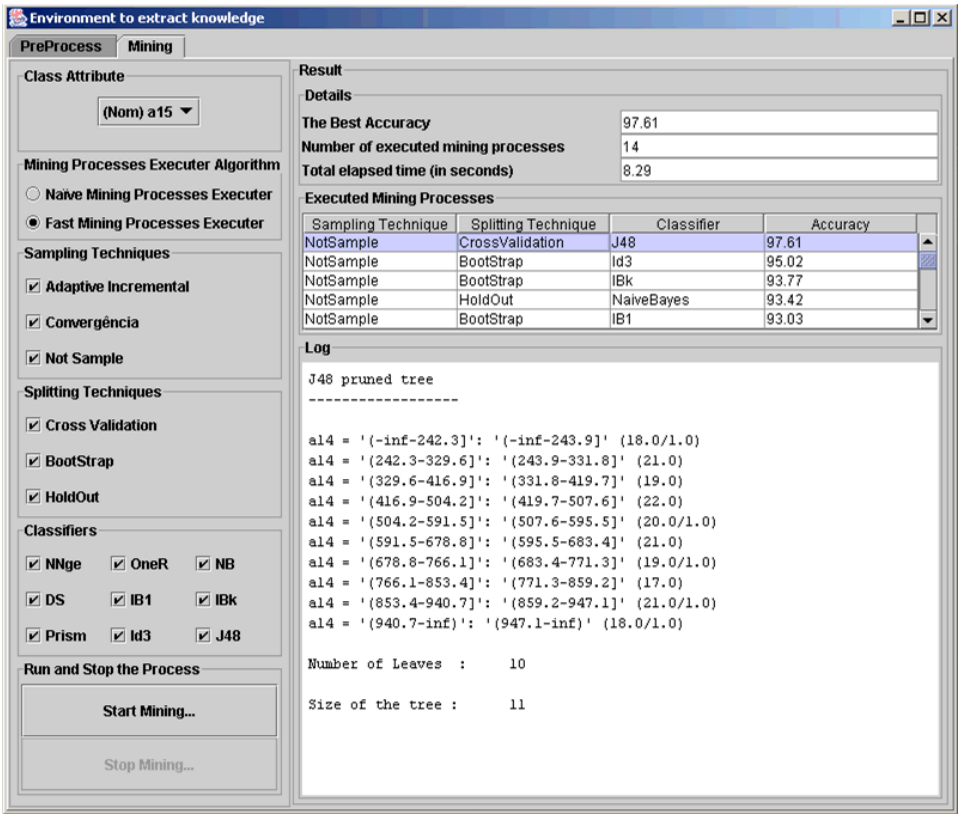


Figura 3.17: Exemplo 1 de interface gráfica com o resultado final da mineração

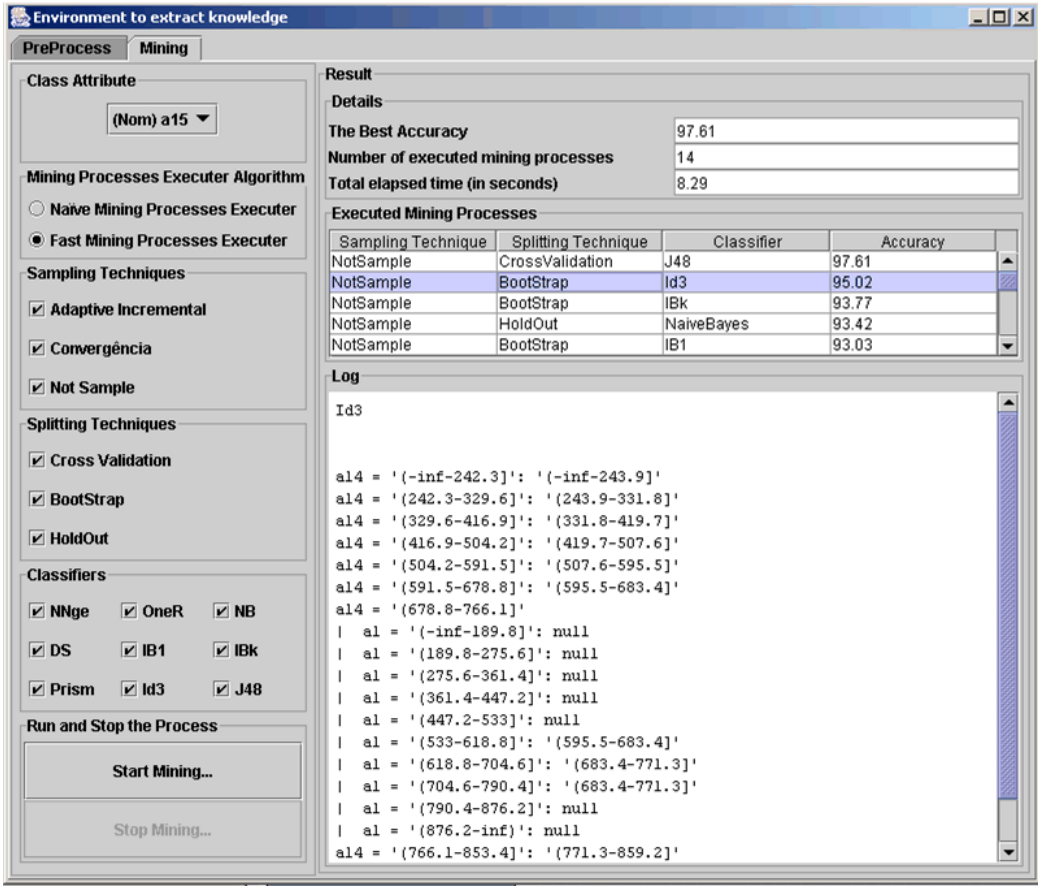


Figura 3.18: Exemplo 2 de interface gráfica com o resultado final da mineração

3.3 Considerações Finais

Neste capítulo, discutimos a confecção de uma ferramenta proposta para que um minerador obtenha um modelo de classificação com acurácia que lhe seja aceitável e de forma rápida, quando comparada ao tempo despendido por uma busca exaustiva.

Infelizmente, apesar de usar uma abordagem de "framework", a ferramenta tem uma limitação que é a seguinte: ao incorporar uma nova técnica à ferramenta seja ela de amostragem, fragmentação ou classificação, a base de heurísticas não cobre os processos concernentes a tal técnica. Em outras palavras, os processos de MD relacionados a uma nova técnica acrescentada ao "framework" sempre farão parte do espaço de busca seja do algoritmo `Mining Process Naïve Executer` ou do algoritmo `Mining Process Fast Executer`.

Capítulo 4

Avaliação Experimental

No intuito de avaliar a eficácia das heurísticas descobertas, realizamos uma avaliação comparativa dos algoritmos `Mining Processes Naïve Executer` (`Naïve`) — execução de cada processo do universo disponível de processos de MD — , e `Mining Processes Fast Executer` (`Fast`) — execução seletiva, apoiada nas heurísticas, de processos do universo. Os dois algoritmos procuram o modelo de classificação com a melhor acurácia. Diferentemente do algoritmo `Naïve`, o algoritmo `Fast` procura atender também aos requisitos de tempo de resposta do minerador. Estas expectativas relativas aos algoritmos devem ser confirmadas pela experimentação.

Mais precisamente, uma vez que as heurísticas permitem afastar do espaço de busca os 'piores' processos de MD, isto é, com desempenho ruim em relação à acurácia, então são esperados dois resultados da avaliação experimental: (i) que as melhores acurácias de ambos os algoritmos sejam equivalentes, e (ii) que a média das acurácias obtidas com `Naïve` seja menor do que a média obtida com `Fast`; todavia, note que, para os nossos propósitos, (i) tem muito mais significação do que (ii).

Para a avaliação, todos os arquivos de dados foram diferentes daqueles usados para a descoberta de heurísticas. Empregamos trinta arquivos, com muita diversidade em relação ao número de atributos, instâncias e classes; isto fica bastante claro na Tabela 4.1.

Os processos de MD são os mesmos usados para a construção da base de conhecimento, 81 ao todo (3 técnicas de amostragem x 3 técnicas de fragmentação x 9 técnicas de indução de modelos de classificação). Além disso, executamos os algoritmos em um computador com a mesma configuração usada anteriormente (microprocessador `Pentium III` 900 MHz e 512 MB de memória RAM).

Arquivo de dados	NA	NI	NC
assing500	13	4332	10
bqp2500	3	2340	10
cap	12	100	4
cnfp10	21	243	4
csp	6	401	10
dv320	3	617	10
dv80	3	5082	10
enstein	3	560	10
gcol	3	5779	10
genelist_new	4	1711	3
indtrack	7	3605	7
ipums_census	52	4941	4
jap_vowels	12	4274	9
mdmkp	37	989	10
mknapct	6	815	10
robot_ef	7	1306	3
scpnrf3-v2	15	217	10
scpnrf3	15	1083	10
scpnrh4	15	1680	10
soluzione-fifo	9	835	10
statistiche-fifo	3	910	8
statistiche	4	1404	4
steine-v2	6	6320	10
steine	11	178	10
te	30	1167	8
elnino	9	782	5
tctodd1	21	3581	17
smni_eeg	3	7363	20
census_inc	41	2585	2
wtpack7	10	2000	10

Tabela 4.1: Valores dos meta-atributos dos arquivos de dados

4.1 `Fast` "versus" `Naïve`

Devido ao fato de o algoritmo `Fast` estar apoiado em heurísticas, na avaliação é esperada uma pequena perda da acurácia, relativamente à acurácia obtida com o `Naïve`. Levando isto em conta, estabelecemos que o melhor processo de MD (isto é, aquele que induz um modelo de classificação com a maior acurácia) indicado por `Fast` não precisa apresentar necessariamente uma acurácia idêntica à acurácia obtida pelo melhor processo apontado com `Naïve`; basta que apresente uma que seja maior do que aquela induzida pelo enésimo melhor processo de MD indicado por `Naïve`. Analisando os resultados dos experimentos, que discutiremos a seguir, observamos que na grande maioria dos casos `Fast` consegue uma acurácia maior do que pelo menos a quarta melhor de `Naïve`. Tendo em vista esta análise, é importante ressaltar que a quarta

melhor acurácia indicada por Naïve, experimentalmente, não tem uma discrepância alta em relação a melhor, ou seja, tem um valor próximo da melhor. Sendo assim, definimos então que a melhores acurácias obtida com Naïve e Fast são equivalentes (ou similares) se a melhor acurácia deste é maior do que a melhor acurácia daquele.

Outro fator que contribui fortemente para que se tenha esta *margem de aceitação* é a aleatoriedade utilizada pelas técnicas de amostragem e fragmentação. Por exemplo, a técnica de fragmentação Holdout escolhe aleatoriamente as instâncias para os conjuntos de teste e treinamento; logo, para cada (execução de um) processo de MD, o par de conjuntos treinamento-teste muito provavelmente será distinto e, consequentemente, as acurácias induzidas também.

Na avaliação dos algoritmos, primeiramente, comparamos o algoritmo Naïve com o algoritmo Fast apoiado somente pelas heurísticas de acurácia (cenário 1). Em seguida, a comparação é feita com Fast usando as heurísticas de acurácia e tempo (cenário 2). Os resultados do primeiro cenário são mostrados na Tabela 4.2.

Arquivo de dados	Naïve						Fast					
	Melhor Processo	AC-DV	"Menor" AC	Média das Acs	DV da Média das ACs	Tempo de Execução	Melhor Processo	AC-DV	Média das Acs	DV da Média das Acs	Tempo de Execução	Num de Processos Executados
assing500	NS-BS-IB1	34,50%	34,11%	13,90%	10,36%	11103,64	NS-BS-IB1	34,41%	18,65%	11,27%	4321,45	42
bqp2500	Conv-BS-IBk	27,64%	24,76%	10,31%	5,25%	551,00	Conv-BS-IB1	26,44%	11,43%	5,89%	214,40	49
cap	NS-BS-IBk	45,18%	43,30%	25,80%	9,33%	36,85	NS-BS-NNge	43,96%	29,32%	10,58%	15,05	42
census_inc	AIF-HO-NNge	100,00%	100,00%	99,78%	1,45%	3005,87	NS-CV-NB	100,00%	100,00%	0,00%	1624,82	47
cmfp10	AIF-BS-NNge	62,91%	59,80%	45,94%	9,42%	113,32	Conv-BS-NNge	64,92%	48,21%	9,09%	34,81	45
csp	NS-CV-NB	62,82%	61,77%	46,74%	14,50%	95,91	Conv-BS-IBk	63,30%	55,57%	5,59%	31,92	42
dv320	AIF-HO-OneR	64,91%	62,98%	52,04%	10,48%	82,75	AIF-CV-NNge	63,41%	53,42%	8,09%	22,48	49
dv80	Conv-CV-NNge	40,14%	36,65%	30,89%	4,30%	2537,32	Conv-BS-IBk	36,71%	31,70%	3,89%	1597,83	49
elnino	NS-BS-IBk	60,29%	59,14%	41,37%	8,94%	262,00	NS-BS-IBk	60,42%	46,56%	7,25%	93,44	42
enstein	Conv-BS-IBk	32,15%	26,18%	13,60%	6,59%	100,39	Conv-BS-IB1	28,88%	14,97%	6,34%	24,29	49
gcool	Conv-CV-OneR	95,14%	93,90%	80,99%	22,54%	1442,45	Conv-HO-OneR	97,83%	88,80%	11,52%	743,22	49
genelist_new	NS-CV-NNge	76,13%	75,29%	69,00%	5,90%	369,77	AIF-BS-J48	82,75%	70,53%	5,25%	150,58	43
indtrack	AIF-BS-J48	95,73%	94,21%	85,24%	11,26%	854,73	Conv-BS-IBk	94,30%	90,22%	3,85%	593,65	42
ipums_census	AIF-CV-J48	94,17%	91,07%	86,97%	3,82%	36000,70	AIF-HO-J48	94,00%	85,71%	4,40%	10179,35	51
jap_vowels	NS-BS-IBk	60,27%	59,25%	34,30%	11,41%	8378,72	AIF-BS-IBk	60,02%	40,93%	10,24%	3729,54	42
mdmkp	AIF-BS-IB1	66,94%	64,40%	45,17%	15,19%	1097,30	AIF-BS-IBk	68,22%	52,81%	9,67%	509,29	48
mknapct	NS-BS-Prism	35,42%	34,92%	16,81%	9,47%	193,94	AIF-BS-NNge	37,67%	21,00%	10,75%	73,96	42
robot_ef	NS-BS-IBk	80,74%	79,45%	66,39%	10,73%	225,28	AIF-BS-IBk	80,65%	72,73%	7,24%	141,92	42
scpnrf3	NS-CV-J48	98,98%	98,71%	79,98%	26,08%	273,19	AIF-CV-NNge	100,00%	88,44%	19,20%	112,37	42
scpnrf3-v2	NS-CV-NNge	97,19%	96,28%	78,20%	23,93%	50,31	NS-CV-NNge	96,79%	83,23%	17,66%	20,88	42
scpnrh4	Conv-CV-Id3	98,39%	98,25%	83,80%	24,05%	518,85	Conv-BS-Id3	100,00%	90,61%	8,91%	450,99	42
smni_eeg	Conv-BS-IBk	29,62%	26,52%	12,82%	6,11%	6140,21	Conv-BS-Prism	28,30%	14,46%	6,20%	3067,42	51
soluzione-fifo	AIF-BS-IBk	51,05%	49,32%	29,77%	10,06%	250,35	AIF-BS-IBk	51,90%	36,14%	9,32%	124,94	42
statistiche-fifo	AIF-CV-Id3	46,68%	44,64%	30,75%	10,16%	94,81	AIF-BS-J48	45,39%	37,88%	6,58%	52,07	49
statistiche	AIF-CV-DS	49,97%	49,65%	42,90%	7,32%	244,58	Conv-BS-J48	53,36%	44,91%	5,03%	113,66	43
steine	AIF-BS-IB1	32,77%	31,19%	13,05%	10,57%	85,71	NS-BS-IB1	33,28%	16,93%	12,38%	23,48	42
steine-v2	AIF-BS-Prism	33,54%	33,41%	14,38%	9,72%	12358,22	NS-BS-Prism	33,52%	18,33%	11,20%	5051,66	42
tctodd1	NS-BS-IBk	69,07%	66,78%	44,53%	17,30%	7247,09	NS-BS-IBk	69,45%	49,82%	15,26%	3205,07	47
tle	NS-BS-IBk	77,30%	74,95%	41,74%	18,55%	1249,91	NS-BS-IB1	77,53%	45,94%	18,81%	565,76	53
wtpack7	AIF-BS-Prism	34,68%	33,87%	14,40%	10,20%	1390,70	AIF-BS-Prism	34,97%	18,97%	11,15%	581,80	42

Tabela 4.2: Avaliação experimental das heurísticas de acurácia

As colunas Naïve e Fast contém uma síntese dos resultados alcançados pelos referidos algoritmos. As colunas AC-DV apresentam a maior acurácia 'pessimista' obtida pelos algoritmos para o respectivo arquivo de dados indicado na coluna Arquivo de dados. Recapitulando, a acurácia 'pessimista' é o valor médio das

acurácias obtidas nas fragmentações com dez iterações menos o desvio padrão. O processo de MD que induziu a maior acurácia está indicado através das colunas Melhor Processo. As colunas Média das ACs indicam a média aritmética de todas as acurácias obtidas com Naïve e Fast, enquanto que as colunas DV da Média das ACs apresentam o desvio padrão desta média. O tempo de execução despendido pelos algoritmos para executar todos os processos de MD do espaço de busca, é informado, em segundos, nas colunas Tempo de execução.

A coluna 'Menor' AC informa a menor acurácia obtida com Naïve, dentro da margem de aceitação. Em outras palavras, é a quarta melhor acurácia de Naïve. A coluna Num de Processos Executados representa a quantidade de processos de MD que foram selecionados por Fast para execução.

Analisando a Tabela 4.2, inicialmente, consideramos o critério tempo de execução. Ficou comprovado, em todos os casos, que o algoritmo Fast executou menos processos do que os 81 processos de MD executados pelo algoritmo Naïve; consequentemente, o tempo de execução também foi, comparativamente, menor. O número de processos de MD executados por Fast variou entre 42 e 53, ou seja, aproximadamente, foram executados pouco mais da metade dos 81 processos disponíveis. Falando em termos relativos de tempo, nos casos extremos, Fast foi quatro vezes mais rápido que Naïve para o arquivo enstein, enquanto que, para o arquivo scpnrh4, foi 14% mais rápido. Em média, Fast conseguiu reduzir pela metade o tempo de execução, se comparado com Naïve.

Quanto à acurácia, verificamos que os algoritmos Fast e Naïve apresentaram acurácias similares — dentro da margem de aceitação esperada — em 100% dos arquivos de dados. Por exemplo, para o arquivo cap, as maiores acurácias obtidas por Naïve e Fast foram 45,18% e 43,96%, respectivamente. Apesar de a segunda ser menor do que a primeira, a acurácia indicada por Fast está coberta pela margem de aceitação, ou seja, é maior que a quarta melhor acurácia de Naïve — pela coluna 'Menor' Acurácia, seu valor é 43,30%. Note ainda que a diferença de valores entre a maior acurácia e a quarta maior de Naïve tem uma variação muito pequena, e isto ocorre para todos os outros arquivos de dados — em média, a diferença é de 1,65, e no caso mais extremo é de 5,97, para o arquivo enstein. Deste modo, fica evidente

que o argumento usado para definir a margem de aceitação é válido uma vez que o valor entre a melhor e quarta melhor acurácia de Naïve não é tão discrepante.

É essencial lembrar que mesmo que se tenha obtido o melhor modelo de classificação para um arquivo de dados, o critério de melhor é relativo. Em termos absolutos, o 'melhor' modelo de conhecimento pode não ser bom – acurácia inferior a 90%, por exemplo. Entretanto, o modelo de conhecimento foi o melhor ao ser comparado com os demais.

Avaliando agora as médias das acurácias obtidas por ambos os algoritmos, fica comprovado que as piores acurácias obtidas com Fast são maiores do que as piores de Naïve. Exemplificando, as médias das acurácias para o arquivo mknapct são 16,81% (Naïve) e 21% (Fast). Este acontecimento ocorre para todos os arquivos de dados, exceto para ipums_census, onde a média calculada para Naïve é 1,26 maior do que a computada para Fast. Para este caso, a média de Fast não foi maior do que Naïve porque as heurísticas de acurácia além de não ter podado muitos processos de MD com piores desempenhos, também descartou alguns com desempenho bom. Resumindo esta análise: as piores acurácias de Fast foram predominantemente melhores que Naïve em 96,7% dos 30 arquivos de dados.

É importante ressaltar ainda que, para ambos os algoritmos, os melhores processos de MD obtidos foram iguais para seis arquivos de dados: assign500, elnino, scpnrf3-v2, soluzione-fifo, tctodd1 e wtpack7. Todavia, mesmo os processos de MD sendo iguais, a acurácia induzida por eles foi diferente. A explicação para esta ocorrência está na aleatoriedade das técnicas de amostragem e fragmentação, discutida no início desta seção.

Também devido a este motivo, explicamos o fato de que, para alguns casos, o melhor processo de MD indicado pelo algoritmo Fast é diferente do melhor indicado pelo algoritmo Naïve, e, mesmo assim, a acurácia deste ser menor do que a acurácia daquele. Em outras palavras, a melhor acurácia obtida por Fast superou até mesmo a melhor acurácia obtida com Naïve. Esta situação pode ser observada em onze arquivos de dados: cnfp10, csp, gcol, genenlist_new, mdmkp, mknapct, scpnrf3, scpnrh4, statistiche, steine e te.

Por fim, avaliando acurácia e tempo de execução simultaneamente, corroborou-se que, para este cenário, em 100% dos arquivos, Fast atingiu sua finalidade principal de obter acurácia similar a Naïve — dentro da margem de aceitação —, com um

tempo de execução comparativamente reduzido. O objetivo secundário também foi cumprido em quase a totalidade dos casos: a média das acurácias obtidas com `Naïve` foi menor do que a média obtida com `Fast` em 96,7% dos arquivos de dados.

No experimento seguinte — cenário 2 —, comparamos de novo os dois algoritmos, agora com `Fast` apoiado nos dois tipos de heurísticas — acurácia e tempo. Os resultados estão na Tabela 4.3. Note que as tabelas (Tabela 4.2 e Tabela 4.3) são estruturalmente iguais, porém os conteúdos para o algoritmo `Fast` diferem, visto que as configurações mudaram.

Arquivo de dados	Naïve						Fast					
	Melhor Processo	AC-DV	"Menor" AC	Média das Acs	DV da Média das ACs	Tempo de Execução	Melhor Processo	AC-DV	Média das Acs	DV da Média das Acs	Tempo de Execução	Num de Processos Executados
assing500	NS-BS-IB1	34,50%	34,11%	13,90%	10,36%	11103,64	Conv-BS-IB1	35,62%	20,02%	11,94%	2187,46	22
bqp2500	Conv-BS-IBk	27,64%	24,76%	10,31%	5,25%	551,00	NS-BS-NNge	14,14%	10,28%	2,16%	207,40	18
cap	NS-BS-IBk	45,18%	43,30%	25,80%	9,33%	36,85	NS-BS-IB1	43,45%	29,68%	8,86%	1,78	16
census_inc	AIF-HO-NNge	100,00%	100,00%	99,78%	1,45%	3005,87	AIF-HO-Id3	100,00%	100,00%	0,00%	147,28	25
cmfp10	AIF-BS-NNge	62,91%	59,80%	45,94%	9,42%	113,32	NS-BS-NNge	63,68%	46,92%	7,97%	10,68	17
csp	NS-CV-NB	62,82%	61,77%	46,74%	14,50%	95,91	NS-CV-J48	64,27%	56,75%	6,73%	7,95	16
dv320	AIF-HO-OneR	64,91%	62,98%	52,04%	10,48%	82,75	NS-CV-NNge	63,58%	55,68%	6,04%	17,45	18
dv60	Conv-CV-NNge	40,14%	36,65%	30,89%	4,30%	2537,32	Conv-BS-Id3	39,85%	32,11%	4,86%	495,06	27
elnino	NS-BS-IBk	60,29%	59,14%	41,37%	8,94%	262,00	NS-BS-IBk	60,87%	47,40%	5,83%	38,77	16
enstein	Conv-BS-IBk	32,15%	26,18%	13,60%	6,59%	100,39	NS-BS-IBk	22,49%	14,94%	4,12%	14,54	18
gcol	Conv-CV-OneR	95,14%	93,90%	80,99%	22,54%	1442,45	AIF-HO-J48	100,00%	82,38%	16,57%	27,69	27
genelist_new	NS-CV-NNge	76,13%	75,29%	69,00%	5,90%	369,77	NS-CV-NNge	75,65%	72,58%	3,51%	72,86	16
indtrack	AIF-BS-J48	95,73%	94,21%	85,24%	11,26%	854,73	AIF-CV-J48	99,13%	87,32%	4,56%	15,53	22
ipums_census	AIF-CV-J48	94,17%	91,07%	86,97%	3,82%	36000,70	AIF-CV-NNge	98,94%	87,10%	5,04%	5967,54	29
jap_vowels	NS-BS-IBk	60,27%	59,25%	34,30%	11,41%	8378,72	AIF-BS-IB1	58,94%	37,94%	9,67%	2050,64	22
mdmkp	AIF-BS-IB1	66,94%	64,40%	45,17%	15,19%	1097,30	NS-BS-IBk	66,56%	56,23%	8,89%	273,06	19
mknapct	NS-BS-Prism	35,42%	34,92%	16,81%	9,47%	193,94	NS-BS-NNge	35,53%	20,89%	10,91%	56,12	16
robot_ef	NS-BS-IBk	80,74%	79,45%	66,39%	10,73%	225,28	NS-BS-IBk	81,37%	75,90%	2,84%	71,57	16
scpnrf3	NS-CV-J48	98,98%	98,71%	79,98%	26,08%	273,19	NS-CV-NNge	99,05%	96,41%	5,39%	104,69	16
scpnrf3-v2	NS-CV-NNge	97,19%	96,28%	78,20%	23,93%	50,31	NS-HO-J48	96,71%	93,01%	2,90%	5,78	16
scpnrh4	Conv-CV-Id3	98,39%	98,25%	83,80%	24,05%	518,85	NS-CV-NNge	98,37%	97,18%	1,04%	436,57	16
smni_eeg	Conv-BS-IBk	29,62%	26,52%	12,82%	6,11%	6140,21	Conv-BS-Prism	27,58%	14,62%	7,34%	1478,97	28
soluzione-fifo	AIF-BS-IBk	51,05%	49,32%	29,77%	10,06%	250,35	NS-BS-IBk	50,57%	39,33%	6,16%	48,76	16
statistiche-fifo	AIF-CV-Id3	46,68%	44,64%	30,75%	10,16%	94,81	NS-CV-NNge	45,70%	39,05%	6,34%	30,84	18
statistiche	AIF-CV-DS	49,97%	49,65%	42,90%	7,32%	244,58	NS-CV-J48	49,65%	44,95%	4,75%	48,02	16
steine	AIF-BS-IB1	32,77%	31,19%	13,05%	10,57%	85,71	NS-BS-Prism	33,12%	17,83%	11,69%	3,78	16
steine-v2	AIF-BS-Prism	33,54%	33,41%	14,38%	9,72%	12358,22	AIF-BS-Prism	33,50%	18,60%	11,32%	2498,37	22
tctodd1	NS-BS-IBk	69,07%	66,78%	44,53%	17,30%	7247,09	AIF-BS-Id3	66,90%	47,20%	14,29%	2018,01	25
te	NS-BS-IBk	77,30%	74,95%	41,74%	18,55%	1249,91	NS-BS-IBk	76,80%	51,98%	16,98%	331,22	20
wtpack7	AIF-BS-Prism	34,68%	33,87%	14,40%	10,20%	1390,70	NS-BS-Prism	34,61%	18,89%	11,33%	379,88	16

Tabela 4.3: Avaliação experimental das heurísticas de acurácia e tempo

Como era esperado, o emprego do algoritmo `Fast` apoiado nas heurísticas de acurácia e tempo diminuiu bastante o tempo de execução — foi, em média, dez vezes mais rápido — em relação ao algoritmo `Naïve`. No caso mais extremo, `Fast` foi, aproximadamente, cinqüenta vezes mais rápido que `Naïve` (arquivo `indtrack`), enquanto que no outro extremo, foi 16% mais rápido (arquivo `scpnrh4`). Observe que a quantidade de processos selecionados por `Fast` foi razoavelmente restringida. Dos 81 processos de MD, não foram executados mais do que 30, e, em média, foram selecionados apenas 19 deles por arquivos de dados.

Avaliando as melhores acurácias, constatamos que, em 27 dos 30 arquivos, os melhores modelos de classificação induzidos com a utilização de `Fast` têm acurácias

dentro da margem de aceitação. O motivo pelo qual Fast não alcançou acurácias similares a Naïve para dois dos três arquivos restantes (arquivos bqp2500 e enstein) está no uso da heurística H82. Relembrando, esta heurística rejeita o uso de amostras para arquivos pequenos (número de instâncias menor ou igual que 2567), isto é, descarta do espaço de busca as técnicas de amostragem Conv e AIF. Foram justamente os processos que continham na sua composição uma das técnicas descartadas, Conv, que levaram a indução de acurácias melhores que os processos executados por Fast, compostos por a técnica de amostragem NotSample. Para os arquivos bqp2500 e enstein, a diferença entre a maior acurácia de Fast e a quarta maior de Naïve foi de fato eminente. Para o outro arquivo de dados, jap_vowels, a acurácia ficou muito próxima: enquanto Fast conseguiu uma acurácia de 58,94%, Naïve obteve 59,25% para a quarta melhor acurácia. Neste arquivo aconteceu o contrário dos outros dois, não foi usado amostras (heurística H81), e isto implicou em uma perda da acurácia.

Quanto às médias das acurácias, em 29 dos 30 arquivos de dados, as médias calculadas para Fast foram superiores do que as de Naïve, evidenciando desta forma, que as piores acurácias deste algoritmo são maiores do que as daquele. Naïve alcançou uma média maior que Fast somente para o arquivo bqp2500, e, mesmo assim, a diferença entre as médias foi 0,03, ou seja, bastante pequena. É bastante compreensível que a média de Fast seja menor que Naïve já que a melhor acurácia obtida por este algoritmo é muito maior do que a melhor obtida por aquele.

Devido a aleatoriedade das técnicas de amostragem e fragmentação, neste experimento também houve (cinco) arquivos de dados em que os melhores processos de MD indicados por ambos os algoritmos foram iguais, mas com acurácias distintas. São eles: elnino, genelist_new, robot_ef, steine-v2 e te.

Em outros arquivos de dados, mesmo os melhores processos de MD apontados por Fast sendo diferentes de Naïve, ainda assim, as acurácias obtidas pelo primeiro foram maiores que o segundo. Isto ocorre para 9 dos 30 arquivos de dados: assign500, cnfp10, csp, gcol, indtrack, ipums_census, mknapct, scpnrf3 e steine.

Finalmente, considerando ambos os critérios, acurácia e tempo de execução, fica confirmado que, em 90% dos arquivos, Fast cumpriu com a meta principal: alcançar uma acurácia similar a obtida por Naïve — dentro da margem de aceitação —,

despendendo um tempo de execução comparativamente reduzido. A meta secundária foi atendida em 96,7% dos 30 arquivos de dados, isto é, a média das acurácias obtidas com `Fast` foi maior do que a média obtida com `Naïve` em 29 arquivos de dados.

4.2 Considerações Finais

Sintetizando os resultados dos experimentos, podemos afirmar que a meta principal do nosso trabalho foi cumprida. Para todos os arquivos de dados minerados, nos dois cenários criados — cenários 1 e 2 —, conseguimos, com as heurísticas, desempenhos relativos à acurácia dos modelos de classificação induzidos tão bons quanto os obtidos sem o auxílio das heurísticas. É importante também salientar que as heurísticas também propiciaram um considerável ganho de tempo para a indução dos modelos de classificação, na comparação com a indução dos modelos sem as heurísticas.

Mais precisamente, com o uso apenas das heurísticas de acurácia, em 100% dos arquivos experimentados, o algoritmo `Fast` conseguiu (i) acurácias próximas as obtidas por `Naïve` — dentro da margem de aceitação —, e (ii) reduzir pela metade, em média, o tempo de execução gasto por `Naïve` ao executar os processos do espaço de busca. Já com o apoio dos dois tipos de heurísticas, acurácia e tempo, `Fast` despendeu um tempo ainda menor do que `Naïve` — em média, dez vezes mais rápido —, porém perdeu um pouco em relação à qualidade da acurácia: em 90% dos 30 arquivos de dados, as melhores acurácias estavam dentro da margem de aceitação.

Ressaltamos que não utilizamos da interação de acompanhar a execução dos processos de MD, o que poderia deixá-la ainda mais rápida. Reiterando, nosso objetivo foi validar experimentalmente as heurísticas.

Ainda avaliamos as piores acurácias obtidas por ambos os algoritmos, `Naïve` e `Fast`, através do cálculo da média de todas as acurácias obtidas. A média computada para `Fast` foi predominantemente maior do que a calculada para `Naïve` em ambos os cenários.

Capítulo 5

Trabalhos Relacionados

Neste capítulo, analisamos cinco trabalhos relacionados com o problema da instabilidade do processo de mineração de dados, terminando por compará-los com o nosso.

São apresentados 5 trabalhos, em 5 seções respectivas. Na Seção 5.1, é apresentada a ferramenta WEKA (Witten & Frank, 1999), a qual não trata o problema da instabilidade, porém foi fundamental na confecção do "framework", por isso abordamo-la. "Zooming" Ranking (Brazdil *et al*, 2002) — Seção 5.2 — tenta resolver o problema da instabilidade focando unicamente em algoritmos classificadores. MiningMart (Morik & Scholz, 2004) — Seção 5.3 —, ADI (Bernstein *et al*, 2002) — Seção 5.4 —, e o ambiente para mineração de dados desenvolvido por (Toebe, 2002) — Seção 5.5 —, consideram várias etapas do processo de MD ao buscar resolver o problema da instabilidade. Finalmente, na Seção 5.6, fazemos um resumo comparativo entre nossa abordagem e os trabalhos descritos nas seções anteriores.

5.1 WEKA (Witten & Frank, 1999)

A biblioteca WEKA (Witten & Frank, 1999) agrega uma variedade de (42) técnicas de pré-processamento, (2 de) amostragem, (2 de) fragmentação e classificação (supervisionada) — ao todo são 38 classificadores disponíveis. Claramente, suas principais contribuições são as técnicas de pré-processamento e classificação. Recapitulando, as técnicas de pré-processamento são focadas na etapa de preparação dos dados, que não está no escopo desta dissertação.

São disponibilizados também algoritmos para outros tipos de classificação: não-supervisionada (técnicas de "clustering") e de regras de associação — generalização da classificação supervisionada, onde o modelo de conhecimento pode conter mais de um

atributo de classificação, o(s) qual(is) é(são) definido(s) dinamicamente de acordo critérios do próprio algoritmo.

Por ser desenvolvida usando a abordagem de "framework", a biblioteca permite o fácil desenvolvimento de aplicações reusando seu código. Dos 38 classificadores disponíveis, reusamos 9 deles, além de uma das interfaces gráficas de sua ferramenta. Ainda devido ao fato de ser um "framework", novas técnicas podem ser adicionadas à biblioteca, sem muito esforço. Por exemplo, como mostramos no Capítulo 3, para adicionar um novo classificador, basta apenas estender a classe `ClassificationTechniqueAbstract`, ou ainda implementar a interface `ClassificationTechnique`.

5.2 "Zooming" Ranking (Brazdil *et al*, 2002)

Em (Brazdil *et al*, 2002), é proposto um ranking (ordenação) de execução dos classificadores disponíveis para minerar um certo arquivo de dados. O ranking é baseado no desempenho destes classificadores em arquivos de dados similares anteriormente minerados. Mais precisamente, assume-se que o desempenho de um algoritmo classificador, quando aplicado a arquivos de dados semelhantes, produz resultado também semelhante.

A ferramenta é processada em duas etapas distintas: "zooming" e ranking.

Na etapa "zooming", são selecionados os arquivos de dados mais semelhantes ao novo arquivo de dados a ser minerado. A similaridade de um arquivo de dados processado em relação ao novo arquivo de dados de entrada é mensurada através do conjunto de meta-atributos extraídos.

Após a seleção dos arquivos de dados mais similares ao novo arquivo, as informações sobre o desempenho dos classificadores para estes arquivos, são utilizadas para gerar uma recomendação em forma de ranking — etapa ranking —, indicando a ordem em que deve ser executado cada classificador — o classificador com melhor desempenho é executado primeiro, enquanto que, o com pior desempenho, por último.

Ao confeccionar o ranking, a ferramenta permite que o minerador escolha o critério a ser utilizado: acurácia e/ou tempo. Desta forma, o ranking pode ser construído baseado no desempenho passado dos classificadores quanto à acurácia, ou tempo de execução, ou considerando ainda ambos os critérios. Por fim, o minerador ainda

seleciona os classificadores (primeiros do ranking) que deseja a serem executados para, então, induzir o(s) modelo(s) de classificação.

Para verificar é eficácia do ranking recomendado, é necessário compara-lo com o ranking real, isto é, o ranking como ele realmente foi executado. Por exemplo, na Figura 5.1, dado um arquivo de dados, a ferramenta recomendou que os classificadores disponíveis X, Y e Z fossem executados exatamente nesta ordem segundo o desempenho em relação à acurácia nos arquivo de dados similares selecionados. Ao término da execução, viu-se que o ranking recomendado errou em duas posições: o classificador X, recomendado como primeiro, ficou na segunda colocação, e o classificador Y, recomendado como segundo, ficou na primeira colocação. Sendo assim, o ranking ideal foi Y, X e Z. À medida que o ranking recomendado se assemelha ao ranking ideal, melhor aquele é avaliado.

Classificadores Disponíveis	Ranking Recomendado	Acurácia após a Execução	Ranking Ideal
X	1	85%	2
Y	2	90%	1
Z	3	78%	3

Figura 5.1: Ranking Recomendado "versus" Ranking Ideal

Na avaliação experimental, os autores construíram o ranking recomendado, sem usar a etapa "zooming" — considerando todos os arquivos já processados — e usando-a, selecionando diferentes números de arquivos similares (dois e quatro). Desta forma, o ranking ideal foi comparado com três diferentes rankings recomendados. Analisando os resultados, foi validado que o uso de "zooming" apresenta melhor resultado quando comparado a não aplicação desta técnica. No entanto, os resultados mostraram que o ranking recomendado ainda fica um pouco aquém do esperado uma vez que há pouca correlação entre o mesmo e o ranking ideal, necessitando, portanto, de um aperfeiçoamento.

Além disto, esta abordagem foca-se apenas na etapa de classificação do processo de MD, sem levar em conta a importância e necessidade das etapas de amostragem e fragmentação. Como já discutimos, é comprovado que o melhor classificador para um arquivo de dados também depende das técnicas de amostragem e fragmentação utilizadas.

5.3 MiningMart (Morik & Scholz, 2004)

MiningMart (Morik & Scholz, 2004) é baseado na idéia de que é mais fácil resolver um caso — definição de quais processos de MD devem ser aplicados aos dados a serem minerados —, se a solução de um caso similar é conhecido; em outras palavras, esta abordagem também é baseada em casos. MiningMart oferece um ambiente para desenvolver, documentar e compartilhar casos que vão desde a preparação dos dados 'brutos' até o resultado final da mineração de dados. Os casos são modelados segundo o meta-modelo MiningMart (M4), e podem ser publicados na Web para ser obtidos via Internet e modificados por outros mineradores para suas próprias aplicações.

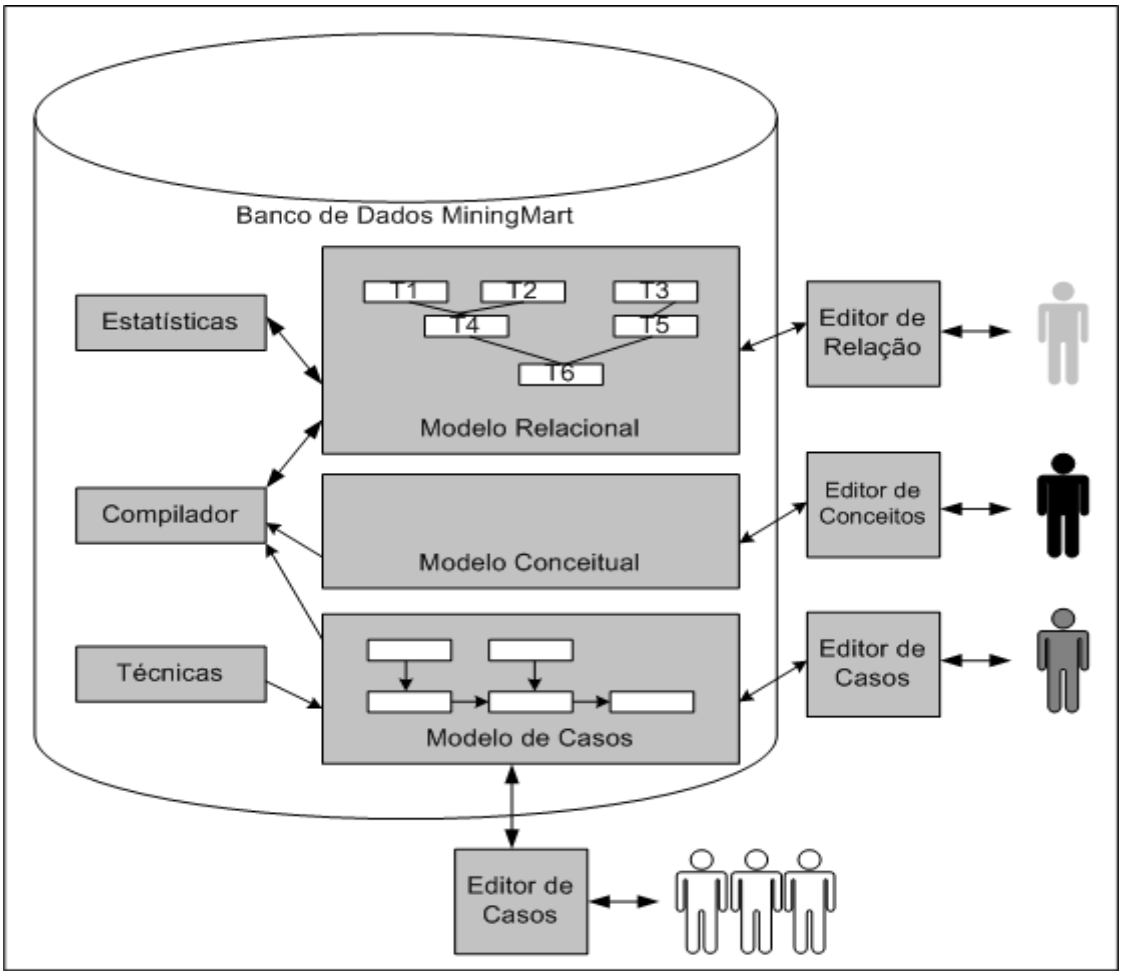


Figura 5.2: Arquitetura do meta-modelo MiningMart (Morik & Scholz, 2004)

O meta-modelo MiningMart (M4) é composto por três modelos: modelo de conceitos, modelo relacional e modelo de casos (Figura 5.2). Primeiramente, um especialista no domínio define conceitualmente, através do `Editor de`

Conceitos, os dados a minerar segundo um modelo de entidades (conceitos) e relacionamentos. Exemplos de conceitos são Cliente e Produto, e um possível relacionamento entre eles é Compra. Todo este formalismo é armazenado em ontologias.

Em seguida, o modelo conceitual dos dados a minerar é mapeado para relações de um banco de dados relacional. A interface gráfica usada para tal atividade é o Editor de Relação, que também disponibiliza algumas estatísticas. Por exemplo, para cada relação do banco de dados, são informados o número de tuplas e a quantidade de atributos categóricos e numéricos.

Dado o modelo relacional para o arquivo de dados, o passo seguinte é selecionar os processos de MD que o minerará. Através do Editor de Casos, o minerador define quais técnicas comporão tais processos (modelo de casos). Por exemplo, a técnica de amostragem Not Sample, juntamente com a técnica de fragmentação Cross Validation e o classificador Prism, formam o processo NS-CV-Prism. O ambiente checa se os processos de MD construídos são válidos. Para ilustrar, considere um arquivo de dados a minerar onde seus atributos são numéricos. O processo NS-CV-Prism é inválido para minerar este arquivo visto que o classificador Prism trata apenas atributos categóricos. Este processo passaria a ser válido se uma técnica de discretização fosse aplicada antes, para transformar os atributos numéricos em discretos (ou categóricos). Logo, note que técnicas de pré-processamento também fazem parte do processo de MD no contexto da abordagem MiningMart.

Enfim, a função do componente Compilador é aplicar os processos de MD construídos aos dados.

Uma vez minerados os dados com sucesso, isto é, com bons resultados, a próxima etapa é compartilhar, através de publicação na Web, o caso descrito em meta-nível (M4). O objetivo esperado é que o mesmo sirva como embasamento para outros casos similares. Mais especificamente, o minerador tem que recuperar o caso da Web, e redefinir o modelo de conceitos e, consequentemente, o modelo relacional, para então reaproveitar os processos de MD para seu novo arquivo de dados.

Inicialmente, os autores oferecem quatro meta-modelos base para servir de referência. Infelizmente, eles não reportam nada sobre a validação destes meta-modelos base e do seu próprio ambiente. Além disso, não é definido 'sucesso', que é um conceito subjetivo: o que pode ser bom para um minerador, pode não ser para outro.

Uma desvantagem de MiningMart é que o próprio minerador é o responsável por selecionar o caso que mais se assemelha ao seu novo caso. Outro ponto negativo é que na definição de um novo caso não é oferecido ao minerador recurso algum (heurísticas, por exemplo) na etapa de composição (seleção) dos processos de MD.

5.4 ADI (Bernstein & Provost, 2001)

Em (Bernstein & Provost, 2001), os autores introduzem o conceito de `Assistente de Descoberta Inteligente` (ADI), que oferece ao minerador (i) uma enumeração sistemática de processos de MD *válidos* — técnicas de pré-processamento também fazem parte dos processos —, e (ii) ordenação eficaz destes processos — através de heurísticas de acurácia ou de tempo —, no intuito de facilitar a escolha de quais dos processos de MD devem ser executados, para um determinado arquivo de dados a minerar.

Assim como a abordagem anterior, este trabalho também usa o conceito de processos *válidos*. Um processo de MD válido não viola nenhuma restrição das técnicas que o compõe. Por exemplo, aplicar diretamente o classificador `Naïve Bayes` a um arquivo de dados que contém atributos numéricos, não resultará em um processo válido porque tal classificador manipula apenas atributos com valores discretos. Entretanto, se uma técnica de pré-processamento discretiza os dados e, em seguida, o conjunto de técnicas `NS-CV-NB` é usado para induzir o conhecimento, então este será um processo de MD válido.

ADIs usa ontologias ao compor os processos de MD válidos. Elas definem as técnicas usadas em um processo, e suas propriedades. Por exemplo, três propriedades da técnica de discretização `Discretize` são (i) pré-condição: estabelece que os dados de entrada sejam numéricos, (ii) efeito esperado: a saída do algoritmo são dados discretizados (ou categóricos), e (iii) indicadores heurísticos: aqui são relatadas informações a respeito do desempenho da técnica em relação à acurácia (heurística de acurácia) e ao tempo de execução (heurística de tempo).

Infelizmente, detalhes sobre como as heurísticas foram descobertas, ou como são usadas para a escolha de processos de MD, são vagamente reportados. Sobre a descoberta das heurísticas, apenas é dito que foram elaboradas subjetivamente, a partir de leitura da literatura e experiência dos autores com as diferentes técnicas.

No contexto da abordagem ADI, as heurísticas de acurácia (tempo) são usadas para ordenar a execução de processos de MD segundo o desempenho de cada técnica em relação ao critério acurácia (tempo de execução). Por exemplo, é designada uma acurácia base para todos os classificadores disponíveis, ou seja, cada algoritmo de classificação tem uma acurácia "default". Assim, os classificadores podem ser ordenados de acordo o valor de sua respectiva acurácia base.

Todas as etapas seguidas por um ADI para minerar um arquivo de dados, estão ilustradas na Figura 5.3.

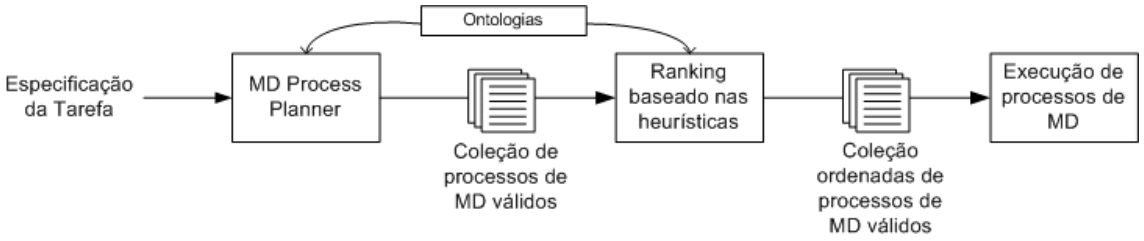


Figura 5.3: Etapas seguidas por um ADI (Bernstein & Provost, 2001)

Na especificação da tarefa, o minerador determina qual arquivo de dados será minerado e qual critério priorizar: acurácia ou tempo de execução. Uma vez tendo sido definida a tarefa, a etapa seguinte é feita pelo componente `MD Process Planner`, que, baseado nas ontologias, busca por um conjunto de processos de MD válidos. Em seguida, os processos de MD válidos são ordenados através das heurísticas de acurácia ou tempo descritas nas ontologias — lembre que é o minerador quem escolhe entre o critério acurácia ou tempo na especificação da tarefa. Depois de ordenados os processos, é permitido ainda que o minerador selecione quais processos deverão ser executados. Por fim, os processos assinalados são enfim executados.

Os autores construíram um protótipo para demonstrar que um ADI pode de fato oferecer enumerações úteis e ordenações eficazes de processos de MD. No entanto, os testes com o protótipo evidenciaram que apenas a ordenação por tempo de execução apresentou bons resultados, ao contrário da ordenação por acurácia, que só demonstrou poucos casos de sucesso (ordenações eficazes).

Finalmente, dado que ADI utiliza ontologia para assistir ao minerador na composição dos processos de MD válidos, não está claro como uma ontologia pode ser adquirida e mantida. Por exemplo, como já foi citado, uma ontologia determina uma acurácia base para cada classificador: esta assertiva não é puramente verdadeira, como

nossos experimentos demonstraram (veja, por exemplo, os resultados de nossa avaliação experimental discutidos no Capítulo 4). A acurácia de um classificador varia bastante de um arquivo de dados para outro — problema da instabilidade. Muito provavelmente, este foi o fator que levou ao desempenho ruim quanto as ordenações de processos de MD usando heurísticas de acurácia.

5.5 "Framework" para mineração de dados (Toebe, 2002)

Em (Toebe, 2002), é proposto um "framework" que tem como objetivo central a automatização da execução de processos de MD levando em conta várias técnicas de amostragem, fragmentação, assim como diversos classificadores existentes. O *"framework"* usa dois algoritmos de automação de processos de MD: `Naïve` e `Expert`.

O algoritmo `Naïve`, apresentado no Capítulo 3, faz uma busca exaustiva do melhor modelo de classificação. Já discutimos que tal algoritmo é eficaz na indicação, porém implica em um alto custo de processamento. Então, a partir da análise empírica dos resultados de experimentos do algoritmo `Naïve`, foram descobertas três heurísticas que resultaram na elaboração do algoritmo `Expert`. O objetivo deste algoritmo é apontar um processo de MD que induza uma acurácia equivalente à melhor de `Naïve`, mas com um tempo comparativamente reduzido.

A fim de avaliar o desempenho de ambos os algoritmos, usamo-los para minerar os 30 arquivos de dados descritos em nossa avaliação experimental (Capítulo 4) com os mesmos 81 processos de MD também apresentados. Os resultados da avaliação experimental (Tabela 5.1) mostraram que o algoritmo `Expert` conseguiu um tempo menor ao despendido por `Naïve`. Entretanto, para treze arquivos de dados (`assign`, bqp2500, `elnino`, `enstein`, `gcol`, `indtrack`, `japanese`, `mknapct`, `scpnrh4`, `smni`, `statistiche`, `statistiche-fifo`, `steine-v2`), o desempenho da acurácia fica a desejar, ou seja, a maior acurácia obtida por `Expert` foi eminentemente discrepante em relação à maior acurácia obtida por `Naïve`, mesmo considerando a margem de aceitação que propomos anteriormente — a melhor acurácia de `Expert` (`Fast`) deve ser maior do que pelo menos a quarta melhor de `Naïve`. Logo, nota-se uma deficiência nestas heurísticas.

Arquivo de dados	Naïve						Fast					
	Melhor Processo	AC-DV	"Menor" AC	Média das Acs	DV da Média das ACs	Tempo de Execução	Melhor Processo	AC-DV	Média das Acs	DV da Média das Acs	Tempo de Execução	Num de Processos Executados
assing500	NS-BS-IB1	34,50%	34,11%	13,90%	10,36%	11103,64	Conv-BS-NB	31,55%	20,69%	10,34%	33,00	15
bcp2500	Conv-BS-IBk	27,64%	24,76%	10,31%	5,25%	551,00	Conv-CV-DS	11,80%	6,55%	3,83%	10,00	15
cap	NS-BS-IBk	45,18%	43,30%	25,80%	9,33%	36,85	AIF-BS-IB1	43,51%	31,29%	10,20%	1,00	15
census_inc	AIF-HO-NNge	100,00%	100,00%	99,78%	1,45%	3005,87	NS-HO-Prism	100,00%	100,00%	0,00%	413,00	15
cmfp10	AIF-BS-NNge	62,91%	59,80%	45,94%	9,42%	113,32	NS-BS-NNge	62,78%	49,78%	6,91%	24,00	15
csp	NS-CV-NB	62,82%	61,77%	46,74%	14,50%	95,91	NS-CV-J48	62,72%	48,20%	12,99%	15,00	15
dv320	AIF-HO-OneR	64,91%	62,98%	52,04%	10,48%	82,75	NS-CV-OneR	63,18%	53,34%	11,80%	8,00	15
dv60	Conv-CV-NNge	40,14%	36,65%	30,89%	4,30%	2537,32	Conv-CV-NNge	39,49%	31,89%	4,96%	12,00	15
elnino	NS-BS-IBk	60,29%	59,14%	41,37%	8,94%	262,00	NS-HO-IBk	46,46%	37,83%	5,75%	32,00	15
enstein	Conv-BS-IBk	32,15%	26,18%	13,60%	6,59%	100,39	AIF-CV-OneR	15,34%	11,78%	2,81%	20,00	15
gcol	Conv-CV-OneR	95,14%	93,90%	80,99%	22,54%	1442,45	NS-BS-NB	93,67%	72,79%	31,79%	1057,00	15
genelist_new	NS-CV-NNge	76,13%	75,29%	69,00%	5,90%	369,77	Conv-CV-NNge	78,39%	69,62%	6,99%	20,00	15
indtrack	AIF-BS-J48	95,73%	94,21%	85,24%	11,26%	854,73	NS-HO-J48	93,64%	87,76%	8,75%	182,00	15
ipums_census	AIF-CV-J48	94,17%	91,07%	86,97%	3,82%	36000,70	AIF-CV-OneR	97,12%	85,87%	5,54%	979,00	15
lap_vowels	NS-BS-IBk	60,27%	59,25%	34,30%	11,41%	8378,72	NS-CV-IBk	49,48%	37,11%	8,73%	1582,00	15
mdmkp	AIF-BS-IB1	66,94%	64,40%	45,17%	15,19%	1097,30	NS-BS-IB1	66,09%	53,38%	14,97%	607,00	15
mknapct	NS-BS-Prism	35,42%	34,92%	16,81%	9,47%	193,94	Conv-BS-IBk	33,82%	22,03%	10,59%	19,00	15
robot_ef	NS-BS-IBk	80,74%	79,45%	66,39%	10,73%	225,28	NS-BS-IBk	81,14%	72,37%	8,37%	198,00	15
scpnrf3	NS-CV-J48	98,98%	98,71%	79,98%	26,08%	273,19	NS-BS-J48	98,84%	90,06%	20,70%	168,00	15
scpnrf3-v2	NS-CV-NNge	97,19%	96,28%	78,20%	23,93%	50,31	NS-HO-OneR	97,03%	87,75%	21,75%	21,00	15
scpnrh4	Conv-CV-Id3	98,39%	98,25%	83,80%	24,05%	518,85	NS-BS-OneR	98,08%	91,10%	20,51%	653,00	15
smni_eeg	Conv-BS-IBk	29,62%	26,52%	12,82%	6,11%	6140,21	NS-BS-NNge	18,01%	14,45%	3,35%	2307,00	15
soluzione-fifo	AIF-BS-IBk	51,05%	49,32%	29,77%	10,06%	250,35	NS-BS-IBk	50,88%	38,07%	9,29%	86,00	15
statistiche-fifo	AIF-CV-Id3	46,68%	44,64%	30,75%	10,16%	94,81	Conv-BS-IBk	40,08%	30,38%	7,59%	17,00	15
statistiche	AIF-CV-DS	49,97%	49,65%	42,90%	7,32%	244,58	Conv-CV-NB	49,24%	44,32%	3,38%	7,00	15
steine	AIF-BS-IB1	32,77%	31,19%	13,05%	10,57%	85,71	NS-BS-IBk	32,94%	23,13%	10,26%	37,00	15
steine-v2	AIF-BS-Prism	33,54%	33,41%	14,38%	9,72%	12358,22	Conv-BS-NNge	31,22%	18,52%	10,31%	16,00	15
tdtodd1	NS-BS-IBk	69,07%	66,78%	44,53%	17,30%	7247,09	NS-BS-IBk	69,96%	54,20%	16,00%	4085,00	15
te	NS-BS-IBk	77,30%	74,95%	41,74%	18,55%	1249,91	NS-BS-IBk	77,83%	50,83%	22,83%	658,00	15
vtpack7	AIF-BS-Prism	34,68%	33,87%	14,40%	10,20%	1390,70	AIF-BS-NNge	35,13%	21,32%	11,50%	540,00	15

Tabela 5.1: Avaliação experimental entre os algoritmos `Naïve` e `Expert`

Por ser "framework", tanto o reuso do seu código como a adição de novas técncias são facilitados. Por exemplo, as três técnicas de amostragem e as três técnicas de fragmentação usados em nosso "framework" foram reusadas de (Toebe, 2002). Como já explicamos no capítulo 3, para que seja adicionada uma técnica de amostragem (fragmentação) basta apenas estender a classe abstrata `SamplingTechniqueAbstract` (`SplittingTechniqueAbstract`) ou implementar a interface `SamplingTechnique` (`SplittingTechnique`).

5.6 Considerações Finais

Uma síntese comparativa do nosso trabalho com os trabalhos correlatos (Witten & Frank, 1999; Brazdil *et al*, 2000; Morik & Scholz, 2004; Bernstein *et al*, 2002; Toebe, 2002) compõe a Tabela 5.2.

Dentre as várias características listadas na tabela, as principais, face nossos objetivos, é 'Desempenho bom em relação à acurácia' e 'Desempenho bom em relação ao tempo de execução'. O primeiro item identifica se a abordagem em questão propõe uma solução eficiente para o problema da instabilidade em relação à acurácia, isto é, se a melhor acurácia indicada pela abordagem é similar à indicada pela busca exaustiva, na grande maioria dos arquivos de dados avaliados em sua respectiva validação

experimental. A segunda característica citada diz respeito à eficácia da abordagem proposta com relação ao tempo de execução, ou seja, se, na grande maioria dos casos avaliados, o tempo para apontar a melhor acurácia é menor que o tempo gasto pela busca exaustiva.

Características	**Abordagens**					
	(Witten & Frank, 1999)	**(Brazdil *et al*, 2002)**	**(Morik & Scholz, 2004)**	**(Bernstein *et al*, 2002)**	**(Toebe, 2002)**	**Nossa abordagem**
Abordagem usando heurísticas	N	N	N	S	S	S
Abordagem usando similaridade	N	S	S	N	N	N
Uso de diversas técnicas de amostragem	S	N	S	S	S	S
Uso de diversas técnicas de fragmentação	S	N	S	S	S	S
Automatização da execução de processos de MD	N	N	S	S	S	S
Validação Experimental	N	S	N	S	S	S
Desempenho bom em relação à acurácia	N	N	N	N	N	S
Desempenho bom em relação ao tempo de execução	N	S	N	S	S	S
Reuso do código	S	N	N	N	S	S
Facilitação da adição de novas técnicas	S	N	S	S	S	S

Tabela 5.2: Síntese dos trabalhos correlatos (S=Sim; N=Não)

Desta forma, ao analisar a tabela considerando essas duas características, podemos concluir que nosso trabalho é a única abordagem que, levando em conta a diversidade de técnicas de amostragem, fragmentação e classificação, consegue acurácias similares às obtidas pela pesquisa exaustiva de modelos de classificação, ao mesmo tempo que reduz comparativamente o custo computacional. Em outras palavras, é a única abordagem que apresenta bom desempenho em relação a ambos os critérios acurácia e tempo de execução.

É importante citar os trabalhos (Bensusan *et al*, 2000; Lindner & Studer, 1999; Peng *et al*, 2002; Todorovski *et al*, 2002; Pfahringer *et al*, 2000) que foram lidos no intuito de contribuir para a definição dos meta-atributos de arquivos de dados que usamos em nossos experimentos. Outra literatura importante que nos ajudou a idealizar o processo de análise de meta-mineração foi (Kalousis & Hilario, 2001).

Capítulo 6

Conclusões e Perspectivas

O foco da dissertação foi a descoberta de heurísticas que podem ajudar a reduzir substancialmente o escopo do espaço de busca do melhor processo de MD, que é aquele que atende aos requisitos de desempenho do minerador, qual sejam: (1) o modelo de classificação com a melhor acurácia, e (2) com custos de processamento aceitáveis.

Um processo de MD, no contexto da dissertação é composto de 3 etapas — amostragem, fragmentação e indução, nesta ordem —; para cada uma destas etapas, dispõe-se de uma variedade de técnicas. Por exemplo, nos nossos experimentos, usamos 3 técnicas de amostragem, 3 de fragmentação, e 9 técnicas de indução; isto resulta em 81 (3x3x9) possíveis processos de MD, que constituíram o nosso espaço de busca. O problema é que, dado um arquivo de dados para minerar, cada um desses 81 processos pode induzir um modelo de classificação diferente. Este fato é conhecido como *o problema da instabilidade* de processos de MD. Uma outra maneira de caracterizar o objetivo geral da dissertação é o de encontrar uma solução para o problema da instabilidade.

Propusemos uma abordagem baseada em heurísticas para a solução do problema, A lógica de tal enfoque baseia-se na seguinte observação: guiado por um conjunto de heurísticas, pode-se descartar preventivamente alguns, ou até muitos, processos de MD de um espaço de busca, sem prejuízo da qualidade do modelo de classificação induzido; como conseqüência da redução do espaço de busca, consegue-se também uma redução do tempo de indução do modelo.

Descobrimos, ao todo, oitenta e três heurísticas, assim tipificadas: (1) oitenta delas são heurísticas de acurácia, isto é, heurísticas que podam processos de MD com desempenho ruim em relação à acurácia; (2) as outras três, são heurísticas de tempo, que tem como objetivo principal descartar processos de MD baseado no tamanho do arquivo de dados a ser minerado (grande ou pequeno), para reduzir, comparativamente, ainda mais o tempo de execução.

Como exemplo de heurística de acurácia, citemos uma das heurísticas descobertas *o processo* `Convergence-CrossValidation-DecisionStump` *deve ser descartado para arquivos a minerar com um número de classes entre 2 e 16*. Fica óbvio, então, que uma ferramenta de mineração de dados, guiada por esta heurística, poderia descartar preventivamente o processo de MD referenciado, desta forma reduzindo o espaço de busca de processos adequados ao caso específico.

Para ilustrar heurísticas de tempo, seja esta outra nossa heurística: *o classificador* `IBk` *deve ser descartado para arquivos a minerar grandes, isto é, com o número de instâncias maior que 2567*. Já uma ferramenta, apoiada por este heurística, poderia podar os processos que continham este classificador, que tem um alto custo computacional, reduzindo assim, consideravelmente o tempo de execução.

Para descobrir as heurísticas, planejamos e executamos um processo de descoberta de heurísticas. O processo em si consistiu em analisar empiricamente o desempenho de 81 processos de MD aplicados em 95 arquivos de dados, segundo os critérios acurácia e tempo de execução. Para isto, armazenamos os resultados destas minerações em uma base de conhecimento. Em seguida, primeiramente, criamos o *algoritmo de poda de processos de MD* que descarta processos de MD com desempenho ruim em relação à acurácia, e assim descobrimos as heurísticas de acurácia. Depois, mineramos a base de conhecimento no intuito de conceituar arquivos grandes e pequenos. Deste modo foi possível: (1) determinar quando é necessário ou não o uso de amostras — se um arquivo é pequeno (grande), então não é (é) preciso usá-las —, e (2) afastar do espaço de busca os processos que tem o classificador com maior custo computacional. Essas são as heurísticas de tempo.

Construímos uma ferramenta que disponibiliza dois algoritmos que automatizam a execução de processos de MD: `Mining Processes Naïve Executer` (`Naïve`) e `Mining Processes Fast Executer` (`Fast`). O primeiro executa todos os processos de MD disponíveis (busca sem heurística), enquanto que o segundo é baseado nas heurísticas descobertas, selecionando assim, um conjunto daqueles processos para execução. Visto que a área de mineração de dados é dinâmica e frequentemente aparecem novos algoritmos, então resolvemos aplicar uma abordagem de "framework" à ferramenta, para, deste modo, possibilitar a adição de novas técnicas de forma simples e rápida, além de prover o reuso de projeto e código.

Na avaliação experimental, comparamos o algoritmo `Fast` com `Naïve` em 30 novos arquivos de dados. O objetivo da validação é que `Fast` obtenha uma acurácia similar — dentro de margem de aceitação proposta — à obtida por `Naïve`, porém com um custo computacional reduzido comparativamente.

Para avaliar ambos os algoritmos, usamos dois cenários. No primeiro, comparamos `Fast`, apoiado apenas nas heurísticas de acurácia, com `Naïve`. Em 100% dos arquivos de dados experimentados, `Fast`, obteve acurácias similares às obtidas via `Naïve`. No segundo cenário, usamos tanto as heurísticas de acurácias quanto as heurísticas de tempo ao comparar `Fast` com `Naïve`. Neste cenário, `Fast` expôs um tempo bastante reduzido em relação ao tempo despendido com `Naïve`. Entretanto, houve uma pequena perda em três arquivos de dados em relação à acurácia. Em outras palavras, em 27 (90%) dos 30 arquivos de dados, `Fast` apresentou acurácias similares as obtidas com `Naïve`.

Os resultados atingidos foram muitos satisfatórios, permitindo-nos concluir que o algoritmo `Fast` consegue reduzir, comparativamente, o escopo do espaço de busca do melhor modelo de classificação, ao mesmo tempo que apresenta acurácias similares às obtidos com `Naïve`. Portanto, estes resultados confirmam a validade das heurísticas usadas pelo algoritmo `Fast`.

Desta forma, baseados nos experimentos realizados ao longo do trabalho, chegamos a uma conclusão a respeito das perguntas levantadas no Capítulo 1:

- dado um arquivo de dados, qual a melhor técnica de amostragem? E a melhor de fragmentação? E a melhor de classificação? A priori estas perguntas continuam sem respostas já que, como demonstramos, todas as etapas têm uma importância fundamental no resultado final da mineração;
- dado um arquivo de dados, qual o melhor processo de MD? Apesar de continuarmos sem saber qual é o melhor processo, conseguimos identificar, através das heurísticas de acurácias, aqueles que produzem os piores resultados para o referido arquivo de dados.

6.1 Perspectivas

Embora os objetivos de nosso trabalho tenham sido cumpridos, existem algumas limitações que podem ser identificadas, no sentido de aperfeiçoá-lo e estendê-lo. Entre elas:

- No intuito de corrigir a limitação do "framework", discutida na seção 3.3, que é a de não gerar heurísticas para uma nova técnica qualquer adicionada, é interessante que o processo de descoberta de heurísticas de acurácia seja automatizado já que a mesma envolve tarefas manuais. Para isto, primeiramente, é necessário que os processos de MD compostos pela nova técnica sejam automaticamente executados em todos os 95 arquivos de dados usados. Em seguida, todos os resultados das minerações seriam armazenados na base de conhecimento. Feita esta etapa inicial, o passo seguinte é implementar o algoritmo de poda de processos de MD, que tem heurísticas como saída. Uma atividade que deve ser realizada para tal fim é fazer com que os modelos de classificação induzidos sejam modelados em forma de classes, pois, atualmente, a saída é analítica. Enfim, deve ser automatizada a conversão das heurísticas descobertas em XML para uso posterior. Sendo assim, quando uma nova técnica for incorporada ao "framework", um processamento será realizado uma única vez, para que sejam geradas, se for o caso, heurísticas que consideram processos de MD concernentes à técnica acrescentada;
- Desenvolver uma interface gráfica para visualização do conhecimento induzido que, em nossa ferramenta, é apresentado de forma textual. Tendo os modelos de classificação modelados em classe, como foi sugerido, facilitaria esta atividade;
- Considerar outros critérios de avaliação do conhecimento induzido, além de acurácia e tempo de execução. Legibilidade e concisão são algumas possibilidades a serem investigadas;
- Descobrir novas heurísticas para reduzir ainda mais o tempo de execução. Para isto, um caminho possível é considerar novos meta-atributos, como entropia de classe;

- Explorar técnicas de paralelismo no sentido de poder executar vários processos de MD em paralelo. Assim, o tempo de execução diminuiria mais ainda.

Referências Bibliográficas

(Bensusan *et al*, 2000) Bensusan, H., Giraud-Carrier, C. and Kennedy, C: "A higher-order approach to meta-learning". In *Proceedings of the ECML'2000 workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, 2000.

(Bernstein & Provost, 2001) Bernstein, A. and Provost, F: "An Intelligent Assistant for the Knowledge Discovery Process". In *Proceedings of the IJCAI-01 Workshop on Wrappers for Performance Enhancement in KDD*. Morgan Kaufmann, Seattle, WA, 2001.

(Blake & Merz, 2002) Blake C. L.; Merz C. J.: "UCI Repository of Machine Leraning Databases http://www.ics.uci.edu/~mlearn/MLRepository.html)". Irvine, CA: University of California, Department of Information and Computer Science, 2002.

(Brazdil *et al*, 2002) Brazdil, P. B., Soares, C. and Costa, J: "Ranking Learning Algorithms". Kluwer Academic Publishers, 2002.

(Brumen *et al*., 2001) Brumen, B.; Welzer, T.; Jaakkola, H.: "Adaptive Incremental Framework for Performance Driven Data Mining". Advances in Databases and Information Systens (ADBIS 2001), Lithuania, September 2001, pp. 193 – 203

(Cabena *et al*., 1997) Cabena, P.; Hadjinian, P.; Stadler, R.; Verhees, J.; Zanasi, A.: "Discovering Data Mining: from concept to implementation". Morgan Kaufmann Publishers, 1997, 195 p.

(Fayyad *et al*., 1996) Fayyad, Usama; Shapiro-Piatetsky Gregory; Smyth Padhraic; Uthurusamy Ramasamy: "Adavances in Knowledge Discovery and Data Mining". AAAI Press, 1996, 611 p.

(Han & Micheline, 2001) Han, J; Micheline, K.: "Data Mining: concepts and techniques". Morgan Kaufmann Publishers, 2001, 550 p.

(Ikizler & Güvenir, 2001) Ikizler, N.; Güvenir, H. A.: "Mining Interesting Rules in Bank Loans Data". In Proceedings of the Tenth Turkish Symposium on Artificial Intelligence and Neural Networks (TAINN'2001), Gazimagusa, T.R.N.C. 2001.

(Kalousis & Hilario, 2001) Kalousis, A.; Hilario, M.: "Model Selection via Meta-Learning: A Comparative Study". International Journal on Artificial Intelligence Tools, Volume 10, No.4, 2001.

(Kohavi, 1996) Kohavi, Ron: "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection". In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1996, pp. 1137 – 1143.

(Landin & Niklasson, 1995) Landin, N.; Niklasson, A.: "Development of Object-Oriented Frameworks". Dissertação de Mestado. Department of Communication Systems, Lund University, 1995.

(Larman, 1998) Larman, C. Applying UML and Patterns: "An Introduction to Objetct-Oriented Analysis and Design". New Jersey: Prentice Hall, 1998, 509p.

(Lindner & Studer, 1999) Lindner, C., & Studer, R..: "AST: Support for Algorithm Selection with a CBR Approach". In *Proceedings of the 16th International Conference on Machine Learning,Workshop on Recent Advances in Meta-Learning and Future Work*. 1999.

(Morik & Scholz, 2004) Morik, K.; Scholz, M.: "The MiningMart Approach to Knowledge Discovery in Databases". In Intelligent Technologies for Information Analysis. Springer-Verlag, 2004.

(Peng *et al*, 2002) Peng, Y., Flach, P., Soares, C.; Brazdil, P.: "Improved Dataset Characterization for Meta-learning". In *Proceedings of the 5th International Conference on Discovery Science*, 2002.

(Pfahringer *et al*, 2000) Pfahringer, B., Bensusan, H.; Giraud_Carrier, C.: "Tell me who can learn you and I can tell you who you are: Landmarking various learning algorithms". In *Proceedings of the 7th International Conference on Machine Learning*. Morgan Kaufman, 2000.

(Pyle, 1999) Pyle, Dorian: "Data Preparation for Data Mining". San Francisco: Morgan Kaugmann Publishers, 1999, 540 p.

(Sauvé, 2005) Sauvé, Jacques.: "Frameworks: notas de aula". Campina Grande: UFCG, 2005. Disponível: http://jacques.dsc.ufcg.edu.br/cursos/map/html/frame/oque.htm [capturado em 30 de mar. 2005].

(Silberschatz & Tuzhilin, 1996) Silberschatz, A.; Tuzhilin, A. "What Makes Patterns Interesting in Knowledge Discovery Systems". IEEE Transac. on Knowledge and Data Eng., 8(6):970--974, 1996.

(Todorovski *et al*, 2002) Todorovski, L., Blockeel, H. and Dzeroski, S. "Ranking with Predictive Clustering Trees". In *Proceedings of the 13th European Conference on Machine Learning*, 2002.

(Toebe, 2002) Toebe, Josué: "Projeto e Construção de um ambiente para extrair conhecimento de banco de dados da Petrobrás". Dissertação de Mestrado, Universidade Federal de Campina Grande, 2002.

(Weiss & Indurkhya, 1998) Weiss, S. M.; Indurkhya, N.: "Predictive Data Mining: a practical guide". Morgan Kaufmann Publishers, 1998, 228 p.

(Witten & Frank, 1999) Witten, Ian H.; Frank, Eibe: "Data Mining: Pratical Machine Learning Tools and Techniques with Java Implementations". San Diego: Morgan Kaufmann Publishers, 1999, 369 p.

Apêndice A

Lista das heurísticas de acurácia

H1 O processo de MD AIF−BS−DS deve ser descartado do espaço de busca quando 3≤NA≤25 ou 34≤NA≤55

H2 O processo de MD AIF−BS−DS deve ser descartado do espaço de busca quando NC≤10 ou 16≤NC≤30

H3 O processo de MD AIF−CV−DS deve ser descartado do espaço de busca quando 3≤NA≤33 ou 35≤NA≤62

H4 O processo de MD AIF−CV−DS deve ser descartado do espaço de busca quando NC≤16 ou 18≤NC≤39

H5 O processo de MD AIF−HO−DS deve ser descartado do espaço de busca quando 3≤NA≤62

H6 O processo de MD AIF−HO−DS deve ser descartado do espaço de busca quando NC≤22

H7 O processo de MD Conv−BS−DS deve ser descartado do espaço de busca quando 6≤NA≤20 ou 34≤NA≤56

H8 O processo de MD Conv−BS−DS deve ser descartado do espaço de busca quando NC≤10 ou 13≤NC≤28

H9 O processo de MD Conv−CV−DS deve ser descartado do espaço de busca quando 3≤NA≤28 ou 35≤NA≤62

H10 O processo de MD Conv−CV−DS deve ser descartado do espaço de busca quando NC≤16

H11 O processo de MD Conv−HO−DS deve ser descartado do espaço de busca quando 3≤NA≤24 ou 34≤NA≤55

H12 O processo de MD Conv−HO−DS deve ser descartado do espaço de busca quando NC≤39

H13 O processo de MD NS−BS−DS deve ser descartado do espaço de busca quando 3≤NA≤28 ou 35≤NA≤62

H14 O processo de MD NS−BS−DS deve ser descartado do espaço de busca quando NC≤10 ou 16≤NC≤29

H15 O processo de MD NS−CV−DS deve ser descartado do espaço de busca quando 4≤NA≤26 ou 34≤NA≤55

H16 O processo de MD NS−CV−DS deve ser descartado do espaço de busca quando NC≤19

H17 O processo de MD NS−HO−DS deve ser descartado do espaço de busca quando 3≤NA≤20 ou 33≤NA≤55

H18 O processo de MD NS−HO−DS deve ser descartado do espaço de busca quando NC≤14

H19 O processo de MD AIF−HO−IB1 deve ser descartado do espaço de busca quando 4≤NA≤23

H20 O processo de MD AIF−HO−IB1 deve ser descartado do espaço de busca quando NC≤18

H21 O processo de MD Conv−CV−IB1 deve ser descartado do espaço de busca quando 3≤NA≤61

H22 O processo de MD Conv−CV−IB1 deve ser descartado do espaço de busca quando NC≤15

H23 O processo de MD Conv−HO−IB1 deve ser descartado do espaço de busca quando 3≤NA≤62

H24 O processo de MD Conv−HO−IB1 deve ser descartado do espaço de busca quando NC≤9

H25 O processo de MD NS−CV−IB1 deve ser descartado do espaço de busca quando 3≤NA≤28

H26 O processo de MD NS−HO−IB1 deve ser descartado do espaço de busca quando 17≤NA≤31

H27 O processo de MD Conv−CV−IBk deve ser descartado do espaço de busca quando 3≤NA≤61

H28 O processo de MD Conv−CV−IBk deve ser descartado do espaço de busca quando NC≤9

H29 O processo de MD Conv-HO-IBk deve ser descartado do espaço de busca quando 3≤NA≤62

H30 O processo de MD Conv-HO-IBk deve ser descartado do espaço de busca quando NC≤15

H31 O processo de MD AIF-CV-Id3 deve ser descartado do espaço de busca quando 3≤NA≤56

H32 O processo de MD AIF-CV-Id3 deve ser descartado do espaço de busca quando NC≤39

H33 O processo de MD Conv-CV-Id3 deve ser descartado do espaço de busca quando 3≤NA≤41

H34 O processo de MD Conv-CV-Id3 deve ser descartado do espaço de busca quando NC≤19

H35 O processo de MD Conv-HO-Id3 deve ser descartado do espaço de busca quando 3≤NA≤41

H36 O processo de MD Conv-HO-Id3 deve ser descartado do espaço de busca quando NC≤13

H37 O processo de MD NS-CV-Id3 deve ser descartado do espaço de busca quando 4≤NA≤35

H38 O processo de MD NS-HO-Id3 deve ser descartado do espaço de busca quando 3≤NA≤23 ou 33≤NA≤55

H39 O processo de MD NS-HO-Id3 deve ser descartado do espaço de busca quando NC≤10 ou 16≤NC≤30

H40 O processo de MD Conv-CV-J48 deve ser descartado do espaço de busca quando 3≤NA≤28

H41 O processo de MD Conv-CV-J48 deve ser descartado do espaço de busca quando NC≤18

H42 O processo de MD Conv-HO-J48 deve ser descartado do espaço de busca quando 4≤NA≤20

H43 O processo de MD Conv-HO-J48 deve ser descartado do espaço de busca quando NC≤16

H44 O processo de MD Conv-HO-NB deve ser descartado do espaço de busca quando 4≤NA≤35

H45 O processo de MD `Conv-HO-NB` deve ser descartado do espaço de busca quando `NC≤11`

H46 O processo de MD `Conv-CV-NNge` deve ser descartado do espaço de busca quando `4≤NA≤16`

H47 O processo de MD `Conv-HO-NNge` deve ser descartado do espaço de busca quando `3≤NA≤28`

H48 O processo de MD `Conv-HO-NNge` deve ser descartado do espaço de busca quando `NC≤18`

H49 O processo de MD `AIF-BS-OneR` deve ser descartado do espaço de busca quando `3≤NA≤45`

H50 O processo de MD `AIF-BS-OneR` deve ser descartado do espaço de busca quando `NC≤22`

H51 O processo de MD `AIF-CV-OneR` deve ser descartado do espaço de busca quando `3≤NA≤22` ou `32≤NA≤55`

H52 O processo de MD `AIF-CV-OneR` deve ser descartado do espaço de busca quando `NC≤10` ou `17≤NC≤30`

H53 O processo de MD `AIF-HO-OneR` deve ser descartado do espaço de busca quando 3`≤NA≤62`

H54 O processo de MD `AIF-HO-OneR` deve ser descartado do espaço de busca quando `NC≤10` ou `17≤NC≤30`

H55 O processo de MD `Conv-BS-OneR` deve ser descartado do espaço de busca quando `3≤NA≤62`

H56 O processo de MD `Conv-BS-OneR` deve ser descartado do espaço de busca quando `NC≤15`

H57 O processo de MD `Conv-CV-OneR` deve ser descartado do espaço de busca quando `3≤NA≤28` ou `33≤NA≤62`

H58 O processo de MD `Conv-CV-OneR` deve ser descartado do espaço de busca quando `NC≤16`

H59 O processo de MD `Conv-HO-OneR` deve ser descartado do espaço de busca quando `4≤NA≤24` ou `33≤NA≤54`

H60 O processo de MD `Conv-HO-OneR` deve ser descartado do espaço de busca quando `NC≤11`

H61 O processo de MD NS-BS-OneR deve ser descartado do espaço de busca quando 3≤NA≤28 ou 33≤NA≤62

H62 O processo de MD NS-BS-OneR deve ser descartado do espaço de busca quando NC≤22

H63 O processo de MD NS-CV-OneR deve ser descartado do espaço de busca quando NC≤11

H64 O processo de MD NS-HO-OneR deve ser descartado do espaço de busca quando 4≤NA≤19 ou 23≤NA≤35 ou 38≤NA≤59

H65 O processo de MD NS-HO-OneR deve ser descartado do espaço de busca quando NC≤39

H66 O processo de MD AIF-BS-Prism deve ser descartado do espaço de busca quando 26≤NA≤62

H67 O processo de MD AIF-BS-Prism deve ser descartado do espaço de busca quando NC≤12

H68 O processo de MD AIF-CV-Prism deve ser descartado do espaço de busca quando 3≤NA≤23 ou 26≤NA≤62

H69 O processo de MD AIF-CV-Prism deve ser descartado do espaço de busca quando NC≤11

H70 O processo de MD AIF-HO-Prism deve ser descartado do espaço de busca quando 3≤NA≤62

H71 O processo de MD AIF-HO-Prism deve ser descartado do espaço de busca quando NC≤11

H72 O processo de MD Conv-BS-Prism deve ser descartado do espaço de busca quando NC≤14

H73 O processo de MD Conv-CV-Prism deve ser descartado do espaço de busca quando 4≤NA≤60

H74 O processo de MD Conv-CV-Prism deve ser descartado do espaço de busca quando NC≤19

H75 O processo de MD Conv-HO-Prism deve ser descartado do espaço de busca quando 3≤NA≤41

H76 O processo de MD Conv-HO-Prism deve ser descartado do espaço de busca quando NC≤19

H77 O processo de MD `NS-CV-Prism` deve ser descartado do espaço de busca quando $3 \leq NA \leq 62$

H78 O processo de MD `NS-CV-Prism` deve ser descartado do espaço de busca quando $NC \leq 22$

H79 O processo de MD `NS-HO-Prism` deve ser descartado do espaço de busca quando $3 \leq NA \leq 62$

H80 O processo de MD `NS-HO-Prism` deve ser descartado do espaço de busca quando $NC \leq 24$

Apêndice B

Lista detalhada das heurísticas de acurácia

A lista detalhada das heurísticas de acurácia está no CD-ROM anexo a esta dissertação.

Apêndice C

Lista das heurísticas de acurácia e tempo em XML

A lista detalhada das heurísticas de acurácia e tempo no formato XML está no CD-ROM anexo a esta dissertação.