UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MURILO ZANGARI DE SOUZA

EXTENSÕES DO ALGORITMO *ANT-MINER* PARA TRATAR O PROBLEMA DE BASES DE DADOS DESBALANCEADAS

Maringá
2012

MURILO ZANGARI DE SOUZA

EXTENSÕES DO ALGORITMO *ANT-MINER* PARA TRATAR O PROBLEMA DE BASES DE DADOS DESBALANCEADAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Ademir Aparecido Constantino

Maringá
2012

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá – PR., Brasil)

S729e

Souza, Murilo Zangari de
Extensões do Algoritmo *Ant-Miner* para tratar o problema de base de dados desbalanceadas / Murilo Zangari de Souza. -- Maringá, 2012.
75 f. : figs., tabs.

Orientador: Prof. Dr. Ademir Aparecido Constantino.
Dissertação (mestrado) - Universidade Estadual de Maringá, Centro de Tecnologia, Departamento de Informática, Programa de Pós-Graduação em Ciência da Computação, 2012.

1. Mineração de Dados. 2. Otimização por colônia de formigas artificiais (ACO). 3. Bases de dados desbalanceadas. 4. Algoritmo *Ant-Miner*. I. Constantino, Ademir Aparecido, orient. II. Universidade Estadual de Maringá. Centro de Tecnologia. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 21.ed. 006.4

GVS-001300

FOLHA DE APROVAÇÃO

MURILO ZANGARI DE SOUZA

Extensões do Algoritmo Ant-Miner para Tratar o Problema de Bases de Dados Desbalanceadas

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Banca Examinadora composta pelos membros:

BANCA EXAMINADORA

Prof. Dr. Ademir Aparecido Constantino
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Wesley Romão
Universidade Estadual de Maringá – DIN/UEM

Profa. Dra. Valéria Delisandra Feltrim
Universidade Estadual de Maringá – DIN/UEM

Profa. Dra. Deborah Ribeiro Carvalho
Pontifícia Universidade Católica – PPGTS/PUC-PR

Aprovada em: 26 de julho de 2012.
Local da defesa: Sala 101, Bloco C56, *campus* da Universidade Estadual de Maringá.

AGRADECIMENTOS

Aos orientadores e professores Ademir Aparecido Constantino e Wesley Romão que me deram a oportunidade de realizar o mestrado.

A professora Valéria Feltrim e demais professores do DIN que contribuíram de alguma forma.

A Maria Inês, secretária do programa de pós-graduação do DIN, pela eficiência e dedicação no seu trabalho.

Aos amigos e colegas de mestrado: Alison, pela ajuda com implementação; Everton, o qual contribuiu com as direções tomadas na pesquisa; Lurdete, pelas conversas e apoio na parte didática e Juliano, nas correções dos artigos em inglês.

Aos amigos fora do mestrado, Claris, Graciele, Glaucia, Willans, Dean e Diego pela amizade e apoio durante este tempo.

A CAPES pelo apoio financeiro.

Aos meus pais, pela paciência e apoio.

Extensões do algoritmo *Ant-Miner* para tratar o problema de bases de dados desbalanceadas

RESUMO

A tarefa de classificação em Mineração de Dados utiliza algoritmos chamados de classificadores para extrair padrões sobre bases de dados. Bases de dados reais podem apresentar um desbalanceamento em suas classes, contendo mais casos de uma classe do que de outras. Algoritmos classificadores têm dificuldade em encontrar padrões de qualidade para as classes minoritárias, pelo fato dos casos pertencentes à classe minoritária possuírem pouca representatividade no conjunto de treinamento. Isto se torna um problema quando a classe minoritária é a de maior interesse para o usuário. O objetivo deste trabalho é o desenvolvimento de extensões para o algoritmo *Ant-Miner* (*Ant Colony-based Data Miner*) para ajudar a encontrar melhores regras para as classes minoritárias. Essas extensões modificam, principalmente, a forma como as regras são construídas e avaliadas. O algoritmo *Ant-Miner* é baseado na meta heurística *ACO* (*Ant Colony Optimization*) e tanto a versão original quanto outros trabalhos relacionados mostram que a técnica é competitiva com outros algoritmos de classificação. Além disso, são analisadas técnicas de balanceamento (*undersampling e oversampling*) e também um estudo da análise ROC (*Receiver Operating Characteristics).* As técnicas de balanceamento visam fazer uma nova amostragem dos dados mudando a distribuição do conjunto de treinamento. A análise ROC realiza avaliações mais apuradas que outras métricas (*e.g.:* taxa de acerto), principalmente quando se trata de bases com classes desbalanceadas. Resultados experimentais mostraram que os algoritmos desenvolvidos contribuíram para a descoberta de melhores regras para as classes minoritárias e também com a simplicidade do modelo de regras.

Palavras-chave: Mineração de dados, tarefa de classificação, classes desbalanceadas, *Ant-Miner*, amostragem, análise ROC.

Extensions to the Ant-Miner Algorithm to Deal with Imbalanced Data Sets

ABSTRACT

The classification task in Data Mining uses algorithms called classifiers to find patterns on data bases. Real data bases can have an imbalance in its classes, when there are more cases of one class then the others. Classification algorithms are sensitive of this imbalance and tend to valorize the majority class and ignore de minority class, because the cases of minority class have low representation on the training set. It is a problem when the minority class is the class of interest. In this work we propose two extensions to the Ant-Miner algorithm to find better rules to the minority classes. These extensions modify, mainly, how rules are constructed and evaluated. The Ant-Miner algorithm is based on ACO (Ant Colony Optimization). The original version and others related works showed that the Ant-Miner is competitive with other standard classifiers. Moreover, we analyzed sampling techniques (undersampling and oversampling) and also a study of ROC (Receiver Operating Characteristics) analysis. The sampling techniques aim to make a new sampling of the data sets changing the classes' proportion of the training set. The ROC analysis can evaluate the results with more accurate than other metrics, mainly when the classifiers are applied in data sets with classes imbalance. Experimental results showed that the developed algorithms contribute to the rule discovery of the minority classes and also contribute with the simplicity of the rules.

Keywords: Data Mining, classification task, class imbalance, Ant-Miner, sampling, ROC analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACO	*Ant Colony Optimization*
AG	Algoritmo Genético
AUC	*Area Under Curve*
CID	Classificação Internacional de Doenças
CNN	*Condensed Nearest Neighbor Rule*
ENN	*Edited Nearest Neighbor Rule*
FN	Falso Negativo
FP	Falso Positivo
KDD	*Knowledge Discovery in Database*
KNN	*K Nearest Neighbor*
MD	Mineração de Dados
ROC	*Receiver Operating Characteristics*
SMOTE	*Synthetic Minority Oversampling Technique*
T(VP)	Taxa de verdadeiros positivos
T(FP)	Taxa de falsos positivos
UCI	*University of California at Irvine*
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

SUMÁRIO

Capítulo

1

Introdução

1.1 Considerações Iniciais

A atual era da informação é caracterizada pela grande expansão no volume de dados gerados e armazenados. Esta situação tem gerado demanda por novas técnicas e ferramentas, que ajudam a transformar os dados armazenados e processados em conhecimento, motivando pesquisas na área de Mineração de dados.

Mineração de dados (MD) consiste em um conjunto de conceitos e métodos com o objetivo de encontrar uma descrição, preferencialmente compreensível e interessante para o usuário, de padrões e regularidades em um determinado conjunto de dados. Um padrão é definido como um tipo de declaração (ou modelo de uma declaração) sobre o conjunto de dados que está sendo analisado (Carvalho, 2005).

A MD é considerada uma etapa do processo de Descoberta de Conhecimento em Banco de dados (do inglês, *Knowledge Discovery in Databases – KDD).* O termo KDD foi estabelecido no primeiro *workshop* sobre o tema em 1989. O processo completo do KDD envolvem as etapas: seleção dos dados, pré-processamento, transformação, MD e interpretação/avaliação dos resultados (Fayyad, 1998).

Todas as etapas são importantes. Uma base de dados que não for devidamente pré-processada pode dificultar/inviabilizar a etapa de MD. Assim como a interpretação dos resultados que só é efetiva se a etapa de MD mostrar resultados padronizados e de fácil compreensão.

Logo, o objetivo da MD é a descoberta de padrões potencialmente úteis ao processo de decisão. A MD se resume em aplicar um algoritmo no qual utiliza uma base de dados como entrada e tem como saída alguma informação (padrões) expresso por algum formalismo. Uma das tarefas de MD é a classificação. A tarefa de classificação pode ser aplicada em diversas áreas, tais como: financeira, científica, médica e etc.

1.2 Contexto e Motivação

Quando um algoritmo indutor de regras é aplicado em dados reais, muitas vezes só a eficiência do modelo previsor não é suficiente, é interessante também extrair alguma informação útil a partir das regras que compõem o modelo. Estas regras podem mostrar alguns padrões não identificados manualmente e assim auxiliar a tomada de decisão dos gestores da aplicação.

As regras encontradas devem possuir um grau de compreensibilidade, pois elas serão utilizadas/interpretadas por seres humanos. Neste caso a compreensibilidade está ligada ao "tamanho" das regras, isto é, regras que possuem muitos atributos antecedentes são consideradas menos compreensíveis.

Muitas vezes, bases de dados reais apresentam um desbalanceamento nas classes, isto é, possui muito mais casos para uma classe X em relação a uma classe Y. As classes de menor proporção são caracterizadas por possuírem casos raros. Algoritmos classificadores, como o C4.5 (Quinlan, 1993), tendem a ignorar os casos raros, pois os algoritmos usam técnicas nas quais induzem a fazer uma generalização ao invés de uma especialização. Segundo Carvalho & Freitas (2003) isto se deve à crença que especializações no conjunto de treinamento são indesejáveis na validação sobre o conjunto de teste ou ao argumento de que os casos raros não deveriam ser incluídos no conjunto de regras descobertas, uma vez que eles tendem a ser uma das causas de erros na classificação no conjunto de teste.

Se os algoritmos tendem a ignorar os casos raros, consequentemente eles deixam de encontrar regras para as classes de menor proporção. Isto se torna um problema quando a classe de menor proporção é a classe de maior interesse, isto é, o algoritmo tende a encontrar apenas padrões óbvios, não descobrindo padrões potencialmente úteis ao processo do KDD, pois padrões óbvios não geram conhecimento novo.

Logo, além de ter um algoritmo classificador que gere um modelo com alto poder preditivo e regras compreensíveis, é relevante que o algoritmo encontre regras para a(s) classe(s) minoritária(s). Desta forma, é possível extrair padrões que realmente possam ser relevantes aos usuários.

Dentre os algoritmos classificadores, existem os algoritmos baseados em técnicas heurísticas, destaque para a ACO *(Anti Colony Optimization),* que faz parte do contexto deste trabalho.

A técnica ACO, proposta por Dorigo *et al.* (1996), é um sistema baseado em agentes que simulam o comportamento natural das formigas na procura por alimentos desenvolvendo

mecanismos de cooperação e aprendizado. A técnica inicialmente foi proposta como uma nova heurística para resolver problemas de otimização combinatorial.

A primeira aplicação do ACO para a tarefa de classificação em MD foi feita por Parpinelli (2001), que resultou no algoritmo *Ant-Miner*. O algoritmo de Parpinelli se mostrou competitivo com o C4.5 (algoritmo baseado em árvores de decisão – muito utilizado na literatura), se tornando uma técnica promissora e motivando novas pesquisas baseadas neste algoritmo.

1.3 Objetivos

O primeiro objetivo deste trabalho é o desenvolvimento de extensões ao algoritmo *Ant-Miner* que, além do poder preditivo e contribuir com a simplicidade do modelo, trate o problema das bases de dados desbalanceadas, encontrando melhores regras para os casos raros, consequentemente para as classes minoritárias.

O segundo objetivo é aplicar os algoritmos desenvolvidos em bases de dados desbalanceadas, comparando os resultados com outros algoritmos existentes na literatura.

Outros objetivos, não menos importantes, são: (1) Aplicar métodos de amostragem (*undersampling e oversampling)* nas bases desbalanceadas; (2) Utilizar a métrica de análise ROC para avaliar e comparar resultados obtidos.

1.4 Organização do Trabalho

Esta dissertação está organizada da seguinte forma:

Capitulo 1: Faz uma introdução sobre o trabalho, apresenta o contexto, motivação, os objetivos e a estrutura da dissertação.

Capítulo 2: Apresenta a revisão bibliográfica dos temas abordados neste trabalho, incluindo a MD, o problema de bases de dados desbalanceadas, a otimização por colônias de formigas, e por fim o algoritmo *Ant-Miner*.

Capítulo 3: Descreve os algoritmos desenvolvidos.

Capítulo 4: Apresenta a avaliação dos resultados experimentais.

Capítulo 6: Apresenta as conclusões sobre o trabalho e as sugestões de trabalhos futuros.

Capítulo

2

Revisão Bibliográfica

2.1 Considerações Iniciais

Este capítulo apresenta uma revisão bibliográfica dos temas que são abordados neste trabalho. São apresentados conceitos de Mineração de Dados, o problema de bases de dados desbalanceadas e a técnica de Otimização por Colônias de Formigas. Além disso, é descrito o algoritmo *Ant-Miner*, desenvolvido por Parpinelli (2001), e outros trabalhos relacionados ao *Ant-Miner*.

2.2 Mineração de Dados

O processo de descoberta de conhecimento é divido em várias etapas nas quais MD refere-se à aplicação de algoritmos para a extração de padrões em um conjunto de dados (Fayyad *et al.*, 1996), isto é, algoritmos de MD apenas descobrem padrões sobre um conjunto de dados. Conhecimento é gerado quando se realiza uma interpretação/avaliação nestes padrões com o auxílio de usuários que possuam domínio sobre a aplicação.

A Figura 2.1 apresenta uma visão dos passos do *KDD* apresentadas por Fayyad *et al.* (1996). Este fluxo pode ter múltiplas iterações, isto é, a sua execução pode voltar para etapas anteriores caso os resultados de uma etapa não estejam satisfatórios.

Geralmente os dados disponíveis para a mineração não estão em um formato adequado. Portanto os dados devem passar por um pré-processamento e transformados antes de serem utilizados por um algoritmo de MD. As etapas que compõem o pré-processamento são: entendimento do domínio da aplicação, escolha de um conjunto de dados alvo, realização de uma limpeza nos dados, seleção de características úteis que melhor representam os dados e selecionar os atributos a serem considerados (Fayyad *et al.* 1996).

A etapa de transformação visa deixar a base de dados compatível com o algoritmo,

isto é, que sirva como entrada para a etapa seguinte.

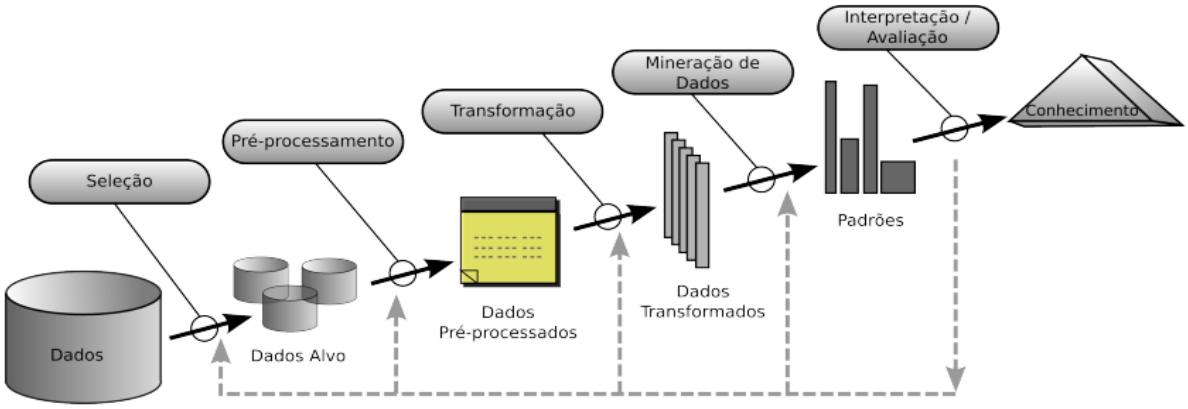


Figura 2.1: Uma visão dos passos do processo de *KDD* (Fayyad *et al.* 1996).

Na etapa de MD, escolhe-se um algoritmo propício para a aplicação em questão que será utilizado para encontrar padrões nos dados. Dependendo do objetivo, pode se aplicar diferentes tipos de tarefas de MD, dentre elas, destaca-se (Fayyad *et al.*, 1996; Freitas, 2002):

- **Associação**: A tarefa de associação usada para descobrir padrões que descrevam características associadas aos dados. Os padrões são representados na forma de regras de implicação ou subconjunto de características.
- **Agrupamento**: A tarefa de agrupamento divide os dados em grupos formados por elementos com características semelhantes. Neste tipo de problema, o sistema deve particionar o conjunto de dados em subconjuntos, no intuito de maximizar a semelhança entre os elementos de um mesmo grupo e minimizar as semelhanças entre exemplos pertencentes a grupos diferentes.
- **Classificação**: A tarefa de classificação tenta encontrar, a partir de um conjunto de exemplos, um relacionamento entre os atributos previsores e atributos meta.

Na fase de pós-processamento, é realizada a validação e interpretação dos resultados obtidos. Esta análise, geralmente, é feita pelos usuários que possuem domínio sobre a aplicação.

2.2.1 A Tarefa de Classificação

A tarefa de classificação é a mais comumente utilizada na MD. A classificação, é um tipo de aprendizado supervisionado, é a tarefa que consiste em classificar um item de dado como pertencente a uma determinada classe dentre várias classes previamente definidas

(Freitas, 2002).

A seguir são descritas as principais terminologias que são usadas no contexto de classificação neste trabalho:

- **Instância/registro/exemplo/caso/dado:** descreve o objeto de interesse por meio de um conjunto de valores (vetor) de atributos.
- **Atributo previsor:** descreve uma característica ou um aspecto do objeto de estudo, no qual cada atributo possui um conjunto de valores possíveis. O conjunto de valores pode ser nominal ou contínuo. Exemplos: *gênero = masculino; idade = 26*.
- **Atributo meta:** é o atributo objetivo, no qual possui um conjunto de rótulos (classes), no qual cada registro possui uma classe associada.
- **Regra**: é uma forma de representação que correlaciona atributos. Muitas vezes representada na forma SE <*antecedentes*> ENTÃO <*consequente*>.
- **Termos**: são as condições da regra que compõem a parte do antecedente. Contém uma combinação lógica formada por atributos previsores.
- **Classe**: corresponde a um valor dentre os valores possíveis do atributo-meta.
- **Classificador**: é o algoritmo de aprendizagem que, dado um conjunto de exemplos de treinamento descobre uma relação entre os atributos previsores e o meta, de forma que, dado um novo exemplo, ele possa predizer com um nível de certeza sua classe.
- **Regras de classificação**: é o conjunto de regras gerado por um classificador que descreve os dados de entrada, também chamado de modelo de classificação.
- **Conjunto de treinamento**: dados utilizados como entrada para induzir o classificador a gerar regras de classificação (modelo).
- **Conjunto de teste**: dados utilizados para avaliar a qualidade das regras de classificação.

O principal objetivo da construção de um classificador é descobrir algum tipo de relação entre os atributos previsores e as classes. Segundo Breiman *et al.* (1984) um modelo de classificação extraído de um conjunto de dados serve para dois propósitos: o primeiro é a predição de um valor e entender a relação existente entre os atributos previsores e a classe, e o segundo propósito é exigir que o modelo não apenas classifique, mas também explicite os padrões extraídos de forma compreensível.

As regras geradas, também conhecidas como regras de produção, são representadas na forma SE <*antecedente*> ENTAO <*consequente*>. Como descrito anteriormente, antecedente possui um conjunto de condições composta pelos atributos previsores, e o consequente o atributo-meta.

Um exemplo simples da tarefa de classificação é visto na Figura 2.2 (Witten e Frank, 2005). Na Figura 2.2a cada linha da tabela representa um registro. Cada registro descreve se a partir de determinadas condições do tempo haverá jogo ou não. Cada registro é composto por quatro atributos previsores (*Outlook, Temperature, Humidity*, *Windy)* e um atributo-meta (*Play)*. A Figura 2.2b apresenta regras de classificação que podem, possivelmente, ser geradas por algum algoritmo de classificação.

a) Condições do tempo. Jogar sim ou não?

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

b) Regras de classificação encontradas a partir dos dados de entrada

SE < *Outlook* = *sunny* E *humidity* = *high*> ENTÃO *play* = *no*

SE <*Outlook* = *rainy* E *windy* = *true*> ENTÃO *play* = *no*

SE <*Outlook* = *overcast*> ENTÃO *play* = *yes*

SE <*humidy* = *normal*> ENTÃO *play* = *yes*

Figura 2.2: Exemplo de tarefa de classificação (Witten e Frank, 2005)

Outra forma de ilustrar o processo de um algoritmo de classificação é a representação da Figura 2.3. Nesta figura cada instância é representada por um ponto no espaço bidimensional formado por dois atributos previsores A_1 e A_2. As coordenadas de cada ponto representam os valores de cada atributo. Cada instância é rotulada com uma classe (“+” ou “-

“) indicando a que classe pertence. O procedimento de construção deste classificador é baseado em particionamentos recursivos do espaço de dados. O espaço é dividido em áreas e a cada estágio é avaliado se cada área deve ser dividida em subáreas, a fim de obter uma separação das classes (Freitas, 2002).

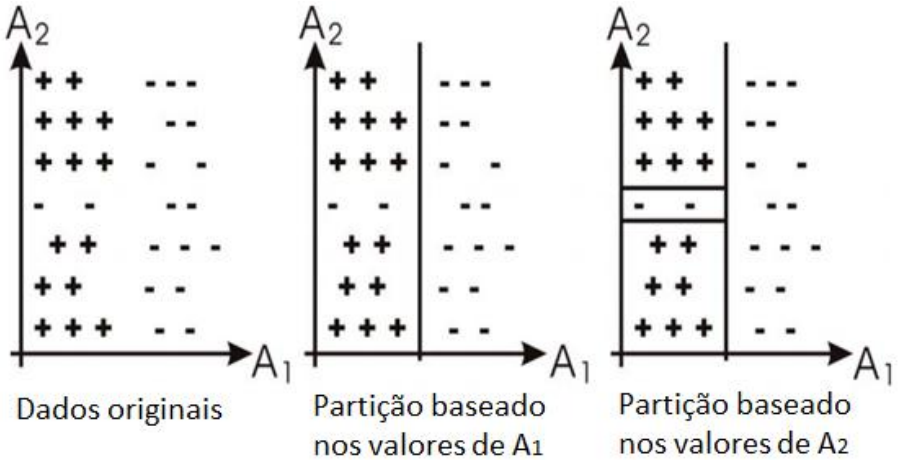


Figura 2.3: Visão bidimensional de um exemplo de classificação (Freitas, 2002)

2.2.2 Critérios para avaliar um modelo de classificação

A Figura 2.4 mostra o processo que o algoritmo executa para chegar até a avaliação, no qual o conjunto de dados está dividido em duas partes: conjunto de treinamento e conjunto de teste. Os métodos mais conhecidos para dividir o conjunto de dados em treinamento e teste são *holdout* e a validação cruzada.

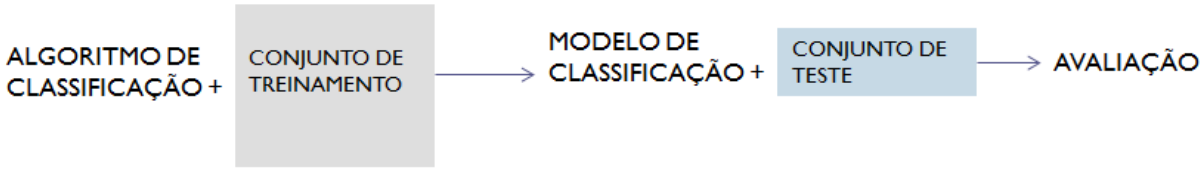


Figura 2.4: Processo executado por um algoritmo indutor de regras

No método *holdout*, os dados são particionados usualmente em 2/3 para treinamento e 1/3 para teste. Este método é mais recomendado quando se tem um conjunto de dados muito grande. No método de validação cruzada todos os dados são utilizados para treinamento e teste. Este método consiste em dividir a base de dados em *k* subconjuntos de mesmo tamanho e executar o algoritmo de classificação *k* vezes, sempre deixando k-1 subconjuntos para treinamento e um para teste. No final realiza-se a média dos *k* resultados.

Geralmente, espera-se que as regras geradas pelo algoritmo classificador sejam corretas, compreensíveis e interessantes para os usuários. Porém, medir a qualidade das regras

às vezes pode não ser uma tarefa fácil, pois alguns critérios podem ser subjetivos, pois dependem do conhecimento prévio dos usuários e seus objetivos com a aplicação (Freitas, 2002; Pappa, 2002).

Medidas de avaliação objetivas

Uma maneira natural de apresentar as estatísticas para a avaliação de um modelo de classificação é por meio de uma tabulação cruzada entre a classe prevista pelo modelo e a classe real dos exemplos. Esta tabulação é chamada de tabela de contingência ou matriz de confusão. Toda informação necessária para avaliar (de forma objetiva) o modelo está contido nesta matriz. A Figura 2.5 descreve uma matriz de confusão 2x2 para uma classificação binária, isto é, que tenha somente duas classes. Sem perda de generalidade, as classes são denominadas como positiva e negativa.

		Valores Reais		
		Positivos	Negativos	
Valores Preditos	Positivos	VP	FP	PPos
	Negativos	FN	VN	PNeg
		Pos	Neg	

Figura 2.5: Matriz de confusão para um modelo de classificação binário

Nessa matriz, *VP*, *FP*, *FN e VN* correspondem, respectivamente, às quantidades de verdadeiros positivos, falsos positivos, falsos negativos e verdadeiros negativos. *Pos* é o total de exemplos positivos, *Neg* é o total de exemplos negativos, *PPos* é o total de exemplos preditos positivos e *PNeg* é o total de exemplos preditos negativos. *VP* e *VN* são as classificações corretas, logo quanto maiores estes valores, melhor foi o desempenho do classificador. A partir desses valores, muitas medidas de avaliação podem ser definidas, tais como:

$$Taxa\ de\ verdadeiros\ positivos = \frac{VP}{VP + FN} = \frac{VP}{Pos} = Recall\ Positivo \quad (2.1)$$

$$Taxa\ de\ verdadeiros\ negativos = \frac{VN}{(VN + FP)} = \frac{VN}{Neg} = Recall\ Negativo \quad (2.2)$$

$$Precisão\ Positiva = \frac{VP}{(VP+FP)} = \frac{VP}{PPos} = Precision \quad (2.3)$$

$$Acurácia\ (taxa\ de\ acerto) = \frac{(VP+VN)}{(Pos+Neg)} \quad (2.4)$$

$$Taxa\ de\ erro = 1 - taxa\ de\ acerto \quad (2.5)$$

A vantagem de reduzir uma matriz de contingência para um único valor é que se torna mais fácil escolher o "melhor" modelo de classificação. Entretanto é comum encontrar casos em que uma medida é mais apropriada/irrelevante que outra em um determinado problema. Na seção 2.3.4 é apresentada em detalhes outra técnica para avaliar modelos de classificação, conhecida como análise ROC.

Compreensibilidade das regras

A principal motivação para descobrir regras de fácil compreensão está no fato de que em MD os resultados são geralmente analisados por seres humanos. Uma melhor interpretação representa uma melhor utilização das regras descobertas. A compreensibilidade das regras geralmente está associada à simplicidade sintática. Logo, quanto maior o número de regras e o número de condições por regras, menos compreensível é o conjunto de regras em questão. Regras com muitas condições nos antecedentes decorrem da complexidade existente entre os atributos previsores e a classe (Freitas, 2002; Carvalho, 2005).

Não se pode dizer que quanto mais compreensível é a regra mais útil ela é. Por exemplo, uma regra simples com apenas um atributo no antecedente pode se tornar muito genérica e óbvia (Parpinelli, 2001).

Interesse nas regras descobertas

É importante também que as regras descobertas sejam avaliadas do ponto de vista do quão interessante elas são para o usuário (Freitas, 2002). Este critério é considerado subjetivo, pois está baseado nos conhecimentos prévios e expectativas do usuário. Porém, existem algumas medidas que tentam estimar esse critério de forma objetiva. Segundo Freitas (2002) e Carvalho (2005), as métricas para medir o interesse do usuário nas regras descobertas são classificadas em: medidas objetivas *(data-driven)* e medidas subjetivas *(user-driven).* O ideal é combinar medidas subjetivas com medidas objetivas, não existe uma medida que possa ser considerada melhor que a outra.

2.2.3 Técnicas utilizadas em MD

Existem algoritmos de MD baseados em várias técnicas existentes como: técnicas estatísticas, árvores de decisão, redes neurais, algoritmos heurísticos, entre outros.

Para a tarefa de classificação, que é o foco neste trabalho, uma técnica muito utilizada é a árvore de decisão. E um dos algoritmos mais conhecido é o C4.5 (Quinlan, 1993), descrito

no Apêndice A.

Algoritmos heurísticos vêm sendo utilizados para resolver as tarefas de MD. Estes algoritmos são inspirados na natureza, alguns deles são inspirados em insetos que vivem em colônias como as formigas, abelhas e cupins. Dentre os algoritmos heurísticos encontram se os algoritmos genéticos (AG) (Holland, 1992; Romão *et al.*, 2004), programação genética (Koza, 1992), *Ant Colony Optimization (ACO)* (Dorigo *et al.*, 1996). A Seção 2.4 descreve o *ACO* em detalhes, pois é a heurística utilizada no desenvolvimento deste trabalho.

2.3 Base de dados desbalanceada

2.3.1 O Problema das bases de dados desbalanceadas

O problema de bases de dados desbalanceadas é uma questão relativamente recente que surgiu quando o aprendizado de máquina evoluiu do seu estado embrionário, puramente científico, para uma tecnologia aplicada, muito usada no mundo dos negócios, indústrias e pesquisas científicas (Machado, 2007). O interesse em estudar bases de dados desbalanceadas aumentou depois da realização de dois workshops, um em 2000 na conferencia AAAI (*Association for the Advancement of Artificial Inteligence)* e outra em 2003 na conferencia ICML (*International Conference on Machine Learning)* e uma edição especial da ACM SIGKDD *Explorations* em 2004 (Japkpowicz, 2000; Chawla, 2003).

Uma base de dados é dita desbalanceada, no domínio de classificação, quando existem muito menos casos de algumas classes do que outras. Em um cenário real, a proporção da classe minoritária para a classe majoritária pode ser drástica tal como 1 para 100, 1 para 1.000, 1 para 10.000 ou mais. Muitas vezes a classe de menor proporção pode ser a de maior interesse (Chawla *et al.*, 2004; Japkowicz & Jo, 2004).

Bases desbalanceadas podem levar a ocorrência de casos raros. Casos raros podem ser vistos como um pequeno número de exemplos de treinamento em áreas especificas do espaço amostral.

Estes desbalanceamentos são propriedades intrínsecas do domínio de muitos problemas como: fraudes em cartões de crédito, diagnósticos médicos, classificação de textos, entre outros. Como exemplo, têm-se as classes fraude e não-fraude no domínio de transações de cartões de crédito. Neste exemplo o desbalanceamento de classes se refere ao número de fraudes serem muito menor do que o de não-fraudes. Um exemplo de casos raros é a

ocorrência de fraudes milionárias. O interesse maior está em descobrir padrões que possam prever a ocorrência de fraudes.

Algoritmos classificadores são muito sensíveis a este tipo de desbalanceamento e tendem a valorizar as classes predominantes e ignorar as classes de menor proporção, assim como os casos raros, pois os algoritmos usam técnicas de indução nas quais tendem a fazer uma generalização dos casos ao invés de uma especialização. Estas técnicas de indução também são conhecidas como "*bias* de indução*",* que significa preferencia por uma hipótese. (Weiss, 2004; Japkowicz & Jo, 2004; Carvalho, 2005).

Casos raros podem levar a ocorrência de pequenos disjuntos. Pequenos disjuntos são regras que possuem baixa cobertura, os quais são conhecidos por serem mais propensos a erros que os grandes disjuntos. Japkowicz & Jo (2004) discutem a relação entre classes desbalanceadas e pequenos disjuntos. Os autores mostram que os pequenos disjuntos são a principal causa do baixo desempenho dos algoritmos classificadores e não a proporção das classes em si. Weiss (2004) também discute esta relação, o autor aponta que as duas formas de raridade (classe rara / casos raros) possuem problemas e soluções similares.

Entretanto, em alguns casos, pequenos disjuntos podem não estar representando casos raros, e sim ruídos. Logo, apenas os pequenos disjuntos mais significativos devem ser considerados. Muitos classificadores tentam prevenir o *overfitting*, consequentemente não aprendem os pequenos disjuntos (Weiss, 2004).

Os modelos resultantes de classificadores com bases de dados desbalanceadas apresentam baixo poder preditivo para a classe rara (tratada como a classe positiva), isto é, apresentam altas taxas de falsos positivos (FP) em suas regras, o que é problemático quando a classe de interesse é justamente a classe de menor proporção. Segundo Liu *et al.* (2008) classes desbalanceadas devem ser cuidadosamente tratadas para se construir um bom classificador.

2.3.2 Soluções para base de dados desbalanceadas

Existem soluções em nível de pré-processamento e em nível de algoritmo. As soluções em nível de pré-processamento visam diminuir o desbalanceamento das classes e são conhecidos como métodos de amostragem. Em nível de algoritmo existem métodos tais como: (1) ajustar o custo de aprender uma determinada classe (conhecidos como, *cost-sensitive learning),* (2) aprender apenas a classe de interesse (conhecidos como, *recognition-based),* (3) ajuste de *bias.* (Chawla *et al.*, 2004; Weiss 2004; Japkowicz & Jo, 2004).

Ajuste de *bias* do classificador é usado muitas vezes na tentativa de fazer o algoritmo encontrar regras para os pequenos disjuntos. Uma *bias* de indução de especialidade pode ser mais adequada neste caso. Porém, com uma *bias* de especialização, o algoritmo pode encontrar regras muito ajustadas, o que degrada o poder preditivo do classificador nos casos de teste.

Weiss (2004) relata que o método *cost sensitive learning* apresenta resultados inferiores que a maioria dos métodos de amostragem. Já o método de aprender apenas uma classe (métodos baseado em reconhecimento), pode ser benéfico em certos domínios. Dentre as técnicas que aprende apenas a classe de interesse, um que apresenta bom desempenho é o SVM (*Support Vector Machine)* (Yan *et al.,* 2003). Para classificar um novo caso basta calcular o nível de similaridade entre ele e a classe alvo e depois compara-lo com o limiar de similaridade adotado.

2.3.3 Métodos de amostragem para balanceamento de dados

Os métodos de amostragem visam mudar a distribuição dos dados do conjunto de treinamento, de modo a aumentar o poder preditivo de seus modelos. Uma nova amostragem pode ser obtida com a eliminação de casos da classe majoritária (conhecido como *undersampling)* ou replicação/adição de casos da classe minoritária (conhecido como *oversamping)* (Chawla *et al.*, 2004). Existem dois tipos de métodos de amostragem: método simples e métodos avançados.

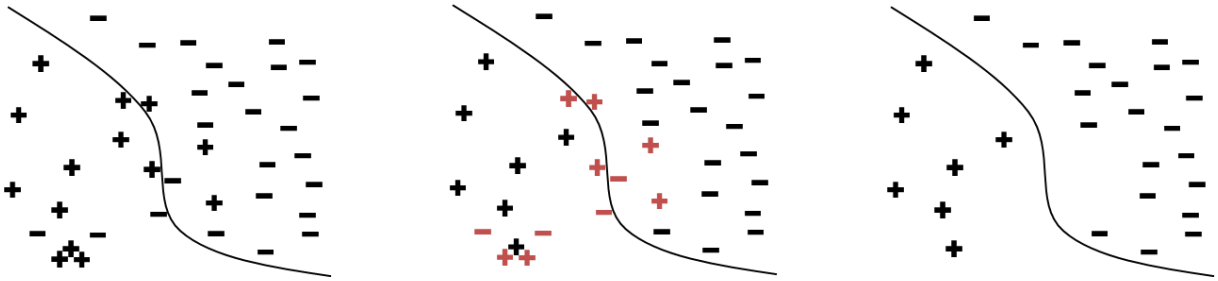
Os métodos de amostragem simples não utilizam uma heurística na eliminação e/ou adição de casos, simplesmente visam obter uma nova distribuição das classes de forma aleatória (*undersampling* aleatório e *oversampling* aleatório), porém os métodos simples possuem alguns problemas. O *undersampling* aleatório pode descartar casos da classe majoritária que podem ser importantes e prejudicar o desempenho do algoritmo. O *oversampling* aleatório faz cópias exatas dos casos da classe rara o que pode aumentar as chances de ocorrer *overfitting* (Weiss, 2004).

Os métodos de amostragem avançados utilizam alguma heurística para remover/adicionar casos. A seguir são descritas algumas destas técnicas:

Undersampling strategy

A técnica *undersampling strategy* é derivada da técnica de limpeza de dados, a qual elimina casos redundantes, ruidosos e limítrofes (*boderlines)* de todas as classes. A ideia de

realizar um *undersampling strategy* é eliminar apenas casos da classe majoritária que podem ser considerados redundantes, ruídos ou limítrofes e deixar apenas os casos seguros. Estas quatro categorias de casos podem ser representados em pontos no espaço R^n, no qual *n* é a quantidade de atributos da base de dados. Os casos redundantes são aqueles que podem ser representados por seus vizinhos mais próximos sem perda de expressividade para o classificador. Os casos ruidosos são aqueles que se encontram no lado errado do limiar de decisão com relação à classe, isto é, diverge da classe de seus vizinhos mais próximos. Os casos limítrofes são aqueles que se encontram próximos ao limiar de decisão. A presença de casos ruidosos quanto à classe, limítrofes e redundantes podem vir a degradar o desempenho dos classificadores (Kubat & Matwin, 1997; Weiss, 2004; Liu *et al.*, 2008).



(a) (b) (c)

Figura 2.6: Eliminação de casos redundantes, ruidosos e limítrofes

A Figura 2.6 apresenta o que é chamado de limpeza de dados. A Figura 2.6(a) apresenta as quatro categorias de casos juntamente com o limiar de decisão, a Figura 2.6(b) apresenta os casos redundantes, ruidosos e limítrofes destacados em vermelho, e a Figura 2.6(c) apresenta o conjunto de dados com a eliminação dos casos destacados, resultando apenas nos casos seguros.

CNN (*Condensed Nearest Neighbor Rule)*

Kubat & Matwin (1997) usam uma variante do método CNN (Hart, 1968) para fazer *undersampling* na classe majoritária. O método consiste em criar um subconjunto *C* a partir de um conjunto de dados *S*. Este método usa primeiramente o algoritmo de classificação *k-NN* (*Nearest Neighbor)* (Apêndice A), que é um classificador que não gera modelo e sim classifica novos casos de forma direta, isto é, um novo caso é associado a uma classe de acordo com os seus *k* vizinhos mais próximos contidos no conjunto de treinamento. A ideia do método CNN é fazer *undersampling* apenas dos casos da classe majoritária retirando os

exemplos redundantes.

Primeiramente o algoritmo cria um subconjunto *C* com todos os casos da classe minoritária e apenas um caso da classe majoritária, os quais são retirados do conjunto *S*. Em seguida, é selecionado um por um dos casos restantes contidos em *S* e classificados com o algoritmo *k-NN* (com $k = 1$) com o conjunto *C* como treinamento. Se o caso de S tiver uma classe diferente da classe c_i retornada pelo k-NN então o caso é retirado de *S* e inserido em *C*. Ao final, o subconjunto *C* terá todos os casos da classe minoritária e apenas os casos mais relevantes da classe majoritária.

Uma variação do CNN é o ENN (do inglês, *Edited Nearest Neigbor Rule)* (Batista et al, 2004), que de forma contrária ao CNN, faz *undersampling* adicionando ao conjunto *C* apenas os casos do conjunto *S* cuja classe coincidir com a classe prevista pelos *k* vizinhos mais próximos retornado pelo *k-NN*. Este processo remove tanto os casos ruidosos quanto os casos limítrofes de forma a propiciar um limiar de decisão mais suave.

Tomek links

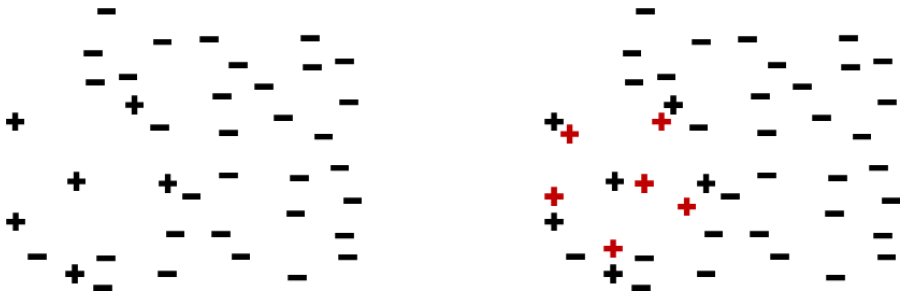
Kubat Matwin (1997) também citam outra técnica para fazer *undersampling*, técnica conhecida como *Tomek links.* Sejam *x* e *y* dois casos de classes diferentes. A distância entre estes dois casos é denotado por *d(x,y).* O par *(x,y*) é chamado de *Tomek link* se nenhum exemplo *z* existir tal que $d(x,z) < d(x,y)$ ou $d(y,z) < d(y,x)$. Se um par *(x,y)* é considerado *Tomek link*, um dos casos é ruído ou os dois casos são limítrofes. Os pares *Tomek links* podem ser utilizados tanto para fazer *undersampling* da classe majoritária (remover ruído) ou ser usado para fazer limpeza em torno do limiar de decisão.

SMOTE (*Synthetic Minority Oversampling Technique*)

Apenas replicar casos já existentes da classe minoritária realmente aumenta o viés do classificador para esta classe. Entretanto, podem ocorrer modelos muito específicos para estes casos, caracterizando o *overfitting*, prejudicando o poder de generalização do modelo. Segundo Chawla *et al.* (2002), classes raras são acompanhadas de casos raros, e estes casos são circundados de casos negativos (casos que possuem outras classes), e o simples ato de replicar um caso raro faz com que a região de decisão seja muito pequena, e isso não ajudará classificar corretamente novos casos da classe rara que venham a cair nas vizinhanças da região.

Dessa forma, Chawla *et al.* (2002) desenvolveram a técnica SMOTE para realizar *oversampling.* Esta técnica adiciona novos casos para a classe rara ao invés de serem

replicados, isto é, são gerados casos sintéticos para a classe minoritária a partir dos casos já existentes. Estes novos casos são gerados na vizinhança de cada caso da classe minoritária. No espaço amostral estes novos casos serão interpolados aleatoriamente ao longo do segmento de reta que liga cada caso da classe minoritária a um dos seus *k* vizinhos mais próximos, escolhidos de forma aleatória. A Figura 2.7(a), apresenta um conjunto de casos em um espaço amostral no qual possui menos casos para a classe "+", e a Figura 2.7(b) apresenta destacado em vermelho os novos casos que a técnica SMOTE adicionou.



(a) (b)

Figura 2.7: *Oversampling* com técnica SMOTE

Esta técnica pode aumentar a generalização dos modelos, ao contrário de uma especialização se apenas replicassem os casos (Weiss, 2004).

Cluster-based Oversampling

Como mencionado, os conjuntos de dados podem possuir casos raros, o que levam a formação de pequenos disjuntos. De acordo com Weiss (2004) os casos raros não são facilmente identificáveis, mas métodos de aprendizagem não-supervisionada, como a clusterização, ajudam na identificação destes casos, sendo que eles podem se apresentar como pequenos *clusters*[1] dentro de uma classe. Nickerson *et al.* (2001) propuseram uma técnica para minimizar este problema de casos raros. Os autores fazem um *oversampling* não somente da classe minoritária, mas também dos casos raros. A ideia é agrupar os dados de treinamento em *clusters* e, então, balancear a distribuição de seus casos. Primeiramente, utiliza-se o algoritmo *K-means* (Apêndice A), que é uma técnica não supervisionada de clusterização para formar os *clusters* de cada classe, e depois faz *oversampling* de cada um destes *clusters* (aumentando a proporção dos pequenos disjuntos). Em seguida é feito novamente um

[1] *Clusters:* Agrupamento de dados que compartilham semelhanças entre si.

oversampling, agora com relação à classe minoritária, até atingir o tamanho da classe majoritária. No final, haverá um conjunto totalmente equilibrado, isto é, com todas as classes e seus *clusters* possuindo a mesma quantidade de casos.

Podem-se usar técnicas de *undersampling e oversampling* ao mesmo tempo, muitos autores propõem algoritmos que combinam as duas técnicas. Outros métodos conhecidos: OSS (*One-sided Selection)* (Kubat & Martwin, 1997), SMOTE + *Tomek links*, SMOTE + ENN (Batista *et al.*, 2004).

2.3.4 Avaliação de algoritmos utilizando análise ROC

Suponha uma base de dados *T* com a seguinte distribuição de classes *distr*(*C1*,*C2*,*C3*) = (98%, 1,25%, 0,75%). Um classificador simples que classifique sempre novos exemplos como pertencentes à classe *C1* teria uma precisão preditiva de 98%, porém é difícil dizer o quão bom é este classificador devido à má distribuição da base. A maioria das medidas de avaliação são altamente suscetíveis ao desbalanceamento das classes, o que pode conduzir a uma conclusão errônea sobre o desempenho do algoritmo classificador.

A análise ROC, também conhecida como gráfico ROC (do inglês, *Receiver Operating Characteristics),* é um método gráfico para avaliação, organização e seleção de sistemas de diagnóstico e/ou predição (Fawcett, 2004). Recentemente, a análise ROC foi introduzida em Aprendizagem de Máquina (AM) e em MD como uma ferramenta útil e poderosa para avaliação de modelos de classificação. A análise ROC é capaz de realizar avaliações mais apuradas de algoritmos de aprendizagem, e é particularmente útil em domínios nos quais existe uma grande desproporção entre as classes (Prati *et al.*, 2008).

O gráfico ROC é baseado em duas medidas: taxa de verdadeiros positivos T(VP) (expressa a qualidade da classificação da classe positiva); e taxa de falsos positivos T(FP) (expressa a qualidade de classificação da classe negativa em termos da taxa de erro). Com as equações 2.6 e 2.7, os valores de T(VP) e T(FP) são calculados a partir da matriz de confusão binária gerada pelo modelo de classificação, na qual a classe de interesse é denotada como classe positiva.

$$\text{T(VP)}=\frac{positivos\ classificados\ corretamente}{total\ de\ positivos}=\frac{VP}{Pos} \tag{2.6}$$

$$T(FP)=\frac{negativos\ classificados\ incorretamente}{total\ de\ negativos}=\frac{FP}{Neg} \quad (2.7)$$

Os valores de T(VP) e T(FP) variam no intervalo de [0,1]. Para se construir o gráfico ROC plota-se T(FP) no eixo das abcissas e T(VP) no eixo das ordenadas. Um modelo de classificação é representado por um ponto no espaço ROC. A Figura 2.8 caracteriza um gráfico ROC com cinco pontos arbitrários representando cinco modelos de classificação diferentes (A, B, C, D e E), para um conjunto de dados também arbitrário.

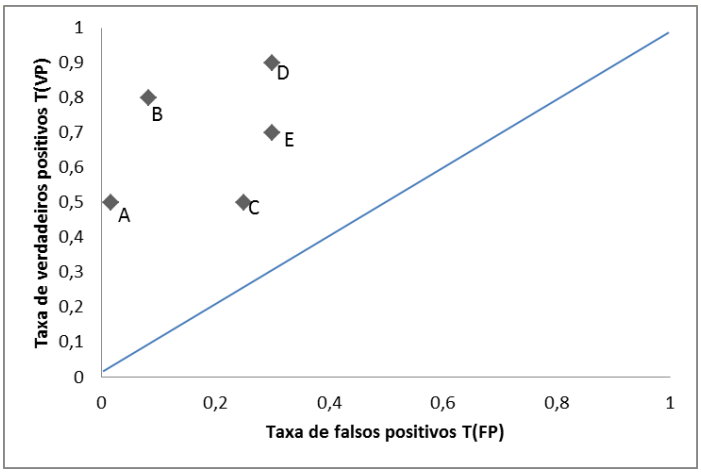


Figura 2.8: Modelos de classificação no espaço ROC

Alguns pontos no espaço ROC merecem destaque. O ponto (0,0) representa a estratégia de nunca classificar um exemplo como positivo, modelos mais próximos a esta região são considerados "conservativos": só fazem uma classificação positiva somente se têm grande segurança na classificação. A estratégia inversa é o ponto (1,1) que classifica um novo exemplo sempre com a classe positiva, modelos mais próximos a esta região são considerados "liberais": eles predizem a classe positiva com maior frequência, porém com alta taxa de falsos positivos. A linha diagonal (0,0) – (1,1) representa modelos de comportamento estocástico (aleatório), pontos acima da diagonal principal são melhores que os abaixo da diagonal principal (Fawcett, 2004; Prati *et al.*, 2008).

O ponto (0,1) é a classificação perfeita, isto é, todos os exemplos das classes positivas e negativas são corretamente classificados. Dessa forma, modelos mais próximos do canto superior esquerdo do gráfico predizem a classe positiva com maior frequência e com baixas taxas de falsos positivos, isto é, estão mais próximos da classificação perfeita (Prati *et al.*,

2008).

Ainda na Figura 2.8, o ponto A é considerado o mais "conservativo": baixa taxa de verdadeiros positivos, porém comete pouco erro falso positivo. O ponto D é considerado o mais "liberal": prediz a classe positiva com maior frequência, porém com alta taxa de falsos positivos. O ponto C é considerado o mais aleatório, pois é o ponto mais próximo da diagonal principal do gráfico.

É possível mostrar que os modelos que se encontram no fecho convexo (Figura 2.9), que mais se aproximam do ponto (0,1), são os modelos que são potencialmente ótimos. A figura mostra também uma linha de isodesempenho, neste caso ela é paralela a diagonal principal, indicando que o custo de classificar erradamente um exemplo como positivo ou negativo é o mesmo, logo B seria a melhor classificação (Fawcett, 2004; Prati *et al.*, 2008).

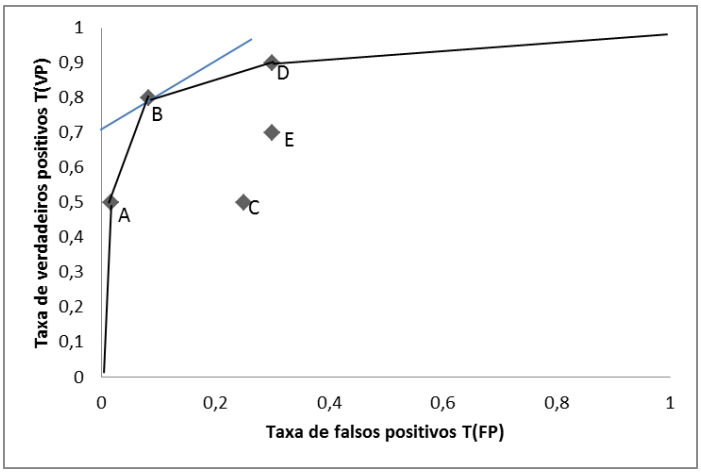


Figura 2.9: Fecho convexo de um gráfico ROC e linha de isodesempenho (adaptado de Prati *et al*., 2008)

Outra vantagem de se utilizar a análise ROC está na avaliação de diferentes limiares, em classificadores *score*, que são os classificadores que não predizem uma classe, mas sim um valor contínuo ou ordinal (por exemplo: *Naive Bayes)*. Ao invés de se escolher um limiar arbitrário e representar o desempenho do sistema para um determinado domínio como um único ponto no espaço ROC, pode-se "simular" a escolha de vários limiares. Varia-se o limiar em todo o seu espectro, desde o valor mais restritivo até o valor mais liberal. Desta maneira, a análise é feita independente da escolha dos limiares. O desempenho do classificador é então representado por uma curva no espaço ROC. A maneira mais eficiente de gerar essa curva é ordenar todos os casos de teste de acordo com o valor contínuo predito pelo modelo. A partir desse conjunto ordenado, para cada caso desse conjunto, dá-se um passo de tamanho 1/Pos na

direção do eixo *y* se o exemplo for positivo ou um passo de tamanho 1/Neg caso o exemplo seja negativo (Prati *et al.*, 2008).

Quanto mais distante a curva estiver da diagonal principal, melhor será o desempenho do classificador para aquele domínio. Ao se comparar duas (ou mais) curvas, caso não haja intersecção, a curva que mais se aproxima do ponto (0,1) é a de melhor desempenho. A Figura 2.10 apresenta um exemplo de curvas ROC. Neste caso, o classificador 1 está mais próximo do ponto (0,1), logo é melhor que o classificador 2.

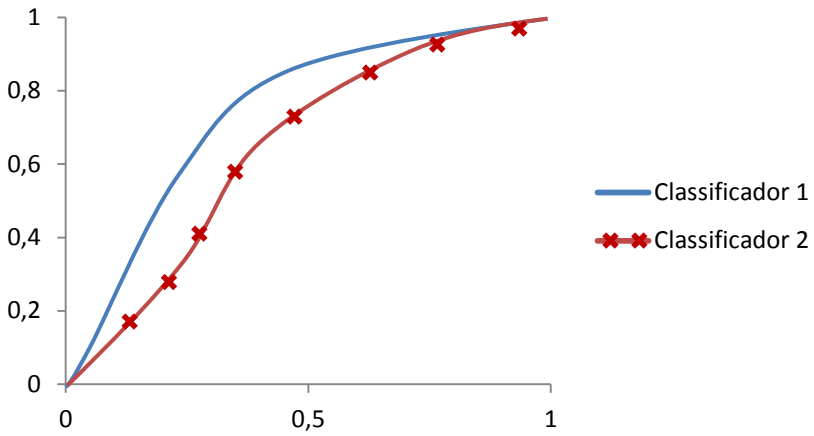


Figura 2.10: Exemplo de curvas ROC (adaptado de Prati *et al.*, 2008)

Outra conexão entre a curva ROC e os diferentes limiares é a área abaixo da curva ROC, chamada AUC (do inglês, *Area Under Curve*). Uma vez que a área abaixo da curva ROC é uma fração da área de um quadrado de lado 1, o seu valor esta sempre entre 0 e 1. A AUC vem gradativamente ganhando espaço como medida de avaliação de modelos em MD, ela contém menos deficiências do que a acurácia (taxa de acerto de classificação) (Pratti *et al.*, 2008).

Huang *et al.* (2003) demonstraram que AUC é, em geral, uma medida melhor que a taxa de acerto; e também afirmam que muitos algoritmos de MD populares deveriam ser reavaliados em termos de AUC.

Entretanto uma desvantagem de se utilizar gráficos ROC é a sua limitação para apenas duas classes. Pois o número de eixos cresce exponencialmente ao número de classes, o que impede a análise. No entanto é possível reduzir a análise para duas classes, fazendo uma-contra-todos (Pratti *et al.*, 2008).

2.4 Otimização por Colônias de Formigas

A meta-heurística otimização por colônias de formigas (do inglês, *Ant Colony Optimization – ACO)* (Dorigo *et al.*, 1996; Dorigo *et al.*, 1999) é um subcampo da *Swarm Intelligence Algorithms,* que é o estudo de algoritmos inspirados no comportamento de insetos sociais. Inicialmente Dorigo *et al.,* (1996) desenvolveu o algoritmo *Ant System,* para a obtenção de soluções de problemas de otimização combinatória, tendo como sua primeira aplicação o problema do caixeiro viajante. Após esta aplicação, muitas outras variações foram elaboradas a fim de resolver diversos tipos de problemas, por isso o nome meta-heurística, pois pode ser adaptada pra resolver um determinado problema.

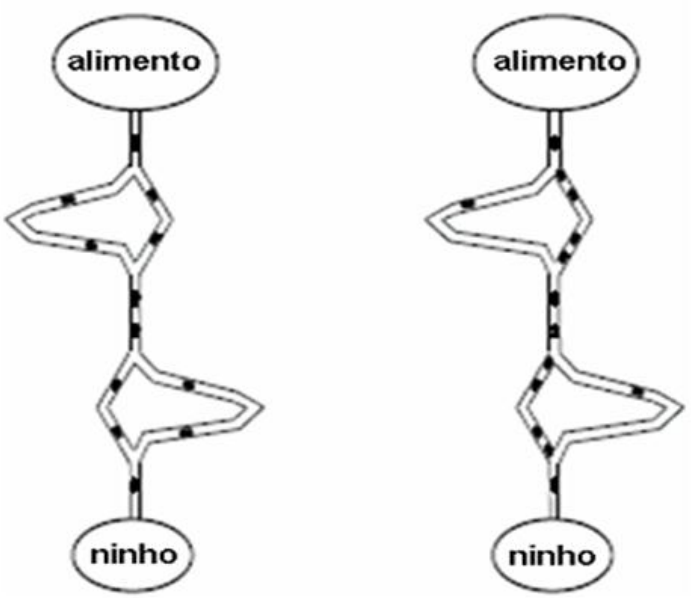
2.4.1 Inspiração nos insetos sociais

Insetos que vivem em colônias, tais como: formigas, abelhas, vespas e cupins; são individualmente limitados, porém quando atuam em conjunto, eles são capazes de solucionar problemas complexos do cotidiano utilizando a cooperação mútua. Uma colônia de insetos é um sistema descentralizado que soluciona problemas como selecionar e coletar materiais, encontrar e estocar comida. Este comportamento coletivo dos insetos sociais tem sido chamado de Inteligência de Enxames (*Swarm Inteligence)* (Dorigo *et al.*, 1999).

Segundo Parpinelli (2001), as principais características do sistema descentralizado dos insetos sociais são a flexibilidade e a robustez com os quais solucionam os problemas: a flexibilidade permite a adaptação da colônia às mudanças do ambiente, enquanto que a robustez permite que a tarefa em andamento seja concluída mesmo com uma possível falha de alguns indivíduos.

Muitas das atividades coletivas realizadas pelos insetos sociais são auto-organizáveis, pois, na maioria das vezes, os insetos se interagem de forma indireta com o uso de substâncias químicas. Isto é, dois indivíduos interagem indiretamente quando um deles modifica o ambiente e o outro responde às novas características do ambiente em um instante de tempo seguinte. Um sistema auto-organizável possui mecanismos que condicionam o surgimento de padrões a nível global a partir da interação entre os componentes que o constituem. Estes componentes interagem entre si tendo como base apenas uma informação local, sem nenhuma referência ao padrão global a ser encontrado, como por exemplo, o comportamento global (melhor rota) das formigas que buscam por alimento e que emerge a partir de trilhas de

feromônio[2] (informação local) (Dorigo *et al.*, 1999).



(a) (b)

Figura 2.11: Experiência da ponte dupla (Dorigo *et al.*, 1999)

A Figura 2.11 mostra o comportamento que uma colônia de formigas tem ao realizar o trajeto do ninho-alimento-ninho. Inicialmente todos os caminhos não contém resíduo de feromônios (Figura 2.11 (a)), deste modo, todos os caminhos (desde o mais curto ao mais longo) tem a mesma probabilidade de serem escolhidos pelas formigas. Após um tempo *t*, algumas formigas já fizeram o trajeto ninho-alimento-ninho mais rápido do que outras, depositando mais feromônio rapidamente. Logo, assim que outra formiga precisa tomar a decisão de qual caminho escolher, a escolha será tendenciosa aos caminhos que tiverem maior quantidade de feromônio depositados. Outra característica é que, com o passar do tempo, este feromônio evapora, ou seja, caso algum caminho esteja sendo menos utilizado, sua quantidade de feromônio decresce. Consequentemente, com o passar do tempo, todas as formigas convergem para um mesmo caminho (caminho mais curto) (Figura 2.11 (b)), obtendo a solução global.

[2] Feromônio: substância química utilizada como meio de comunicação indireta entre indivíduos da mesma espécie.

2.4.2 Colônias de formigas artificiais

Assim como nas colônias reais, um algoritmo *ACO* é composto por uma população de formigas artificiais que cooperam globalmente entre si para encontrar uma boa solução. Esta cooperação surge a partir da informação que eles concorrentemente leem/escrevem nos "estados do problema". Esta informação, assim como nas colônias reais, é chamada de feromônio.

Enquanto as formigas reais depositam uma substancia química, as formigas artificias modificam alguma informação numérica armazenada localmente no estado do problema que estão visitando. Por analogia, esta informação numérica local é chamada de trilha de feromônio. A quantidade de feromônio depositada em uma trilha é feita em função da qualidade da solução encontrada, isto é, as formigas artificiais apenas atualizam as trilhas de feromônios depois de terem gerado uma solução. Um mecanismo de evaporação do feromônio permite que as formigas artificiais aos poucos esqueçam o seu passado, podendo direcionar sua busca para novas soluções sem a influência das decisões tomadas tempos atrás.

Assim como as formigas reais, as formigas artificiais têm preferência probabilística por caminhos com maior quantidade de feromônio. Logo, quando uma formiga artificial tiver de escolher entre dois ou mais caminhos, o caminho (solução) que tiver sido mais frequentemente escolhido por outras formigas no passado terá maior probabilidade de ser escolhido por esta formiga atual. Dessa forma, as formigas artificiais vão reforçando as melhores soluções causando uma convergência.

Existem algumas características nas formigas artificiais que as reais não possuem. Nas colônias artificiais o tempo é discreto, as formigas podem possuir alguma memória (*e. g.* para armazenar o caminho de volta) e não utilizam apenas o feromônio para realizar a escolha dos caminhos, as formigas artificiais usam também outro fator (*e.g.:* informação heurística) o qual influencia a probabilidade de decisão (Dorigo *et al.*, 1999; Parpinelli, 2001).

Dorigo *et al.* (1996) descreveram uma representação genérica do *ACO*, no qual deve ser modelado conforme o problema que se queira resolver. Para implementar um *ACO*, primeiro deve-se definir:

- Representação apropriada do problema: permite que as formigas de forma incremental construam ou modifiquem soluções a partir do uso de uma regra probabilística de transição baseada na quantidade de feromônio (τ) na trilha e em uma heurística local (*η);*

- Método para forçar a construção de soluções válidas: diz respeito a restrições que o problema pode ter;
- Uma função heurística (η): mede a qualidade dos itens que podem ser adicionados à solução parcial atual;
- Uma regra para atualizar o feromônio (τ): especifica como modificar a quantidade de feromônio (τ) correspondente a cada item;
- Uma regra probabilística de transição: esta transição é baseada no valor da função heurística (η) e no valor do feromônio (τ) correspondente a cada item.

2.5 Ant-Miner

Esta seção descreve o algoritmo *Ant-Miner (Ant Colony-based Data Miner)* proposto por Parpinelli (2001). O *Ant-Miner* é um algoritmo de MD para resolver a tarefa de classificação. Como visto na Seção 2.2, o intuito de um algoritmo indutor de regras é descobrir padrões (regras de classificação) a partir de um conjunto de dados. O algoritmo é baseado na meta-heurística *ACO*, no qual cada formiga incrementalmente constrói/modifica uma solução para determinado problema alvo, neste caso, o problema alvo é a descoberta das regras de classificação.

As regras descobertas são do tipo SE <*termo1* E *termo2* E ...> ENTÃO <c*lasse*>. Cada *termo* é composto por uma tupla <*atributo, operador, valor*>, no qual *atributo* é um dos possíveis atributos antecedentes, o *operador* é um operador relacional (*e.g.:* operador de igualdade "="), e *valor* é um dos valores possíveis deste atributo específico, por exemplo, <*gênero=feminino*>. A parte <*classe*> é o consequente da regra e contém a classe prevista, composto por uma tupla <*atributo_meta, operador, valor*>. Cada valor possível do *atributo_meta* é chamado de classe. O *Ant-Miner* segue uma sequência de passos para descobrir uma lista ordenada de regras que cubra todos os casos do conjunto de treinamento. A Figura 2.12 descreve o pseudocódigo do algoritmo.

Inicio
ConjuntoTreinamento = todos os casos do conjunto de treinamento;
ListaDeRegras = {};
Calcule o valor heurístico para cada termo <atributo = valor>;
Enquanto (ConjuntoTreinamento > Max_casos_n_cobertos)
    i = 1; // índice de cada formiga.
    j = 1; // índice do teste de convergência.
    Inicialize o feromônio de cada termo com a mesma quantidade;
    Repita
        Ant_i começa com uma regra vazia e incrementalmente constrói uma regra de classificação R_i, adicionando um termo por vez;
        Pode a regra R_i;
        Atualize o feromônio de todos os termos. Os termos contidos em Ant_i são incrementados (proporcional à qualidade da regra R_i) e o feromônio dos outros termos são decrementados;
        Se (R_i é igual a R_{i-1})
            Então j = j +1; // incrementa teste de convergência.
            Senão j = 1;
        Fim Se
        i = i + 1;
    Até (i >= Num_formigas) **ou** (j >= Num_regras_converg)
    Escolha a melhor regra R_m dentre todas as regras R_i construídas pelas formigas;
    Adicione a regra R_m na ListaDeRegras;
    ConjuntoTreinamento = ConjuntoTreinamento – {conjunto de casos cobertos pela regra R_m};
Fim Enquanto
Adicione a regra padrão na ultima posição da ListaDeRegras;

Figura 2.12: Pseudocódigo do algoritmo *Ant-Miner* (Parpinelli *et al., 2002)*

Inicialmente a ListaDeRegras é uma lista vazia, o conjunto de treinamento possui todos os casos para treinamento e todos os termos são inicializados com sua informação heurística. No inicio do laço **Enquanto**, todos os termos recebem a mesma quantidade de feromônio. Cada iteração do laço **Enquanto** corresponde a uma regra descoberta, a qual foi considerada a melhor regra dentre as *n* regras candidatas construídas pelo laço interno **Repita-até**. Cada regra descoberta por este laço interno **Repita-até** corresponde a uma formiga. Logo, a cada iteração do laço **Enquanto**, a melhor regra[3] (melhor formiga) é selecionada e adicionada a ListaDeRegras e os casos de treinamento que são corretamente cobertos por esta regra são removidos do conjunto de treinamento. Casos corretamente cobertos significa que

[3] Melhor regra: por analogia com as formigas reais, a melhor regra corresponde à formiga que obteve o melhor “caminho”.

satisfazem os antecedentes da regra e têm a classe prevista pelo consequente. Esta iteração, de descobrir uma nova regra e remover os casos cobertos por ela, é executada enquanto o número de casos de treinamento for maior que um limiar (parâmetro) especificado pelo usuário, chamado de *max_casos_n_cobertos*.

Detalhando o laço **Repita-até** tem-se, para cada iteração deste laço, três passos: a construção de uma regra Ant_i, a poda da regra e a atualização do feromônio. Na construção da regra (Seção 2.5.3), cada formiga Ant_i inicialmente está vazia, isto é, com nenhum termo no antecedente, e adiciona um termo por vez à sua regra parcial, no qual cada termo adicionado é escolhido de forma probabilística, por analogia as formigas reais, correspondendo à escolha de uma direção para o caminho. Esta escolha probabilística depende da função heurística dependente do problema (η) (Seção 2.5.2) e da quantidade de feromônio (τ) associada a cada termo.

A formiga Ant_i interrompe a adição de termos a sua regra em duas situações:

- Quando a adição de um novo termo fizer com que a regra cubra um número de casos menor que um limiar especificado pelo usuário (*min_casos_por_regra)*;
- Quando todos os atributos já tiverem sido usados pela formiga.

Uma restrição importante é que cada atributo pode aparecer apenas uma vez em cada regra, pois regras como "SE (<*sexo=feminino*> E <*sexo=masculino*> E ...)" são inválidas.

Ao término da construção da regra Ant_i, o segundo passo é a poda da regra (Seção 2.5.4). Esta poda auxilia a remoção de termos irrelevantes que podem ter sido adicionados desnecessariamente na regra.

Após a poda, o terceiro passo é a atualização do feromônio dos termos usados pela regra (Seção 2.5.5). Esta atualização é feita de acordo com a qualidade da regra da Ant_i (seção 2.5.4). Desta forma, as próximas formigas irão construir suas regras usando as novas quantidades de feromônio para guiar sua busca (escolha de termos). A execução destes passos do laço interno **Repita-até** é interrompido quando umas das duas seguintes condições são alcançadas:

- O número de regras candidatas construídas for maior ou igual a um limiar especificado pelo usuário. Este limiar (*num_formigas*) corresponde à população de formigas, isto é, o número máximo de iterações para encontrar a melhor regra a cada laço **Enquanto**;

- Quando a formiga atual (Ant_i) tiver construído uma regra exatamente igual a um limiar de regras consecutivas (teste de convergência). Especificado pelo parâmetro *num_regras_converg*. Corresponde às formigas reais convergirem para um mesmo caminho.

Com o término do laço **Repita-até**, é feita a escolha da melhor regra dentre as regras construídas pelas formigas (de acordo com um critério de qualidade, Seção 2.5.4) e é adicionada à lista ordenada de regras descobertas, e os casos corretamente cobertos por esta regra são removidos do conjunto de treinamento. O algoritmo inicia uma nova iteração do laço **Enquanto**, reiniciando o feromônio de todos os termos e agora com um conjunto de treinamento reduzido. Logo, o laço **Enquanto** é chamado novamente com um novo conjunto de treinamento. O processo é repetido até encontrar um número de regras que cubra um limiar de casos no conjunto de treinamento.

Ao término do laço **Enquanto**, uma regra padrão é adicionada ao final da lista de regras descobertas. Esta regra padrão tem parte do antecedente vazio e tem como consequente a classe de maior frequência dos casos de treinamento que não foram cobertos por nenhuma regra.

Com a lista ordenada de regras pronta, o algoritmo pode aplica-las ao conjunto de teste. A primeira regra que cobrir o novo caso é aplicada, isto é, o novo caso recebe a classe da regra que o cobriu.

2.5.1 Inicialização do feromônio

Já foi descrito que um termo é composto por <*atributo, operador, valor*>. Para identificar todos os termos possíveis da base de dados em questão, considere $termo_{ij}$ como sendo na forma $A_i=V_{ij}$, no qual A_i é o *i-ésimo* atributo e V_{ij} é o *j-ésimo* valor do domínio de A_i.

Inicialmente todos os termos possuem a mesma quantidade de feromônio $\tau_{ij}(t=0)$, e este valor é inversamente proporcional ao número total de termos, definido pela Equação 2.8:

$$\tau_{ij}(t=0) = \frac{1}{\sum_{i=1}^{a} bi} \tag{2.8}$$

Onde:

- a é o número total de atributos;
- b_i é o número total de valores do domínio do atributo i.

2.5.2 Valor heurístico

Cada *termo*$_{ij}$, além de possuir um valor de feromônio associado, possui também um valor heurístico que é sempre o mesmo para um determinado *termo*$_{ij}$, isto é, diferente do feromônio, o valor heurístico não muda no decorrer da execução.

Esta função heurística η_{ij} é uma estimativa da qualidade associada a cada termo e tem a habilidade de melhorar a escolha de termos que devem ser adicionados a regra. Nas formigas reais isto corresponde a uma percepção entre dois caminhos, qual é o mais curto (visão local). De acordo com Parpinelli (2001), esta função heurística é baseada na Teoria da Informação no qual envolve a medida de entropia[4] associada com cada *termo*$_{ij}$. Liu *et al.* (2002) apresentou uma forma mais fácil de calcular o valor de η_{ij} (Equação 2.9). A ideia é que o algoritmo não necessite de um valor preciso para η_{ij} desde que o feromônio (τ_{ij}) compense os pequenos erros dos η_{ij}.

$$\eta_{ij} = \frac{|majority\_classT_{ij}|}{|T_{ij}|} \tag{2.9}$$

Onde:

- T_{ij} é a quantidade de vezes que o *termo*$_{ij}$ aparece no conjunto de treinamento;
- $majority\_classT_{ij}$ é a quantidade de vezes da ocorrência da classe majoritária para o *termo*$_{ij}$ no conjunto de treinamento.

2.5.3 Construção das regras

A probabilidade do *termo*$_{ij}$ ser adicionado na regra parcial atual de uma formiga é dada pela Equação 2.10.

$$P_{ij} = \frac{\tau_{ij}(t) \times \eta_{ij}}{\sum_i^a \sum_j^{bi} \tau_{ij}(t) \times \eta_{ij}, \forall\, i \in I} \tag{2.10}$$

Onde:

- η_{ij} é o valor da função heurística do *termo*$_{ij}$. Quanto maior o valor de η_{ij}, maior a relevância do *termo*$_{ij}$ para a classificação e maior é a probabilidade deste ser escolhido para fazer parte do antecedente da regra;
- $\tau_{ij}(t)$ é a quantidade de feromônio associado ao *termo*$_{ij}$ no tempo *t*. Quanto maior a quantidade de feromônio maior é a probabilidade do *termo*$_{ij}$ ser escolhido. O valor de

[4] Entropia: corresponde a quantidade de informação

$\tau_{ij}(t)$ depende dos caminhos seguidos pelas formigas anteriores, similarmente a comunicação indireta das formigas reais;

- *a* é o número total de atributos;
- b_i é o total de valores possíveis do atributo a_i;
- *I* é o conjunto de atributos que ainda não foram usados pela formiga atual na construção da regra (restrição de um atributo não poder aparecer mais de uma vez em uma regra).

Dessa forma, a formiga Ant_i adiciona termos a sua regra parcial com a restrição de não aparecer o mesmo atributo mais de uma vez, e um $termo_{ij}$ não pode ser adicionado se isto fizer com que a regra parcial cubra menos casos do que um limiar pré-definido de casos (*min_casos_por_regra).*

Ao término da construção do antecedente, é então associado uma classe ao consequente, esta classe é a de maior frequência entre os casos de treinamento cobertos pela parte do antecedente.

2.5.4 Cálculo da qualidade

Incrementar a quantidade de feromônio associado aos termos $termo_{ij}$ de uma regra corresponde, nas formigas reais, a incrementar a quantidade de feromônio do caminho completado por uma formiga. No algoritmo, este incremento é proporcional à qualidade da regra construída pela formiga. A qualidade da regra, denotada por *Q*, é calculada pelo produto: *Q* = *sensibilidade* × *especificidade* (Hand, 1997), definido pela Equação 2.11.

$$Q = \left(\frac{VP}{VP+FN}\right) \times \left(\frac{VN}{FP+VN}\right) \tag{2.11}$$

Onde:

- *VP* (verdadeiro positivo): é o número de casos de treinamento cobertos pelo antecedente da regra e a classe destes casos é igual à classe prevista;
- *FP* (falso positivo): é o número de casos de treinamento cobertos pelo antecedente da regra e a classe destes casos é diferente da classe prevista;
- *FN* (falso negativo): é o número de casos de treinamento que não são cobertos pelo antecedente da regra, porém a classe destes casos é igual à classe prevista;

- *VN* (verdadeiro negativo): é o número de casos de treinamento que não são cobertos pelo antecedente da regra e a classe destes casos é diferente da classe prevista.

Logo, *VP* e *VN* representam as classificações corretas, enquanto *FP* e *FN* representam as errôneas. O valor de *Q* varia entre 0 e 1.

2.5.5 Poda das regras

Ao término da construção da regra e cálculo da qualidade correspondente, é feito uma poda na tentativa de diminuir o número de antecedentes, melhorar a precisão preditiva e evitar *overfitting*[5] (Parpinelli *et al.*, 2002; Liu *et al.*, 2003). O procedimento de poda iterativamente remove um termo por vez da regra enquanto este processo estiver melhorando a qualidade da regra, isto é, o procedimento remove temporariamente cada um dos termos da regra e calcula a qualidade da regra resultante; se a regra com um termo removido temporariamente melhorar a qualidade, então o termo é efetivamente removido. Esta iteração é repetida até que a regra não contenha mais termos que, uma vez removidos, resulte em melhora da qualidade.

2.5.6 Atualização do feromônio

Seguindo o algoritmo, após os passos de construção da regra, cálculo da qualidade e poda da regra, o passo seguinte é atualizar o feromônio de todos os termos $termo_{ij}$ que aparecem na regra. A atualização é feita pela Equação 2.12.

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \tau_{ij}(t) \times Q \text{ , } \forall\, termoij \in \text{regra atual} \qquad (2.12)$$

Dessa forma, a quantidade de feromônio de todos os $termo_{ij}$ que constituem a regra terá seu feromônio atualizado proporcionalmente à qualidade *Q*.

Em seguida o feromônio de todos os $termo_{ij}$ é decrementado para simular a evaporação. Este efeito é alcançado a partir de uma normalização, calculada pela divisão do valor de cada τ_{ij} pela soma de todos $\tau_{ij}(t)$. Todos os $termo_{ij}$ terão seus valores normalizados (decrementados), porém os termos $termo_{ij}$ usados na Equação 2.7 terão com certeza suas

[5] *Overfitting*: quando uma regra ou o modelo é muito específico para o conjunto de treinamento. Modelos com *overffiting* apresentam poder preditivo elevado para os casos conhecidos, mas não é geral o suficiente para a predição de novos casos.

quantidades de feromônio aumentadas.

2.5.7 Considerações finais sobre o *Ant-Miner*

Ao final do laço **Enquanto** os exemplos que não foram cobertos por nenhuma regra, serão classificados pela regra padrão, a qual não possui termos no antecedente e tem como consequente, a classe da maioria destes exemplos. Caso não haja exemplos não cobertos, a regra padrão será adicionada da mesma forma, porém como consequente a classe majoritária de todos os exemplos de treinamento.

Ao todo são quatro parâmetros (limiares), e eles têm influência sobre os resultados do algoritmo. Logo, dependendo do conjunto de dados a ser aplicado, o usuário deve considerar/avaliar quais valores atribuir para os parâmetros:

- *num_formigas:* é o número máximo de regras candidatas a cada iteração do laço **Enquanto**, visto que cada formiga constrói uma regra. Em cada iteração deste laço, a melhor regra construída é considerada a regra descoberta. Quanto maior o número de *num_formigas* maior é o número de regras candidatas;
- *min_casos_por_regra:* cada regra deve cobrir um número mínimo de casos. Isto ajuda a forçar a generalidade das regras e consequentemente evita o *overfitting*;
- *max_casos_n_cobertos:* é o número máximo de casos não cobertos no conjunto de treinamento. A cada iteração do laço **Enquanto**, o conjunto de treinamento é reduzido, isto é, os casos corretamente cobertos são eliminados. Quando esta redução atinge um limiar, o laço é interrompido e os casos restantes são classificados pela regra padrão*;*
- *num_regras_converg:* É o número de regras consecutivas e iguais. Se a formiga atual construir uma regra que é exatamente igual às regras construídas pelas (*num_regras_converg* - 1) formigas anteriores, o algoritmo conclui que as formigas convergiram para o mesmo caminho e então a construção das regras é interrompida.

Após a construção das regras, elas são aplicadas sobre o conjunto de teste. As regras são aplicadas na ordem em que foram inseridas na lista de regras descobertas. Cada caso do conjunto de teste é classificado (associado a uma classe) assim que a primeira regra da lista ordenada o cubra.

2.6 Trabalhos relacionados ao *Ant-Miner*

Parpinelli *et al.* (2002) avaliaram o *Ant-Miner* sobre seis bases de dados de domínio público da *University of California at Irvine* (UCI, 2012). O algoritmo foi comparado com outros algoritmos de classificação e obteve bons resultados. Estes resultados demonstraram que o *Ant-Miner* é uma técnica promissora para descoberta de regras de classificação, tornando-se uma área foco de pesquisa. Dessa forma, uma série de pesquisas e modificações tem sido propostas a fim de melhorar sua eficiência (Salama *et al.*, 2011). A seguir são descritos brevemente alguns trabalhos relacionados ao *Ant-Miner*.

***Ant-Miner2*:** A função heurística baseada no ganho de informação da versão original é complexa. Liu *et al.* (2002) apresentaram uma técnica mais fácil para computar o valor da função heurística para cada $termo_{ij}$. A ideia é que o algoritmo não necessite de um valor preciso desta função desde que o feromônio compense os pequenos erros nos valores da função heurística.

$$\eta_{ij} = \frac{|majority\_classT_{ij}|}{|T_{ij}|} \tag{2.13}$$

Onde:

- T_{ij} é a quantidade de vezes que o $termo_{ij}$ aparece no conjunto de treinamento;
- $majority\_classT_{ij}$ é a quantidade de vezes de ocorrências da classe majoritária para o $termo_{ij}$ no conjunto de treinamento.

Os resultados comparativos mostraram que o *Ant_Miner2* apresentou resultados idênticos à versão original, com a vantagem de menor custo computacional.

Ant-Miner3: Liu *et al.* (2003) apresentaram uma nova forma de atualização do feromônio. No *Ant-Miner*, quando uma formiga constrói uma regra, a quantidade de feromônio associado a cada termo constituído na regra é incrementado pela Equação 2.12 e a evaporação dos termos é efetuada pela normalização. No *Ant-Miner3* a quantidade de feromônio associada a cada termo que ocorre na regra é atualizada ao mesmo tempo em que é efetuada a normalização, e é dada pela Equação 2.14:

$$\tau_{ij}(t) = (1 - \rho) \cdot \tau_{ij}(t-1) + \left(1 - \frac{1}{1+Q}\right) \cdot \tau_{ij}(t-1) \tag{2.14}$$

Onde:

- ρ é a taxa de evaporação de feromônio, o qual controla a velocidade que os caminhos antigos evaporam. Um valor alto de ρ indica maior velocidade na taxa de evaporação. O valor 0.1 foi fixado e usado nos experimentos;
- Q é a qualidade da regra construída, que varia no intervalo [0,1].

Resultados experimentais mostraram que o *Ant-Miner3* precisava de mais formigas para convergir e encontrar uma solução e a precisão preditiva das regras descobertas foi melhor que a obtida na versão original.

Unordered Rule Set Ant-Miner: Smaldon & Freitas (2006) modificam a forma com que o modelo de regras é produzido. A versão original produz uma lista ordenada de regras, a qual é aplicada ao conjunto de teste na ordem em que foram descobertas. Isto dificulta a interpretação das regras, já que para entender uma regra é necessário compreensão das regras precedentes contidas na lista. Esta nova versão, proposta pelos autores, produz um conjunto desordenado de regras que são aplicadas ao conjunto de teste em qualquer ordem. Isto faz que as regras sejam mais fáceis de ser interpretadas pelo usuário, porque a interpretação de cada regra se torna independente das outras. Resultados experimentais mostraram que o algoritmo proposto obteve resultados semelhantes ao original com relação ao poder preditivo. Com a vantagem de descobrir regras mais modulares.

Ant-Miner+: O trabalho proposto por Martens *et al.* (2007) é considerado outra importante extensão do *Ant-Miner*, com modificações que melhoram a performance do algoritmo. Uma das principais diferenças está no uso da medida *MAX-MIN*, proposto por Stutzle & Hoos (2000) para o *Ant System*. O *MAX-MIN* contém um valor máximo e um mínimo do feromônio para a construção das regras. Inicialmente o feromônio de todos os termos é inicializado com *Tmax*. Conforme o feromônio é atualizado nas construções das regras (incremento ou evaporação), a quantidade de feromônio não pode exceder o intervalo de *Tmax* e *Tmin*. A ideia é melhorar a exploração dos termos e evitar convergências prematuras das formigas.

Outra mudança no *Ant-Miner+* está na representação dos caminhos, pois o feromônio não está associado aos termos e sim às arestas. Cada *atributo-valor* ($A_i=V_{ij}$) é conectado a todos os outros valores do próximo atributo A_{i+1}.

Outra característica do *Ant-Miner+* é que o consequente (classe) pode ser selecionado antes da construção da regra, diferente da versão original em que a classe é associada apenas

no término da construção dos antecedentes. Inicialmente uma classe é selecionada de acordo com a quantidade de feromônio associado. Este feromônio indica qual classe contribui melhor na construção de uma regra. Os próximos termos (parte do antecedente) são selecionados com base na Equação 2.15.

$$\eta_{Vi,j} = \frac{|T_{ij}\ \&\ Class = C_t\ |}{|T_{ij}|} \tag{2.15}$$

Onde:

- $|T_{ij}|$ é a quantidade de casos que contem T_{ij};
- *Ct* é a classe atual selecionada.

Martens *et al.* (2007) mostram uma tabela com um *overview* de algumas extensões, mostrando as diferenças em cada procedimento: função heurística, feromônio, escolha do termo e de poda.

***cAnt-Miner*:** O *Ant-Miner* original trabalha apenas com atributos discretos, porém muitos problemas de classificação do mundo real podem apresentar atributos contínuos. Otero *et al.* (2008) propuseram uma extensão chamada *cAnt-Miner* a qual pode dinamicamente criar limiares nos atributos contínuos no momento da construção da regra e consequentemente evitar a necessidade de uma discretização como passo de pré-processamento. Resultados apresentados pelos autores mostraram que o processo de construção facilita a descoberta de regras mais precisas e significativamente simples.

***Multiple Pheromone types Ant-Miner*:** No *Ant-Miner* original, a parte do consequente das regras é escolhida depois que a parte dos antecedentes é construída. O trabalho proposto por Salama *et al.* (2011) utiliza múltiplos feromônios na construção das regras no qual a classe (consequente) é escolhida antes da construção dos antecedentes e os termos antecedentes são selecionados com relação à classe atual escolhida. Entretanto, a formiga que pré-selecionou uma determinada classe apenas é influenciada pelo feromônio correspondente a esta classe o qual foi depositado pelas formigas anteriores que também possuíam esta classe.

Antes cada $termo_{ij}$ <*atributo, valor*> possuía apenas um feromônio correspondente. Agora a ideia é cada $termo_{ij}$ <*atributo, valor*> possuir *n* feromônios correspondentes, no qual *n* é o número de classes existentes. Isto é, tem se um $termo_{ijk}$ como <*atributo, valor, classe*> o qual possui um feromônio correspondente. A cada construção de uma nova regra *Ri*, apenas

os feromônios dos $termo_{ijk}$ correspondente à classe *k* da regra R_i tem seus valores atualizados.

Esta técnica contribui para que termos irrelevantes não sejam adicionados nas regras. Os resultados mostraram que o *Ant-Miner* com múltiplos feromônios generaliza regras melhores com relação à precisão preditiva e simplicidade.

Capítulo

3

Desenvolvimento

3.1 Considerações Iniciais

A maioria das extensões do *Ant-Miner* visa melhorar o poder preditivo e a compreensibilidade do modelo. Neste trabalho as extensões realizadas visam adaptar o algoritmo para encontrar melhores regras quando aplicado a base de dados com classes desbalanceadas, isto é, que seja capaz de encontrar regras para a classe de menor proporção, pois algoritmos classificadores são muito sensíveis a este tipo de desbalanceamento e tendem a valorizar a classe predominante e ignorar a classe de menor proporção, o que é problemático principalmente quando a classe minoritária é a de maior interesse.

A Seção 2.5 descreve o algoritmo classificador *Ant-Miner*, já as Seções 3.2 e 3.3 deste capítulo descrevem os algoritmos desenvolvidos neste trabalho.

Quando comparado com o algoritmo baseado em árvore de decisão, o *Ant-Miner,* na maioria das vezes, apresenta um número menor de regras. Logo, quando aplicado a uma base de dados com classes desbalanceadas, as chances do algoritmo *Ant-Miner* encontrar regras para as classes raras são ainda menores que as chances do C4.5.

A implementação partiu apenas do pseudo-código do algoritmo original. Desenvolvidos na linguagem *Java* no ambiente de desenvolvimento *Eclipse* e recebem bases de dados no padrão *arff*, assim como a ferramenta *Weka*[6]. Desta forma elas podem ser avaliadas com bases de dados já conhecidas na literatura. Assim como a versão original, as extensões trabalham apenas com atributos discretos. As extensões do *Ant-Miner* neste trabalho foram denominadas: *Ant-MinerCI (Ant-Miner Class Imbalance)* e *Ant-MinerCIP* (*Ant-Miner Class Imbalance Precision*).

[6] Machine Learning Group at University of Waikato. Disponível em: http://www.cs.waikato.ac.nz/~ml/weka

3.2 Ant-MinerCI (Ant-Miner Class Imbalance)

A Figura 3.1 apresenta o pseudocódigo do algoritmo *Ant-MinerCI*.

Inicio
ConjuntoTreinamento = todos os casos do conjunto de treinamento;
ListaDeRegras = {};
Calcule o valor heurístico para cada termo <*atributo* = *valor*> com relação à *classe_interesse;*
Enquanto (ConjuntoTreinamento > *Max_casos_n_cobertos*)
    i = 1; // índice de cada formiga
    j = 1; // índice do teste de convergência
    Inicialize o feromônio de cada termo com a mesma quantidade;
    Repita
        Se (Primeira Iteração do laço Enquanto)
            Ant_i adiciona como consequente a classe de interesse;
        Fim Se
        Enquanto (número de antecedentes da regra Ant_i <= *max_num_antec)*
            Ant_i começa com uma regra vazia e incrementalmente constrói uma regra de classificação R_i, adicionando um termo por vez;
        Fim Enquanto
        Atualize o feromônio de todos os termos. Os termos contidos em Ant_i são incrementados (proporcional à qualidade da regra R_i) e decrementados os feromônios dos outros termos;
        Se (R_i é igual a R_{i-1})
            Então j = j +1; // incrementa teste de convergência
            Senão j = 1;
        Fim Se
        i = i + 1;
    Até (i >= Num_formigas) **ou** (j >= Num_regras_converg)
    Escolha a melhor regra R_m dentre todas as regras R_i construídas pelas formigas;
    Adicione a regra R_m na ListaDeRegras;
    ConjuntoTreinamento = ConjuntoTreinamento – {conjunto de casos cobertos pela regra R_m};
Fim Enquanto
Adicione a regra padrão na ultima posição da ListaDeRegras;

Figura 3.1: Pseudocódigo do algoritmo *Ant-MinerCI*

3.2.1 Valor heurístico

No *Ant-Miner2* (Seção 2.6) o cálculo do valor heurístico para cada *termoij* é feito pela Equação 2.13, que é a frequência de vezes que um $termo_{ij}$ aparece com relação à classe majoritária, isto é, as formigas tendem a achar mais "interessantes" os caminhos (termos) que levam a descoberta de regras para a classe majoritária.

O cálculo do valor heurístico para cada $termo_{ij}$ no *Ant-MinerCI* é feito não mais com relação à classe majoritária, e sim com relação à classe de maior interesse, que pode ser ou não a classe minoritária. O cálculo é feito pela Equação 3.1.

$$\eta_{ij} = \frac{|CI \,\&\, T_{ij}|}{|T_{ij}|} \tag{3.1}$$

Onde:

- $|T_{ij}|$ é a quantidade de vezes que o *termoij* aparece no conjunto de treinamento;
- $|CI \,\&\, T_{ij}|$ é a quantidade de vezes de ocorrência da classe de interesse para o *termoij*.

A ideia é fazer com que as formigas artificias Ant_i tenham preferência por termos que sejam mais relevantes para o encontro de regras para a classe de interesse. Porém, este cálculo não é suficiente, pois com as iterações das formigas e as atualizações dos feromônios, as formigas ainda tendem a convergir para caminhos que levam a regras com as classes majoritárias.

3.2.2 Construção das regras

Na versão original o consequente (classe) das regras é escolhido após a construção dos antecedentes. Já no *Ant-Miner+* (Seção 2.6), o consequente é escolhido primeiro de forma probabilística, isto é, o consequente é selecionado de acordo com a quantidade de feromônio associado, este feromônio indica qual classe contribui melhor na construção da regra.

Na primeira iteração para encontrar a melhor regra, o *Ant-MinerCI* seleciona primeiro o consequente, porém não de forma probabilista, mas obrigatoriamente seleciona a classe de interesse. Isto faz com que as formigas obrigatoriamente busquem caminhos (selecionem termos) para sua regra parcial que maximizem a qualidade das regras que tenham como consequente a classe de interesse. Portanto, as formigas tendem a convergir para o caminho mais curto (melhores regras) com relação à classe de interesse. Esta classe de interesse deve ser especificada pelo usuário antes que o algoritmo inicie a construção das regras.

Uma formiga interrompe a adição de termos ao seu antecedente em duas situações:

- Quando a adição de um novo *termoij* fizer com que a regra cubra um número de casos menor que um limiar especificado pelo usuário, *min_casos_por_regra;*
- Quando todos os atributos já estejam compondo a regra.

No *Ant-MinerCI* há ainda mais uma condição para interromper a adição de termos. É o *max_num_antec,* este parâmetro limita o número de antecedentes que uma regra pode conter, pois regras que possuem um número muito grande de termos antecedentes dificulta a compreensibilidade das regras.

Após a primeira iteração do laço **Enquanto** (laço mais externo) da Figura 3.1, a melhor regra R_m dentre todas as R_i (que possuem a classe de interesse) é selecionada e adicionada à lista de regras descobertas. A partir da segunda iteração do laço **Enquanto**, o algoritmo pode encontrar regras para todas as classes possíveis do atributo-meta. A fórmula para calcular a qualidade da regra é a mesma do *Ant-Miner* original, *Q = sensibilidade* × *especificidade,* definido na Equação 2.11.

Dessa forma, a lista ordenada de regras descobertas pelo algoritmo irá conter no mínimo uma regra para a classe de interesse, mesmo quando esta for à classe minoritária, que é mais difícil descoberta.

É importante que a regra com a classe de interesse seja a primeira, pois a primeira regra sempre é gerada com relação ao conjunto de treinamento inteiro. Isto porque na segunda iteração, os registros que foram corretamente cobertos pela primeira regra são eliminados do conjunto de treinamento. Logo, a *n-ésima* regra descoberta é condicionada as *n-1* regras anteriores.

Isso significa que a primeira regra encontrada é de mais fácil entendimento, dado que, para entender a *n-ésima* regra, é preciso levar em consideração as *n-1* regras anteriores (Smaldon & Freitas, 2006; Carvalho, 2005).

3.2.3 Parâmetros do *Ant-MinerCI*

Além dos parâmetros convencionais (Seção 2.5.7) o *Ant-MinerCI* contém outros dois:

- *classe_interesse:* o usuário especifica qual a classe, dentre as possíveis do atributo-meta, deve ser utilizada para a descoberta da primeira regra;
- *max_num_antc:* é o número máximo de termos que uma regra pode conter no seu antecedente.

3.3 *Ant-MinerCIP (Ant-Miner Class Imbalance Precision)*

Como mencionado no início deste capítulo, uma situação que pode ocorrer é a classe de interesse ser a classe minoritária (pouca representatividade no conjunto), logo as regras

descobertas para esta classe possuem poucos casos cobertos com relação ao total de casos do conjunto de treinamento.

Quando se usa o cálculo *Q = sensibilidade × especificidade,* a tendência é que as regras tenham um número razoável de casos cobertos com relação ao total de casos do conjunto de treinamento, pois este cálculo "olha" para todas as situações que podem ocorrer na regra.

Como exemplo, considere uma base de dados em que a meta é dizer se um paciente tem ou não uma determinada doença. O atributo-meta tem duas classes, a classe "não" e a classe "sim". A distribuição dessas classes no conjunto de treinamento é 98% para a classe "não" e 2% para a classe "sim". A maioria dos algoritmos indutores de regras provavelmente irão encontrar regras apenas para a classe "não", porém o interesse maior está nos 2% da classe "sim", que significa saber que o paciente tem a doença.

Mesmo que o *Ant-MinerCI* encontre regras para a classe minoritária, estas regras podem ter uma baixa taxa de acerto utilizando a métrica *Q = sensibilidade × especificidade.* Por exemplo: a base de dados com classes desbalanceadas, citada no exemplo acima, contém 50.000 casos para treinamento. Uma formiga encontra uma regra R_1 (para a classe "sim") que cobre 500 casos, sendo *VP* = 400, *FP* = 100, *FN* = 600 e *VN* = 48900. Utilizando *Q = sensibilidade × especificidade,* tem-se:

$$Q = \left(\frac{VP}{VP+FN}\right) \times \left(\frac{VN}{FP+VN}\right) = \left(\frac{400}{400+600}\right)\left(\frac{48900}{100+48900}\right) = 0{,}3912 \qquad (3.2)$$

Dado o valor da Equação 3.2 para a qualidade de uma regra R_1, a probabilidade desta regra R_1 ser escolhida pelo algoritmo como a melhor regra é baixa, apesar desta regra ter uma precisão preditiva de 80%, pois o algoritmo seleciona como melhor regra a que tiver a maior taxa de *Q*, isto é, que possui cobertura VP alta com relação ao total de casos, independente de qual classe se trata. Nesta situação, em que se possui classe rara, as regras descobertas para esta classe pode possuir uma baixa precisão preditiva, o que prejudica o modelo e a descoberta de regras relevantes.

O *Ant-MinerCIP* utiliza diretamente o cálculo da precisão (Equação 3.3) para avaliar a qualidade da regra, pois a precisão visa analisar somente os casos cobertos pela regra. Quanto maior a precisão, menor o número de FP na regra.

$$Precisão = Q = \left(\frac{VP}{VP+FP}\right) \quad (3.3)$$

Onde:

- *VP* (verdadeiro positivo): é o número de casos de treinamento cobertos pelo antecedente da regra e possuem a mesma classe do consequente prevista;
- *FP* (falso positivo): é o número de casos de treinamento cobertos pelo antecedente da regra e possuem uma classe diferente da classe prevista pela regra.

Logo, a métrica *precisão* auxilia não somente o encontro de regras para as classes raras, como também o encontro de regras para os pequenos disjuntos (casos raros). E não compromete o encontro de regras para as outras classes.

Uma situação que pode ocorrer é a descoberta de regras muito ajustadas que podem caracterizar *overfitting.* Para evitar este problema, deve se utilizar o parâmetro do algoritmo *min_casos_por_regra* (Seção 2.5.7), que indica o valor mínimo de casos que uma regra deve cobrir. Escolhendo um limiar adequado, é possível evitar *overfitting.*

3.4 Considerações Finais

Os algoritmos propostos apresentam mais informações nos resultados (*output)* que uma versão do algoritmo original disponível, chamado *GUI-Ant-Miner* (Meyer & Parpinelli, 2005). A Tabela 3.1 apresenta um comparativo das informações que aparecem no *output* de cada algoritmo.

Tabela 3.1 Descrição das informações que aparecem no *output.*

	GUI-Ant-Miner	*Ant-MinerCI / Ant-MinerCIP*
Lista de regras descobertas	X	X
Taxa de acerto e cobertura de cada regra descoberta		X
Matriz de confusão		X
Taxa acerto no treinamento	X	X
Taxa acerto no teste	X	X
Taxa de VP e Taxa de FP		X

Com a taxa de acerto e cobertura de cada regra, pode-se analisar a validade das regras para a classe de interesse. Com as taxas de VP e FP os resultados podem ser avaliados por gráficos ROC.

Capítulo

4

Resultados Experimentais e Análise dos Resultados

4.1 Validação dos Algoritmos Desenvolvidos

Como mostrado na Seção 3.2, as principais modificações nos algoritmos desenvolvidos são:

- Possuem limitador para o número de antecedentes que compõem uma regra;
- É escolhida a classe de interesse para o cálculo do valor heurístico;
- A primeira regra descoberta é sempre com relação à classe de interesse.

Essas modificações auxiliam na compreensibilidade das regras e também na descoberta de regras para as classes minoritárias.

Primeiramente, foram utilizadas bases de dados de domínio público para avaliação e validação dos algoritmos. As bases estão disponíveis no repositório do UCI (2012). Das sete bases de dados utilizadas, apenas uma possui atributos contínuos, porem seus valores foram categorizados (pré-processamento).

Tabela 4.1 Bases de dados utilizadas para validação dos algoritmos

Base de dados	No. de casos	No. de atributos	No. de classes	Positiva (%)
Ljubljana breast cancer	286	9	2	29,7%
Winsconsin breast cancer	683	9	2	34,4%
tic-tac-toe	958	9	2	34,6%
dermatology	366	34	6	5,5%
Votes	435	16	2	38,6%
Letter-a	20.000	17	2	4%
Letter-vogal	20.000	17	2	19,4%

A Tabela 4.1 descreve as bases de dados com relação ao número de casos, número de atributos, número de classes e a proporção da classe minoritária (classe positiva).

A base *Ljubljana breast cancer* possui 286 registros, cada um com 9 atributos previsores e a meta é determinar os pacientes que terão reincidência do câncer de mama.

A base *Winsconsin breast cancer* possui 683 registros, cada um com 9 atributos previsores e a meta é determinar se os pacientes possuem câncer de mama benigno ou maligno.

A base *tic-tac*-toe é composta por 958 registros, cada um com 9 atributos previsores e a meta é prever o melhor movimento no "jogo da velha".

A base *dermatology* possui 366 registros, cada um com 34 atributos previsores e a meta é determinar se o paciente possui alguma das seis possíveis patologias dérmicas registradas na base.

A base *Votes* é composta por 435 registros, cada um com 16 atributos previsores e a meta é prever se determinado candidato é democrata ou republicano (eleições nos Estados Unidos).

A base *Letter* possui 20.000 registros, cada um com 16 atributos e a meta é identificar um retângulo de pixels como uma das 26 letras maiúsculas do alfabeto Inglês. A base adaptada *Letter-a* tem como meta identificar quais destes retângulos de *pixels* são a letra "A". A base adaptada *Letter-vogal* tem como meta identificar quais destes retângulos de *pixels* são vogais, ambas em consonância com Batista *et al.* (2004) e Machado (2007).

Foram utilizados dois algoritmos para comparação com o *Ant-MinerCI* e *Ant-MinerCIP*: algoritmo C4.5 e o algoritmo *GUI-Ant-Miner*. O C4.5 (Quinlan, 1993) (Apêndice A), é um algoritmo indutor de regras que usa a representação de árvores de decisão e possui implementação na ferramenta *Weka*. O *GUI-Ant-Miner* (Meyer & Parpinelli, 2005), é uma versão do *Ant-Miner* que contém todas as características originais do algoritmo, porém com uma melhoria na interface gráfica para iteração do usuário com o sistema, e utiliza arquivos de entrada no mesmo padrão do *Weka*.

Para uma comparação mais justa, não foram feitos ajustes nos parâmetros dos algoritmos a fim de otimizar seus desempenhos, pelo fato que os parâmetros ótimos podem ser diferentes para cada base de dados. A Tabela 4.2 apresenta os valores dos parâmetros, em consonância com (Parpinelli *et al.*, 2002).

Tabela 4.2 Parâmetros utilizados

Parâmetros	**Valores**
num_formigas	1000
min_casos_por_regra	10
max_casos_n_cobertos	15
num_regras_converg	10
max_num_antec	4
classe_interesse	Classe minoritária

A Tabela 4.3 apresenta os resultados dos quatro algoritmos com relação à taxa de acerto no conjunto de teste. A Tabela 4.4 mostra os resultados com relação ao número de regras que compõe cada modelo. A Tabela 4.5 apresenta os resultados com relação ao maior número de termos na regra mais extensa que compõe cada modelo. O método de avaliação utilizado foi a validação cruzada (Seção 2.2.2) com 10 partições.

Tabela 4.3 Taxa de acerto (%) Ant-MinerCI x Ant-MinerCIP x GUI-Ant-Miner x C4.5

	Taxa de Acerto (%)			
Base de dados	***Ant-MinerCI*** **(μ%)**	***Ant-MinerCIP*** **(μ%)**	***GUI-Ant-Miner*** **(μ%)**	**C4.5** **(μ%)**
Ljubljana breast cancer	74,73	**75,91**	74,62	75,2
Winscon breast cancer	93,79	94,42	94,12	**94,99**
tic-tac-toe	72,14	69,48	72,22	**84,9**
dermatology	80,67	84,02	86,49	**93,98**
Vote	95,86	95,86	95,45	**96,3**
Letter-a	96,17	97,89	97,82	**98,39**
Letter-vogal	81,35	89,53	84,09	**90,06**

Tabela 4.4 No. de Regras do Modelo Ant-MinerCI x Ant-MinerCIP x GUI-Ant-Miner x C4.5

	No. de Regras			
Base de dados	***Ant-MinerCI*** **(μ)**	***Ant-MinerCIP*** **(μ)**	***GUI-Ant-Miner*** **(μ±DP)**	**C4.5** **(μ)**
Ljubljana breast cancer	**3,1**	9	4,4	4
Winscon breast cancer	**5,8**	7	7,2	19
tic-tac-toe	**6**	38,6	6,5	95
dermatology	**5**	18	6,5	30
Vote	5,1	10,5	**4,8**	6
Letter-a	**4,2**	12,8	7,2	37
Letter-vogal	**5**	67	14	185

Tabela 4.5 Maior No. de termos da regra mais extensa Ant-MinerCI x Ant-MinerCIP x GUI-Ant-Miner x C4.5

	Maior No. de Antecedentes			
Base de dados	***Ant-MinerCI***	***Ant-MinerCIP***	***GUI-Ant-Miner***	**C4.5**
Ljubljana breast cancer	***1***	*3*	*3*	*2*
Winscon breast cancer	***2***	*4*	***2***	*3*
tic-tac-toe	*2*	*4*	***1***	*5*
dermatology	***2***	*4*	*5*	*5*
Vote	*4*	*4*	***3***	*4*
Letter-a	***2***	*4*	***2***	***2***
Letter-vogal	***3***	*4*	***3***	*6*

Como mencionado anteriormente à taxa de acerto não é uma métrica muito efetiva para avaliar classificadores, principalmente quando aplicados a bases de dados desbalanceadas. Logo a Tabela 4.6 apresenta os resultados com relação à métrica ROC, no qual avalia o poder preditivo dos classificadores.

Tabela 4.6 Ranking dos algoritmos com análise ROC

	Ranking dos algoritmos com ROC		
Base de dados	**1º**	**2º**	**3º**
Ljubljana breast cancer	*Ant-MinerCI*	*Ant-MinerCIP*	C4.5
Winscon breast cancer	*Ant-MinerCIP*	C4.5	*Ant-MinerCI*
tic-tac-toe	C4.5	*Ant-MinerCI*	*Ant-MinerCIP*
Vote	*Ant-MinerCI*	*Ant-MinerCIP*	C4.5
Letter-a	C4.5	*Ant-MinerCIP*	*Ant-MinerCI*
Letter-vogal	C4.5	*Ant-MinerCIP*	*Ant-MinerCI*
GRUPO_CID_IIIA	*Ant-MinerCI*	*Ant-MinerCIP*	C4.5
GRUPO_EVENTO_09	*Ant-MinerCI*	C4.5	*Ant-MinerCIP*

4.1.1 Análise

O C4.5 obteve uma taxa de acerto significativamente melhor nas bases de dados *tic-tac-toe, dermatology* e *Letter-vogal* porém obteve um número de regras muito maior do que os outros dois algoritmos. Isto mostra que o *Ant-MinerCI*, *Ant-MinerCIP* e o *GUI-Ant-Miner* sacrificam a taxa de acerto para construírem um modelo com um menor número de regras, o que contribui com a simplicidade e compreensão das regras.

Com as bases *Ljubljana breast cancer, Wisconsin breast cancer, Vote* e *Letter-a* os três algoritmos alcançaram taxas de acerto muito próximas, porém o C4.5 obteve maior número de regras nas bases *Wisconsin breast cancer* e *Letter-a*, possivelmente por estas bases possuírem maior número de registros.

Observando os resultados do *Ant-MinerCI e Ant-MinerCIP* em relação ao *GUI-Ant-Miner*, nota-se que encontrar primeiramente a classe minoritária não diminui o poder preditivo do classificador.

O modelo gerado pelo C4.5 é composto por um conjunto de regras que são independentes uma das outras, consequentemente não possuem uma ordem. Isto significa que, para uma regra ser aplicada a um caso de teste, não importa quantas regras o caso já tenha sido testado, só haverá uma regra que irá cobri-lo.

Os modelos gerados pelos algoritmos *Ant-Miner* são organizados na forma de uma lista ordenada de regras. Isto significa que, para uma regra R_i ser aplicada a um caso de teste,

as regras de R_0 à R_{i-1} não devem cobrir o caso. Desta forma, suas regras não são tão modulares e independentes como as regras descobertas pelo C4.5, o que resulta em um efeito que reduz um pouco a simplicidade das regras descobertas pelo *Ant-MinerCI* (Smaldon & Freitas, 2006). Porém, este efeito é compensado pelo fato de que, geralmente, o tamanho da lista de regras descobertas pelo *Ant-MinerCI* é muito menor do que o tamanho do conjunto de regras descobertas pelo C4.5.

Comparando o *Ant-MinerCI* e o *Ant-MinerCIP,* nota-se que o primeiro encontra um número menor de regras para o modelo, devido à tendência do algoritmo de fazer uma generalização, logo encontra regras mais gerais e assim precisa de um número menor de regras para cobrir todos os casos. Já o *Ant-MinerCIP* tende a encontrar regras mais especificas, logo precisa de um número maior de regras para cobrir todos os casos.

Os algoritmos desenvolvidos por terem um limitador no número de antecedentes de cada regra encontram no máximo quatro termos no antecedente, pois este foi o parâmetro especificado, já os outros algoritmos que utilizam a poda, em alguns casos, encontram regras com um número maior que quatro termos para a parte do antecedente.

Diferente da taxa de acerto no qual diz que o C4.5 obteve os melhores resultados, a análise ROC mostrou que em alguns casos os algoritmos desenvolvidos são mais eficientes (maior poder preditivo) que o C4.5.

4.2 Comparativo *Ant-MinerCI x Ant-MinerCIP*

O cálculo da qualidade das regras no *Ant-MinerCI* é realizado pela equação *Q = sensibilidade x especificidade*. No *Ant-MinerCIP* é utilizada a Precisão para o cálculo da qualidade de cada regra. Isso modifica a forma como as regras são avaliadas e escolhidas para fazerem parte da lista de regras descobertas. Como mencionado na Seção 3.3, o parâmetro ***min_cas_por_regra*** é utilizado para ajudar a evitar o *overfitting.* O objetivo do *Ant-MinerCIP* é auxiliar na descoberta de regras para as classes raras com maiores taxa de acerto e simplicidade. Logo, os próximos resultados desta seção estão focados na primeira regra descoberta por cada algoritmo sobre base de dados com classes desbalanceadas.

Para este estudo foi utilizado à base de dados *Letter-a*, *Letter-vogal* e também uma base de dados de um plano de saúde (Barros *et al.,* 2011). Estas bases de dados possuem a característica de classes desbalanceadas, ou seja, apresentam muito menos casos para a classe de interesse. A Tabela 4.7 mostra a proporção das bases *Letter-a* e *Letter-vogal.*

Tabela 4.7 Distribuição das bases *Letter-a* e *Letter-vogal*

Base de dados	Classes	Distribuição (%)
Letter-a	(a, outras)	(4%, 96%)
Letter-vogal	(vogal, outras)	(20%, 80%)

A base do plano de saúde contém 50.082 registros e não possui um atributo-meta específico. Inicialmente é selecionado um atributo-meta dentre os 54 atributos da base, assim os outros 53 ficam sendo atributos dos antecedentes (previsores). Uma tabela com todos os atributos está em Barros *et al.* (2011).

Foram escolhidos dois atributos meta para os experimentos: GRUPO_CID_IIIA, GRUPO_EVENTO_09. A Tabela 4.8 apresenta o significado e a proporção dos dois atributos-meta da base do plano de saúde selecionados. Estes atributos foram selecionados em consonância com a análise apresentada em Barros *et al.* (2011).

Tabela 4.8 Distribuição de classes dos atributos-meta da base do plano de saúde

Atributo-meta	Descrição	SIM (%)	NAO (%)
GRUPO_CID_IIIA	Doenças Endócrinas	2,26	97,74
GRUPO_EVENTO_09	Procedimentos – Câncer de Mama	3,62	96,39

Novamente não foi realizado ajustes de parâmetros, isto é, não se sabe os valores ótimos. Estipulou-se o valor 50 para o parâmetro *min_casos_por_regra*, pois estas bases possuem um número maior de exemplos com relação aos dados do UCI. Os outros parâmetros foram o mesmo da seção anterior.

As tabelas 4.9 a 4.12 apresentam os resultados da precisão, cobertura e o número de antecedentes que compõem a **primeira** regra encontrada pelos algoritmos. A primeira regra sempre é relacionada à classe de interesse (classe minoritária). Neste caso a cobertura se refere ao número de casos cobertos corretamente para a classe minoritária dividido pelo número total de casos da classe minoritária no conjunto de treinamento. Esta forma de cobertura é denominada *Recall.*

Tabela 4.9 Precisão, Cobertura e No. de termos no antecedente da melhor regra para a classe "A" da base de dados *Letter-a. Ant-MinerCI* x *Ant-MinerCIP*

Letter-A	**Classe**	**Precisão (μ%±DP%)**	**Cobertura (μ%±DP%)**	**No. de termos no antecedente**
Ant-MinerCI	A	66,07 ± 17,56	39,58 ± 15,31	1,5 ± 0,5
Ant-MinerCIP	A	98,74 ± 0,02	10,64 ± 2,78	3,7 ± 1,0

Tabela 4.7 Precisão, Cobertura e No. de termos no antecedente da melhor regra para a classe "vogal" da base de dados *Letter-a*. *Ant-MinerCI* x *Ant-MinerCIP*

Letter-vogal	**Classe**	**Precisão (μ%±DP%)**	**Cobertura (μ%±DP%)**	**No. de termos no antecedente**
Ant-MinerCI	vogal	28,46 ± 1,61	31,17 ± 2,2	1,28 ± 0,45
Ant-MinerCIP	vogal	100 ± 0	3,58 ± 0,8	3 ± 1,2

Tabela 4.8 Precisão, Cobertura e No. de termos no antecedente da melhor regra para a classe SIM do atributo-meta GRUPO_CID_IIIA. *Ant-MinerCI* x *Ant-MinerCIP*

GRUPO_CID_IIIA	**Classe**	**Precisão (μ%±DP%)**	**Cobertura (μ%±DP%)**	**No. de termos no antecedente**
Ant-MinerCI	SIM	4,56 ± 2,08	37,5 ± 9,45	4,0 ± 0,0
Ant-MinerCIP	SIM	42,42 ± 5,82	2,78 ± 0,21	4,0 ± 0,0

Tabela 4.9 Precisão, Cobertura e No. de termos no antecedente da melhor regra para a classe SIM do atributo-meta GRUPO_EVENTO_09. *Ant-MinerCI* x *Ant-MinerCIP*

GRUPO_EVENTO_09	**Classe**	**Precisão (μ%±DP%)**	**Cobertura (μ%±DP%)**	**No. de termos no antecedente**
Ant-MinerCI	SIM	23,16 ± 17,42	36,94 ± 18,61	3,7 ± 0,64
Ant-MinerCIP	SIM	72,45 ± 9,93	3,924 ± 3,43	4,0 ± 0,0

4.2.1 Análise

As tabelas 4.9 a 4.13 mostram os resultados obtidos para a melhor regra encontrada para a classe minoritária das quatro bases. O *Ant-MinerCI*, que utiliza a formula *Q = sensibilidade x especificidade* na função de avalição, encontra regras mais gerais, isto é, cobrem um número maior de casos, consequentemente obtém baixa precisão por possuírem muitos casos errôneos que são os *FP* das regras. Já o *Ant-MinerCIP*, por utilizar uma *bias* para a especialização na função de avaliação das regras, encontra regras mais específicas (baixa cobertura), porém com uma maior precisão. Apesar destas coberturas baixas, as regras sempre possuem no mínimo 50 registros cobertos, pois neste caso foi o limite especificado para o parâmetro *min_casos_por_regra*.

Dessa forma, conclui-se que o *Ant-MinerCI* é mais apropriado quando se quer um **modelo** de regras para se utilizar como um previsor e o *Ant-MinerCIP* é mais apropriado quando se quer encontrar regras para uma determinada classe do atributo-meta, principalmente quando se trata da classe rara.

4.3 Balanceamento e avaliação com análise ROC

A abordagem de balanceamento de uma base de dados ajuda a diminuir a diferença de proporção das classes, o que contribui para que algoritmos, ao serem aplicados sobre a base de dados, encontrem melhores regras para as classes minoritárias.

Os três algoritmos utilizados foram o C4.5, *Ant-MinerCI* e *Ant-MinerCIP*, os quais são ajustados para descobrir regras com no mínimo 50 casos cobertos. Os outros parâmetros do *Ant-MinerCI* e *Ant-MinerCIP* são os mesmos da seção anterior.

Foram utilizadas as bases: *Letter-a,* e a base do plano de saúde com os atributos meta GRUPO_CID_IIIA e GRUPO_EVENTO_09. As bases de dados originais são denominadas aqui de P0.

Foram utilizados dois métodos de amostragem: *oversampling* SMOTE e *undersampling* CNN. Os métodos de amostragem visam mudar a distribuição dos dados de treinamento, os quais foram exemplificados na Seção 2.3.3. O método *oversampling* SMOTE (*Synthetic Minority Oversampling Technique)* possui implementação na ferramenta *Weka,* e resultou na base denominada P1 que possui o dobro de registros para a classe minoritária. Estes novos registros foram casos sintéticos gerados pelo algoritmo a partir dos vizinhos mais próximos de cada caso da classe minoritária. O método *undersampling* CNN (*Condensed Nearest Neighbor Rule)* faz *undersampling* estratégico, resultou na base denominada P2 que possui todos os registros para a classe minoritária e apenas os exemplos da classe majoritária considerados mais relevantes, isto é, os casos considerados redundantes foram eliminados.

A Tabela 4.10 apresenta, para cada base de dados, as três diferentes proporções resultantes pelas técnicas de balanceamento. Para cada proporção é apresentada a quantidade de casos e a porcentagem destes casos que possuem a classe de interesse chamada de classe positiva.

Tabela 4.10 Distribuição das bases de dados utilizadas: P0, P1 e P2.

Base de dados	P0 - original		P1 - *oversampling*		P2 – *undersampling*	
	Quantidade de casos	Positivo (%)	Quantidade de casos	Positivo (%)	Quantidade de casos	Positivo (%)
Letter-a	20.000	3,95	20.789	7,59	1.122	70,32
GRUPO_CID_IIIA	50.082	2,24	51.202	4,37	4.140	27,05
GRUPO_EVENTO_09	50.082	3,64	51.903	7,02	4.555	39,98

Quando se trata de bases desbalanceadas, a taxa de acerto não é uma boa métrica, pois

ela passa uma impressão errônea do classificador, ocultando os resultados para a classe positiva (minoritária). Logo, para este estudo, foi utilizada a análise ROC (descrita na Seção 2.3.4), considerada na literatura como uma boa forma de avaliar o desempenho de classificadores, principalmente quando aplicados a bases desbalanceadas. No gráfico ROC, quanto mais próximo o resultado estiver do ponto (0,1) mais próximo está da classificação perfeita; pode se dizer também que obteve melhores resultados para a classe positiva (minoritária).

4.3.1 Resultados

No total são nove gráficos, cada uma das três bases de dados possuem um gráfico para cada proporção (P0, P1 e P2).

Os gráficos resultantes da aplicação da análise ROC sobre a base de dados *Letter-a* com as proporções P0, P1 e P2 são apresentados pelas figuras 4.1, 4.2 e 4.3 respectivamente.

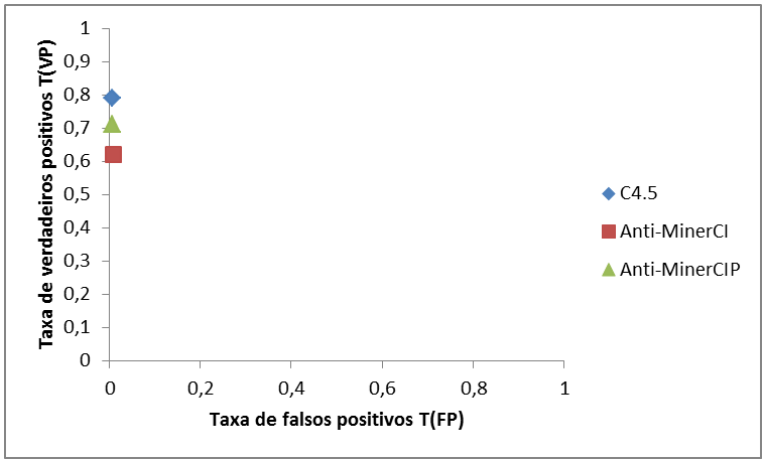


Figura 4.1: Gráfico ROC da base *Letter-a* com a proporção P0

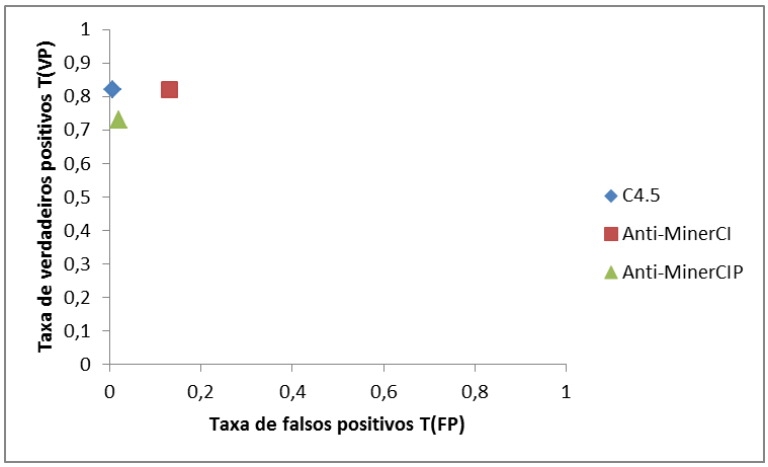


Figura 4.2: Gráfico ROC da base *Letter-a* com a proporção P1 (*oversampling)*

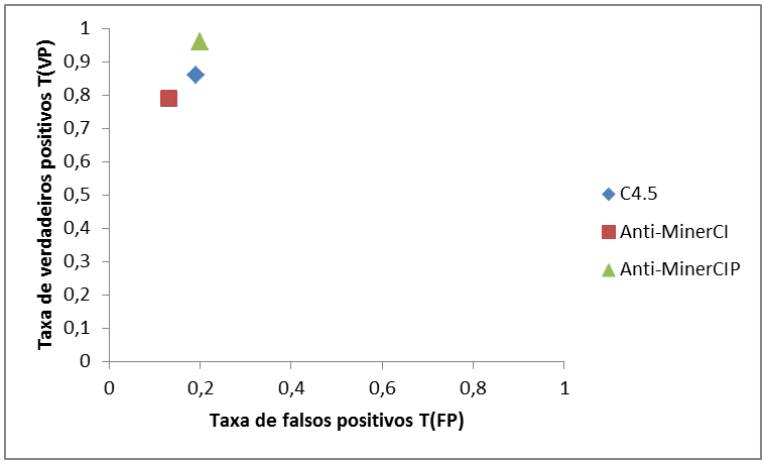


Figura 4.3: Gráfico ROC da base *Letter-a* com a proporção P2 (*undersampling)*

Com a base P0, o C4.5 obteve o melhor desempenho sobre os outros dois classificadores. Os três classificadores estão com baixa taxa de T(FP), logo cometem poucos erros de falsos positivos, isto é, quando classificam um caso negativo como positivo. Com *oversampling* (P1), o *Ant-MinerCI* alcançou a mesma taxa de verdadeiros positivos que o C4.5, porém com taxa maior de falsos positivos. Com *undersampling* (P2), o Ant-*MinerCIP* obteve melhor desempenho.

Os gráficos resultantes da aplicação da análise ROC sobre a base de dados GRUPO_CID_IIIA com as proporções P0, P1 e P2 são apresentados pelas figuras 4.4, 4.5 e 4.6 respectivamente.

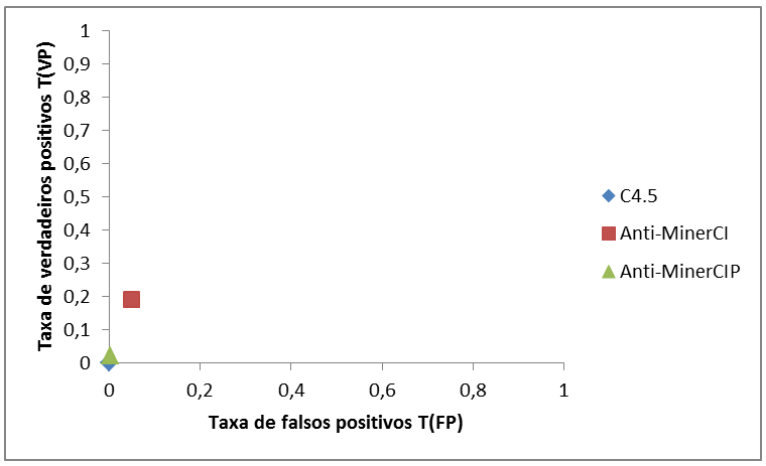


Figura 4.4: Gráfico ROC da base GRUPO_CID_IIIA com a proporção P0

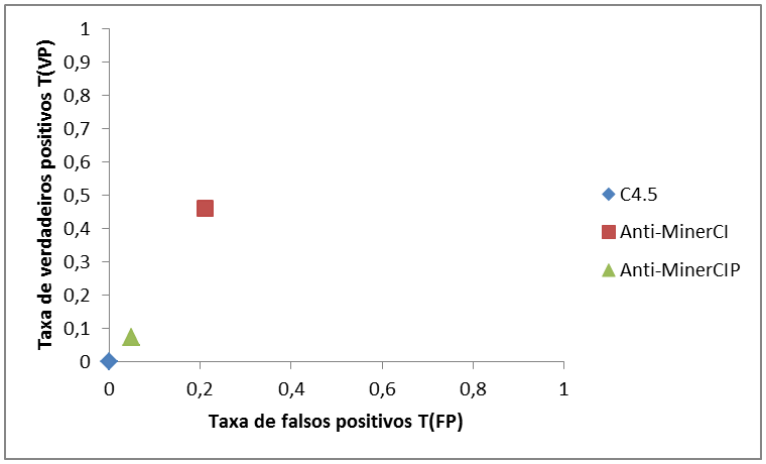


Figura 4.5: Gráfico ROC da base GRUPO_CID_IIIA com P1 (*oversampling*)

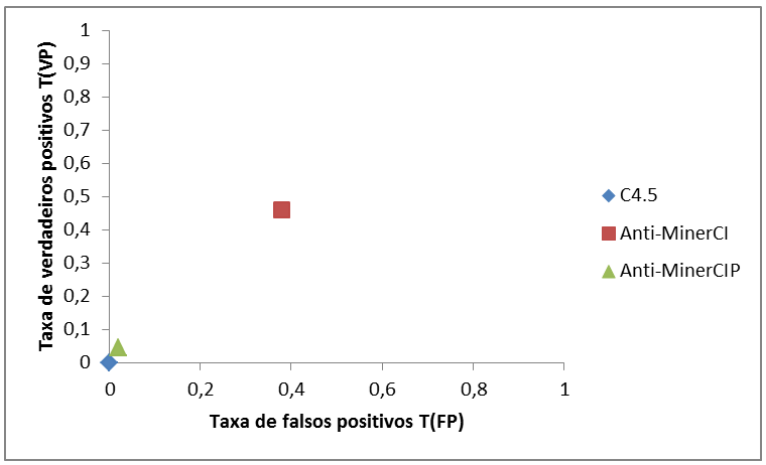


Figura 4.6: Gráfico ROC da base GRUPO_CID_IIIA com P2 (*undersampling)*

Nota-se que nas figuras 4.4, 4.5 e 4.6, o C4.5 não classificou nenhum caso como positivo, consequentemente suas taxas de verdadeiros positivos e falsos positivos são zero. Isso ocorreu porque os modelos de regras gerados pelo C4.5 não possuem regras para a classe positiva, o classificador gerou apenas uma regra, sem antecedentes e como consequente a classe negativa. Já o *Ant-MinerCI* e o *Ant-MinerCIP* possuem no mínimo uma regra para a classe positiva, o que faz com que apareça a taxa de verdadeiros positivos e falsos positivos.

Uma observação importante: se a métrica utilizada fosse taxa de acerto, seria dito que o C4.5 obteve melhor desempenho que os outros dois, o que seria errôneo em termos de classes desbalanceadas.

Os gráficos resultantes da aplicação da análise ROC sobre a base de dados GRUPO_EVENTO_09 com as proporções P0, P1 e P2 são apresentados pelas figuras 4.7, 4.8

e 4.9 respectivamente.

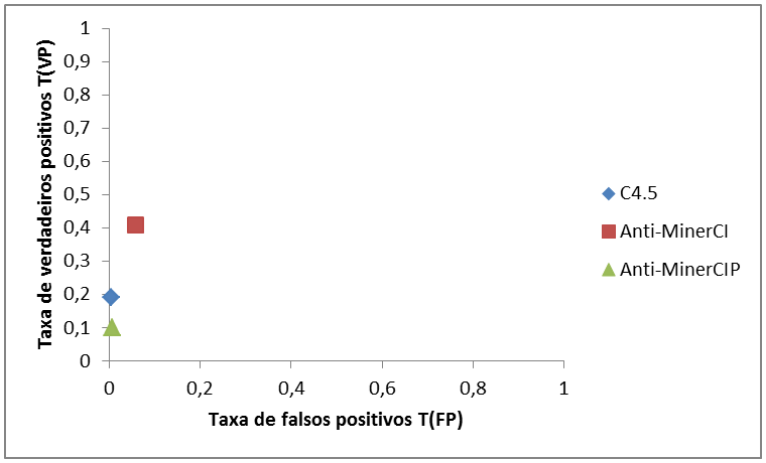


Figura 4.7: Gráfico ROC da base GRUPO_EVENTO_09 com a proporção P0

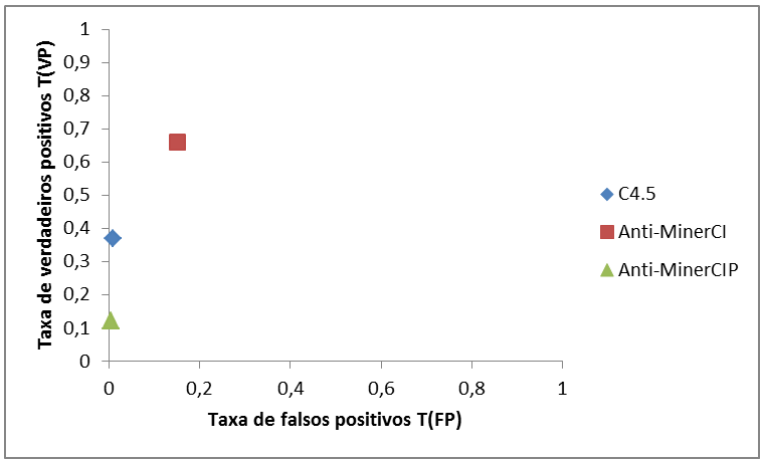


Figura 4.8: Gráfico ROC da base GRUPO_EVENTO_09 com P1 (*oversampling)*

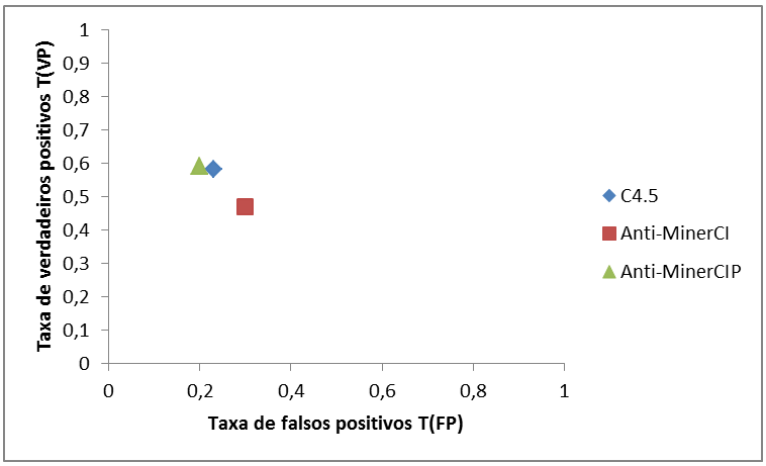


Figura 4.9: Gráfico ROC da base GRUPO_EVENTO_09 com P2 (*undersampling)*

Com P0 e P1 (*oversampling),* o *Ant-MinerCI* obteve melhor desempenho que os outros dois algoritmos, pois está mais próximo do ponto (0,1). Com P2 (*undersampling),* o

Anti-MinerCIP foi melhor que os outros dois.

4.3.2 Análise geral

Dos nove gráficos (nove resultados), o *Ant-MinerCI* foi melhor em cinco deles, principalmente com as bases de proporção P0 e P1. Os melhores resultados do *Ant-MinerCIP* foram com as bases de proporção P2.

O *Anti-MinerCI* é mais liberal que o *Ant-MinerCIP*, isto é, prediz a classe positiva com mais frequência, consequentemente possuem altas taxas de falsos positivos. O *Anti-MinerCIP* é mais conservador que o *Anti-MinerCI*, isto é, comete menos erros de falsos positivos, consequentemente possuem baixas taxas de verdadeiros positivos.

Se fosse utilizada a métrica taxa de acerto, o C4.5 seria considerado o melhor classificador nos nove resultados. Como mencionado anteriormente, esta métrica, quando utilizada em classes desbalanceadas, pode ocultar o erro de predição da classe minoritária (considerada a classe positiva).

Com relação às técnicas de amostragem, os melhores resultados foram com o método *oversampling*, logo aumentar o número de casos da classe minoritária é mais eficiente do que eliminar casos da classe majoritária, mesmo que isso aumente o custo computacional.

Outra questão importante é o número de regras que compõem os modelos, como mostrado nos resultados da Seção 4.1. A lista de regras descobertas pelo *Ant-MinerCI* é, geralmente, menor que o conjunto de regras gerado pelo C4.5. O mesmo ocorre com os resultados desta seção, enfatizando a vantagem do *Ant-MinerCI* com relação à simplicidade das regras.

Capítulo

5

Conclusões e Trabalhos Futuros

5.1 Conclusões

Bases de dados reais podem apresentar um desbalanceamento nas classes, possuindo mais registros para uma classe do que de outras. Este trabalho mostrou que algoritmos classificadores, como o C4.5, podem ter dificuldades em encontrar regras para as classes minoritárias, e quando encontradas elas podem possuir muitos casos errôneos (baixo poder preditivo).

Os dois algoritmos desenvolvidos neste trabalho são baseados no algoritmo *Ant-Miner*, e foram denominados de *Ant-MinerCI* e *Ant-MinerCIP*. Considerando os critérios de avaliação: poder preditivo, simplicidade do modelo e descoberta de regras para a classe de interesse, os resultados mostraram que os dois algoritmos são competitivos com os existentes na literatura.

Eles contribuem para a descoberta de regras para as classes minoritárias, geram modelos com número menor de regras e podem limitar o número de antecedentes que compõem uma regra, o que contribui com a compreensibilidade.

A principal diferença entre o *Ant-MinerCI* e o *Ant-MinerCIP* é que o primeiro encontra regras mais gerais para a classe minoritária, consequentemente com maior taxa de falsos positivos, já o segundo encontra regras mais específicas, consequentemente com menor taxa de falsos positivos.

A aplicação de técnicas de balanceamento ajudou a diminuir a diferença de proporção das classes no conjunto de treinamento, auxiliando os algoritmos a encontrarem melhores regras para as classes minoritárias, aumentando a taxa de verdadeiros positivos.

A métrica taxa de acerto, quando utilizada em classes desbalanceadas, pode ocultar o erro de predição da classe minoritária. Sendo a classe minoritária a classe positiva (de maior interesse), um classificador que nunca classifica um caso como positivo, não tem valia

alguma, independente se sua taxa de acerto ser alta.

Logo, a análise ROC contribuiu para a avaliação dos resultados obtidos pelos algoritmos de maneira mais justa, o que evidenciou que os algoritmos desenvolvidos são competitivos com os existentes na literatura, principalmente quando aplicados a bases de dados com classes desbalanceadas.

5.2 Trabalhos Futuros

Algumas direções de pesquisas futuras podem ser:

- Desenvolvimento de outras duas versões do *Ant-Miner*, uma delas utiliza as duas formas de indução (generalização e especialização); a outra versão é um algoritmo hibrido em consonância a Carvalho (2005).
- Melhorar interface gráfica dos algoritmos desenvolvidos, para a interação do usuário com o sistema, e mostrar os resultados em forma de gráficos ROC;
- Mostrar os resultados em forma de AUC, que é a área sobre uma curva ROC;
- Realizar novos experimentos com outras bases de dados que possuam classes desbalanceadas, e utilizar outros algoritmos na comparação;
- Utilizar algumas medidas de avaliação do grau de interesse do usuário nas regras descobertas, medidas *user-driven.*

Referências

BARROS, E.F., ROMÃO, W., CONSTANTINO, A. A.: "Pré-processamento para mineração de dados sobre beneficiários de planos de saúde suplementar." In: *Journal of Health Informatics*, pp. 19-26, 2011.

BATISTA, G.E.A.P.A., PRATI, R.C., MONARD, M.C.: "A study of the behavior of several methods for balancing machine learning training data." In: *ACM SIGKDD Explorations*, vol. 6, pp. 20-29, 2004.

BREIMAN, L., FRIEDMAN, J. H., OLSHEN, R. A., STONE, C. J.: "Classification and regression trees." In: *Wadsworth and Brooks*, Monterey, CA, 1984.

CARVALHO, D. R.: "Árvore de decisão / algoritmo genético para tratar o problema de pequenos disjuntos em classificação de dados." Tese de Doutorado, COPPE / UFRJ, 2005.

CARVALHO, D. R., FREITAS, A.A.: "A hybrid decision tree/genetic algorithm method for data mining". In: *Information Sciences Journal – special issue on Soft Computing Data Mining*, 2003.

CHAWLA, N. V., JAPKOWICZ, N., KOLCZ, A.: "Editorial: Special issue on learning from imbalanced data sets." In: *ACM SIGKDD Explorations*, vol. 6, no. 1, pp. 1-6, 2004.

CHAWLA, N. V., JAPKOWICZ, N., KOLCZ, A. *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Data Sets*, 2003.

CHAWLA, N. V., HALL, L. O., BOWYER, K. W., KEGELMEYER, W. P.: "SMOTE: synthetic minority oversampling technique". In: *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.

DE CASTRO, M. S. M.; CARVALHO, M. S.: "Agrupamento da classificação internacional de doenças para análise de reinternação hospitalares." In: *Cad. Saúde Pública*, vol. 21, pp. 317-323, 2005.

DORIGO, M., COLORNI, A., MANIEZZO, V.: "The Ant System: Optimization by a colony of co-operating agents." In: *IEEE Transactions on Systems, Man and Cybernetics, vol. 26, pp. 29-41*, 1996.

DORIGO, M., DI CARO, G.: “The ant colony optimization meta-heuristic. In: *New Ideas in Optimization”, pp.11-32*, 1999.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P.; UTHURUSAMY, R.: “Advances in knowledge discovery and data mining.” In: *American Association for Artificial Intelligence.* Menlo Park, CA: MIT Press, 1996.

FAYYAD, U.: “Mining Databases: Towards algorithms for knowledge discovery.” In: *Data Engineering IEEE Computer Society*. Washington, pp.39-48, 1998.

FAWCETT, T.: “ROC graphs – notes and practical considerations.” In: *Kluwer Academic Publishers, Netherlands,* 2004.

FREITAS, A.A.: “Data mining and knowledge discovery with evolutionary algorithms.” *Springer,* 2002.

HAND, D. J.: “Construction and assessment of classification rules.” In: *New York: John Wiley & Sons,* 1997.

HART, P. E.: “The condensed nearest neighbor rule.” In: *IEEE Transactions on Information Theory IT-14,* pp.515-516, 1968.

HOLLAND, J. H.: “Adaptation in natural and artificial systems.” In: *Cambridge, MA, USA: MIT Press*, 1992.

HUANG, J., LU, J., LING, C. X.: “Comparing Naïve Bayes, Decision Trees, and SVM with AUC and Accuracy”. In: *Proceedings of the Third IEEE International Conference on Data Mining*, pp. 553-556, 2003.

JAPKOWICZ, N.: *Proceedings of the AAAI’2000 Workshop on Learning from Imbalanced Data Sets, AAAI Tech Report WS-00-05,* AAAI, 2000.

JAPKOWICZ, N., JO, T.: “Class imbalances versus small disjuncts.” In: *ACM SIGKDD Explorations,* vol. 6, no. 1, pp. 40-49, 2004.

KOZA, J.: “Genetic programming: on the programming of computers by means of natural selection.” In: *The MIT press, 1992.*

KUBAT, M., MATWIN, S.: “Addressing the curse of imbalanced training sets: One-sided

selection." In: *Proceedings of the 14th International Conference on Machine Learning,* pp. 179-186, 1997.

LIU, B., ABBASS, H. A., MCKAY, B.: "Density-based heuristic for rule discovery with ant-miner" In: *Proc. 6th Australasia-Japan Joint Workshop on Intelligence Evolutionary System,* pp.180-184, 2002.

LIU, B., ABBASS, H. A., MCKAY, B.: "Classification rule discovery with ant colony optimization." In: *Proc. IEEE/WIC International Conference Intelligence Agent Technology,* pp. 83-88, 2003.

LIU, X., Wu, J., Zhou, Z.: "Exploratory under-sampling for class-imbalance learning." In: *IEEE Transactions on Systems, Man and Cybernetics, Part B,* pp. 1-14, 2008.

MACHADO, E. L.: "Um estudo de limpeza em base de dados desbalanceadas e com sobreposição de classes" *Dissertação de Mestrado, Programa de Mestrado em Informática, Universidade de Brasília, Brasília,* 2007.

MARTENS, D., BACKER, M. D., HAESEN, R., VANTHIENEN, J., SNOECK, M., BAESENS, B.: "Classification with ant colony optimization." In: *IEEE Transactions on Evolutionary Computation,* vol. 11, pp. 651–665, 2007.

MEYER, F.; PARPINELLI, R.S.: "Gui Ant-Miner: uma versão atualizada do minerador de dados baseado em colônias de formigas". In: *I Congresso Sul Catarinense de Computação: UNESC – Criciúma,* 2005.

NICKERSON, A., JAPKOWICZ, N., MILLOS, E.: "Using unsupervised learning to guide resampling in imbalanced data sets." In: *Proceedings of the 8th International Workshop on AI and Statistics,* pp. 261-265, 2001.

OTERO, F., FREITAS, A.A., JOHNSON, C.G.: "cAnt-Miner: an Ant Colony Classification algorithm to cope with continuous attributes." In: *Ant Colony Optimization and Swarm Intelligence. Lecture Notes in Computer Science,* vol. 5217, pp. 48-59, 2008.

PAPPA, G. L.: "Seleção de atributos utilizando algoritmos genéticos multiobjectivos." *Dissertação de Mestrado, Programa de Pós-graduação em Informática Aplicada,* Pontifícia Universidade Católica do Paraná, Curitiba, 2002.

PARPINELLI, R. S.: “Um algoritmo baseado em colônias de formigas para a classificação em data mining”. *Dissertação apresentada ao CEFET-PR para obtenção do título Mestre em Ciências.* Curitiba, 2001.

PARPINELLI, R.S., LOPES, H.S., FREITAS, A.A.: “Data mining with an ant colony optimization algorithm.” In: *IEEE Transactions on Evolutionary Computation*, vol. 6, pp.321–332, 2002.

PHUA, C., ALAHAKOON, D., LEE, V.: “Minority report in fraud detection: classification of skewed data.” In: *ACM SIGKDD Explorations,* vol. 6, no. 1, pp. 50-59, 2004

PRATI, R. C., BATISTA, G. E. A. P. A., MONARD, M. C.: “Curvas ROC para avaliação de classificadores” In: *IEEE Latin America Transactions,* vol.6, no. 2, pp. 215-222, 2008.

QUINLAN, J. R.: “C4.5: Programs for machine learning.” In: *Morgan Kaufmann Publishers,* San Mateo, CA, 1993.

ROMÃO, W.; FREITAS, A.A., GIMENES, I. M.: “Discovery interesting knowledge from a science & technology database with a genetic algorithm.” In: *Applied Soft Computing Journal,* vol. 4, no. 2, pp. 121-137, 2004.

SALAMA, K.M., ABDELBAR, A.M., FREITAS, A.A.: “Multiple pheromone types and other extensions to the ant-miner classification rule discovery algorithm.” In: *Swarm Intelligence,* vol. 6234, pp 1-34, Lecture Notes in Computer Science, 2011.

SMALDON, J.;FREITAS, A.A.: “A new version of the Ant-Miner algorithm discovering unordered rule sets”. In: *Proceedings Genetiv and Evolutionary Computation Conference (GECCO),* pp.889, 2006.

SIAU, K.: “Health care informatics.” In: *Information Technology in Biomedicine, IEEE Transactions on,* vol. 7, no. 1, pp. 1-7, 2003.

STUTZLE, T., HOOS, H. H.: “MAX-MIN Ant System”. In: *Future Generation Computation System,* pp. 889, 2000.

UCI (*University of California at Irvine*). Repositório de aprendizagem de máquina da Universidade Califórnia Irvine. Disponível em: http://archive.ics.uci.edu/ml/, 2012.

WEISS, G.: “Mining with rarity: a unifying framework.” In: *ACM SIGKDD Explorations*, vol. 6, no. 1, pp. 7-19, 2004

WITTEN, I. H., FRANK, E.: “Data Mining: practical machine learning tools and techniques.” In: *Morgan Kaufmann Publishers, Second Edition,* 2005.

YAN, R., LIU, Y., JUN, R., HAUPTMANN, A.: “On predicting rare classes with SVM ensembles in scene classification.” In: *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2003.

Apêndice A

C4.5

O algoritmo C4.5 utiliza a aprendizagem por árvores de decisão. Cada folha representa uma classe, cada nó de decisão especifica um teste a ser feito sobre um atributo que pode levar a uma folha ou alguma outra sub-árvore.

Árvores de decisão podem ser representadas como um conjunto de regras de classificação. Cada caminho iniciado no nó raiz e finalizado no nó folha representa uma regra de classificação.

Basicamente árvore de decisão constrói uma árvore recursivamente, conforme segue:

1. Um atributo é selecionado como nó raiz e ramificações são criadas para cada valor de atributo possível.
2. O conjunto de dados de treinamento é dividido em subconjuntos (um para cada ramo que estende do nó).
3. Pare se todas as instâncias (conjunto de dados) associadas a um ramo possuem a mesma classe.
4. Repita os passos 1 a 3 para cada ramo que possua mais de uma classe, usando apenas as instâncias a eles associadas.

A questão a ser analisada é qual atributo deve ser selecionado no passo 1. Deve ser selecionado aquele capaz de produzir a menor árvore de decisão. Uma boa heurística é escolher o atributo que produz os nós mais puros. Um nó é puro se as instancias associadas a ele possuem apenas uma classe. Frequentemente, entropia (medida da Teoria da Informação) é utilizada como medida de pureza e o ganho de informação maior como critério para selecionar o atributo a ser utilizado.

Cada caminho da árvore (que representa uma regra) pode ser expresso na forma de regras do tipo Se-Então.

k-NN

Usualmente, um algoritmo classificador precisa ter dois conjuntos de instâncias: o conjunto de treinamento e o conjunto de teste. O algoritmo utiliza o conjunto de treinamento para gerar o modelo que é utilizado para classificar casos do conjunto de teste. O classificador k-NN (do inglês, *Nearest Neighbor*) não gera modelo, pois a classificação de novos casos é feita de forma direta. Os casos de um conjunto de dados são interpretados como pontos distribuídos no espaço Euclidiano. Quando um novo caso é apresentado ao classificador k-NN, ele o classifica (associa a uma classe) conforme a classe de maior frequência entre seus k vizinhos mais próximos.

k-Means

Os algoritmos citados até agora fazem parte da tarefa de classificação em Mineração de Dados. O algoritmo *k-means* resolve a tarefa de clusterização (agrupamento).

Neste método, cada caso é visto como um ponto no plano cartesiano. O algoritmo consiste em dividir estes casos em k subconjuntos de casos mais próximos. Inicialmente, k casos da base de dados são eleitos, de forma aleatória, para serem os centroides $c_1, c_2, ..., c_k$ de cada cluster $cl_1, cl_2, ..., cl_k$, respectivamente. Um caso x pertencerá a um cluster cl_k, se, e somente se, a distância $d(x, c_i)$ for menor do que a distancia $d(x, c_m)$ para todo m, $m \neq i$. Depois de agrupado todos os casos em seus respectivos clusters, o centroide de cada cluster é recalculado. Para cada cluster cl_i, a média aritmética de seus casos é atribuída ao seu centroide c_i. Desta forma, cada cluster cl_i tem seu centroide c_i deslocado para o centro geométrico de seus casos. Novamente, todos os casos da base de dados são reagrupados ao cluster com centroide mais perto. Este processo é repetido até que não haja mais mudanças em nenhum valor dos k centroides.