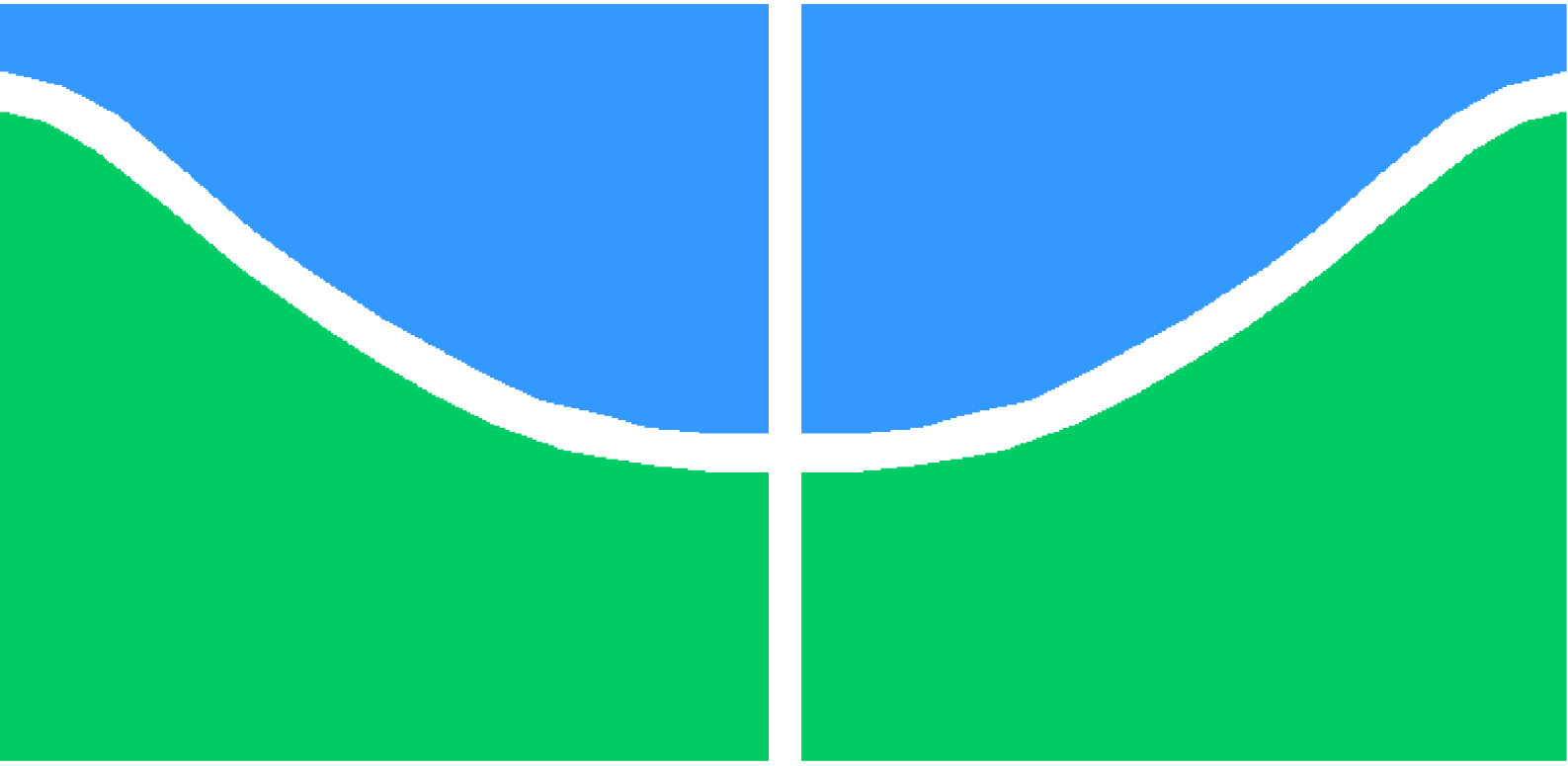

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Aprendizado de Máquina voltado para Mineração de Dados: Árvores de Decisão

Autor: Hialo Muniz Carvalho
Orientador: Prof. Dr. Nilton Correia da Silva

Brasília, DF
2014

Hialo Muniz Carvalho

Aprendizado de Máquina voltado para Mineração de Dados: Árvores de Decisão

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Nilton Correia da Silva

Brasília, DF

2014

Hialo Muniz Carvalho
Aprendizado de Máquina voltado para Mineração de Dados: Árvores de Decisão/ Hialo Muniz Carvalho. – Brasília, DF, 2014-
96 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Nilton Correia da Silva

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2014.

1. Aprendizado de Máquina. 2. Mineração de Dados. I. Prof. Dr. Nilton Correia da Silva. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Aprendizado de Máquina voltado para Mineração de Dados: Árvores de Decisão

CDU 02:141:005.6

Hialo Muniz Carvalho

Aprendizado de Máquina voltado para Mineração de Dados: Árvores de Decisão

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 16 de junho de 2014:

Prof. Dr. Nilton Correia da Silva
Orientador

Prof. Dr. Luiz Augusto Fontes Laranjeira
Convidado 1

Prof. Dr. Fabricio Braz
Convidado 2

Brasília, DF
2014

Resumo

Mineração de Dados é uma área que vem crescendo rapidamente nos últimos anos, devido ao crescimento do volume de dados gerado e a necessidade do mercado por ferramentas que sejam capazes de identificar e extrair conhecimento destes grandes repositórios de dados de forma automatizada. Grande parte deste avanço é oriundo de conceitos recebidos da área de Aprendizado de Máquina, principalmente no que se refere a métodos de classificação, predição e análise de grupos com base em dados históricos. Este trabalho apresenta um estudo voltado para a aplicação de conceitos de Aprendizado de Máquina para Mineração de Dados em uma abordagem que utiliza Árvores de Decisão em processos de classificações supervisionadas. Soluções alcançadas por meio de Árvores de Decisão podem ser utilizadas para análises de níveis de importância das diferentes características utilizadas na construção do modelo. Para tanto, o modelo será submetido a uma fase de análise estatística com o propósito de verificar sua acurácia em diferentes contextos.

Palavras-chaves: Mineração de Dados, Aprendizado de Máquina, Árvores de Decisão

Abstract

Data Mining is a research area that has been growing quickly in recent years due to the increasing volume of generated data and the market's needs for methods and tools that are able to identify and extract knowledge into these large repositories in an automated way. Much of this progress comes from concepts received from the Machine Learning area, mostly about classification and predition methods, and group analysis using historical data. This work shows a study on the application of concepts of Machine Learning on Data Mining, using a Decision Tree approach in supervised classification processes. Solutions achieved through Decision Trees can be used for analysis of the features significance levels used to construct the model. Thus, the model will be submitted to a statistical analysis phase in order to verify its accuracy in different settings.

Key-words: data mining, machine learning, decision trees

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

MD	Mineração de Dados
AM	Aprendizado de Máquina
SQL	*Structured Query Language*
ID	Identificador
EC	*Engenharia de Características*
ENIAC	*Electronic Numerical Integrator And Computer*
YaDT	*Yet Another Decision Tree*
ID3	*Iterative Dichotomiser 3*
CNH	Carteira Nacional de Habilitação
ROC	*Receiver Operating Characteristics*
MAPA	Ministério da Agricultura, Pecuária e Abastecimento
PNCRC	Plano Nacional de Controle de Resíduos e Contaminantes
SIF	Serviço de Inspeção Federal
SISRES	Sistema de Controle de Resíduos e Contaminantes
LMR	Limite Máximo de Resíduo

Sumário

1 Introdução

1.1 Motivação

Devido à crescente evolução da tecnologia em nossa sociedade, a quantidade de dados gerada cresceu de forma explosiva nos últimos anos. Segundo o Business Intelligence Observatory (2013), cerca de 90% dos dados de hoje foram gerados nos últimos dois anos, tendo como fontes documentos de textos, e-mails, streaming de vídeos e áudio, transações comerciais, telecomunicações e diversas outras. Unindo este crescimento gigantesco na quantidade de informação gerada com o grande progresso realizado na área de armazenamento de dados, o resultado é uma verdadeira mina de ouro escondida em terabytes, ou até mesmo petabytes de dados.

Ferramentas para análise destes dados a fim de se encontrar informações valiosas se tornaram necessárias, já que a quantidade de dados se tornou tão grande a ponto de ser uma tarefa impossível para um humano verificar toda a base de dados de uma grande empresa, por exemplo. Grandes empresas começaram a perceber a importância de se encontrar padrões em suas bases de dados, a fim de maximizar o lucro obtido de acordo com o comportamento de seus clientes. Motores de busca podem obter padrões de pesquisa interessantes, a fim de predizer certos eventos de acordo com o aumento em certos termos pesquisados. Uma rede social pode melhorar a sua política de anúncios de acordo com informações disponibilizadas pelas pessoas por meio de conversas ou postagens. As possibilidades são gigantescas, de acordo com o que se deseja procurar.

Com isso, a procura por ferramentas que sejam capazes de obter informações dentro de vastas Bases de Dados e organizar tais informações em conhecimentos úteis de modo a ser utilizado na tomada de decisões, vem crescendo cada vez mais. Com isso, a Mineração de Dados começou a ganhar força. Mineração de Dados é uma área voltada para o desenvolvimento de métodos de extração de padrões – não-triviais, implícitos, previamente desconhecidos e potencialmente úteis – de grandes bancos de dados (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996). E como a Mineração de Dados é uma área multidisciplinar, incorporando técnicas de diversas disciplinas, tais disciplinas também experimentaram um grande desenvolvimento em seus estudos. Uma das áreas beneficiadas por esta sede de conhecimento foi a área de Aprendizado de Máquina (AM).

A área de AM é considerada um ramo da área de Inteligência Artificial, sendo uma área especializada no estudo e construção de sistemas que sejam capazes de aprender de forma automatizada a partir de dados (BRINK; RICHARDS, 2014). Ou seja, a área de AM tem como foco a construção de modelos capazes de aprender a identificar padrões de

acordo com a sua capacidade de aprendizado com experiências passadas.

A disciplina de AM é muitas vezes confundida com a própria disciplina de Mineração de Dados em si, já que ambas possuem muitos conceitos em comum, e muitas vezes, são usadas em conjunto. Assim como a disciplina de Mineração de Dados, a área de AM vem ganhando cada vez mais importância no contexto tecnológico atual. Estima-se que o mercado de Inteligência Artificial movimentou 700 milhões de euros em 2013, e é esperado que este valor cresça de forma exponencial, chegando a exceder 27 bilhões de euros em 2015 (OBSERVATORY, 2013).

Portanto, é possível observar que as áreas de Mineração de Dados e AM (e por que não considerar a área de Inteligência Artificial como um todo) estão ganhando cada vez mais importância no contexto atual da área de tecnologia. E é muito comum ver estas duas áreas correlacionadas no desenvolvimento de novos algoritmos que facilitem a descoberta de novos padrões em grandes conjuntos de dados, visto que a área de AM "empresta" muitos de seus conceitos e definições para a área de Mineração de Dados, sendo considerada uma das disciplinas que mais contribuiu com a evolução da Mineração de Dados nos últimos anos (de acordo com Zhou (2003), "sem as técnicas de análise de dados avançadas doadas pela área de AM, a mineração de dados seria como buscar uma agulha em um palheiro").

Uma das técnicas oriundas da área de AM é o algoritmo de Árvores de Decisão, sendo que, provavelmente, é o algoritmo de AM mais estudado para aplicações em Mineração de Dados (WITTEN; FRANK, 2000). Os motivos são variados: é um modelo que possui suporte a diversos tipos de características (categóricas e numéricas), sua representação de conhecimento adquirido é facilmente compreendida

Dito isso, a motivação para o desenvolvimento deste trabalho consiste em realizar um estudo prévio sobre Árvores de Decisão, para o desenvolvimento de um modelo de AM que realize a mineração de dados em uma base de dados previamente definida por meio deste algoritmo.

1.2 Objetivos

Este trabalho tem como objetivo principal a implementação e validação de um sistema de Mineração de Dados, utilizando o algoritmo de Árvores de Decisão.

Em relação aos objetivos específicos, a preparação da Base de Dados para a análise também possui uma grande importância no processo de mineração. Portanto, as atividades de pré-processamento de dados, como limpeza, seleção e integração de dados, cujo objetivo é preparar a base de dados para que o modelo de AM desenvolvido realize a análise e a mineração de dados, também serão abordadas neste trabalho, juntamente com o processo

de Engenharia de Características (EC).

Após a preparação da base de dados para a mineração e implementação do modelo de AM, é necessário verificar se o mesmo obteve resultados consistentes. Portanto, a avaliação do modelo de AM gerado também faz parte dos objetivos do trabalho, mais especificamente, a verificação de resultados e análise de acurácia do modelo.

1.3 Estrutura do Trabalho

A fim de organizar o trabalho de forma a facilitar o entendimento do tema abordado, o conteúdo presente foi dividido nos seguintes tópicos:

- No capítulo 2 consiste na definição do que é Mineração de Dados, contendo uma explicação básica de diversos fatores que fazem parte da área em questão. Neste capítulo, são abordados tópicos como contexto histórico da Mineração de Dados, definição, descrição do processo, abordagens e métodos utilizados, além de correlacionar o tema de Mineração em si com a área de AM.
- No capítulo 3, é abordado o referencial teórico referente à área de AM. Neste capítulo, são abordados tópicos como aprendizado supervisionado e não-supervisionado, pré-processamento de dados, EC, algoritmos usados para o desenvolvimento de AM e verificação de um modelo de AM, por meio de técnicas estatísticas.
- No capítulo 4, é apresentada a definição de Árvores de Decisão, detalhando o funcionamento do algoritmo, cálculos para a construção da árvore, técnicas de poda, e o detalhamento do principal algoritmo para a construção de Árvores de Decisão, o C4.5.
- No capítulo 5, é apresentada a proposta detalhada do trabalho, o fluxo de trabalho proposto e detalhamento das atividades a serem realizadas.
- No capítulo 6, são apresentados os resultados obtidos nas bases de dados analisadas.
- No capítulo 7, são apresentadas as conclusões e o cronograma que foi seguido durante o trabalho.

2 Mineração de Dados

2.1 Contexto Histórico

Atualmente, pode-se considerar que esta é a Era da Informação. Quantidades massivas de dados são produzidas todos os dias: transações comerciais, resultados de pesquisas científicas, tráfego na Internet, armazenamento de informações como conversas na rede e e-mails, e muitos outros. Como a tecnologia, mais do que nunca, faz parte do cotidiano da vida das pessoas, a quantidade de dados gerados só tende a aumentar, e com a evolução da tecnologia, estes dados podem ser armazenados em grandes repositórios de dados (bancos de dados, *Data Warehouses*, entre outros). Porém, esta quantidade gigantesca de dados não possui importância significativa se estes dados não puderem ser analisados e entendidos. Devido à quantidade de dados, é uma tarefa muito difícil, para não dizer impossível, toda esta quantidade de dados ser analisada por humanos sem a ajuda de ferramentas voltadas para este fim.

Como mencionado por Witten e Frank (2000), vivemos em uma situação "rica em dados, mas pobre em informação". Ou seja, existe uma mina de ouro em dados brutos, mas sem as ferramentas necessárias, não é possível obter o que se tem de valioso dentro desta mina. Para suprir esta demanda, a área de gerenciamento de dados vem evoluindo sistematicamente, principalmente na subárea de análise de dados.

Segundo Han e Kamber (2001), após o estabelecimento de diversos sistemas de gerenciamento de banco de dados (DBMS), a tecnologia de banco de dados começou a evoluir para o desenvolvimento de sistemas de banco de dados avançados. Com isso, a Mineração de Dados começou a ganhar força, juntamente com a arquitetura de repositório de dados conhecida como *Data Warehouse*.

2.2 Data Warehouse

Segundo Han e Kamber (2001), *Data Warehouse* é um repositório com múltiplos dados, unificados sob um único esquema, facilitando o gerenciamento dos mesmos. Imagine uma grande rede de supermercados, com diversas unidades espalhadas por um país, cada uma com o seu banco de dados. Certamente lidar com cada uma destas fontes de dados em separado seria trabalhoso. A união de todos estes dados em um único local facilita a análise do grande volume de dados que estas unidades possuem.

O processo de criação de um *Data Warehouse* é conhecido como *Data Warehousing*. Este processo tem como objetivo principal integrar e gerenciar dados obtidos de diversas

fontes, a fim de se obter uma visão mais detalhada de parte ou de todo um negócio (CHAUDHURI; DAYAL, 1997).

Apesar de possuir eficientes métodos de OLTP (Online Transaction Processing, ou Processamento de Transações Online), a indústria precisava de algo ainda melhor. Com a adoção da arquitetura de *Data Warehouse*, foi possível a utilização de métodos de OLAP (Online Analytical Processing, ou Processamento Analítico Online), ou seja, técnicas de análise que permitem a visualização dos dados de maneira multidimensional. A principal diferença entre OLTP e OLAP é, que enquanto o primeiro método trabalha com cenários pré-definidos, como por exemplo, consultas de transações em bancos de dados relacionais, o segundo pode trabalhar com uma quantidade muito maior de dados em caminhos que não precisam ser pré-definidos, permitindo a manipulação e análise de dados de forma multidimensional, afetando uma quantidade de dados muito maior.

Segundo Rezende et al. (2003), as técnicas de análise de dados usadas em um *Data Warehouse* geralmente não extrapolam a realização de consultas SQL simples e a utilização de ferramentas OLAP. O problema na utilização desta abordagem, é que o usuário necessita ter conhecimento suficiente para realizar a análise dos dados. Como dito anteriormente, a quantidade de dados disponível hoje em dia torna esta tarefa algo quase impossível, já que muitos padrões escondidos nos dados não são encontrados devido ao volume. Portanto, a necessidade de se minerar estes dados foi se tornando cada vez mais necessária.

2.3 Definição

Na literatura, existem diversas definições para o que é conhecido como Mineração de Dados. A definição de Mineração de Dados segundo Han e Kamber (2001)seria “o processo de descoberta de padrões interessantes em grandes blocos de dados, armazenados em banco de dados, *Data Warehouses*, ou outros repositórios de informação”. Segundo Hand, Smyth e Mannila (2001), seria “a análise de grandes conjuntos de dados observacionais para encontrar relações não-suspeitas e resumir tais dados em estados que possam ser entendidos e utilizados pelo dono dos dados”.

Por ser um campo de estudo muito amplo, não é surpresa que este termo tenha tantas definições diferentes. Até mesmo o termo Mineração de Dados não engloba todo o processo de descoberta de informações. Muitos autores consideram a mineração de dados como um passo para o processo conhecido como KDD (*Knowledge Discovery from Data*, ou Descoberta de Conhecimento a partir de Dados, em tradução livre). É muito comum a confusão entre estes dois termos, pois não há um consenso em relação ao fato da Mineração de Dados ser um passo do processo de KDD, ou se o termo pode ser utilizado como sinônimo para este processo. O KDD é uma sequência iterativa de passos, que são

descritos a seguir, na Fig. (1).

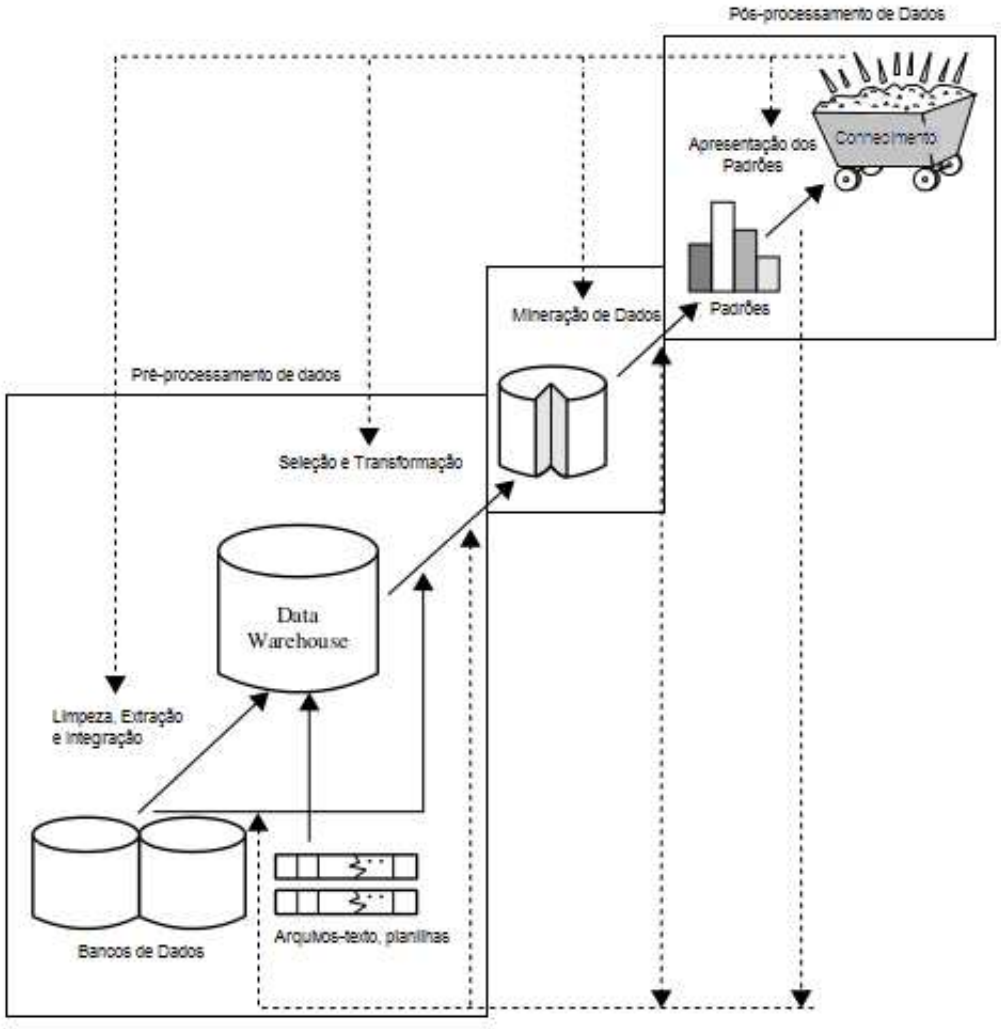

Figura 1: Sequência de passos do processo de KDD. (HAN; KAMBER, 2001)

Como pode ser visto, o KDD pode ser dividido em três grandes etapas: Pré-Processamento de Dados, Mineração de Dados e Pós-Processamento de Dados. Como a Mineração de Dados pode ser considerada o passo crucial do processo de KDD, devido a sua importância, acabou se tornando um sinônimo do processo completo. Neste trabalho, serão discutidas principalmente as etapas de Pré-Processamento de Dados e a atividade de Mineração de Dados em si. Portanto, o termo Mineração de Dados será utilizado como sinônimo do processo de KDD, e não apenas como parte do processo. Conforme mencionado por Han e Kamber (2001), o termo Mineração de Dados está se tornando mais popular que o termo KDD. Portanto, este termo será utilizado para descrever o processo completo.

Com base na Fig. (1), os passos de 1 a 4 são formas de se preparar os dados para serem analisados e minerados de forma eficiente, sendo chamados de Pré-Processamento de Dados. Apesar de alguns autores (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996)

dividirem esta etapa de forma diferente, a etapa de Pré-Processamento de Dados geralmente consiste nestes quatro passos: Extração e Integração, Limpeza, Seleção e Transformação. É interessante notar que, apesar de não serem obrigatórios, geralmente os passos de Limpeza e Extração e Integração são realizados durante a execução do processo de Data Warehousing.

O objetivo da Extração e Integração de Dados é unir os dados em uma única fonte. Muitas vezes, os dados a serem minerados estão armazenados em diversas fontes, como vários bancos de dados, arquivos-texto, planilhas, e outras fontes. Isto ocorre com muita frequência, principalmente em grandes empresas, com diversas filiais espalhadas, aonde cada uma possui um modo específico de armazenamento de dados. Além dos dados estarem armazenados de diversas maneiras, tais dados podem estar armazenados de formas diferentes nestas fontes. Por exemplo, uma filial de um supermercado pode manter os registros de venda de acordo com o ID do comprador, número de itens comprados e valor total da compra, enquanto outra filial mantém os registros usando apenas o ID do comprador e o valor total da compra. Portanto este passo é necessário para juntar os dados de diversas fontes em um único local, em um formato padrão que possa ser utilizado como entrada para as ferramentas de análise. Como pode ser visto, este passo pode ser considerado como a atividade principal no processo de Data Warehousing, que consiste na união de diversas fontes de dados em um único repositório.

A Limpeza de Dados é executada principalmente para remover possíveis inconsistências nos dados, ou ruído, que possam vir a atrapalhar o processo de descoberta de padrões. Uma atividade repetitiva como a obtenção dos dados podem vir a trazer dados incorretos, como registros com valor inválido, ou até mesmo instâncias completas inconsistentes, ainda mais se a atividade de obtenção dos dados não for realizada de forma automatizada. Como os objetivos deste passo são identificar e eliminar estes possíveis erros é recomendável que esta tarefa seja realizada por alguém que possua conhecimento do domínio do qual os dados pertencem.

A Seleção de Dados consiste em analisar quais dados podem ser importantes para o processo de mineração, e remover o que não possui relevância para a atividade. Dependendo dos padrões que se deseja obter de um conjunto de dados, muitas informações não possuem relevância para a análise destes padrões, ou estão de tal forma incompletos que a sua utilização pode vir a prejudicar os resultados obtidos com a análise.

Este passo deve ser realizado com cautela, pois dependendo da quantidade de dados a serem analisados, principalmente se tais dados possuem diversas features (características), alguns dados removidos podem eliminar certos padrões que seriam encontrados por meio da mineração destes dados. Conforme mencionado por Brink e Richards (2014), ao selecionar uma série de características, o responsável pode se encontrar em um dilema: ao utilizar todo tipo de característica, inclusive as que não possuem relevância à análise

de padrões desejada, o modelo de análise pode ter a precisão prejudicada, pois o modelo não será capaz de diferenciar o que é um padrão e o que pode vir a ser apenas o que se considera ruído, ou seja, conjuntos de dados que destoam dos padrões que estão presentes na base de dados. Por outro lado, selecionar apenas um pequeno conjunto de características pode vir a omitir diversos padrões que possam vir a ser de grande importância. Ou seja, a precisão da análise é prejudicada novamente, já que informações essenciais para a descoberta de padrões (por mais que não pareçam) foram eliminadas da análise. Assim como a Limpeza de Dados, é recomendável que a Seleção seja realizada por alguém que possua conhecimento do domínio em que os dados se encontram.

O último passo do Pré-Processamento é a Transformação de Dados. Este passo consiste em transformar e consolidar os dados em formas apropriadas para a mineração, por meio de operações de redução e/ou agregação (HAN; KAMBER, 2001). Segundo Weiss e Indurkhya (1995), a redução de dados pode ser feita de três maneiras: reduzindo o número de instâncias, o número de atributos das instâncias ou os valores de um atributo de uma instância, geralmente por meio de discretização dos valores. Ao se usar a primeira alternativa, deve-se ter cuidado ao remover certos exemplos a fim de se manter as características do conjunto de dados original. Ao se remover o número de atributos das instâncias, isto pode ser feito por meio da remoção de atributos que não possuam importância nos exemplos em si, ou seja, que não afetem o resultado final, ou por meio da agregação de dois ou mais atributos em um novo atributo. A terceira forma consiste na discretização ou normalização de valores de atributos, sendo a mais utilizada. Valores contínuos podem ser discretizados para certos intervalos. E como diversos valores podem ter diferentes pesos na análise, geralmente tais valores são normalizados para um intervalo único, como por exemplo, de 0 a 1.

Além dos quatro passos descritos acima, existe um novo passo que pode ser realizado para melhorar a qualidade dos dados que serão analisados nesta etapa de Pré-Processamento. Este passo é chamado de EC. Resumidamente, esta etapa consiste em analisar os dados de forma a obter novos dados que possam ajudar a melhorar a descoberta de padrões. Como esta etapa é parte essencial da abordagem de AM, a EC será abordada com maior detalhe no capítulo 3.

Após a conclusão desta etapa, os dados estão prontos para o processo de mineração. Este processo pode ser realizado de diversas maneiras, que serão descritas no próximo tópico.

2.4 Técnicas e Métodos

Segundo Zhou (2003), a área de Mineração de Dados recebeu diversas contribuições de várias áreas, como Estatística, Computação Distribuída, ramos da Inteligência Artificial

como AM, entre outros. Portanto, é comum observar certos padrões presentes em outras áreas ao estudar Mineração de Dados. Este trabalho é focado na utilização da Mineração de Dados usando abordagens presentes na área de AM.

Diversos tipos de dados podem ser trabalhados utilizando mineração de dados, a fim de se obter padrões que possam ser úteis. Por exemplo, podem-se minerar dados brutos sobre previsão do tempo para descobrir certos padrões de chuva em determinada época do ano, ou para se tentar prever o tempo no dia seguinte com base em dados de anos anteriores. Existem diversos tipos de padrões que podem ser obtidos em uma certa base de dados, e o padrão a ser encontrado depende do tipo de dado a ser analisado e mais importante, o que se deseja descobrir sobre tais dados.

Existem alguns tipos principais de técnicas para a obtenção de padrões em certa base de dados. Tais padrões podem ser classificados em:

- Caracterização.
- Discriminação.
- Regras de Associação.
- Classificação e Regressão.
- Clustering.

Estas técnicas podem ser divididas em dois grupos principais: técnicas para mineração de dados preditiva e mineração de dados descritiva. A mineração de dados preditiva é utilizada para induzir modelos ou teorias (como árvores de decisão ou conjunto de regras) a partir de dados classificados em conjuntos (FURNKRANZ; GAMBERGER; LAVRAC, 2012) . Estes modelos então são utilizados para realizar predições, auxiliando à tomada de decisões. As principais técnicas de mineração de dados preditiva são: classificação, regressão e clustering. Já na mineração de dados descritiva, os dados são categorizados de acordo com as suas características, sendo úteis em atividades de exploração de dados. As principais técnicas de mineração de dados descritiva são: caracterização, discriminação e regras de associação.

Apesar de este tópico tratar de forma breve estas seis técnicas, a proposta deste trabalho envolve a utilização da técnica de Classificação, com a utilização do algoritmo de Árvores de Decisão. Ambos serão descritos mais detalhadamente no capítulo 4.

2.4.1 Caracterização e Discriminação

A Caracterização de dados consiste em descrever uma classe de certo conjunto de dados, de acordo com o que se deseja descobrir. A caracterização consiste em resumir

um conjunto de dados em termos gerais, a fim de se facilitar o entendimento daquela classe. A discriminação de dados é a comparação das características de uma determinada classe com uma ou mais classes, chamadas de classes de contraste. A saída destas técnicas pode ser expressa na forma de regras ou gráficos de comparação, no caso da técnica de Discriminação (HAN; KAMBER, 2001).

2.4.2 Regras de Associação

Segundo Furnkranz, Gamberger e Lavrac (2012), a técnica de Regras de Associação pode ser definida como: dado uma série de exemplos (instâncias), no qual cada exemplo é uma série de características, uma Regra de Associação é expressa na forma LHS -> RHS (Left Hand Side e Right Hand Side, respectivamente). LHS e RHS podem ser uma ou mais características, e a regra geralmente é expressa na forma SE... ENTÃO.

Regras de associação são amplamente utilizadas na descoberta de padrões de comportamento em um conjunto de dados. A técnica de Regras de Associação difere da técnica de Classificação pelo motivo que tanto LHS quanto RHS podem ter uma série de condições, não havendo nenhuma distinção entre classes. Uma classe pode aparecer de qualquer lado da regra, e pode haver uma série de classes em apenas uma regra de associação.

Para facilitar o entendimento e explicar como que as muitas regras de Associação são refinadas, considere o exemplo abaixo. Durante a análise de um conjunto de dados de uma grande rede de supermercados, a seguinte Regra de Associação foi encontrada:

compra (X, “notebook”) ˆ compra (X, “impressora”) -> sexo (X, “masculino”) [suporte = 4/40, confidência = 4/5]

Na qual X é uma variável representando o consumidor. Esta regra pode ser lida da seguinte forma:

SE compra = “notebook” **E** compra = “impressora”, **ENTÃO** sexo = “masculino”

Sendo a forma mais comum de se representar uma Regra de Associação. Existem dois valores importantes a serem considerados ao se avaliar uma Regra de Associação: suporte e confidência. O suporte de uma Regra é a quantidade de instâncias que confirmam a Regra (ou seja, a quantidade de instâncias em que um consumidor do sexo masculino comprou um notebook e uma impressora), de ambos os lados, em relação à quantidade total de instâncias. Ou seja, de 40 instâncias testadas, 4 possuem as características necessárias para a confirmação da regra.

Já a confidência de uma Regra é a quantidade de instâncias que possuem todos os valores verdadeiros do lado esquerdo da Regra (LHS) em relação à quantidade de instâncias que possuem valores verdadeiros dos dois lados da regra. Voltando ao exemplo, de 5 instâncias nas quais o consumidor comprou um notebook e uma impressora, 4 destes

consumidores são do sexo masculino.

O suporte e a confidência de uma Regra geralmente são utilizados como parâmetros para definir se uma Regra pode conter um padrão interessante ou não. Como diversas Regras podem ser encontradas, de acordo com a quantidade de características de uma instância, é comum que se determine um limiar para determinar se a Regra possui uma informação interessante ou não. De volta ao exemplo, supõe-se que o limiar determinado para o suporte de uma Regra seja 15% e o limiar de confidência seja 80%. Como a regra não alcançou o limiar de suporte, a mesma é descartada.

2.4.3 Classificação e Regressão

As técnicas de classificação e regressão são de certa forma, conceitualmente similares. Porém, nas áreas de AM e Mineração de Dados, a maioria das pesquisas é voltada para problemas de classificação, que são mais comuns na vida real do que problemas de regressão Weiss e Indurkhya (1995).

Classificação é o processo de identificar um modelo (ou função) que descreve e identifica classes de dados. O modelo é derivado com base na análise de uma série de dados de teste (conjuntos de dados aonde as suas classificações são conhecidas) (HAN; KAMBER, 2001). O modelo então é utilizado para predizer a classe de um conjunto de dados que ainda não foi classificado. Resumidamente, a técnica de classificação mapeia conjuntos de dados de entrada em categorias, ou classes, de acordo com um modelo criado após o treinamento com um conjunto de dados cuja classificação é conhecida. Este conceito é conhecido como treinamento supervisionado, e será abordado com detalhes no capítulo 3.

Já a técnica de regressão se diferencia da técnica de classificação no fato em que o resultado predito é um valor contínuo, e não discreto (como a divisão em classes, como é realizada na técnica de classificação). Por exemplo, uma determinada rede de lojas deseja prever qual será a quantidade que certo produto irá vender em um mês com base nas vendas deste mesmo produto nos anos anteriores. Ou seja, a técnica de regressão é usada para predizer valores numéricos. Geralmente, as técnicas de regressão consistem em encontrar uma relação entre um conjunto de dados de entrada e uma saída numérica.

Técnicas de classificação e regressão podem trabalhar tanto com modelos lineares quanto não-lineares. Os algoritmos mais usados na criação de modelos que utilizam estas técnicas serão descritos no capítulo 3.

2.4.4 Clustering

Em alguns problemas de mineração de dados, as classes de determinado conjunto de dados podem não ser conhecidas, sendo necessário dividir as instâncias de acordo com as suas similaridades. As técnicas de clustering são aplicadas nestas situações, agrupando

as instâncias de dados em grupos em que as instâncias pertencem a aquele grupo possuam alta similaridade. Cada cluster criado pode ser visto então como uma classe de objetos.

Esta divisão pode ser feita de diversas formas: cada grupo pode ser exclusivo, ou seja, cada instância pertence a apenas determinado cluster. Pode-se também utilizar a abordagem probabilística, aonde cada instância possui uma probabilidade de pertencer a cada grupo. A abordagem utilizada depende da necessidade de divisão das instâncias, ou seja, como que se deseja encontrar as diferentes divisões que podem ocorrer em uma base de dados, sendo que cada abordagem possui o seu próprio método de divisão.

O algoritmo mais simples e conhecido da técnica de clustering é o algoritmo de particionamento, também conhecido como k-médias. De forma resumida, este algoritmo determina k pontos de forma aleatória, que serão os centróides do cluster. Calculando a distância Euclidiana de cada instância de dados, cada instância é designada para o cluster mais próximo. Após todas as instâncias serem designadas, a média das distâncias das instâncias em cada cluster é calculada e uma nova iteração do algoritmo é realizada, utilizando a média calculada como o novo centróide do cluster. O algoritmo é repetido até que o centróide do cluster não se modifique. A Fig.(2) demonstra o funcionamento do algoritmo para k = 3.

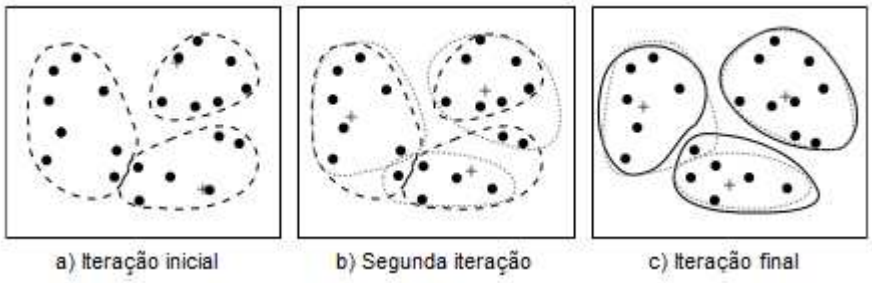

Figura 2: Algoritmo de k-médias para k = 3. (HAN; KAMBER, 2001)

Outros algoritmos de clustering conhecidos são: algoritmos baseados em hierarquia (Chameleon, BIRCH), algoritmos baseados em densidade (DBSCAN, OPTICS). (HAN; KAMBER, 2001) apresentam a descrição e exemplos de funcionamento de muitos destes algoritmos.

2.5 Pós-Processamento de Dados

Após a mineração de dados ser realizada, pode ser necessário uma nova etapa de processamento nos dados que foram obtidos na mineração. Esta etapa é conhecida como Pós-Processamento de Dados, e consiste em transformar os dados obtidos na mineração de dados em padrões observáveis e que possam ser entendidos facilmente.

Esta etapa pode ser dividida em dois passos: selecionar quais padrões são realmente interessantes para o domínio dos dados, chamada de Preparação de Padrões, e apresentar este padrões em forma de conhecimento. Dependendo da técnica que se queira utilizar para realizar a mineração, muitos padrões podem ser encontrados, sendo que apenas alguns deles são realmente importantes ou novos. Então, a seleção destes padrões é necessária.

Um modo de se selecionar padrões de acordo com a sua importância para o usuário final, seria os limiares de suporte e confidência vistos na técnica de Regras de Associação. Ao determinar um limite mínimo para os dois parâmetros, o número de Regras obtido pode ser drasticamente reduzido, além de filtrar para que apenas padrões que atendam uma quantidade razoável de instâncias sejam obtidos. Outra forma muito conhecida de se selecionar padrões é a "poda" de árvores de decisão. As árvores de decisão geradas podem ser muito complexas, sendo necessário "podar" alguns nós e folhas a fim de se facilitar o entendimento da mesma. Além disso, este procedimento ajuda a evitar o conhecido problema de overfitting. Este assunto será tratado no capítulo 4.

Ao se selecionar quais padrões são mais importantes, estes padrões devem ser apresentados de uma maneira em que o usuário final seja capaz de interpretá-los de forma simples. Esta etapa é conhecida como Visualização do Conhecimento. Nesta etapa, o conhecimento adquirido por meio dos padrões obtidos é apresentado, por meio de gráficos, regras do tipo SE... ENTÃO, grafos, redes e etc., dependendo do tipo de padrões que foram descobertos durante a atividade de Mineração de Dados.

2.6 Mineração de Dados e Aprendizado de Máquina

Como mencionado em tópicos anteriores, a área de Mineração de Dados foi beneficiada com diversos conceitos oriundos da área de AM. Muitos dos conceitos vistos neste capítulo são vistos em livros voltados para a área de AM, como as técnicas de Classificação, Regressão, Regras de Associação e Clustering. Os dois grupos principais de Mineração de Dados, preditiva e descritiva, são semelhantes à maneira em que os tipos de Aprendizado são divididos: Aprendizado Supervisionado e Não-Supervisionado.

Portanto, o próximo capítulo trata de alguns conceitos que foram mencionados neste capítulo, sendo que tais conceitos serão abordados de forma mais detalhada, voltada para a área de AM.

3 Aprendizado de Máquina

3.1 Contexto Histórico

Assim como a área de Mineração de Dados, a área de AM ganhou muita força com o advento da "Era da informação", onde a tecnologia está cada vez mais presente em nosso cotidiano. De acordo com o Business Innovation Observatory (2013), o valor investido no mercado de AM em 2013 chegou a 700 milhões de euros, e a expectativa é que este valor cresça de forma exponencial nos próximos anos, chegando a 27 bilhões de euros em 2015. Ou seja, é possível verificar que esta área vem se expandindo de forma significativa nos últimos anos.

A área de AM é considerada um ramo da área de Inteligência Artificial, sendo uma área especializada no estudo e construção de sistemas que sejam capazes de aprender de forma automatizada a partir de dados (BRINK; RICHARDS, 2014). Esta capacidade de aprender com experiências passadas é algo que é desejado desde a criação dos primeiros computadores, como o ENIAC, na década de 40. O benefício que uma máquina pudesse aprender como um humano seria imensurável: sistemas capazes de realizar diagnósticos com base em históricos médicos, sistemas que otimizassem o uso de energia elétrica em uma residência, de acordo com o modo em que seus ocupantes usam, e muitas outras.

Apesar de uma máquina ainda não possuir a capacidade de aprendizado de um humano, a área de AM caminha a passos significativos. Diversos algoritmos funcionam de forma eficiente em certas tarefas de aprendizado, como por exemplo, detecção de spams (SILVA; YAMAKAMI; ALMEIDA, 2012) e reconhecimento de letras escritas à mão (AHMAD MARZUKI KHALID, 2002). E uma das áreas que mais se beneficiam com o uso de AM é a área de Mineração de Dados. Durante este capítulo, será possível notar o motivo pelo qual muitas pessoas ainda confundem estas duas disciplinas: grande parte dos conceitos apresentados aqui foram mencionados no capítulo anterior.

3.2 Definição

Segundo Brink e Richards (2014), AM é "uma abordagem guiada a dados para a resolução de problemas, na qual padrões ocultos em um conjunto de dados são identificados e utilizados para ajudar na tomada de decisões". Já Mitchell (1997), apresenta uma definição mais formal sobre a área: "Diz-se que um sistema é capaz de aprender com experiência E, com respeito a um grupo de tarefas T e índice de performance P, se a performance medida por P em T aumenta com E".

Ou seja, pode-se considerar que um sistema "aprende" se o mesmo consegue melhorar sua performance em determinada tarefa à medida em que o mesmo é alimentado com dados de experiências passadas. Então, é possível verificar que, para a criação de um sistema que utilize esta abordagem, são necessários três pontos: o grupo de tarefas (o que se espera que o modelo de AM faça), o índice de performance do modelo (o nível de precisão com o qual o modelo é capaz de resolver as tarefas delegadas ao mesmo) e uma fonte de dados em que o modelo possa aprender (que seriam as experiências passadas).

Como exemplo, tem-se um sistema que reconheça letras do alfabeto escritas à mão. O grupo de tarefas que se espera do sistema é simples: que ele reconheça e classifique letras do alfabeto escritas à mão. A porcentagem de acertos do sistema seria o índice de performance, e uma base de dados contendo uma série de imagens de letras, já classificadas de acordo com o seu significado, seria a fonte de dados.

O principal objetivo de um algoritmo de AM é adquirir a capacidade de generalizar além das instâncias presentes no conjunto de dados de treinamento. Segundo Domingos (2012), um erro comum cometido por iniciantes na área de AM é achar que, pelo seu modelo ter sido eficiente na etapa de treinamento, o modelo gerado é eficiente em um conjunto de dados real. Por isso, é necessário possuir um conjunto de dados separado para testes, a fim de evitar um sério problema conhecido como *overfitting* (que será tratado mais a frente).

3.3 Abordagens de AM

3.3.1 Aprendizado Supervisionado

Algoritmos de Aprendizado Supervisionado são os mais comuns na área de AM, sendo que a maioria dos problemas em que a área de AM pode ajudar a resolver é supervisionada por natureza (BRINK; RICHARDS, 2014). Juntamente com a abordagem de Aprendizado Não-Supervisionado, a grande maioria dos algoritmos de AM pertence a estes dois grupos. Existem outras abordagens menos usuais, como Aprendizado Semi-Supervisionado e Classificação de Multi-Classes, que não são o foco deste trabalho. Portanto, apenas as duas primeiras abordagens serão tratadas neste capítulo.

A abordagem de Aprendizado Supervisionado consiste em utilizar uma série de exemplos (chamados de instâncias), já classificados, para induzir um modelo que seja capaz de classificar novas instâncias de forma precisa, com base no aprendizado obtido com o conjunto de dados de treinamento. É comum que este modelo seja chamado também de classificador. Portanto, é importante que exista um conjunto de dados de treinamento de qualidade, para que o modelo criado possa ser capaz de predizer novas instâncias de forma eficiente.

Um classificador é um modelo que recebe como entrada uma série de características, que consistem de valores discretos e/ou contínuos, geralmente na forma de um vetor, e retorna como saída um único valor discreto, a classe da instância que foi utilizada como entrada. A Eq. (3.1) exemplifica o processo de um classificador:

$$Entrada : \{x_i, y_i\}, aonde x_i = \{x_{i1}, x_{i2}, ..., x_{in}\} e y_i é a saída da esperada \tag{3.1}$$

Com base nesta definição, é possível notar que a abordagem de Aprendizado Supervisionado é bem semelhante à definição da técnica de Mineração de Dados Preditiva. De fato, ambas possuem o mesmo objetivo: induzir modelos ou teorias (como árvores de decisão ou conjunto de regras) a partir de dados classificados em conjuntos (FURNKRANZ; GAMBERGER; LAVRAC, 2012). Estes modelos então são utilizados para realizar predições. É comum encontrar na literatura autores que considerem estas duas abordagens sinônimas (DOMINGOS, 2012).

3.3.2 Aprendizado Não-Supervisionado

Assim como a abordagem de Aprendizado Supervisionado é considerada sinônimo da Mineração de Dados Preditiva, a abordagem de Aprendizado Não-Supervisionado equivale à técnica de Mineração de Dados Descritiva, ou seja, é uma abordagem utilizada para descoberta de padrões em dados não-categorizados, sendo muito utilizado para atividades de exploração de dados.

O conjunto de dados utilizado na abordagem de Aprendizado Não-Supervisionado não possui classificação, ou seja, a saída do conjunto de dados de treinamento não possui uma saída esperada para cada uma de suas instâncias. É a abordagem indicada quando o objetivo do sistema não é construir um modelo de predição, e sim um modelo cuja função seja encontrar regularidades nos dados que possam vir a ser úteis.

Apesar de frequentemente ser utilizada para atividades de exploração de dados (FURNKRANZ; GAMBERGER; LAVRAC, 2012), em conjunto com algoritmos supervisionados para criar conjuntos de dados de teste ou encontrar novos padrões em determinado conjunto de dados, o Aprendizado Não-Supervisionado também pode ser utilizado para fins preditivos. A técnica de *Clustering* é uma técnica típica de Aprendizado Não-Supervisionado, mas que pode ser utilizada para classificar uma série de instâncias em um determinado subgrupo: neste caso, o *cluster* seria uma espécie de classe para a instância.

Como o foco deste trabalho é a respeito dos algoritmos de Aprendizado Supervisionado, mais especificamente os algoritmos de Classificação, os algoritmos de Aprendizado Não-Supervisionado não serão tratados neste trabalho. No capítulo 2, há um detalhamento breve do algoritmo mais utilizado de Aprendizado Não-Supervisionado, o algo-

ritmo de particionamento, também conhecido como k-médias. Para maiores detalhes, ver (HAN; KAMBER, 2001).

3.4 Entradas de um Modelo de AM

Para que um modelo de AM obtenha êxito ao analisar uma vasta quantidade de dados, não basta apenas escolher um algoritmo que realize as predições necessárias para responder as perguntas esperadas: é necessário que estes dados passem por um processo de pré-processamento para que possam alimentar o modelo de AM projetado.

Segundo Domingos (2012), iniciantes na área de AM se surpreendem com quão pouco tempo em um projeto de AM é gasto com a implementação do modelo de AM . De fato, a preparação dos dados para o modelo de AM leva uma quantidade considerável de tempo e esforço. A etapa de Pré-Processamento de Dados do processo de KDD visto no capítulo 2, também se encaixa na área de AM: uma base de dados no mundo real muitas vezes possui muitas inconsistências que precisam ser fixadas a fim de não afetar a precisão do modelo de AM, por meio de atividades como Seleção, Limpeza e Transformação de Dados.

Os maiores problemas encontrados em bases de dados são instâncias com características incompletas ou ausentes. Abaixo, será visto como lidar com estes problemas, além de descrever brevemente o processo de EC, fator importante na otimização de um modelo de AM.

3.4.1 Tipos de Características

Geralmente, os algoritmos utilizados para criação de modelos de AM suportam dois tipos de características: numéricas e categóricas. As características numéricas, como o próprio nome sugere, são representadas por um valor numérico, já as características categóricas são caracterizadas por dividir seus valores em diferentes categorias, cuja ordem não importa (por exemplo, o sexo de uma pessoa pode ser Masculino ou Feminino: duas categorias diferentes).

Em relação às características numéricas, é necessário entender como que o algoritmo de AM utilizado lida com este tipo de característica. Geralmente, tais algoritmos lidam com características tratando-as como escalas ordinárias, ou escalas de razão (WITTEN; FRANK, 2000). Algoritmos que tratam características numéricas como escalas de razão, como o algoritmo K-Vizinho Mais Próximo (que utiliza a métrica da Distância de Euclides como parâmetro), precisam que as características numéricas de uma instância passem por um processo conhecido como normalização. A normalização apenas modifica a variação de todas as características para uma variação única, geralmente de 0 a 1.

Já as características categóricas inspiram um pouco mais de cuidado. Alguns algoritmos, como Árvores de Decisão, possuem suporte para características categóricas, não sendo necessário nenhum tipo de processamento. Mas grande parte dos algoritmos de AM, como Regressão Linear e K-Vizinho mais Próximo, trabalha apenas com valores numéricos. Portanto, é necessário transformar as características categóricas em uma espécie de característica numérica.

A saída mais simples para este tipo de problema é realizar esta transformação de forma binária: se uma instância possui uma característica com k categorias, k novas colunas serão criadas. A coluna equivalente à categoria em que a característica se encontra é marcada com o valor 1 e as $k - 1$ colunas restantes são marcadas com um 0. Apesar de ser um método rudimentar, é considerado eficiente.

3.4.2 Características Ausentes

É comum em bases de dados alguns dados estarem ausentes, seja por problemas no recolhimento dos mesmos, medição ineficiente, entre outros. Estas características devem ser tratadas antes do modelo de AM recebê-las, pois estas ausências podem vir a afetar a eficiência e precisão do modelo.

Primeiramente, é necessário avaliar o seguinte aspecto: a característica ausente é significativa para o aprendizado do modelo? De acordo com a resposta a esta pergunta, as características ausentes podem ser tratadas de forma diferente.

Caso a característica seja significativa para o modelo, estas características recebem o seu próprio significado: seja uma nova categoria (por exemplo, "Ausente"), caso a característica seja categórica, ou um valor fora da variação padrão de uma característica numérica, como -1.

O caso em que se deve tomar mais cuidado é o que a resposta à pergunta anterior seja "não". Como a característica não possui significância para o modelo, não é possível criar uma categoria/novo valor para a mesma, pois ao fazer isso, existe o risco de se adicionar um padrão não-existente ao modelo. Remover todas as instâncias que possuem características ausentes resolveria o problema, porém na maioria das vezes acaba por trazer mais problemas: caso as características ausentes não sejam uniformes em relação ao resto das instâncias, isso acaba por modificar o padrão da base de dados, além de diminuir a precisão do modelo.

Portanto, o que resta é tentar "adivinhar" um valor para a característica, que não afete a consistência da base de dados. Os métodos mais utilizados são substituir o valor da característica pela média das características que possuem valor definido, ou então utilizar o valor da instância anterior. Assim como a escolha do algoritmo, não existe uma "bala de prata" ao se lidar com características ausentes. Estes casos devem ser analisados com

cuidado e, se possível, por alguém que possua conhecimento do domínio a qual os dados pertencem.

3.4.3 Engenharia de Características

Segundo Brink e Richards (2014), EC é "a utilização de características existentes para gerar novas características que aumentem o valor da base de dados original, utilizando conhecimento dos dados ou do domínio em questão".

É nesta etapa que o valor de um especialista no domínio dos dados a serem minerados se torna significativo. Com base no conhecimento que este especialista possui no domínio dos dados, algumas características inicialmente sem significado para o modelo de AM (como localização ou datas de determinado evento), podem trazer diversos padrões antes ocultos, após serem transformados em dados que possam ser utilizados pelo modelo de AM.

Portanto, o primeiro passo para se realizar EC em uma base de dados é entender o escopo dos padrões que se deseja obter desta base de dados. Ou seja, entender o problema que se deseja resolver e o que se espera obter ao final da resolução do mesmo. Com este conhecimento, gera-se uma série de experimentos para a resolução do problema e analisam-se os resultados. Com base nestes resultados, se obtém mais conhecimento dentro do escopo, o que leva a uma nova série de experimentos.

Segundo Domingos (2012), AM não consiste em apenas criar um modelo e alimentá-lo com dados, e sim "um processo iterativo em que o modelo é criado, alimentado, os resultados são analisados, então se modifica o conjunto de dados e/ou o modelo, e repete-se o processo".

Portanto, pode-se dizer que EC nada mais é que um ciclo, contendo os seguintes passos:

- Selecionar uma série de características que possam ser significativos para o modelo de AM.
- Gerar o modelo de AM utilizando um conjunto de dados de teste que contenham instâncias com estas características.
- Verificar a eficiência e precisão do modelo.
- Com base nos resultados obtidos, verificar quais novas características podem melhorar o modelo criado, e realizar novamente todos os passos.

Porém, deve-se tomar cuidado com a análise das características utilizadas. Conforme mencionado no capítulo 2, dependendo das características removidas/adicionadas, a

precisão do modelo pode ser prejudicada, seja por *overfitting* ou *underfitting* (que também será tratado mais a frente).

3.5 Algoritmos de Aprendizado Supervisionado

Na área de AM, Classificação pode ser descrita como a predição de novos dados em baldes (classes) utilizando um classificador modelado por um algoritmo de AM (BRINK; RICHARDS, 2014). É a técnica mais utilizada para a resolução de problemas e mineração de dados. Já a técnica de Regressão é utilizada quando se deseja como saída um valor numérico, e não um valor discreto, como uma classe.

Existe uma grande quantidade de algoritmos disponíveis para classificação/regressão, sendo os principais:

- Regressão Linear
- K-Vizinho Mais Próximo (*K-Nearest Neighbor*, em tradução livre)
- Redes Neurais Artificiais
- Árvores de Decisão

Existem outros algoritmos conhecidos, como o algoritmo *Naive Bayes*, que é uma aproximação do Teorema de Bayes aplicado à AM, entre outros, porém, apenas os algoritmos acima serão descritos. Com exceção do primeiro algoritmo que, como o próprio nome dá a entender, é utilizado apenas para problemas de regressão, todos os algoritmos podem ser utilizados tanto para classificação quanto para regressão.

3.5.1 Regressão Linear

É um algoritmo estatístico e paramétrico no qual, resumidamente, consiste em expressar a saída desejada na forma de uma função linear, aonde cada instância é relacionada com um peso. Sua principal vantagem é a sua simplicidade, sendo reconhecidamente o algoritmo mais simples e utilizado para a criação de modelos de regressão. Já suas desvantagens consistem em trabalhar apenas com dados numéricos, e sua eficiência em problemas não-lineares é baixíssima, devido a sua limitação de tentar transformar uma função não-linear em um modelo linear simples. Ou seja, sua utilização é muito limitada. Para maiores detalhes, ver (WITTEN; FRANK, 2000).

3.5.2 K-Vizinho Mais Próximo

O algoritmo de K-Vizinho Mais Próximo é um poderoso algoritmo não-paramétrico de classificação/regressão, sendo utilizado desde a década de 1950 na área de Estatística. É

um algoritmo lento, mas eficiente, sendo recomendado para bases de dados que contenham muitas instâncias, já que dependendo do valor escolhido para k (geralmente um número ímpar), pode ser perigoso utilizar o algoritmo em bases pequenas, tornando o algoritmo suscetível a ruído no conjunto de dados e afetando de forma significativa a precisão do mesmo.

Seu funcionamento é relativamente simples: ao se classificar uma nova instância, o algoritmo busca as k instâncias que possuem a menor distância em relação à nova instância. Uma grande desvantagem deste algoritmo (além da lentidão ao se predizer novas instâncias), e que deve ser analisada com cuidado ao se considerar a sua utilização, é que todas as características de uma instância possuem o mesmo peso ao se calcular a distância. Ou seja, caso uma característica tenha uma importância maior do que as outras na hora de se classificar uma instância, esta importância acaba sendo descartada por este algoritmo. Para maiores detalhes, ver (WITTEN; FRANK, 2000).

3.5.3 Redes Neurais Artificiais

O estudo de Redes Neurais Artificiais foi inspirado, em grande parte, pela observação de que os sistemas de aprendizado biológicos são formados por complexas redes de neurônios interligados (MITCHELL, 1997). Portanto, os algoritmos de Redes Neurais Artificiais muitas vezes possuem como objetivo tentar simular as capacidades de processamento de um cérebro humano, por meio de unidades (ou nós) de processamento simples que modelam um neurônio biológico.

Na área de AM, Redes Neurais Artificiais são consideradas algoritmos poderosos, sendo utilizadas para resolução de problemas lineares ou não-lineares, tenso suporte tanto a Aprendizado Supervisionado quanto Não Supervisionado. Sua utilização mais frequente é na resolução de problemas não-lineares de Classificação, por meio de uma estrutura de rede chamada de Perceptron Multi-Camada (*Multi-Layer Perceptron*, ou MLP). O algoritmo mais conhecido de aprendizado para Redes Neurais Artificiais é o algoritmo de Retropropagação (backpropagation), sendo que a modelagem de Redes Neurais Artificiais voltadas para a área de AM são implementadas utilizando estes dois algoritmos em conjunto.

As Redes Neurais Artificiais são recomendadas para bases de dados que contenham muito ruído, ou seja, cujo conjunto de dados de treinamento apresente muitas inconsistências. Estas inconsistências podem afetar severamente o modelo gerado por outros algoritmos de Aprendizado Supervisionado como Árvores de Decisão, mas os algoritmos de Redes Neurais Artificiais conseguem ser relativamente “imunes” a este ruído.

Apesar de serem extremamente robustas e eficientes, as Redes Neurais Artificiais são pouco utilizadas em atividades de Mineração de Dados (CRAVEN; SHAVLIK, 1997),

pois os padrões gerados pelo algoritmo são, em sua maioria, incompreensíveis, além de ser um algoritmo cuja função de aprendizado é consideravelmente lenta, devido às inúmeras iterações realizadas pelo algoritmo de Retropropagação para atualizar os pesos da rede. Para maiores detalhes, ver (MITCHELL, 1997), (CRAVEN; SHAVLIK, 1997).

3.5.4 Árvores de Decisão

Árvores de Decisão é, provavelmente, o algoritmo de AM mais estudado para aplicações em Mineração de Dados (WITTEN; FRANK, 2000). Por ser o tema deste trabalho, uma explicação mais detalhada do algoritmo é necessária: o capítulo 4 contém uma descrição detalhada do funcionamento de um algoritmo padrão de Árvores de Decisão, além de detalhar o algoritmo C4.5, provavelmente a implementação mais conhecida de Árvores de Decisão.

3.6 Validação do Modelo de AM

Após a escolha do algoritmo e criação do modelo de AM, é necessário verificar se o modelo apresenta resultados consistentes. Apenas testar o modelo utilizando um conjunto de dados de treinamento não é o suficiente: isso pode ser enganoso, pois o modelo pode funcionar perfeitamente para as instâncias de treinamento, mas apresentar um desempenho pífio na classificação de novas instâncias.

Verificar a consistência dos resultados gerados pelo modelo de AM, portanto, não deve ser deixado de lado. A seguir, serão descritos os dois maiores problemas que podem ocorrer em um modelo de AM (*overfitting* e *underfitting*) e a técnica mais comum (e considerada a mais eficiente) para evitar estes problemas: Validação Cruzada.

3.6.1 Overfitting

Overfitting é considerado o "bicho-papão" dos modelos de AM, devido a facilidade em que este problema pode ocorrer caso não se tome cuidado ao gerar um modelo. Mitchell (1997) fornece uma definição formal sobre o problema: "dada um modelo H, é dito que H causa *overfitting* no conjunto de dados de treinamento se existe uma segunda hipótese H', em que a taxa de erros de H $<$ H', em relação aos dados de treinamento, mas a taxa de erros de H' $<$ H em relação ao conjunto total de dados."

Ou seja, um modelo que trabalha com precisão em um conjunto de dados de treinamento não necessariamente irá trabalhar bem com novas instâncias. Por exemplo, um classificador com 100% de precisão no conjunto de dados de treinamento, mas apenas 50% nas instâncias de teste, está causando *overfitting* nos dados de treinamento.

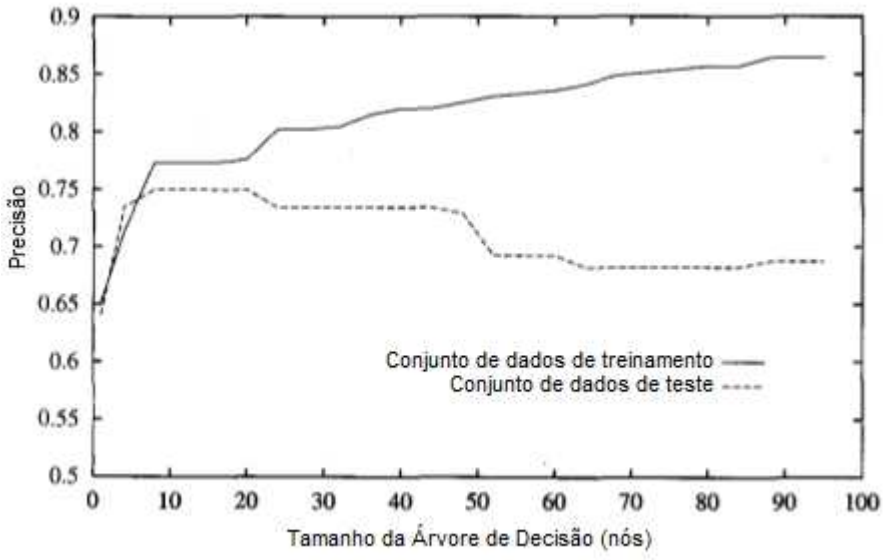

Figura 3: *Overfitting* em um modelo de Árvore de Decisão.

Na Figura (3), pode-se verificar como o *overfitting* acontece. Neste caso, o eixo horizontal equivale à quantidade de nós gerados por um modelo de árvore de decisão, enquanto o eixo vertical é a precisão do modelo geral. A linha sólida mostra a precisão do modelo em relação ao conjunto de dados de treinamento, e a linha pontilhada mostra a precisão do modelo em relação ao conjunto de dados de teste. É possível notar que, à medida que a árvore cresce em quantidade de nós, a precisão do modelo nos dados de treinamento acompanha este crescimento, mas quando o modelo foi utilizado para predizer a classificação dos dados de teste, a precisão diminuiu depois que a árvore cresceu mais que 22 nós, aproximadamente. Mais detalhes sobre este exemplo, ver (MITCHELL, 1997).

Geralmente, o *overfitting* é causado por ruídos no conjunto de dados de treinamento, porém pode ser causado por outros motivos: um conjunto de dados de treinamento que não representa de forma adequada a base de dados completa, um conjunto de dados de treinamento pequeno demais, entre outros. Portanto, os dados de treinamento devem ser escolhidos com cuidado, para que este tipo de problema possa ser evitado, mas nem sempre isso é o suficiente. A Validação Cruzada é amplamente utilizada para evitar tanto o *overfitting* quanto o *underfitting*, descrito abaixo.

3.6.2 Underfitting

Como o próprio nome sugere, o *underfitting* pode ser considerado um complemento ao problema de *overfitting*. Enquanto um modelo com *overfitting* trabalha muito bem com um conjunto de dados de treinamento, mas não com novas instâncias, um modelo que sofre com o problema de *underfitting* apresenta problemas no próprio conjunto de dados de treinamento. Um modelo de AM com *underfitting* não consegue identificar os padrões

ocultos em um conjunto de dados de treinamento.

O *underfitting* ocorre normalmente quando o modelo gerado por determinado algoritmo não consegue trabalhar bem com as instâncias do conjunto de dados utilizados. Isso significa que o algoritmo utilizado não é adequado para este determinado problema.

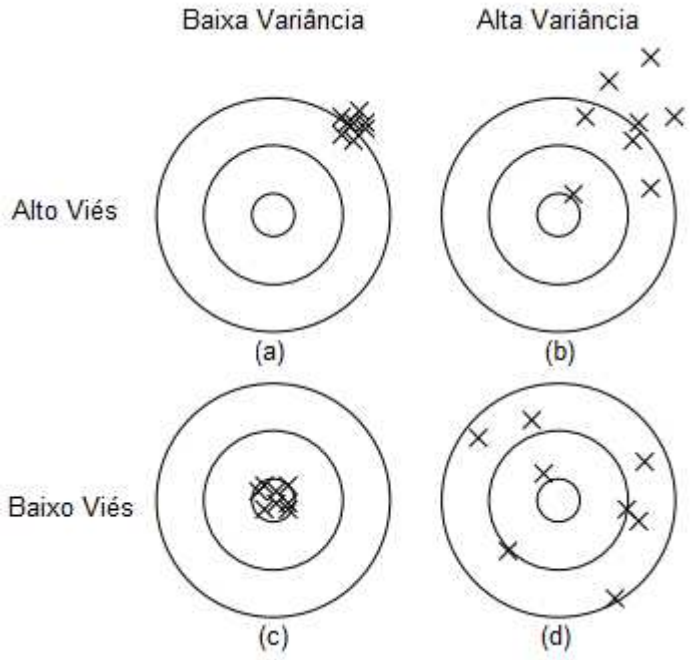

Figura 4: Classificação de algoritmos com base em variância e viés.

A Figura (4) exemplifica os problemas de *overfitting* e *underfitting* em relação a duas variáveis: variância e viés. Viés é a tendência de um modelo de AM aprender a mesma coisa errada de forma consistente, já a variância é a tendência de aprender coisas de forma randômica (DOMINGOS, 2012). Um modelo de AM com *overfitting* apresenta uma variância alta, porém um viés baixo (d), enquanto o mesmo modelo com *underfitting* possui uma variância baixa, porém um viés alto (a).

Portanto, além de se escolher com cuidado os dados de treinamento, deve-se verificar se determinado algoritmo é capaz de solucionar o problema proposto.

3.6.3 Validação Cruzada

Conforme visto nos tópicos anteriores, uma má escolha no conjunto de dados de treinamento pode acarretar em sérios problemas na construção do modelo de AM. E o que pode ser ainda mais grave: tais problemas podem não ser visíveis até o momento que novas instâncias sejam inseridas no modelo. Neste caso, é dito que o conjunto de dados de treinamento não é representativo o suficiente em relação ao conjunto inteiro de dados.

As causas podem variar: pode ser que no conjunto de dados de treinamento, todas as instâncias pertencentes a uma determinada classe estejam ausentes, ou que a variância

em determinada característica da instância esteja ausente dos dados de treinamento. É necessário que o conjunto de dados de treinamento contenha instâncias que representem de forma satisfatória a base de dados inteira, ou seja, que o conjunto de testes mantenha a representatividade de cada classe, em um modelo de Classificação, por exemplo.

Para garantir que o modelo de AM seja criado com um conjunto de dados de treinamento adequado, a técnica de Validação Cruzada é amplamente utilizada para calcular se o modelo está trabalhando com uma precisão satisfatória, auxiliando a evitar os problemas de *overfitting* e *underfitting* por uma má escolha dos dados de treinamento. A Validação Cruzada pode ser aplicada de duas formas: o método de *holdout*, ou o método de k-partições.

No método de *holdout*, o conjunto de dados é dividido em dois: o conjunto de dados de treinamento e o conjunto de testes. Esta divisão é realizada de forma randômica, e é recomendável que seja realizada diversas vezes. A cada iteração se obtém a margem de erro, e a média entre as margens de erros obtidas no final é a margem de erro do modelo. Este método ainda apresenta problemas, pois pode ocorrer uma grande variância dependendo do conjunto de dados de treinamento escolhido, o que afetará a margem de erro final.

Portanto, a melhor abordagem para a Validação Cruzada é a utilização do método de k-partições. Neste método, o conjunto de dados é dividido em k partições (segundo Witten e Frank (2000), testes extensivos em diversas bases de dados, utilizando diversos algoritmos, identificaram o valor de k para identificar a melhor margem de erro como sendo 10), também de forma randômica. Então, o conjunto de dados de treinamento é criado com $k - 1$ partições, e apenas uma partição é utilizada para testes. São realizadas k iterações, aonde cada partição é utilizada uma vez para testes enquanto as outras são utilizadas para treinamento. Após todas as partições terem sido utilizadas para teste, a margem de erro de cada iteração é somada e a média das k iterações se torna a margem de erro do modelo.

Diversos autores ((WITTEN; FRANK, 2000), (BRINK; RICHARDS, 2014)) demonstram preferência para o método de k-partições. De fato, este método apresenta resultados superiores ao método de *holdout*, apesar do custo computacional do método de k-partições ser maior. Portanto, é recomendável a utilização deste método para validação da eficiência do modelo de AM gerado.

4 Árvores de Decisão

4.1 Definição

Uma árvore de decisão é um modelo de classificação/regressão cuja estrutura consiste em um determinado número de nós e arcos (também conhecidos como ramos) (FURNKRANZ; GAMBERGER; LAVRAC, 2012). Este modelo composto por uma estrutura no formato de uma árvore, aonde cada nó interno da árvore representa um determinado teste em uma característica de uma instância, e os arcos representam o resultado do teste realizado. Os nós externos da árvore (chamados de nós terminais, ou nós-folha) são referentes às classes de classificação. Para classificar uma determina instância, a Árvore de Decisão é percorrida de cima para baixo, seguindo pelos arcos dos nós aonde as características das instâncias satisfazem os testes realizados em cada nó até alcançar um nó-folha, que contém a nova classificação da instância. Na Fig. (5) é mostrada uma árvore de decisão simples, e como a mesma realiza a classificação de uma determinada instância.

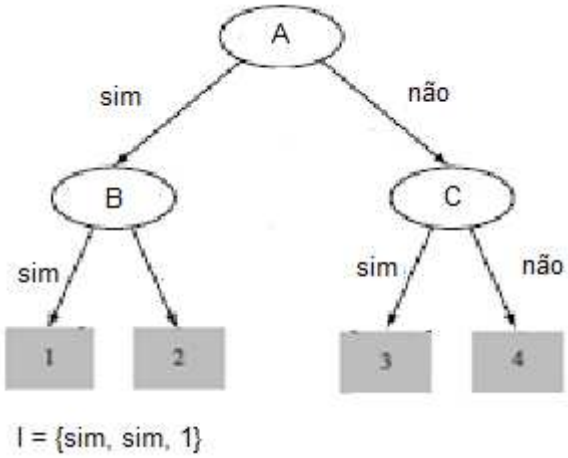

Figura 5: Exemplo de uma árvore de decisão.

O exemplo fornecido pela Fig. (5) acima é bem rudimentar, mas é facilmente compreendido. A instância I contém duas características: sim e sim. A árvore então executa um teste em cada nó, e compara com a característica presente na instância. A terceira característica é a classe que a instância pertence: a classe 1. Ao verificar o resultado dos dois testes, a árvore rotula a instância de acordo com a classe presente no nó-folha.

Árvores de Decisão é, provavelmente, o algoritmo de AM mais estudado para aplicações em Mineração de Dados (WITTEN; FRANK, 2000). Os motivos são variados: é um modelo que possui suporte a diversos tipos de características (categóricas e numéricas), sua representação de conhecimento adquirido é facilmente compreendida, e o seu processo

de aprendizado e treinamento é relativamente rápido, comparado a outros algoritmos como Redes Neurais Artificiais.

4.2 Implementação

Para a construção de uma Árvore de Decisão, a abordagem mais comum é a utilização do algoritmo *dividir-e-conquistar*: este algoritmo trabalha dividindo o conjunto de dados de treinamento em subconjuntos, de maneira recursiva. O algoritmo inicia escolhendo a árvore de decisão mais geral, contendo apenas o nó inicial, chamado de nó-raiz. Com base neste nó, o algoritmo trabalha recursivamente em cada nó filho, refinando a árvore até que cada subconjunto pertença a uma única classe.

4.3 Criação da Árvore

O fator crítico na criação de uma Árvore de Decisão é a escolha da característica que irá ser utilizada como nó na árvore. O critério de escolha da característica é determinado de forma que cada divisão realizada na árvore seja feita da forma mais "pura" possível, ou seja, que o número máximo possível de instâncias em cada subconjunto pertença a uma única classe.

É necessário um critério de seleção para a escolha de uma característica que se tornará um nó na árvore. Geralmente, o critério de seleção de um nó é realizado por meio do cálculo do Ganho de Informação, ou Entropia das características de cada instância presente no subconjunto.

Após a árvore ser construída, é necessário "podá-la". Este passo é importante ao se utilizar árvores de decisão, pois este algoritmo é muito suscetível ao problema de overfitting. Basicamente, ruído nos dados de treinamento pode levar a construção de ramos e nós que atrapalham a generalização de novas instâncias. Além disso, o modelo construído pode ser muito específico (como, por exemplo, possuir uma folha para cada instância do conjunto de dados de treinamento): este modelo mostra uma precisão enorme nos dados de treinamento, mas irá se mostrar ineficiente ao ser alimentado com instâncias do conjunto de dados de teste.

As duas abordagens mais comuns de poda são: a pré-poda e a pós-poda. Na primeira abordagem, a construção da árvore é interrompida devido a um determinado critério de parada em tempo de execução, ou seja, a árvore é podada durante a sua construção. Apesar de ser uma abordagem interessante, pois evita o trabalho de gerar uma árvore completa para podá-la apenas depois, este tipo de abordagem traz problemas em relação ao critério de parada: este critério precisa ser escolhido com muita cautela, pois um critério fraco irá podar muito pouco da árvore, enquanto um critério de parada forte (como

um limiar de Entropia alto) pode gerar uma árvore simplificada demais, prejudicando a precisão do modelo.

A segunda abordagem é a mais comum, consistindo em construir a árvore completa e então, remover determinadas sub-árvores, as substituindo por um nó folha. Este nó folha geralmente é a classe mais comum da sub-árvore substituída. A decisão de se substituir uma sub-árvore por uma folha é dado pelo cálculo da estimativa de erro de um determinado nó N, aonde N é o pai da sub-árvore, em comparação com a estimativa de erro de N com a sub-árvore podada, ou seja, apenas com um nó folha. Se a estimativa de erro da folha for menor, a sub-árvore é podada.

4.4 Complexidade

A complexidade computacional do algoritmo de Árvore de Decisão é $O(nTlog(T))$, aonde n é o número de características de cada instância e T é o número de instâncias que compões o conjunto de dados de treinamento (HAN; KAMBER, 2001). Ou seja, a árvore de decisão irá crescer no máximo até $nTlog(T)$.

4.5 Algoritmo C4.5

O algoritmo C4.5 foi desenvolvido por J. R. Quinlan, na década de 90, sendo até hoje considerado o algoritmo de referência para o desenvolvimento e análise de novos algoritmos de Classificação (RUGGIERI, 2004). O C4.5 é uma evolução do algoritmo ID3, também desenvolvido por Quinlan em meados da década de 80, apresentando diversas melhorias, principalmente em relação ao cálculo utilizado para o critério de decisão e ao algoritmo de poda.

Assim como o ID3, o C4.5 realiza a construção da árvore por meio de um algoritmo guloso, utilizando a estratégia de *dividir-e-conquistar*. Um algoritmo guloso é um algoritmo que trabalha resolvendo problemas indicando a melhor escolha de forma local, com a expectativa que estas escolhas levem à melhor escolha global (também chamada de ótimo global). No caso de árvores de decisão, o algoritmo C4.5 calcula a melhor característica que possa servir como um nó de decisão e trabalha recursivamente a cada nó até que a árvore esteja montada. Ou seja, o algoritmo trabalha com a melhor escolha local a cada nó da árvore, sem reconsiderar suas escolhas prévias.

Com base em um conjunto de dados de treinamento T, o algoritmo inicia com todas as instâncias de treinamento em um único nó, que será a raiz da árvore. Este conjunto de dados de treinamento é dividido em subconjuntos menores a medida que a árvore é construída, até que todos os subconjuntos presentes em um nó N são da mesma classe. Então, estes nós são chamados de nós-folha, e a construção da árvore está completa.

4.5.1 Ganho de Informação

A principal tarefa realizada pelo algoritmo é verificar qual característica que será utilizada como teste em cada nó. Naturalmente, o interessante é escolher uma característica que melhor classifique os subconjuntos que serão criados ao realizar o teste no nó. Este critério de seleção é calculado, e a característica com melhor resultado é utilizada como teste no nó. Porém, como calcular este critério de seleção?

O ID3 utiliza uma propriedade estatística conhecida como Ganho de Informação, que calcula como determinada característica divide o conjunto de casos de teste de acordo com a sua classificação (MITCHELL, 1997). O C4.5 utiliza um critério de seleção mais elaborado, conhecido como Razão de Ganho, porém é necessário o entendimento de como o Ganho de Informação é calculado para entender a Razão de Ganho.

Para calcular o Ganho de Informação, é necessário primeiramente calcular a Entropia da característica a ser testada. A Entropia da Informação é uma métrica utilizada para calcular a "impureza" em um determinado subconjunto, e com base nesta métrica, "é possível minimizar a quantidade de informação necessária para classificar uma determinada instância, e garantir que uma árvore simplificada seja obtida" (HAN; KAMBER, 2001).

A Entropia da Informação em um conjunto de dados de treinamento T é calculada por meio da Eq. (4.1):

$$E(T) = -\sum_{i=1}^{\mathrm{C}} p_i log_2(p_i) \tag{4.1}$$

$$p_i = \frac{C_{i,T}}{T} \tag{4.2}$$

Aonde pi é a probabilidade de uma instância aleatória i pertencer a uma classe Ci. A variável pi é calculada conforme a Eq. (4.2), aonde Ci,T é a quantidade de instâncias de T que pertencem a Ci. O logaritmo base 2 é utilizado pois a Entropia é medida em bits. Por meio deste cálculo, a informação necessária para se classificar uma instância é obtida.

O Ganho de Informação nada mais é do que a redução na quantidade de informação necessária para se classificar uma determinada instância com base na característica que está sendo calculada. Ou seja, é o ganho de informação obtido ao se partir o conjunto de dados com base na característica calculada, dado pela diferença entre a Entropia original e a Entropia calculada após o particionamento do conjunto. Com isso, é necessário calcular a Entropia do conjunto após o particionamento. Este cálculo é realizado utilizando a Eq. (4.3):

$$E_c(T) = -\sum_{i=1}^{\mathrm{V}} \frac{T_i}{T} E(T_i) C_v \tag{4.3}$$

Aonde, v é a quantidade de valores que uma característica C pode assumir, Ti é o subconjunto de instâncias aonde C assume um valor Cv, e $E(Ti)$ é a Entropia do subconjunto. Basta calcular este valor para cada característica, e a característica que possuir o menor valor, e consequentemente, o maior Ganho de Informação Eq. (4.4), será a característica escolhida como nó de decisão.

$$Ganho(C) = E(T) - E_c(T) \tag{4.4}$$

4.5.2 Características Discretas e Contínuas

Ao se analisar como que o Ganho de Informação é calculado, é possível notar que o mesmo se restringe a um conjunto discreto de valores. Ou seja, caso a característica seja discreta, basta calcular o Ganho de Informação normalmente.

Para o cálculo do Ganho de Informação para Características Contínuas, é necessária a Discretização das mesmas. O processo de Discretização classifica os valores contínuos em um conjunto discreto que possua o maior Ganho de Informação para as instâncias. Ou seja, o valor que possui o maior Ganho de Informação é usado como limiar para os valores contínuos, e o nó de decisão executa apenas um teste simples, do tipo MAIOR QUE... ou MENOR OU IGUAL QUE....

Para o cálculo do limiar, primeiramente os valores contínuos são ordenados em ordem crescente. O algoritmo C4.5 utiliza a média entre dois valores adjacentes para o cálculo do Ganho de Informação. Ou seja, para um conjunto de valores contínuos de tamanho v, existem $v - 1$ possíveis limiares. Com base nisso, é calculado o Ganho de Informação para cada um destes possíveis limiares, o que possuir o maior valor, é escolhido como critério de seleção.

4.5.3 Razão de Ganho

Apesar de ser uma métrica eficiente, o Ganho de Informação possui um problema: este tipo de cálculo favorece características que possuam uma grande quantidade de valores. Como a base de dados pode possuir características que identificam uma instância de forma única, o resultado obtido pelo Ganho de Informação para esta característica será um resultado ótimo, resultando em um Ganho de Informação máximo, mesmo sendo considerado um nó de decisão inútil.

O algoritmo C4.5 utiliza uma extensão do Ganho de Informação, chamada Razão de Ganho. Esta extensão introduz um novo termo, que penaliza características que

possuam muitos valores (MITCHELL, 1997). Este novo termo, chamado de Divisão de Informação, é definido pela Eq. (4.5):

$$DivInfo_c(T) = -\sum_{i=1}^{\mathrm{V}} \frac{T_i}{T} log_2(\frac{T_i}{T}) \tag{4.5}$$

Este novo termo nada mais é que a Entropia do conjunto de dados T em relação à característica C. Com base nisso, a Razão de Ganho é dada pela Eq. (4.6) :

$$RazãoGanho(T) = \frac{Ganho(C)}{DivInfo_c(T)} \tag{4.6}$$

Para evitar problemas com uma Razão de Ganho muito alto por conta de o denominador assumir valores muito próximos de zero, quando não há uma variância alta entre os valores de determinada característica (ou seja, grande parte das instâncias possuem o mesmo valor nesta característica), o algoritmo C4.5 aplica a Razão de Ganho apenas quando o valor obtido pelo Ganho de Informação é maior que a média dos Ganhos de todas as características.

4.5.4 Poda

O algoritmo C4.5 utilizada a técnica de pós-poda, por meio de um método conhecido como Poda Pessimista. Segundo Witten e Frank (2000), este método consiste em podar a árvore com base em uma estimativa de erro calculada. O C4.5 realizada o cálculo desta estimativa de erro em cima do conjunto de dados de treinamento, adicionando uma penalidade por conta da utilização do conjunto de treinamento para a estimativa de erro, pois estimativas de erro ou precisão realizadas em cima do conjunto de dados de treinamento são, em grande parte, muito otimistas (HAN; KAMBER, 2001). A estimativa de erro pessimista é calculada de acordo com a Eq. (6.1):

$$p = f + z\sqrt{\frac{f(1+f)}{N}} \tag{4.7}$$

Aonde f é a estimativa de erro calculada com base no conjunto de treinamento, z é a dispersão correspondente ao fator de confidência c utilizado, e N é o tamanho do subconjunto de treinamento. O valor padrão utilizado no algoritmo C4.5 é 25%, e z = 0.69 (WITTEN; FRANK, 2000).

Obtida a estimativa de erro pessimista, a abordagem de poda é semelhante à descrita no tópico. Utilizando uma abordagem bottom-up (ou seja, iniciando de baixo para cima), o algoritmo percorre a árvore, e para cada sub-árvore, realiza o cálculo de estimativa de erro pessimista. Caso a estimativa de erro da sub-árvore for maior do que a estimativa de erro calculada caso a sub-árvore inteira fosse substituída por um nó-folha, a

sub-árvore é podada. A Figura (6) mostra a execução da abordagem de Poda Pessimista em uma árvore de decisão.

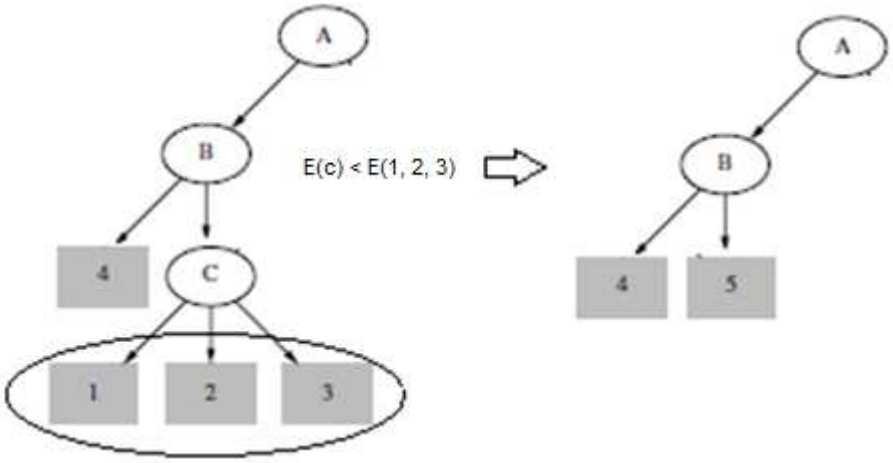

Figura 6: Abordagem de Poda Pessimista em uma sub-árvore.

Neste exemplo, a estimativa de erro foi calculada para cada um dos três nós (1, 2, 3). Então, a estimativa de erros dos três nós são combinadas, com a quantidade de instâncias que cada um abrange em relação ao total de instâncias abrangido pela sub-árvore como pesos. A estimativa de erro do pai foi calculada. Como a estimativa de erro do pai é menor que a estimativa de erro dos filhos, os três nós são podados e substituídos por um nó-folha (5).

5 Desenvolvimento

Este trabalho propõe a criação e validação de um modelo de AM baseado no algoritmo de Árvores de Decisão C4.5. Segundo Witten e Frank (2000), o C4.5 é o principal algoritmo de classificação presente no mercado, sendo a referência para a criação de novos algoritmos. Com este trabalho, pretendeu-se analisar a capacidade de predição do modelo em diversos contextos.

5.1 Fluxo de Trabalho

A execução do projeto proposto seguiu o fluxo de trabalho descrito na Fig. (7):

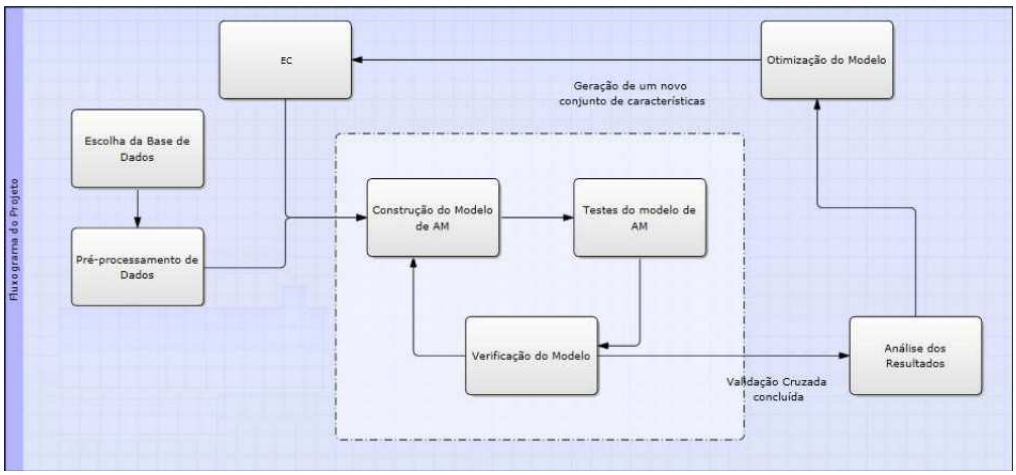

Figura 7: Fluxograma do projeto.

É possível notar que a execução do projeto seguiu um fluxo de trabalho padrão para o desenvolvimento de um modelo de AM. Primeiramente, realizou-se a escolha da base de dados supervisionada que será utilizada como conjunto de dados de treinamento e testes. Após a escolha da base de dados, a mesma passou por uma etapa de pré-processamento, com o objetivo de se remover possíveis inconsistências (o chamado ruído). Após a etapa de pré-processamento, o modelo de AM foi gerado por meio do sistema YaDT, uma conhecida implementação do algoritmo C4.5, o algoritmo de Árvore de Decisão escolhido para este trabalho.

Para a criação e validação do modelo, tem-se início um processo iterativo: o modelo será criado utilizando-se o conjunto de dados de treinamento obtido. Após o treinamento, a estimativa de erro é calculada de acordo com os resultados obtidos pelo modelo na etapa de criação e poda da árvore. Então, tem-se início a validação do modelo gerado pelo conjunto de dados. Este processo é realizado iterativamente, por meio da técnica

de Validação Cruzada. A cada iteração, são obtidas as estimativas de erro e a matriz de confusão de cada conjunto de treinamento e testes. Ao término das iterações, a estimativa de erro final é calculada, e os dados obtidos são analisados para otimização tanto do algoritmo, quanto da base de dados (por meio do processo de EC).

Nos tópicos a seguir, este fluxo de trabalho é detalhado desde a coleta de dados para escolha da base de dados que será utilizada, passando pela etapa de pré-processamento e suas atividades, tecnologias utilizadas para a escolha da implementação do algoritmo, e detalhamento da etapa de verificação do modelo.

5.2 Coleta de Dados

O principal pré-requisito ao se escolher as bases de dados para este comparativo foi que as mesmas fossem voltadas para atividades de classificação supervisionada. Ou seja, a classe a qual cada instância pertence (chamada de característica-alvo) deve estar presente juntamente com o restante das características de cada instância.

Após uma análise de diversas bases de dados candidatas no início do trabalho, optou-se pela escolha de duas bases: uma para validação do modelo de AM, para que se pudesse ter certeza da eficácia do algoritmo de Árvores de Decisão escolhido, e outra que representasse um problema do mundo real, para demonstrar como a implementação do C4.5 pode ser utilizado para a descoberta de padrões nos mais variados problemas.

Como base de dados para validação do algoritmo, afim de se demonstrar seus resultados, foi escolhida uma base de dados conhecida na comunidade científica: a base de dados do Titanic, disponiblizada pela Kaggle. Por ser uma base que já foi extensamente trabalhada, passando pelos mais diversos algoritmos de AM, é uma opção interessante para que se possa verificar a eficácia do modelo de AM gerado pelo YaDT.

A segunda base de dados que foi analisada neste trabalho são um conjunto de dados disponibilizado pelo Ministério da Agricultura, relacionados ao teste de substâncias que podem causar danos à saúde humana em carnes bovinas. Esta segunda etapa tem como objetivo principal a descoberta de padrões que possam ajudar a prever certos problemas que possam vir a acontecer, como por exemplo, determinada substância ter um pico de detecções mais aparente em determinada época do ano. As bases de dados candidatas são descritas com maior detalhes no Capítulo 6.

5.3 Pré-Processamento de Dados

Após a escolha da base de dados que será utilizada para alimentar o algoritmo de Árvore de Decisão desenvolvido, é necessário verificar se os dados estão consistentes o suficiente para que não haja problemas na hora de treinar o modelo. Conforme descrito no

capítulo 2, a atividade de Pré-Processamento de Dados possui importância significativa para que o modelo de AM consiga minerar padrões ocultos e, consequentemente, classificar os dados de forma precisa, pois é nesta fase que as inconsistências são identificadas e removidas da base de dados.

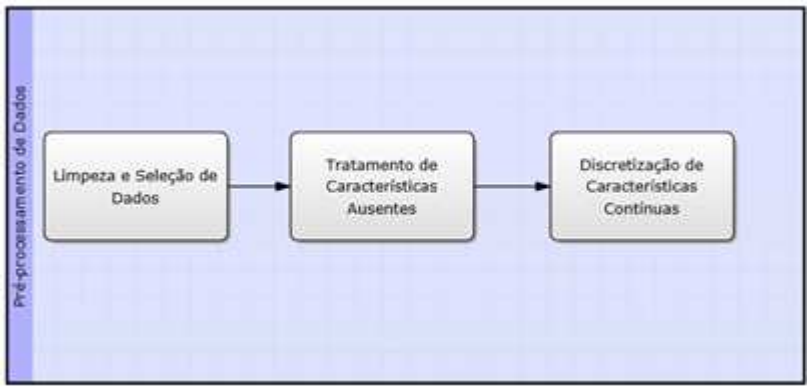

Figura 8: Fluxo de trabalho da etapa de Pré-Processamento de Dados

A etapa de Pré-Processamento de Dados foi dividida em três fases, conforme mostrado na Fig. (8): Limpeza e Seleção de Dados, Tratamento de Características Ausentes e Discretização de Características Contínuas. Destas três etapas, duas foram realizadas antes da implementação do algoritmo de Árvores de Decisão, enquanto a etapa de Discretização de Características Contínuas está inclusa dentro do algoritmo, conforme descrito no capítulo 4.

5.3.1 Limpeza e Seleção de Dados

As etapas de Limpeza e Seleção de Dados foram realizadas para identificar e remover o chamado ruído, que são inconsistências nas características de determinadas instâncias, ou até mesmo instâncias inteiras destoantes do restante da base, e removê-las para que a acurácia do modelo de AM não seja prejudicada durante a atividade de classificação.

Tais atividades exigem certo conhecimento do domínio no qual a base de dados pertence, portanto após a escolha da base de dados a ser minerada, um estudo do domínio dos dados foi realizado antes da execução destas atividades. Sem este conhecimento, o ruído da base de dados pode ser incorretamente identificado e instâncias que não possuem inconsistências podem ser removidas, prejudicando o aprendizado do algoritmo ao ser alimentado com o conjunto de dados de treinamento "limpo" incorretamente.

As atividades que foram realizadas nesta etapa são:

- Identificação de possíveis discrepâncias nos dados.
- Identificar possíveis erros tipográficos.

- Verificar possíveis inconsistências em valores numéricos.
- Verificar possíveis inconsistências em categorias.
- Lidar com características ausentes.

Erros tipográficos são comuns em bases de dados do mundo real. Um erro tipográfico em uma característica contendo um tipo de categoria pode vir a criar uma categoria inexistente, afetando a árvore de decisão criada. Esta atividade foi realizada identificando todos os valores possíveis para uma característica discreta, e analisando a quantidade de instâncias que pertencem a cada categoria.

Valores numéricos inconsistentes também são relativamente comuns. Para identificar tais inconsistências, foram gerados gráficos contendo todos os valores numéricos possíveis de uma determinada característica. Por meio da análise destes gráficos, é possível encontrar pontos "fora da curva", que possam vir a ser valores inconsistentes. Então, tais pontos foram analisados e, identificada a inconsistência, devidamente tratados.

A última tarefa, e possivelmente a de maior importância, é a tarefa de tratamento de características ausentes.

5.3.2 Tratamento de Características Ausentes

Características ausentes são comuns em bases de dados do mundo real, e são consideradas um sério problema no treinamento de um modelo de AM se não tratadas adequadamente. Como pode ser visto na análise das bases de dados candidatas no próximo capítulo, a quantidade de instâncias com características ausentes é considerável (principalmente na base de dados Titanic, onde 1/5 das instâncias que compõem a base possuem ao menos uma característica ausente), a alternativa mais simples, de se remover todas as instâncias que contém características ausentes se torna inviável, pois o risco de se inserir uma variância antes inexistente na base é alto.

Portanto, a abordagem para tratamento de características ausentes foi realizada da seguinte forma: como descrito no capítulo 3, características ausentes devem ser tratadas de acordo com a sua importância na base de dados. Foi realizada então uma análise para identificar se o valor ausente de cada característica carrega algum significado. Exemplificando, ao realizar um cadastro em determinada empresa, pode ser pedido ao usuário para que o mesmo cadastre a sua CNH. Mas e se o candidato não possuir o documento? Então este campo é deixado em branco. Neste caso, é dito que a ausência de valores em determinada característica carrega significado, pois o valor está ausente não por erro de coleta de dados ou problemas de medição, e sim por uma necessidade.

Então, caso o resultado da análise seja "sim", uma nova categoria foi criada para a ausência da característica em questão, sendo que tal categoria será indicada por um

"Unknown" (desconhecido, em inglês) para valores categóricos, e "0 a – 999" para valores numéricos.

Caso a resposta seja "não", é necessária uma nova abordagem no tratamento destas características. Neste caso, o tratamento será o seguinte: o valor da característica ausente será igual ao valor mais comum dentre os valores para tal característica do subconjunto de dados de treinamento.

5.3.3 Discretização de Valores Numéricos

Apesar de possuir suporte à características contínuas, o algoritmo de Árvore de Decisão não consegue trabalhar diretamente com estas características, pois os valores testados nos nós de decisão e a predição realizada pelo algoritmo precisam ser características discretas. Portanto, é necessária a classificação das características contínuas em intervalos discretos. Este processo é conhecido como discretização, e é realizado internamente pelo algoritmo C4.5, portanto não é necessário realizar a discretização diretamente na base de dados.

O processo de discretização realizado pelo algoritmo C4.5, conforme descrito mais detalhadamente no capítulo 4, consiste em definir um limiar de intervalo de forma dinâmica, por meio do cálculo do Ganho de Informação (ou Entropia). Este cálculo é realizado para que o limiar possua o maior ganho de informação, ou seja, que este intervalo trabalhe com o maior número de instâncias dentro de um subconjunto. Definido este intervalo, o nó de decisão executa apenas uma comparação booleana: se o valor contínuo da instância é maior ou menor/igual ao limiar calculado. Para mais detalhes, ver capítulo 4.

5.4 Criação do Modelo de AM

Após se determinar a base de dados que será utilizada para avaliação do modelo de AM, teve início a etapa de criação do modelo, por meio do algoritmo C4.5. Nos tópicos seguintes, estão presentes a justificativa pela escolha do algoritmo, e as principais ferramentas que auxiliarão na utilização e avaliação do sistema.

5.4.1 Algoritmo C4.5

Conforme dito nos tópicos anteriores, o algoritmo utilizado para a geração do modelo é um conhecido algoritmo de Árvores de Decisão, o C4.5. A escolha deste algoritmo se deve aos seguintes motivos:

- Robustez do algoritmo
- Suporte a valores categóricos

- Tratamento de características ausentes
- Facilidade na visualização do modelo de AM gerado

Conforme mencionado por Witten e Frank (2000), Árvores de Decisão provavelmente é o algoritmo mais estudado em aplicações de Mineração de Dados utilizando a abordagem de AM. Por conta disso, é um algoritmo já consolidado, e como o C4.5 é considerado o algoritmo de referência para o desenvolvimento e propostas de algoritmos de Classificação utilizando Árvores de Decisão (RUGGIERI, 2004), sua escolha é facilmente explicada.

Seu suporte a valores categóricos é interessante, visto que este suporte evita a necessidade de se transformar tais características em valores numéricos, e assim, criando uma grande quantidade de novas características, aumentando a dimensionalidade do problema, Além disso, o C4.5 é capaz de lidar com características ausentes internamente, em determinadas situações.

Porém, todas as vantagens citadas também estão presentes em algoritmos de Redes Neurais Artificiais. Por que escolher o C4.5? Como este trabalho tem como objetivo, além da implementação do algoritmo, a análise dos resultados obtidos pelo modelo de AM, a principal desvantagem das Redes Neurais vêm a tona: as Redes Neurais Artificiais são pouco utilizadas em atividades de Mineração de Dados (CRAVEN; SHAVLIK, 1997), pois os padrões gerados pelo algoritmo são, em sua maioria, incompreensíveis. Já o algoritmo de Árvore de Decisão gera um modelo que é facilmente compreendido, facilitando a análise dos resultados para que o modelo possa ser otimizado.

5.4.2 Linguagens e Ambiente de Desenvolvimento

A ideia inicial do trabalho era a implementação do algoritmo C4.5 utilizando a linguagem C++. A escolha por esta linguagem foi feita pelo suporte ao paradigma orientado-a-objetos, e pela facilidade ao se programar por meio desta linguagem. Porém, como o objetivo do trabalho é demonstrar a utilidade da Árvore de Decisão para a descoberta de padrões ocultos em base de dados, optou-se por utilizar uma implementação já conhecida, cujo desempenho já tenha sido testado.

A implementação escolhida foi o YaDT (Yet Another Decision Tree), desenvolvida por Salvatore Ruggieri em 2004. O YaDT é uma implementação melhorada do algoritmo C4.5, conhecida por EC4.5 (Enhanced C4.5, em inglês), escrita em C++ e apresentando suporte ao paradigma orientado-a-objetos. O YaDT ainda apresenta diversas melhorias em relação à outras implementações conhecidas do C4.5, como o Weka, principalmente em questões de velocidade e gerenciamento de memória. Para maiores detalhes dos resultados obtidos pelo YaDT em comparação à outras implementações, ver (RUGGIERI, 2004).

Para a criação de scripts para lidar com as bases de dados e executar tarefas como Limpeza e Seleção de Dados e Tratamento de Caracerísticas Ausentes, foram utilizadas as linguagens C++ e Python. A primeira devido à facilidade de programação, permitindo um maior aproveitamento de tempo, e a segunda pela existência de módulos que facilitam a manipulação dos dados em formatos de tabela, como arquivos .xls e .csv.

O sistema operacional utilizado como ambiente de desenvolvimento é o Linux Debian Wheezy 7.0 64-bits. Como editor de texto padrão, foi utilizado o Sublime Text 2, e como compilador padrão, o G++ 4.72. O ambiente Python utilizado foi a versão 2.7.3.

5.4.3 Ferramentas de Desenvolvimento

Além do ambiente de desenvolvimento, algumas ferramentas foram utilizadas para auxiliar no desenvolvimento do projeto.

Para controle de versão do projeto, foi utilizada a ferramenta GitHub. O GitHub é um serviço web para usuários que utilizam o sistema de controle de versionamento Git. O repositório do projeto está disponível, podendo ser acessado pelo seguinte link: <https://www.github.com/hialo/decisiontreebuilder>.

Para a geração de gráficos, foi utilizada a ferramenta GnuPlot. Esta ferramenta foi utilizada principalmente nas atividades de Pré-Processamento de Dados (onde a plotagem de valores numéricos em gráficos de dispersão é importante na identificação de valores que estejam fora do padrão, indicando possível inconsistência) e Verificação do Modelo.

5.5 Validação do Modelo de AM

Criado o modelo, é necessário verificar a sua acurácia em um conjunto de dados de teste. Porém, árvores de decisão são suscetíveis ao problema conhecido como *overfitting*: a árvore de decisão gerada pelo algoritmo é muito específica em relação ao conjunto de dados de treinamento utilizado para gerar o modelo. Ou seja, a árvore de decisão gerada não consegue generalizar adequadamente novas instâncias de teste, tendo a sua capacidade de predição muito abaixo da precisão obtida ao se utilizar o conjunto de dados de treinamento.

Para evitar este problema, será utilizada a técnica de Validação Cruzada para avaliar a estimativa de erro obtida pela árvore de decisão.

5.5.1 Validação Cruzada

A técnica de Validação Cruzada é utilizada para verificar se o conjunto de dados de treinamento está representativo o suficiente em relação à base de dados na qual se deseja predizer certas instâncias.

O método de Validação Cruzada utilizado neste trabalho foi o método de k-partições. Como o objetivo do trabalho consiste em analisar os resultados obtidos em uma base de dados de teste, todas as instâncias já estarão classificadas, como nas bases de dados candidatas analisadas anteriormente. Portanto, a base de dados inteira foi o conjunto de dados de treinamento e teste.

Então, a base de dados foi dividida em k partições diferentes, cada partição com o mesmo tamanho (caso o número de instâncias seja divisível por k . Caso contrário, as instâncias restantes serão distribuídas pelas partições). Estas partições então foram rotuladas com valores de 1 a k . Então por k iterações, cada uma das partições irá ser o conjunto de casos de teste enquanto as k *-1* restantes irão ser o conjunto de casos de treinamento. A cada iteração realizada, a estimativa de erro do modelo, que consiste em quantas classificações incorretas foram feitas no conjunto de casos de teste, foi calculada. A estimativa total do erro do modelo será a soma das estimativas obtidas em cada iteração dividida por k.

5.5.2 Matriz de Confusão

Além da técnica de Validação Cruzada, a técnica de Matriz de Confusão também foi utilizada para avaliar a precisão do algoritmo. Como a Validação Cruzada apenas avalia qual foi a estimativa de erro calculada para todas as instâncias (ou seja, quantas instâncias foram incorretamente classificadas), a Matriz de Confusão fornece uma estimativa mais detalhada, em relação à classificação incorreta entre classes. A tabela 1 mostra como que é formada uma matriz de confusão para um conjunto de classificação contendo duas classes-alvo:

Verdadeiros Positivos (VP)	Falsos Positivos (FP)
Verdadeiros Negativos (VN)	Falsos Negativos (FN)

Tabela 1: Exemplo de matriz de confusão.

Por meio da matriz de confusão, é possível verificar quantas instâncias foram corretamente classificadas, e quantas foram incorretamente classificadas. Uma matriz de confusão para N classes-alvo possui tamanho NxN: como o exemplo mostrado na tabela 1 contém apenas duas classes-alvo, a matriz possui tamanho 2x2, sendo quatro valores possíveis: quantas instâncias da classe X foram corretamente classificadas (chamados verdadeiros positivos, ou VP), quantas instâncias da classe X foram incorretamente classificadas na classe Y (falso positivo, ou FP), quantas instâncias da classe Y foram corretamente classificadas (verdadeiro negativo, ou VN) e quantas instâncias foram incorretamente classificadas na classe X (falso negativo, ou FN).

Portanto, é fácil notar que a matriz de confusão é uma excelente ferramenta para verificar a precisão do algoritmo, fornecendo maior detalhamento da estimativa de erro

obtida pela Validação Cruzada. Quando usadas em conjunto, oferecem um excelente material para que se possa trabalhar na otimização do modelo de AM.

5.5.3 Análise dos Resultados

Com base na matriz de confusão obtida na execução do modelo, diversas métricas podem ser obtidas para avaliar o desempenho do modelo. As duas mais importantes para a análise são:

- Sensibilidade: é a capacidade do modelo em predizer corretamente a classe da instância testada.
- Especificidade: capacidade do modelo em predizer corretamente que determinada instância NÃO pertence à determinada classe.

Estas duas métricas também são conhecidas como Taxa de Verdadeiro Positivo (TVP) e Taxa de Verdadeiro Negativo (TVN). Como se pode notar, estas duas métricas são complementares entre si.

A análise dos resultados de cada modelo, além da matriz de confusão, foi feito por meio de um gráfico conhecido como curva ROC. Um exemplo de curva ROC pode ser visto na Fig. (9):

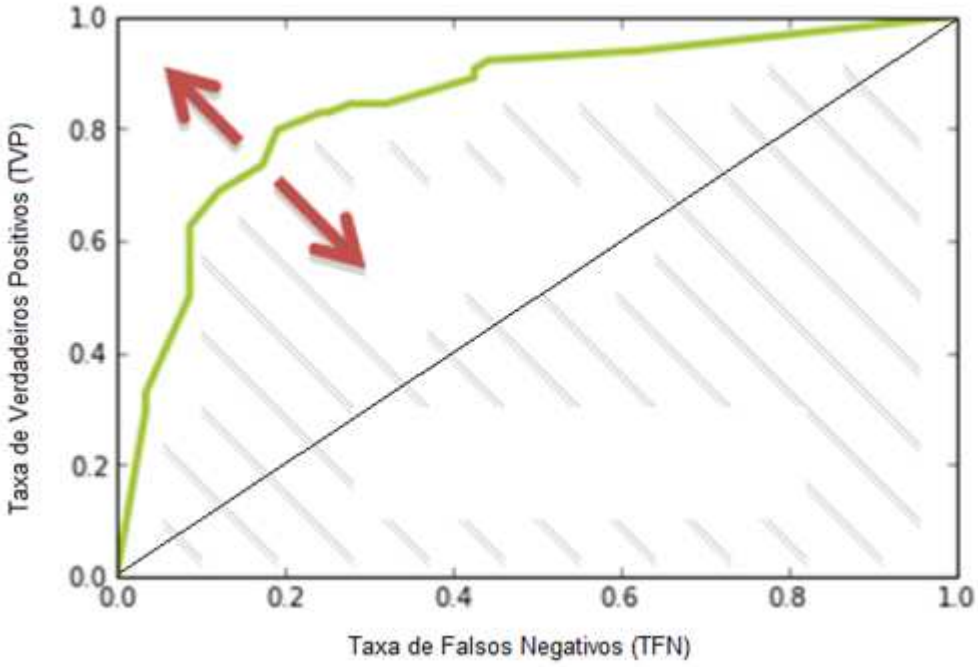

Figura 9: Exemplo de uma curva ROC. (BRINK; RICHARDS, 2014)

Este gráfico nada mais é do que a plotagem de duas métricas calculadas com base na matriz de confusão: TVP e a inversa de TVN, a Taxa de Falsos Positivos (TFP). Cada modelo então se torna um ponto no gráfico, e o seu desempenho pode ser avaliado pela sua proximidade com o canto superior esquerdo do gráfico (TVP = 1.0 e TFP = 0.0, ou seja, o modelo classifica todas as suas instâncias corretamente, sem FPs ou FNs).

Uma linha diagonal corta o gráfico, representando a probabilidade de uma classificação aleatória. Modelos que são classificados acima da linha são considerados bons, e abaixo da linha são descartados.

5.6 Otimização do Modelo de AM

Primeiramente, o modelo de AM foi treinado e testado utilizando a base de dados completa, ou seja, com todas as características que compõem cada instância. Obtidas as estimativas de erro e a matriz de confusão para esta iteração, os dados serão analisados para verificar se é possível melhorar o desempenho de predição do algoritmo.

Esta otimização pode ser feita de duas formas: aplicando a técnica de EC na base de dados, ou aperfeiçoando o algoritmo em si. Em relação à melhoria da qualidade do conjunto de dados de teste/treinamento, foi aplicada a técnica de Redução de Dimensionalidade, que remove as características que possuem menor importância na construção da Árvore de Decisão.

5.6.1 Engenharia de Características

Conforme descrito no capítulo 3, EC é um processo pelo qual as características que formam a base de dados são trabalhadas para que o modelo de AM tenha seu desempenho melhorado.

A técnica que foi utilizada para realizar EC na base de dados do sistema é a técnica de Redução de Dimensionalidade. Apesar de não ser limitado a três dimensões como os humanos, quanto maior a dimensionalidade, maior a dificuldade do modelo de AM lidar com o conjunto de dados (BRINK; RICHARDS, 2014). Esta técnica tem como objetivo reduzir o número de características que o modelo de AM irá lidar durante as etapas de treinamento e avaliação, visando á melhora de desempenho do algoritmo.

A técnica de Redução de Dimensionalidade possui várias abordagens, e a que foi utilizada é a de Seleção de Características. Novamente, a abordagem de algoritmo guloso será utilizada, para encontrar o melhor subconjunto de características em um conjunto completo. Esta abordagem é conhecida como Seleção de Atributos reversa: iniciando-se com o conjunto de característica completo, a pior característica medida neste conjunto é eliminada.

6 Resultados

6.1 Base de Dados - Titanic

6.1.1 Descrição da Base de Dados

A primeira base de dados analisada é uma base de dados fornecida pela Kaggle (2012), uma conhecida plataforma onde vários desafios na área de análise e mineração de dados são propostos, na busca do melhor modelo preditivo para determinado problema. Esta base de dados, fornecida em setembro como parte de um destes desafios, consiste em informações sobre os passageiros que estavam no famoso navio Titanic, em sua estréia ocorrida em 1917.

A base de dados analisada contém 1309 instâncias, cujas características consistem em dados sobre os passageiros que estavam no navio na sua primeira e única viagem, quando o mesmo afundou ao bater em um iceberg. A classe de predição é uma característica categórica, que informa se o passageiro sobreviveu (1) ou não (0). É possível ver na Tabela 2 a ordenação das características na base de dados.

Característica	**Descrição**	**Tipo**	**Valores não-nulos**
pclass	Classe do passageiro	categórico	1309
survived	Binário se o passageiro sobreviveu (1) ou não (0)	categórico	1309
name	Nome do passageiro	categórico	1309
sex	Sexo do passageiro	categórico	1309
age	Idade do passageiro	numérico	1046
sibsp	Número de irmãos/esposa presentes	numérico	1309
sex	Número de pais/filhos presentes	numérico	1309
ticket	Número do *ticket*	categórico	1309
fare	Tarifa de embarque	numérico	1308
cabin	Cabine do passageiro	categórico	295
embarked	Porto em que o passageiro embarcou	categórico	1307
boat	Identificação do barco salva-vidas	categórico	486
body	Número de identificação do corpo	categórico	121
home_dest	Destino	categórico	745

Tabela 2: Descrição da base de dados do Titanic

Cada instância contém 13 características, além da característica-alvo (que indica que o passageiro sobreviveu ou não): classe em que o passageiro estava, nome, sexo, idade, número de irmãos/esposas à bordo, número de pais/filhos à bordo, número da passagem, tarifa da passagem, cabine, localidade em que o passageiro embarcou, em qual barco salva-vidas o passageiro estava, se o mesmo conseguiu embarcar em algum dos barcos disponíveis quando o navio afundou, número de identificação do corpo, e localidade de destino do passageiro. Destas, 10 características são categóricas e 4 são numéricas. A porcentagem de características ausentes é cerca de 20.1%, e sua distribuição é: 61.8% dos passageiros não sobreviveram ao acidente, e 38.2% sobreviveram.

6.1.2 Limpeza e Seleção de Dados

Iniciando o pré-processamento da base antes da criação do modelo de AM, primeiramente foi realizada a Limpeza e Seleção de Dados da base. Como descrito anteriormente, esta fase é realizada com o propósito de deixar a base de dados consistente o suficiente para que não ocorra problemas na criação e validação do modelo, removendo possíveis inconsitências que a base possa vir a ter, e que podem atrapalhar a mineração dos dados.

O Pré-Processamento da base de dados foi realizado por meio de um script em Python, utilizando os módulos *Python Data Analysis Library* (conhecido como *pandas*) para manipulação da base de dados em um *dataframe*, e o NumPy em conjunto com a *Open Source Library Library of Scientific Tools*, conhecido como SciPy, para o uso de funções matemáticas. As atividades realizadas na base são descritas abaixo.

6.1.2.1 Identificação de possíveis erros tipográficos

O YaDT trabalha com bases de dados em um formato específico, então foi necessário tratar alguns problemas presentes nesta base. Neste caso, o algoritmo não trabalha com características que possuam aspas, então foi necessário retirá-las das características *name* e *home_dest*. Para isso, a função *replace()* foi aplicada em cada um dos elementos presentes na lista referente à cada característica, com o auxílio das funções *map()* e *lambda()*. Além de remover as aspas, foi necesário também remover as vírgulas de dentro das características, pois isso acarretaria em problemas na leitura do YaDT.

Além disso, optou-se pela divisão do nome completo do passageiro em duas colunas distintas: *name* e *surname*. Esta divisão foi executada por meio da execução da função *split()*, dividindo o nome e o sobrenome por meio do token passado como parâmetro (no caso, o hífen). O snippet de código pode ser visto abaixo.

```
import pandas as pd
import numpy as np
import string as st
from scipy.stats import mode
```

```
def clean_database(df):
    df.name = df.name.map(lambda x: str(x).replace(',', '-'))
    df.name = df.name.map(lambda x: str(x).replace('"', ''))

    df['surname'] = df.name.map(lambda x: str(x).split("-")[0])
    df.name = df.name.map(lambda x: str(x).split("-")[1])

    df.home_dest = df.home_dest.map(lambda x: str(x).replace(',', '␣-'))
    df.home_dest = df.home_dest.map(lambda x: str(x).replace('"', ''))
```

6.1.2.2 Identificação de possíveis inconsistências em valores numéricos e em categorias

Para identificar inconsistências nas características das instâncias, as seguinte atividades foram realizadas:

- Para características categóricas, foram verificadas as categorias que a característica possui e se todas as instâncias estão classificadas em uma destas características.
- Para características numéricas, seus valores foram plotados em um gráfico, onde cada ponto representa uma instância. Com a ajuda do gráfico, possíveis inconsistências podem ser encontradas em valores muito baixos ou muito altos, que ultrapassem o valor esperado para aquela característica.

A base de dados do Titanic possui quatro características numéricas (*age, fare, sibsp e parch*), sendo o restante características categóricas. Porém, como *sibsp* e *sex* possuem poucos valores únicos, estas duas características foram analisadas juntamente com as características categóricas.

Para analisar as características categóricas, novamente foi utilizado um script em Python para verificar quais categorias cada uma das características possui. Isto foi feito por meio da função *unique()*, presente no módulo NumPy, em conjunto com a função *ravel()*, também presente no NumPy. O retorno destas funções utilizadas em conjunto é um *array* contendo os valores únicos, referente às categorias categóricas, encontrados em cada uma das características. A tabela 3 mostra os valores encontrados em cada uma das características.

Como é possível notar na tabela 3, algumas características possuem tantas categorias que as mesmas quase que funcionam como valores únicos para cada instância: apesar de ser interessante observar que haviam homônimos no navio, *name* é quase um índice para cada instância. As características *surname*, *ticket*, *cabin* e *boat* também possuem um alto número de categorias, fazendo com que as instâncias sejam classificadas em pequenos grupos. Estas características não são interessantes para a generalização do modelo, então devem ser tratadas adequadamente. No processo de Engenharia de Características, que será descrito mais à frente, será mostrado como essas caracteristícas foram tratadas neste trabalho para melhorar a generalização do modelo.

Nome da instância	Categorias	Descrição	Instâncias classificadas
name	1137	———	1309
surname	870	———	1309
sex	2	male, female	1309
sibsp	7	0, 1, 2, 3, 4, 5, 8	1309
sex	8	0, 1, 2, 3, 4, 5, 6, 9	1309
ticket	929	———	1309
cabin	187	———	295
embarked	4	C, Q, S, NaN	1307
boat	28	———	486
body	121	———	121
home_dest	370	———	1309
survived	2	0, 1	1309

Tabela 3: Número de categorias de cada característica categórica.

As características restantes (*sex*, *sibsp*, *sex*, *embarked* e *home_dest*) possuem poucas categorias em relação às outras, e não possuem inconsistências em relação à classificação das instâncias na base de dados. Portanto, é necessário apenas tratar as características ausentes em algumas destas características.

Em relação às características *age* e *fare*, a análise foi feita por meio de um gráfico, ou seja, uma simples visualização da relação entre uma característica numérica e o número de instâncias contido na base. Para a geração dos gráficos, um novo script em Python foi criado, para gerar um arquivo-texto apenas com os pontos que interessam ao gráfico.

```
def verifyingDataConsistency(df, column, choice):
    df[column] = df[column].fillna(0.0)
    target = '/home/hialo/UnB/TCC2/codes/databases/' + column + '_data.csv'
    target_sorted = '/home/hialo/UnB/TCC2/codes/databases/' + column + '_data_sorted.csv'

    if (column == 'age'):
        new_df = df[[column]]
        new_df_sorted = new_df.sort([column])

        pdf = pd.DataFrame(new_df_sorted)
        pdf.to_csv(target_sorted, index = False)

    elif (column == 'fare'):
        new_df = df[['pclass', column]]
        new_df = new_df.apply(lambda x: x['fare'] if x['pclass'] == choice else 0, axis =
            1)

        target = '/home/hialo/UnB/TCC2/codes/databases/' + column + '_data_pclass' + str(
            choice) + '.csv'

    pdf = pd.DataFrame(new_df)
    pdf.to_csv(target, index = False)
```

Este método, presente no script principal, busca a coluna referente à característica que se deseja plotar, e cria um novo objeto contendo apenas os dados daquela coluna.

Então,é criado um novo objeto com os dados ordenados de forma crescente, e dois arquivos-texto são criados: um com os dados fora de ordem, apenas removidos das instâncias, e outro contendo os dados ordenados.

No caso de *age*, bastou apenas armazenar os dados ordenados e não-ordenados em arquivos distintos, e por meio do Gnuplot, os mesmos foram plotados, em relação ao número total de instâncias da base. Os gráficos podem ser vistos abaixo:

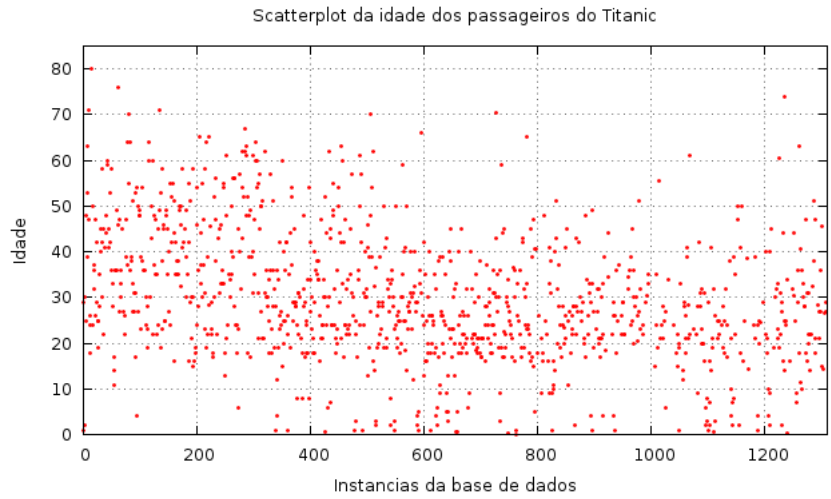

Figura 10: Gráfico da idade dos passageiros em relação à quantidade de instâncias

É possível verificar que a idade dos passageiros é bem distribuída entre as instâncias, se mantendo nos valores esperados para esta característica (entre 0 e 80 anos). O valor máximo encontrado para *age* foi 80, e o menor valor foi 0.16666. Este valor foi encotnrado pelo fato do criador da base ter optado por representar a idade dos recém-nascidos menores que 1 ano presentes no navio na forma de um valor em ponto flutuante. Ou seja, 0.16666 seria o equivalente à idade de 2 meses. Não foram encontradas instâncias com o valor igual a 0, então pode-se concluir que *age* está consistente.

Já a característica *fare* possui ligação direta com outra característica: *pclass*. Os valores para passageiros da classe 1, 2 e 3 possuem diferentes variações, então é interessante dividir estes valores de acordo com o valor de *pclass*. Passando como parâmetro o valor de *pclass* desejado, o método retorna apenas os valores de *fare* cuja instância possui o mesmo valor de *pclass*. Com isso, os seguintes gráficos foram plotados, também em relação ao número total de instâncias:

Como é possível verificar na Fig. (11), a grande maioria das tarifas pagas oscilam entre £25 e £260. Porém, existem 4 pontos destoantes da maioria, na faixa de £512. De acordo com a Titanica (2013), estas 4 instâncias consistem em dois membros da família Cardoza, e seus dois serviçais, Anna Ward e Gustave J. Lesueur. Os Cardoza ocuparam uma das duas suítes mais luxuosas do Titanic, e de acordo com registros históricos, esta suíte custou o valor que consta na base de dados. Portanto, apesar de estarem destoan-

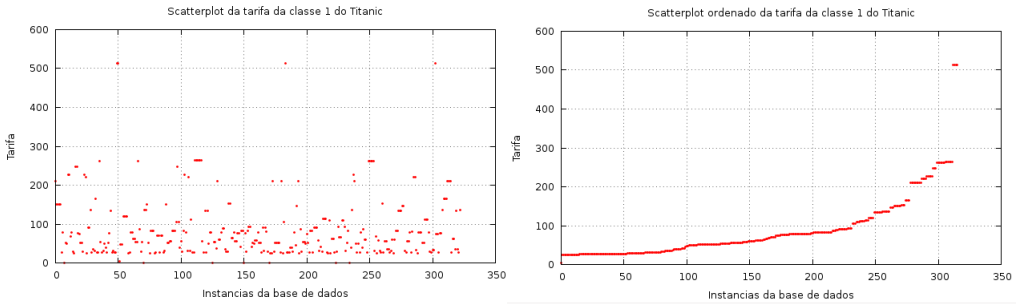

Figura 11: Gráfico da tarifa paga pelos passageiros da 1 classe

tes da maioria, estes valores foram mantidos por haver registros que corroboram estas instâncias.

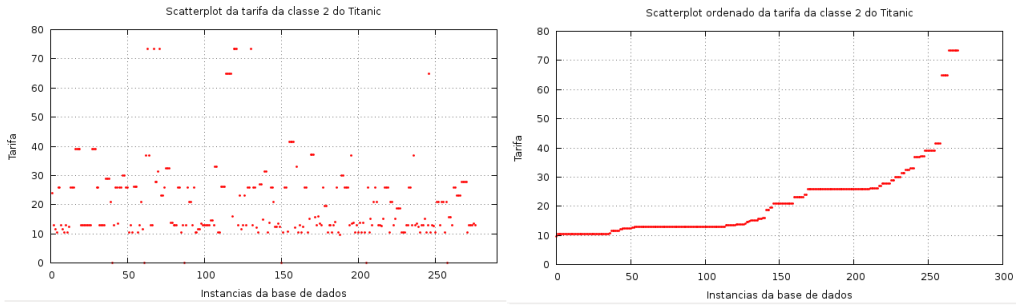

Figura 12: Gráfico da tarifa paga pelos passageiros da 2 classe

Já entre os passageiros da segunda classe, é possível verificar que a maioria das tarifas pagas oscilam entre £10 e £40. Os pontos mais destoantes estão na faixa de £65 e £73. Estes valores são referentes a dois grupos de pessoas que viajaram utilizando o mesmo *ticket*: os membros da família Hermann e o seu filho adotivo, e um grupo de amigos.

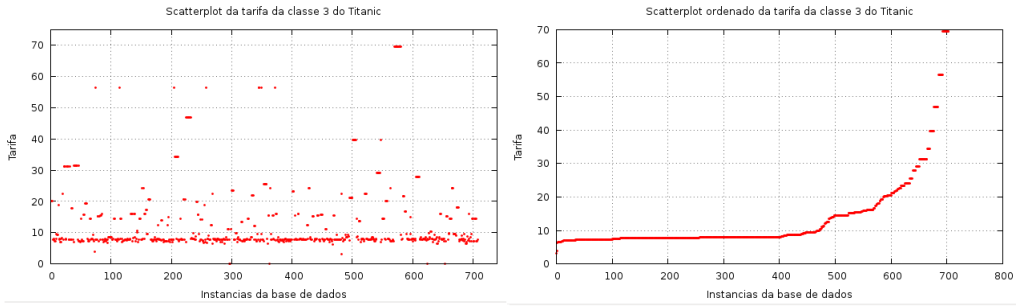

Figura 13: Gráfico da tarifa paga pelos passageiros da 3 classe

A terceira é a que possui a maior quantidade de pontos destoantes em relação às outras duas curvas formadas pelos dados ordenados. A grande maioria dos dados oscilam na faixa entre £7 e £30, e há uma alta variação entre £34 e £69. Como o gráfico dá a entender, estas variações menores tratam de grupos de pessoas que embarcaram no navio

utilizando o mesmo ticket, daí o alto valor comparado às outras instâncias. De acordo com a Titanica (2013), estres grupos de dados são:

- £34.375 - 5 instâncias, referentes à família Ford.
- £39.687 - 7 instâncias, referentes à família Panula e a sua vizinha, Susanna Juhantytar Riihivouri, que acompanhava a familia na viagem.
- £46.899 - 8 instâncias, referentes à família Goodwin.
- £56.495 - 8 instâncias, referentes àos passageiros Lee Bing, Chang Chip, Choong Foo, Ling Hee, Ali Lam, Len Lam, Fang Lang e Lee Ling, um grupo de amigos que viajava junto para Nova Iorque.
- £69.549 - 11 instâncias, referentes à família Sage, que embarcara no navio utilizando o mesmo *ticket*.

Portanto, a diferença visível nos gráficos se dá pelo fato da tarifa estar relacionada com o *ticket* em si, e não com o passageiro. Ou seja, *ticket*s únicos para grandes famílias possuem um valor mais elevado do que um *ticket* para uma única pessoa, o que levou ao aparecimento destes pontos destoantes no gráfico. Porém, foi verificado que estes pontos estão corretos, então a única atividade a ser realizada é limpar a caracterítisca de possíveis valores iguais a zero ou ausentes.

6.1.2.3 Tratamento de Características Ausentes

Conforme pode ser visto na tabela 2, a base de dados do Titanic possui muitas características ausentes. Mais precisamente, 7 das 14 características presentes possuem valores ausentes, sendo que algumas das características estão ausentes em mais de 70% das instâncias. O YaDT possui suporte à características ausentes, porém ainda assim é necessário que estas características sejam tratadas antes de alimentar o algoritmo.

As características *cabin*, *boat*, *home_dest* e *body* foram tratadas da seguinte forma: nas três primeiras, todos os valores ausentes foram preenchidos por *Unknown* (desconhecido, em inglês), e na quarta, pelo valor *0*. Esta abordagem foi escolhida pelo fato de que estas características possuem tantas categorias que as tornam, de certa forma, quase únicas, sendo difícil substituí-las por algum valor válido sem inserir um grande viés na base, até pela grande quantidade de valores ausentes.

Como a característica *embarked* possui apenas 2 valores ausentes, não é necessária a criação de uma nova classe como *Unknown*. Então, neste caso, optou-se por preencher estes valores pelo valor que aparece com mais frequência dentro desta característica. Então, utilizando a função *mode()* presente no módulo SciPy, obteve-se a moda da coluna

embarked, e por meio do método *fillna()*, preencheu-se os dois valores ausentes com o valor mais frequente obtido.

As características *age* e *fare* possuem poucos valores ausentes, assim como *embarked*, então foi utilizada uma abordagem semelhante para as duas. Para a característica *age*, obteve-se a média das idades de todos os passageiros, e os valores ausentes foram preenchidos com esta média. Já em relação à característica *fare*, levando-se em consideração sua alta correlação com a característica pclass, optou-se pelo seguinte procedimento: assim como foi foi feito na análise dos dados, a média da tarifa de cada classe foi calculada, e o valor da característica ausente foi preenchido de acordo com a sua classe. Se a instância era da classe 1, sua tarifa seria preenchida com a média das tarifas da classe 1, e assim por diante.

```
def clean_database(df):
    mean_age = np.mean(df.age)
    df.age = df.age.fillna(mean_age)

    df.fare = df.fare.map(lambda x: np.nan if x == 0 else x)
    classmeans = df.pivot_table('fare', rows='pclass', aggfunc='mean')
    df.fare = df[['fare', 'pclass']].apply(lambda x: classmeans[x['pclass']] if pd.isnull
        (x['fare']) else x['fare'], axis = 1)

    df.cabin = df.cabin.fillna('Unknown')

    embarked = mode(df.embarked)[0][0]
    df.embarked = df.embarked.fillna(embarked)

    df.boat = df.body.fillna('Unknown')

    df.body = df.body.fillna('0')

    df.home_dest = df.home_dest.fillna('Unknown')
    df.home_dest = df.home_dest.map(lambda x: str(x).replace(',', ', -'))
    df.home_dest = df.home_dest.map(lambda x: str(x).replace('"', ''))

    return df
```

6.1.3 Engenharia de Características

Ao terminar a Limpeza e Seleção dos Dados, foi aplicada a técnica de Redução de Dimensionalidade na base de dados para a remoção de algumas características por meio da Seleção de Características, descrita anteriormente. Para isso, algumas características foram analisadas e removidas, pois o seu Ganho de Informação seria muito baixo ou a sua presença transformaria o modelo em algo muito simples para a descoberta de padrões.

Antes da E.C ter sido aplicada, uma árvore de decisão de teste foi gerada utilizando a base de dados "crua", ou seja, todas as características presentes. A árvore gerada com esta base pode ser vista na Fig. (14):

Como é possível verificar, a árvore de decisão criada é muito pobre: basicamente, se

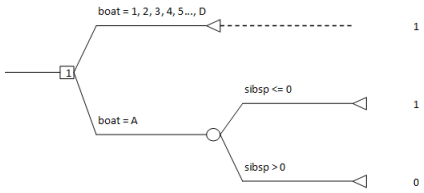

Figura 14: Árvore de Decisão gerada com os todos os dados da base Titanic.

estava no barco salva-vidas, sobreviveu. Se não estava, morreu. Apesar de ter conseguido um excelente critério de acerto na classificação das instâncias de teste, para mineração de dados esta árvore não auxilia em nada na descoberta de novos padrões. Ao analisar estas duas características em conjunto, é possível notar o porquê da criação de uma árvore como esta: de 500 sobreviventes, 477 estavam em um barco salva-vidas identificado, e dos 809 que perderam a vida, apenas 9 estavam em barcos salva-vidas.

Analisando um pouco mais à fundo, é possível notar outra característica perigosa: *body*. Apesar de possuir apenas 121 valores válidos, ela pode reduzir drasticamente o tamanho da árvore, já que sempre que ela possuir um valor válido, a árvore irá corretamente classificar a instância como 0, já que apenas mortos possuem um identificador válido.

Portanto, é possível notar que *boat* e *body* são duas características que tornam a árvore muito fraca para mineração de dados. Portanto, foram removidas na primeira rodada do processo de E.C. Juntamente com essas, porém pelo motivo oposto, outras três características foram removidas: *name*, *surname* e *ticket*. Devido às suas grandes quantidades de categorias, estas características servem praticamente como um identificador da instância, o que as tornam inúteis para a generalização esperada do modelo.

6.1.4 Criação do Modelo

Após as etapas de Pré-Processamento da base de dados e da Engenharia de Características estarem concluídas, finalmente o primeiro modelo de AM foi criado. Porém, antes de visualizar o modelo que foi gerado pelo algoritmo C4.5, é necessária uma pequena descrição de como o sistema YaDT funciona.

Além da base de dados, o sistema exige um segundo arquivo contendo os "metadados" da base, ou seja, a descrição de cada característica das instâncias que compõem a base. Estes metadados devem estar no seguinte formato:

name, data_type, column_type

Sendo que *name* é o nome da característica que será apresentado no modelo gerado, *data_type* é o tipo do dado (inteiro, float, string ou nulo), e *column_type* é o tipo da característica (numérica, categórica ou classe). O YaDT possui suporte à outros tipos de característica, como pesos, porém neste trabalho não é necessário o entendimento destes

tipos. O arquivo contendo os metadados usados para a geração do modelo foi o seguinte:

```
pclass,integer,discrete
sex,string,discrete
age,float,continuous
sibsp,integer,continuous
parch,integer,continuous
ticket,string,discrete
fare,float,continuous
cabin,string,discrete
embarked,string,discrete
home_dest,string,discrete
survived,integer,class
```

Neste caso, discrete (discreto, em português) seria o equivalente às características categóricas, e continuous (contínuo, em português) seria o equivalente às características numéricas, sendo apenas uma questão de nomenclatura.

O YaDT possui uma GUI, porém é recomendável que o sistema seja executado pela linha de comando, devido ao maior controle sobre os parâmetros ajustáveis do C4.5 por este método. Para a criação dos modelos apresentados neste trabalho, a seguinte linha foi utilizada:

dTcmd64 -fm *METADATA_FILE* -fd *DATABASE_FILE* -c4.5 -c *CONFIDENCE* -m *SPLIT* -x *TREE_XML_FILE* -tb *TREE_BINARY_FILE* -t *MATRIX_OUT_FILE* -l *LOG_OUTPUT_FILE*

Em que:

- METADATA_FILE: Arquivo contendo os metadados da base.
- DATABASE_FILE: Base de dados cujas instâncias serão utilizadas para a criação do modelo.
- CONFIDENCE: Parâmetro do fator de confidência utilizado para a poda. O Capítulo 4 possui mais detalhes de como a poda do algoritmo C4.5 funciona.
- SPLIT: Número minímo de casos para que um nó-folha seja criado.
- TREE_XML_FILE e TREE_BINARY_FILE: O sistema salva a árvore de decisão gerada em dois arquivos, um no formato XML e o outro em formato binário. É possível verificar a árvore de decisão carregando-a na interface do sistema.

- MATRIX_OUT_FILE: Arquivo de texto contendo a porcentagem de erro obtida pela árvore de decisão criada, a matriz de confusão, e a árvore em si.

- LOG_OUTPUT_FILE: Arquivo de texto contendo o log de execução do sistema.

Como é possível ver na linha de comando utilizada, existem dois fatores utilizados na construção da árvore que chamam a atenção: *CONFIDENCE* e *SPLIT*. Mitchell (1997) e Witten e Frank (2000) recomendam o uso do valor padrão para o fator de confidência, 25%.

Para *SPLIT*, foi realizado o seguinte teste para verificação de qual seria o melhor valor a ser utilizado na geração do modelo: foram criadas 498 árvores, com a mesma base de dados, apenas variando o valor de *SPLIT* entre 2 e 500, utilizando CONFIDENCE = 0.25. Então, a porcentagem de erro da árvore gerada no conjunto de treinamento é utilizada como parâmetro para definição do valor de *SPLIT* ótimo.

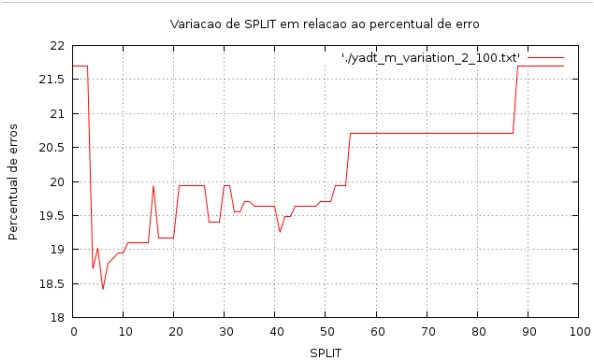

Figura 15: Variação de SPLIT indo de 2 a 100.

Como é possível verificar na Fig. (15), o ponto ótimo de SPLIT ocorre entre 0 e 10, mais precisamente quando SPLIT é igual a 6. Há uma variação nos percentuais de erro das árvores com o valor de SPLIT entre 10 e 50, e a partir deste valor, o percentual tende a subir indefinidamente.

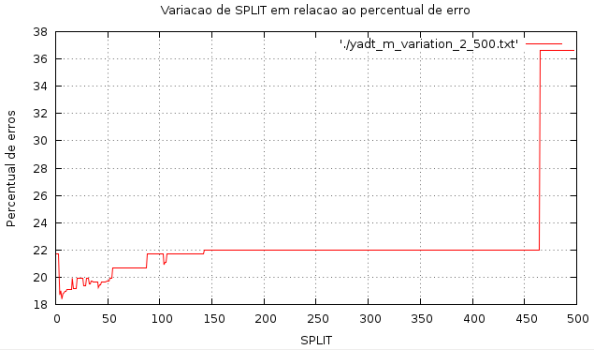

Figura 16: Variação de SPLIT indo de 2 a 500.

A Fig. (16) apresenta os percentuais de erro das 498 árvores, em que é possível ver de forma mais clara este comportamento: quando o valor de SPLIT se aproxima de 500, o erro de classificação da árvore chega a ultrapassar os 36%.

Tendo o valor ótimo de SPLIT para este modelo, este valor foi utilizado para verificar qual seria o melhor valor para CONFIDENCE. O mesmo processo para a descoberta do SPLIT ótimo foi utilizado, sendo criadas 20 árvores distintas variando o valor de CONFIDENCE entre 0.05 e 1.0 (de 5% a 100%, variando de 5 em 5%). A tabela 4 apresenta os resultados obtidos:

CONFIDENCE	**Perc.Erro**	**Tamanho da Árvore**
0.05	19.9389	8
0.10	19.9389	8
0.15	18.7166	21
0.20	18.7166	21
0.25	18.7166	21
0.30	18.7166	21
0.35	18.7166	21
0.40	18.7166	21
0.45	18.7166	21
0.50	18.7166	21
0.55	18.7166	21
0.60	18.7166	21
0.65	18.7166	21
0.70	18.7166	21
0.75	18.7166	21
0.80	18.7166	21
0.85	7.63942	950
0.90	7.63942	950
0.95	7.63942	950
1.00	7.41024	1320

Tabela 4: Testes para o valor de CONFIDENCE ótimo - Titanic

No caso de CONFIDENCE, seus valores foram analisados levando em consideração o tamanho da árvore gerada, além do percentual de erro. Nos casos em que CONFIDENCE é menor que 10% obtiveram-se as menores árvores, porém com um percentual de erro maior em relação ao intervalo entre 15% e 80%. De 85% em diante obteve-se percentuais de erros ótimos, porém ao custo de uma árvore gigantesca. Ou seja, uma árvore com problemas de *overfitting*, já que a sua construção não foi podada adequadamente. Ao se analisar estas árvores, percebeu-se que estes percentuais de erro foram pelo fato de novos ramos relacionados à todas as categorias de *ticket* ou de home_dest, por exemplo, terem sido criados e não podados, criando uma árvore gigantesca.

Portanto, os parâmetros utilizados para a geração do modelo de AM apresentado foram: $CONFIDENCE = 0.25$, já que é um valor considerado padrão e apresentou bons

resultados no seu teste, e $SPLIT = 6$. Então, finalmente o primeiro modelo foi gerado.

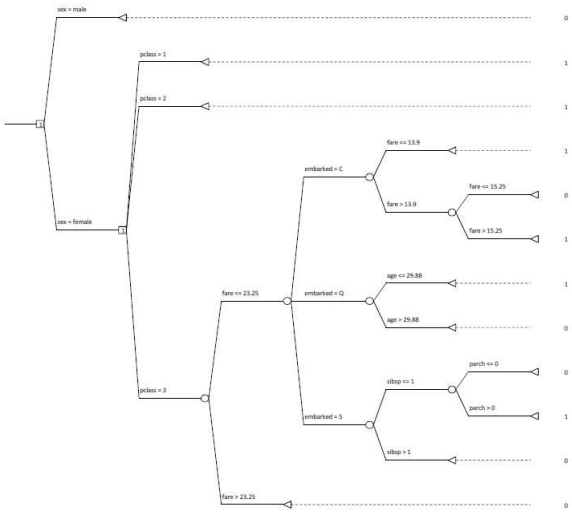

Figura 17: Árvore de Decisão gerada com os dados da base Titanic.

Como é possível verificar na Fig. (17), a característica home_dest não está presente no modelo, pois devido ao seu grande número de categorias, o Ganho de Informação gerado por esta característica foi baixo, sendo seus nós possivelmente podados da árvore. A característica com o maior Ganho de Informação foi a característica *sex*, portanto o nó-raiz da árvore foi criado utilizando esta característica como critério de divisão. O restante das características foi utilizado para a criação dos nós de decisão restantes.

6.1.5 Validação do Modelo

O percentual de erro calculado pelo YaDT foi de 18.71%. Na tabela 5 é possível visualizar a matriz de confusão gerada para este modelo.

VP = 765	FP = 201
FN = 44	VN = 299

Tabela 5: Matriz de confusão do modelo gerado - Titanic

A quantidade de instâncias corretamente classifcadas como 0 é igual a 765/809, e a quantidade de instâncias corretamente classificadas como 1 é igual a 299/500. Em relação às taxas de Falsos Positivos e Negativos, a quantidade de instâncias incorretamente classificadas como 0 é igual a 201/500, ou seja, cerca de 40% do total de instâncias, e a quantidade de instâncias incorretamente classificadas como 1 é 44/809, cerca de 5.4%. A principal causa para uma discrepância tão grande foi a poda realizada pelo algoritmo no nó de decisão: como para o modelo, todas as instâncias do sexo masculino morreram, a classificação acaba sendo prejudicada por conta disso. Já quando a instância é do sexo feminino, o modelo realiza uma classificação bem mais precisa.

Juntamente com a matriz de confusão, foi utilizada a técnica de Validação Cruzada para a validação do modelo, conforme descrito nos capítulos anteriores. Mais precisamente, foi utilizada a abordagem de k-partições. Esta validação tem como objetivo obter uma estimativa de erro mais próxima do real para o modelo, já que a estimativa de erro obtida no YaDT é, geralmente, um pouco otimista.

Primeiramente, assim como na definição dos valores ótimos de SPLIT e CONFIDENCE, obteve-se qual seria o k ótimo. Para isso, foi desenvolvido um código em C++ que executa os seguintes passos: o algoritmo recebe a base de dados total e o valor de k desejado. Então, realiza a leitura da base de dados e armazena as instâncias como um objeto do tipo Instance, formando um vetor destes objetos. Então, este vetor é embaralhado e dividido em k partições, dentro de um loop. Ao final, serão geradas 2k bases de dados: k bases de testes, variando de 1 até k, e k bases de treinamento, que consistem no número de instâncias que não estão presentes na base de testes equivalente.

```
#ifndef INSTANCE_H
#define INSTANCE_H

using namespace std;

#include <vector>
#include <string>

class Instance {
  public:

    Instance();
    Instance(vector<string> attributes);
    Instance(vector<string> attributes, int classification);
    int getClassification();
    vector<string> getAttributes();

  private:

    vector<string> attributes;
    int classification;
};

#endif
```

O código completo se encontra no repositório do trabalho: <http://github.com/hialo/decisiontreebuilder>. No snippet acima, a classe Instance criada para receber as instâncias da base e armazená-las. Como as duas bases utilizadas neste trabalho possuem classes que produzem umr esultado inteiro, optou-se por utilizar o tipo *int* na variável *classification*.

Então, um script em Shell Script foi criado para executar o algoritmo de Validação Cruzada variando o valor de k de 0 até o tamanho da base menos um, ou seja, 1308 (que é uma técnica de Validação conhecida e bastante utilizada, chamada Leave One Out). Este script funciona de acordo com o pseudo-código apresentado abaixo.

```
1 for each k entre 2 e tamanho da base - 1
2   kfoldtests(k) \\ gerando as bases de dados
3
4   for each i entre 0 e k
5       dTcmd64 -fm METADATA_PATH -fd TRAIN_PATH + i -ft TEST_PATH + i...
```

Ou seja, para cada valor de k, o script irá criar todas as k árvores possíveis utilizando o método de k-partições, e irá armazenar o resultado de cada um deles. Após o script terminar de gerar todos os resultados possíveis dentro do intervalo, um script em Python realiza a leitura do percentual de erro dentro do arquivo matrix.txt gerado pelo YaDT de todos os k modelos gerados e retorna a média do percentual de erro de cada iteração do algoritmo. O script pode ser visualizado abaixo:

```
def kfold():
    path = '/home/hialo/UnB/TCC2/codes/results/generated_trees/'
    var = 0
    NUMBER_OF_CASES = 1308
    LINE = 10

    for first_index in range (3, NUMBER_OF_CASES):
        var = 0
        for second_index in range (0, first_index):
            full_path = path + str(first_index) + '/' + str(second_index) + '/matrix.txt'

            f = open(full_path)
            lines = f.readlines()
            lines[LINE] = lines[LINE].split(": ")[1]

            lines[LINE] = lines[LINE].split("%")[0]
            var += float(lines[LINE])

            f.close()

        print var/first_index
```

Após a execução do algoritmo e a obtenção dos pontos, o seguinte gráfico foi obtido:

É possível verificar que as maiores variações ocorrem logo nos primeiros valores de k, e mais à frente o gráfico tende a estabilizar um pouco. Como há muitas variações de acordo com os conjuntos de dados de treinamento e testes obtidos, não é possível obter um valor para k que seja ótimo, pois um valor de k cujo percentual de erro seja o menor obtido pode não ser mais válido se uma nova iteração do algoritmo de Validação Cruzada. Portanto, o valor escolhido para k é igual a 10, pelos seguintes motivos: é considerado o valor padrão na literatura (Witten e Frank (2000), Mitchell (1997)) e não exige muito poder de processamento, já que o custo computacional para a execução do algoritmo

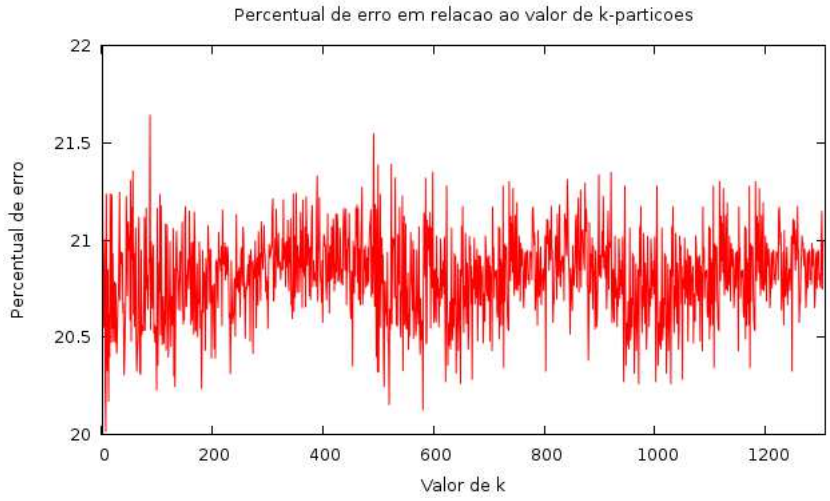

Figura 18: Percentual de erro em relação ao valor de k

cresce de forma linear.

Então, foram criados 10 bases de dados de treinamento e 10 bases de teste, e em cada conjunto de dados de treinamento e teste, o erro foi calculado. Ao final, a média destes erros foi obtida. O erro médio obtido por meio da Validação Cruzada foi de 20.75%, cerca de 2% acima do erro calculado na geração do modelo, o que era esperado. O código utilizado para leitura de todos os percentuais de erro gerados durante as iterações da Validação Cruzada é o mesmo utilizado para a leitura dos percentuais de erro para a descoberta do k ótimo, apenas modificando o endereço das bases.

Além disso, com o auxílio das Matrizes de Confusão geradas por cada modelo durante a execução da técnica de k-partições, a curva ROC referente a este modelo foi plotada. A cada modelo gerado, os valores referentes às suas Taxas de Verdadeiros Positivos (TVP) e Taxas de Falso Positivo (TFP) foram calculadas. Estas taxas foram calculas segundo as equações abaixo:

$$TVP = \frac{VP}{(VP + FN)} \tag{6.1}$$

$$TVN = 1 - \frac{VN}{(FP + VN)} \tag{6.2}$$

Como é possível ver, estas taxas são calculadas com base nos quatro valores que compõem a matriz de confusão de cada modelo. Estas taxas então foram plotadas em um gráfico na forma de uma curva.

A Fig. (19) mostra a curva ROC gerada por uma única iteração da técnica de k-partições. Como é possível verificar, o modelo está acima da curva, que significa que a Árvore de Decisão gerada possui performance satisfatória.

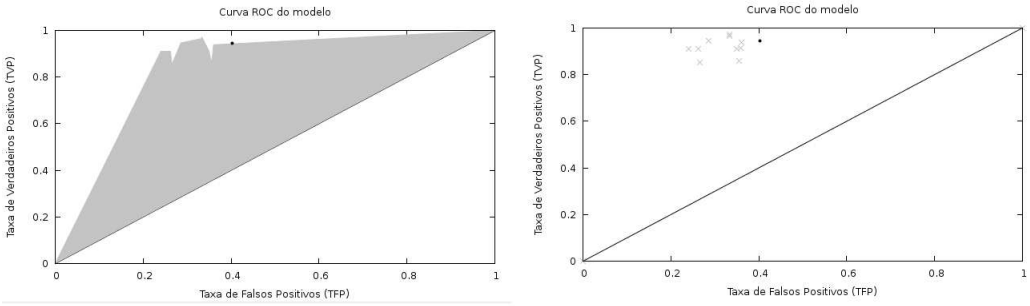

Figura 19: Curva ROC do modelo - Única iteração

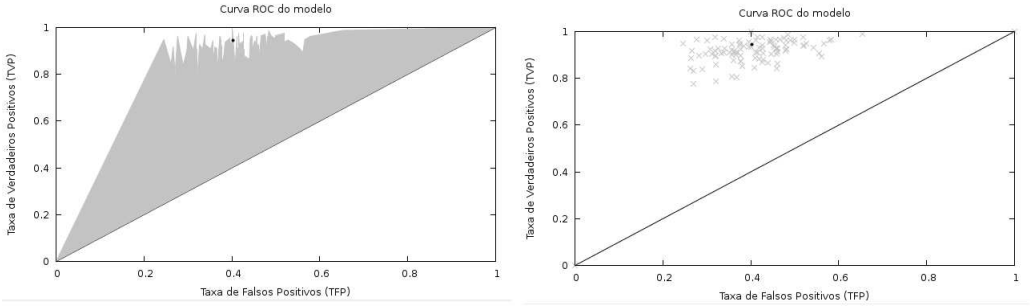

Figura 20: Curva ROC do modelo - 10 iterações

A Fig. (20) mostra a curva ROC gerada por 10 iterações da técnica de k-partições, ou seja, esta curva é composta por 100 pontos distintos. Nesta curva, é possível visualizar que o modelo está no limiar da curva.

6.2 Base de Dados - PNCRC/Bovinos

6.2.1 Descrição da Base de Dados

A base de dados analisada nesta etapa foi cedida pelo MAPA (Ministério da Agricultura, Pecuária e Abastecimento). Esta base faz parte dos dados obtidos pelo PNCRC (Plano Nacional de Controle de Resíduos e Contaminantes), um programa federal de inspeção e fiscalização das cadeias produtivas de alimentos, cujo objetivo é a monitoração da efetividade dos controles implementados pelos sistemas de produção e a respectiva qualidade e segurança dos produtos de origem animal e vegetal disponibilizados ao comércio e ao consumo.

O PNCRC é dividido em duas grandes áreas: O PNCRC/Animal e o PNCRC/-Vegetal, sendo que cada um possui uma regulamentação específica referente à sua área. O PNCRC/Animal é composto pelos seus programas setoriais, para o monitoramento em carnes (PNCRC/Bovinos, PNCRC/Aves, PNCRC/Suínos, PNCRC/Equinos, PNCRC/Avestruz e PNCRC/Caprinos e Ovinos) e demais produtos de origem animal (PNCRC/-Leite, PNCRC/Mel, PNCRC/Ovos e PNCRC/Pescado)(AGRICULTURA, 2014), sendo que a base de dados analisada neste trabalho faz parte do programa PNCRC/Bovinos.

De forma resumida, o PNCRC funciona da seguinte forma: Por meio de um plano estatístico definido, são coletadas diversas amostras em estabelecimentos registrados no SIF (Serviço de Inspeção Federal). Todos os frigoríficos registrados no SIF são inseridos para participar dos sorteios semanais para a coleta de amostras. Um sistema informatizado, o SISRES (Sistema de Controle de Resíduos e Contaminantes), que é responsável por gerenciar o encaminhamento de amostras e os respectivos resultados laboratoriais, gerencia as informações do plano e realiza o sorteio aleatório dos estabelecimentos onde as amostras serão coletadas pelos fiscais do Ministério da Agricultura. São, em média, 52 sorteios semanais, com coletas realizadas em todo o País, entre 1º de janeiro e 31 de dezembro de cada ano.

Após a análise e uma vez identificada violação do LMR (Limite Máximo de Resíduo) ou a utilização de uma droga de uso proibido no País, o Ministério da Agricultura adota ações regulatórias imediatas, a fim de evitar que o produto contaminado vá para o mercado. Além disso, são colocadas em prática ações na propriedade, com a finalidade de identificar as causas dessa infração.

A base de dados analisada pertence ao grupo dos bovinos, e contém as análises realizadas em amostras deste tipo nos períodos de 2002 a 2014. Esta base contém as análises apenas de um subgrupo de substâncias analisadas pelos laboratórios, como Antiparasitários. A classe de predição é uma característica categórica multi-classe, contendo as possíveis classificações: não houve detecção da substância testada na amostra, houve deteccção mas o valor obtido não ultrapassou o LMR, e o valor detectado ultrapassou

o LMR referente à substância testada. A base analisada contém 32114 instâncias, sendo que cada instância é formada por 23 características. A tabela 6 apresenta um pequeno resumo das instâncias da base:

Característica	**Descrição**	**Tipo**	**Valores Não-Nulos**
ANO	Ano em que a análise foi realizada	categórico	32114
N_ANALISE	N. de identificação da análise	categórico	32114
SIF	Código do Serviço de Inspeção Federal	categórico	32114
NOME_PROPR	Nome da propriedade em que a amostra foi obtida	categórico	31893
UF	UF da propriedade	categórico	31941
COD_PROPR	Código de identificação da propriedade	categórico	712
NOME_PROPRIET	Nome do proprietário	categórico	32002
NOME_MUN	Nome do município	categórico	31995
CEP	5 primeiros dígitos do CEP da propriedade	categórico	30934
CEP2	3 últimos dígitos do CEP da propriedade	categórico	25044
COD_ESPECIE	Código da espécie da amostra	categórico	32114
COD_TIPAN	Código do grupo do qual a substância testada pertence	categórico	32114
QTD_ANIMAIS	Quantidade de animais do lote testado	numérico	31849
SEMANA	Semana da análise	categórico	32114
END_PROPR	endereço da propriedade	categórico	14487
TIPO_PESSOA		categórico	14641
UF_PROPRIET	UF do proprietário	categórico	14487
MUN_PROPRIET	Município de origem do proprietário	categórico	14470
CEP_PR	CEP do proprietário	categórico	14487
COD_RESIDUO	Código da substância	categórico	32114
RESULTADO	Resultado da análise da substância	numérico	13459
SIT_VIOLAC	Se houve violação (1) da amostra ou não (0)	categórico	21187
STATUS	Classe que descreve a ação que será tomada	categórico	32114

Tabela 6: Descrição da base de dados das amostras bovinas do PNCNR

6.2.2 Limpeza e Seleção de Dados

Assim como na base de dados do Titanic, foi realizado o pré-processamento da base de dados antes que a mesma alimentasse o algoritmo. Conforme descrito nos próximos tópicos, algumas características foram tratadas em relação à erros de classificação da instância, além da etapa de tratamento de valores ausentes. Estas etapas foram realizadas novamente por meio de um script em Python, utilizando os módulos *pandas* e SciPy.

6.2.2.1 Identificação de possíveis inconsistências tipográficas

Assim como realizado na base de dados do Titanic, nesta etapa foram removidas possíveis inconsistências que pudessem atrapalhar na leitura dos dados realizada pelo YaDT.

As características numéricas ANO, N_ANALISE, SIF, COD_TIPAN, CEP e QTD_ANIMAIS, devido ao filtro do SISRES, vieram com vírgulas em valores acima de 1000 ao serem salvas na tabela, além das aspas. Novamente, utilizando a função *replace()* em conjunto com a função *map()* para aplicar a função em todos os elementos do *dataframe*, as vírgulas e aspas foram removidas de tais características.

Além disso, a característica CEP2 possuía valores iguais a 0. Estes valores foram substituídos pela sua representação correta, 000, novamente por meio das funções *map()* e *lambda()*.

6.2.2.2 Tratamento de Características Ausentes

Conforme pode ser visto na tabela 6, 16 das 23 características presentes na base possuem valores ausentes, sendo que algumas características possuem mais de 80% de seus valores ausentes. Apenas algumas características foram tratadas nesta etapa, já que algumas das características que compõem a base seriam removidas na etapa de E.C, sendo desnecessário o tratamento de tais características.

As características tratadas nesta etapa foram: UF, NOME_MUN, CEP, CEP2, QTD_ANIMAIS e RESULTADO. Primeiramente, as características UF, NOME_MUN, CEP e CEP2 foram tratadas em conjunto. Por se tratarem de características referentes à localização das propriedades onde as amostras foram recolhidas, tais características podem ser consideradas complementos uma da outra. Ou seja, é possível inferir o valor de determinada característica de acordo com o valor de outra. Como pode ser visto na tabela 6, a característica NOME_MUN possui o menor número de valores ausentes, 119, em relação às outras características. Com isso, foram inferidos os valores de UF e CEP nas instâncias em que NOME_MUN era conhecido. Com isso, foi possível preencher de 64 estados e CEPs com exatidão.

Em relação à CEP e CEP2, existia uma diferença de cerca de 5000 valores ausentes a mais em CEP2. Ao se analisar os dados de CEP, foi possível verificar que, quando o valor de CEP existia e o de CEP2 não, CEP2 era igual a 000. Então, estes valores foram preenchidos de acordo.

Com isso, restaram cerca de 120 instâncias com valores ausentes para estas características, que não puderam ser inferidos por conta do valor de todas estas características (UF, NOME_MUN, CEP e CEP2) estarem ausentes na instância. Neste caso, optou-se pela abordagem padrão, substituir estes valores ausentes por *Unknown*.

Em QTD_ANIMAIS, optou-se por preencher os valores ausentes pela média de todas as instãncias, mesmo procedimento adotado na característica *age* na base de dados do Titanic. Utilizando a função *mean()* do módulo NumPy, esta média foi calculada e os valores ausentes foram preenchidos. Já em RESULTADO, de acordo com o Ministério da Agricultura, os valores ausentes foram um erro nos filtros do SISRES, e deveriam ser considerados iguais a 0. Portanto, todos os valores ausentes desta característica foram preenchidos de acordo.

6.2.3 Engenharia de Características

Nesta etapa, a primeira etapa realizada foi uma substituição. Algumas das características presentes nas instâncias são confidenciais ao Ministério, então tais características foram trocadas por um valor mapeado e sem significado aparente.

Estas características são o SIF e o NOM_PROPR, referente ao nome do estabelecimento em que a amostra foi recolhida. Como cada SIF é único (um mesmo estabelecimento pode ter mais de um SIF), foi criada uma tabela adicional em que cada SIF e a sua respectiva propriedade foram mapeados em um índice, chamado de SIF_INDEX, e estas duas características foram substituídas por este índice, afim de se preservar a confidencialidade dos fornecedores das amostras.

Também foi solicitado também que fossem removidos os dados referentes aos proprietários. Portanto, as seguintes características foram removidas: NOME_PROPRIET, END_PROPRIET, TIPO_PESSOA, UF_PROPRIET, MUN_PROPRIET e CEP_PR. Por este motivo, as mesmas não passaram pela análise das categorias no tópico anterior. Como grande parte destas características possuiam uma grande quantidade de características (mais de 50% das instâncias), sua remoção não trouxe prejuízos.

Por esta razão, optou-se pela remoção da característica COD_PROPR. Com apenas 712 ocorrências, ou seja, presente em apenas 2% das ocorrências, esta característica serve como uma espécie de índice (como *name* na base do Titanic, por exemplo). Já para a característica COD_ESPECIE, como este valor é único para toda a base (afinal, esta base consiste apenas de testes com amostras de carnes bovinas), não há necessidade de

mantê-lo, então o mesmo também foi removido.

Em relação às características CEP e CEP2, elas foram unidas em uma única característica, CEP.

Finalizando, os valores presentes nas categorias COD_RESIDUO e COD_TIPAN foram substituídos pelo nome da substância referente ao código presente, para facilitar a visualização dos padrões no modelo gerado. Ao final, obteve-se os seguintes metadados:

```
ANO,integer,discrete
SEMANA,integer,discrete
N_ANALISE,integer,discrete
SIF_INDEX,integer,discrete
UF,string,discrete
CEP,string,discrete
QTD_ANIMAIS,integer,continuous
COD_TIPAN,string,discrete
COD_RESIDUO,string,discrete
RESULTADO,float,continuous
STATUS,integer,class
```

O código do script utilizado para as atividades de Pré-Processamento de Dados e E.C pode ser visto no repositório deste trabalho.

6.2.4 Geração do Modelo

Terminadas as etapas de Pré-Processamento de Dados e E.C, o modelo de AM foi gerado, por meio do YaDT. Assim como na base de dados do Titanic, primeiramente foram calculados quais os melhores valores para SPLIT e CONFIDENCE, antes de se gerar o modelo definitivo utilizando todas as instâncias da base.

A metodologia utilizada foi a mesma: primeiramente, tentou-se obter o valor ótimo de SPLIT variando o mesmo de 2 a 500, utilizando o valor padrão para CONFIDENCE, 0.25. Então, o valor calculado de SPLIT é utilizado para se obter o valor ótimo de CONFIDENCE.

Na Fig. (21), é possível ver a variação do percentual de erro em relação ao valor de SPLIT. Diferentemente da base de dados do Titanic, a taxa de erro manteve um comportamento mais estável, sem mostrar variação de 0 até cerca de 70. Então, até cerca de 120 a taxa de erro sobe consideralmente e então, volta a ficar estável. Como é possível perceber, também não foi possível obter um valor de SPLIT ótimo. Portanto, o valor utilizado para a geração do modelo foi o valor mínimo para SPLIT, 2.

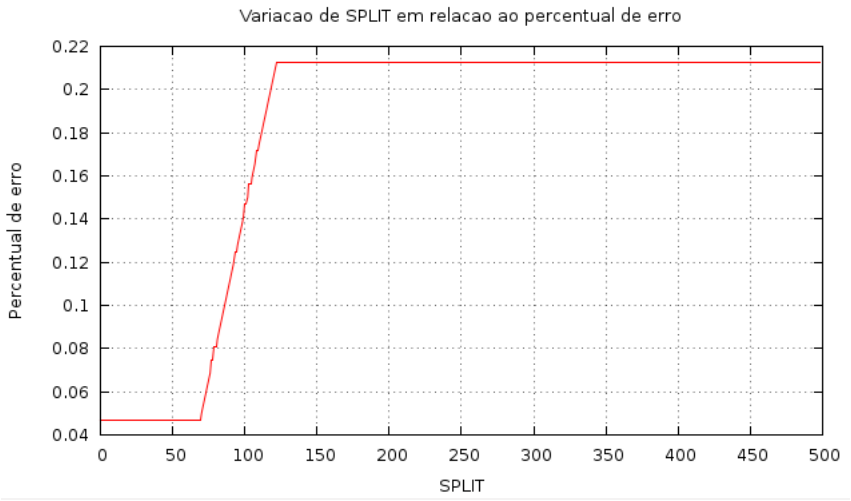

Figura 21: Variação de SPLIT indo de 2 a 500

Com o valor de SPLIT, foi calculado qual seria o melhor valor de CONFIDENCE para a geração do modelo. O mesmo procedimento foi adotado: variando o percentual de 5 em 5%, obteve-se todos os percentuais de erro no intervalo de 5 a 100%, valor máximo permitido. A tabela 7 apresenta os resultados obtidos.

CONFIDENCE	**Perc.Erro**	**Tamanho da Árvore**
0.05	0.046861	5
0.10	0.046861	5
0.15	0.046861	5
0.20	0.046861	5
0.25	0.046861	5
0.30	0.046861	5
0.35	0.046861	5
0.40	0.021868	20
0.45	0.021868	20
0.50	0.021868	20
0.55	0.021868	20
0.60	0.021868	20
0.65	0.018744	73
0.70	0.018744	73
0.75	0.018744	73
0.80	0.018744	73
0.85	0.018744	73
0.90	0.018744	73
0.95	0.018744	73
1.00	0.000000	6282

Tabela 7: Testes para o valor de CONFIDENCE ótimo - PNCRC/Bovinos

Na escolha de CONFIDENCE, o tamanho da árvore também foi levado em consideração. As árvores geradas nos intervalos de 5 a 35% foram consideradas muito simples,

sendo constituídas por apenas uma característica. As árvores geradas nos intervalos de 40 a 60% apresentaram uma taxa de erros menor e complexidade maior, porém ainda sim sendo muito simples. Entre 65 e 95% as árvores obtiveram uma taxa de erro um pouco menor, porém seu tamanho ficou bem mais elevado. Com CONFIDENCE igual a 100%, temos uma árvore com graves problemas de *overfitting*: uma acurácia perfeita, porém totalmente presa à esta base de dados em específico, apresentando resultados bem piores caso um conjunto de casos de teste desconhecido fosse apresentado.

Portanto, os valores escolhidos foram SPLIT = 2, já que os resultados foram os mesmos (ou seja, não foi possível obter um SPLIT ótimo, como no caso da base do Titanic), e CONFIDENCE = 0.40. Optou-se pela mesma abordagem utilizada anteriormente, a escolha de uma taxa de confidência cuja árvore apresentasse uma boa taxa de erros com um tamanho razoável. Então, utilizando os valores descritos, o seguinte modelo foi obtido:

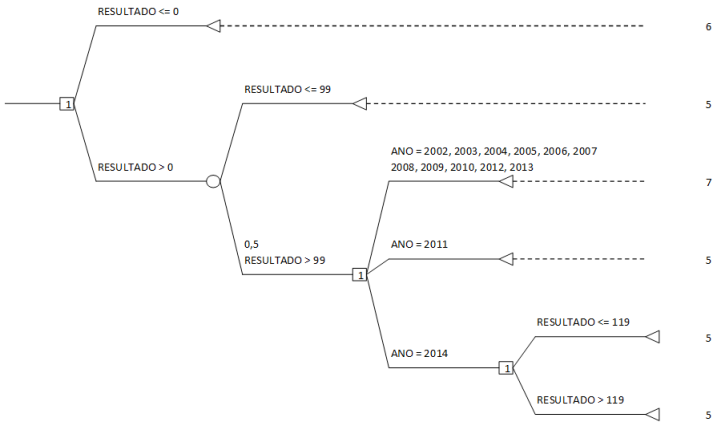

Figura 22: Árvore de Decisão gerada com os dados da base PNCRC/Bovinos.

Como é possível verificar, obteve-se uma árvore pequena para a obtenção de padrões, mas com uma alta acurácia: o modelo gerado classificou apenas 7 instâncias de forma incorreta, de um total de 32114, ou seja, uma taxa de erro de apenas 0.02%. Este valor fica ainda mais aparente quando comparado ao valor obtido no modelo anterior, gerado a partir da base do Titanic, que ficou em cerca de 18%. Apesar de suspeito (um valor tão baixo é uma forte suspeita de um modelo com problemas de *overfitting*), tal resultado é discutido no próximo tópico, na validação do modelo.

6.2.5 Validação do Modelo

Conforme mencionado na seção anterior, o percentual de erro do modelo gerado em relação à base de dados foi muito baixo, cerca de 0.02%. A tabela 8 contém a matriz de confusão do modelo em questão:

Como é possível verificar, o modelo gerado não errou nenhuma instância cuja classe esperada era a não-detecção (STATUS igual a 6), ou seja, 31127/31127. Isto já era

Esperada / Obtida	**STATUS = 5**	**STATUS = 6**	**STATUS = 7**
STATUS = 5	814	0	7
STATUS = 6	0	31127	0
STATUS = 7	0	0	61

Tabela 8: Matriz de confusão do modelo gerado - PNCRC/Bovinos

esperado, já que uma amostra não-detectada é simplesmente uma amostra que obteve resultado 0 no teste. É interessante notar que o único problema que o modelo teve foi na classificação de instâncias cujo valor de detecção foi maior que o LMR permitido com STATUS = 5, e não igual a 7. Já as instâncias cujas amostras não ultrapassaram o LMR foram todas corretamente classificadas. O número de instâncias incorretamente classificadas como 5 foi de 7/68.

Novamente, a técnica de Validação Cruzada foi utilizada para se verificar a qualidade do modelo gerado, ou seja, para se ter uma estimativa de como o modelo se comportaria caso fosse necessário realizar a classificação de um novo conjunto de dados de teste, por exemplo. Neste caso, optou-se por k = 10 novamente, devido à grande flutuação existente no teste para se descobrir o k ótimo, por conta da aleatoriadade do algoritmo que gera os conjuntos de dados de treinamento e testes. Portanto, optou-se por utilizar o valor que é consenso na literatura. Foram criadas 10 bases de dados de treinamento e 10 bases de teste, e em cada conjunto de dados de treinamento e teste, o erro foi calculado. Ao final, a média destes erros foi obtida.

A média de erro alcançada pelo modelo utilizando a técnica de Validação Cruzada foi de 0.06%, ainda sim um valor irrisório. Porém, foi possível observar que houve erros em certas iterações referentes à classe 5, o que mostra que o fato do modelo ter acertado todas as instâncias neste caso não significa que a generalização tenha sido perfeita. Este comportamento fica mais aparente ao se verificar a curva ROC gerada para este modelo.

Juntamente com a Validação Cruzada, foi utilizada a curva ROC para visualização da performance do modelo. Neste caso, a curva ROC é especialmente importante por conta do chamado *trade-off* de informação. Como é possível visualizar em uma matriz de confusão ou em uma curva ROC, à medida em que a TFP aumenta, a TVP diminui. Em casos do mundo real, como este, este *trade-off* deve ser tratado com cuidado. Existem casos em que vale a pena se trocar precisão por uma queda nos Falsos Positivos, e vice-versa. No caso do PNCRC, é extremamente grave uma amostra contaminada, ou seja, cuja substância testada ultrapassou o LMR estabelecido, ser classificada como sadia, portanto, deve-se evitar a todo custo um modelo que possa realizar este tipo de erro, mesmo que por esta certeza, tenha-se que pagar com menor precisão. Conforme dito por Brink e Richards (2014) diz, *não existe almoço grátis*.

Então, a curva ROC do modelo foi gerada, por meio da técnica de *um-contra-todos*. A curva ROC é uma técnica cuja aplicação é restrita a apenas em classificadores binários, já que são utilizadas as taxas de positivos e negativos das classes para a geração da curva. Porém, por meio desta técnica é possível adaptar uma curva ROC para classificadores multi-classe.

A técnica de *um-contra-todos* consiste em usar uma das n classes do modelo como a classe positiva e todas as outras como negativas, e desenhar uma curva de acordo. Ou seja, ao final, serão n curvas em um gráfico, cada uma contendo uma das n classes como a classe positiva e todas as outras como a classe negativa. Em essência, lembra um pouco a técnica de Validação Cruzada.

Portanto, a curva ROC gerada para este modelo contém 30 pontos no total: 3 curvas geradas por 10 pontos cada, calculadas de acordo com a matriz de confusão gerada por cada uma das k iterações da Validação Cruzada.

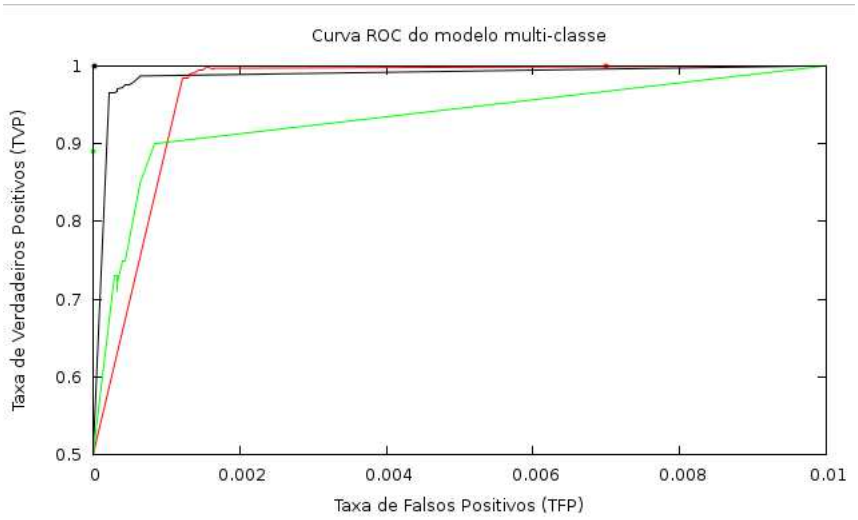

Figura 23: Curva ROC do modelo multi-classe

Na Fig. (23), tem-se a curva ROC do modelo. Como é possível observar, a curva ficou muito próxima do classificador ideal (um classificador com 100% de Verdadeiros Positivos e sem Falsos Positivos), de tal modo que chega a ser necessário ampliar a figura para identificar os pontos referentes ao modelo. A curva de cor preta são referentes à curva da classe 5, a cor vermelha é referente à curva da classe 6 e a cor verde é referente à curva da classe 7, e os pontos são referentes ao modelo de AM gerado, sendo que cada cor é referente a sua curva (por exemplo, o ponto verde é a posição do modelo gerado em relação à curva da classe 7, e assim em diante). Devido à natureza do gráfico, os pontos ficaram muito próximos do limiar do gráfico, dificultando sua visualização.

A Fig. (24) apresenta a curva ROC ampliada no eixo X variando de [0:0.002] e no eixo Y de [0.5:1]. O ponto referente à classe 6 não se encontra neste gráfico pois a mesma está fora deste intervalo. É possível notar que a curva referente à classe 7 apresentou os

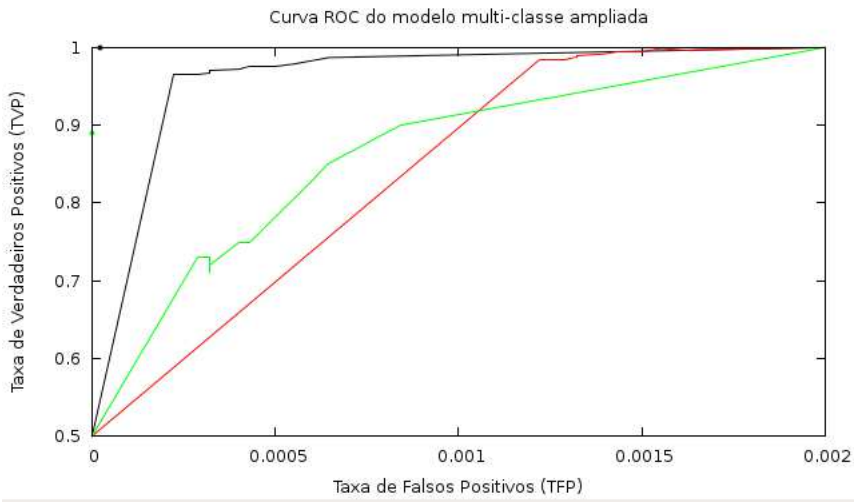

Figura 24: Curva ROC do modelo multi-classe ampliada

piores resultados em relação ao eixo Y, ou seja, é a classe que possui os menores valores de Verdadeiros Positivos, o que era esperado, já que é a classe mais propensa a erros.

Portanto, estes são os resultados obtidos no trabalho ao se analisar estas duas bases de dados tão distintas. Os resultados apresentados neste capítulo são discutidos mais à frente, além de propostas para trabalhos futuros.

7 Considerações Finais

7.1 Primeira Etapa

Esta primeira etapa do projeto proposto serviu para que o autor deste trabalho obtivesse um referencial teórico de qualidade, para que o projeto possa ser implementado na segunda etapa, no TCC2. Este trabalho teve início em março de 2013, com a definição do tema e início do estudo do referencial teórico necessário para a compreensão do tema em geral, além de entender o funcionamento dos algoritmos de Aprendizado de Máquina a fim de se escolher qual seria a melhor abordagem para a implementação do sistema de Mineração de Dados.

Ou seja, durante o TCC1, a curva de aprendizado necessária para o planejamento e implementação do projeto foi percorrida, para que na segunda etapa consista na implementação do projeto em si e posterior avaliação dos resultados obtidos. Espera-se também que o trabalho desenvolvido neste TCC tenha utilidade para aplicações do mundo real, principalmente em projetos que envolvam Mineração de Dados. Atualmente, existe um projeto dentro do curso que pode vir a se beneficiar deste trabalho: o Projeto SALTAR. Este projeto se propõe a realizar pesquisa e implementação de detecção de anomalias em redes de comunicação de dados e está incluído dentro do plano de ação do Centro de Defesa Cibernética do Exército (CDCiber), possuindo uma área específica para Mineração de Dados.

7.1.1 Cronograma

A tabela 9 consiste no cronograma executado na segunda etapa da proposta, além de englobar o que já foi feito nesta primeira etapa. As principais atividades consistem em:

1. Escolha da base de dados de treinamento/testes.
2. Atividades de pré-processamento de dados.
3. Treinamento do modelo de AM gerado
4. Validação do modelo
5. Engenharia de Características
6. Análise dos resultados
7. Escrita do TCC2.

	Julho	Agosto	Setembro	Outubro	Novembro
A	*	*			*
B		*	*		*
C			*		*
D			*		*
E			*	*	*
F				*	*
G		*	*	*	*

Tabela 9: Cronograma do projeto

7.2 Segunda Etapa

A segunda etapa deste trabalho, realizada entre Julho e Dezembro de 2014, teve como objetivo avaliar os resultados obtidos pelo algoritmo de Árvores de Decisão para Mineração de Dados. Conforme dito nos tópicos anteriores, a implementação do algoritmo C4.5 foi o YaDT. Apesar de seus resultados serem superiores à outras implementações existentes, sentiu-se a necessidade de, primeiramente, validar o algoritmo antes de sua aplicação em um problema do mundo real.

Portanto, este é o principal motivo pelo qual estas duas bases de dados foram trabalhadas: a base de dados do Titanic, por ser amplamente conhecida e estudada por diversos especialistas na área de Mineração de Dados, mostrou-se uma excelente oportunidade de se verificar a eficácia do algoritmo, utilizando os diversos resultados que existem mundo afora sobre esta base, sendo que os resultados obtidos podem ser validados utilizando a própria história do navio. Após validado o algoritmo, o mesmo foi utilizado em um problema do mundo real, na base de dados do PNCRC/Bovinos.

7.2.1 Resultados - Titanic

De acordo com os resultados obtidos, é razoável afirmar que o YaDT produz um modelo de AM consistente, já que os resultados obtidos se mantiveram em um patamar aceitável. A acurácia do modelo foi calculada entre cerca de 79% a 81%, o que o classifica entre os 300 primeiros da competição da Kaggle (2012), onde a base de dados foi obtida.

Na construção da árvore, foi possível verificar a obtenção de certos padrões que corroboram a história do Titanic. Por exemplo, a característica mais preponderante na árvore foi a característica sex, seguida da característica pclass. De acordo com a Titanica (2013), cerca de 81% da quantidade total de passageiros homens presentes no navio perderam a vida, sendo que esta porcentagem cai 28% no caso das mulheres. Ou seja, a generalização causada pela poda da árvore em relação à instância ter morrido caso a mesma seja do sexo masculino se mantém na taxa de erro esperada pelo modelo, já que neste caso, o modelo iŕa classificar 19% das instâncias do sexo masculino de forma errada.

Em relação à classe dos passageiros, cerca de 97% das mulheres que viajavam na primeira classe e cerca de 90% das que viajavam na segunda classe foram salvas: ou seja, novamente uma generalização corroborada pelos dados dos sobreviventes do acidente. Em relação à terceira classe, apenas 46% das mulheres sobreviveram, o que explica o porquê a árvore continuou a crescer neste ponto, para que a generalização do modelo não fosse prejudicada.

Também foi possível verificar que o YaDT permite vários ajustes na construção da árvore, desde o valor de confidência utilizado na poda pessimista realizado pelo algoritmo C4.5 até o número mínimo de casos para a criação de um nó-folha. Isto permite um ajuste mais fino na construção de um modelo para um determinado caso, buscando os valores ótimos para estas variáveis, assim como foi realizado neste caso.

Concluindo, acredita-se que tanto o algoritmo quanto o modelo obtiveram resultados bons, o que justifica o seu uso no segundo caso que será apresentado à frente, ou seja, em uma aplicação do mundo real.

7.2.2 Resultados - PNCRC/Bovinos

Ao se verificar os resultados obtidos com a base de dados do Titanic, esperava-se bons resultados em relação à obtenção de padrões ocultos na base de dados fornecida pelo MAPA, já que o algoritmo obteve alguns padrões interessantes relacionados à história do navio.

Porém, não foi o que foi constatado durante a geração do modelo. A árvore gerada pelo algoritmo, apesar de possuir uma alta acurácia, não é de muita serventia na verificação de certos padrões, sejam eles temporais, regionais, ou de acordo com a substância testada. Ao se criar uma árvore com apenas duas características, sendo que uma delas (RESULTADO), é basicamente um indicador prévio da classe, acabou-se tendo um classificador confiável, mas que não atende ao propósito principal do trabalho.

Vale ressaltar que este fato não é responsablidade do algoritmo em si: pelo contrário, o YaDT gerou uma árvore próxima da perfeição levando-se em consideração o que se esperava dele, um classificador eficiente e que gere uma árvore com boa generalização. Acredita-se que os resultados ineficientes tenham sido causados pelo próprio modelo da base em si: com pouquíssimas instâncias que realmente interessavam (instâncias com classes 5 representavam apenas cerca de 2% da base, e instâncias com classe 7, apenas são apenas cerca de 0.2%), o algoritmo não foi capaz de verificar padrões dentro de tão poucos dados a serem minerados. Ou seja, a base de dados não possui representatividade suficiente para a procura dos padrões desejados pelo MAPA.

7.3 Conclusões

Concluindo, acredita-se que o trabalho alcançou o seu objetivo.

A primeira parte do trabalho correu como esperada, a base de dados do Titanic apresentou bons resultados e se mostrou uma excelente oportunidade de se demonstrar a eficácia do YaDT. Seus padrões são corroborados pela própria história que cerca estes dados, e sua acurácia se mostrou na média entre diversas implementações utilizadas na competição da Kaggle. Já a modelagem relativa ao novo problema (PNCRC/Bovinos), demonstrou-se que a base de dados não possui amostras suficientemente distribuídas para que o modelo adotado pudesse identificar padrões de comportamento que poderiam ajudar o MAPA a prevenir as detecções de substâncias, ou seja, o aparecimento de classes 5 e 7. Notoriamente, para que se obtenha um modelo satisfatório para este caso, sugere-se realizar uma amostragem mais equiprobabiliśtica rem relação às classes de interesse.

Para este trabalho, talvez a escolha da base não tenha sido a ideal para apresentar o poder do algoritmo de Árvores de Decisão para a tarefa proposta, Mineração de Dados, por não apresentar resultados satisfatórios nesta aplicação do mundo real. Porém, acredita-se que, levando em consideração todo o trabalho, é possível observar que o algoritmo funciona visivelmente bem, sendo um classificador poderoso e apresentando os resultados de forma simples e clara, diferentemente de outros algoritmos conhecidos de AM.

Para trabalhos futuros, sugere-se um estudo mais profundo dos dados fornecidos pelo MAPA, a fim de encontrar as características necessárias para uma Mineração de Dados mais adequada do que a realizada na base deste trabalho.

Referências

AGRICULTURA, P. e. A. Ministério da. *PNCRC - Plano Nacional de Controle de Resíduos e Contaminantes*. 2014. <http://www.agricultura.gov.br/portal/page/portal/Internet-MAPA/pagina-inicial/pncrc>. [Online; acessado pela última vez em 11 de novembro de 2014]. Citado na página 80.

AHMAD MARZUKI KHALID, R. Y. A. R. Machine learning using support vector machines. MSTC 2002, Johor Bahru, 2002. Citado na página 33.

BRINK, H.; RICHARDS, J. *Real World Machine Learning*. [S.l.]: Manning Publications C.O, 2014. Citado 11 vezes nas páginas 9, 19, 26, 33, 34, 38, 39, 44, 61, 62 e 87.

CHAUDHURI, S.; DAYAL, U. An overview of data warehousing and olap technology. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 26, n. 1, p. 65–74, mar. 1997. ISSN 0163-5808. Citado na página 24.

CRAVEN, M. W.; SHAVLIK, J. W. Using neural networks for data mining. *Future Gener. Comput. Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 13, n. 2-3, p. 211–229, nov. 1997. ISSN 0167-739X. Citado 3 vezes nas páginas 40, 41 e 58.

DOMINGOS, P. A few useful things to know about machine learning. *Commun. ACM*, ACM, New York, NY, USA, v. 55, n. 10, p. 78–87, out. 2012. ISSN 0001-0782. Citado 4 vezes nas páginas 34, 35, 36 e 43.

FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P. Advances in knowledge discovery and data mining. In: FAYYAD, U. M. et al. (Ed.). Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996. cap. From Data Mining to Knowledge Discovery: An Overview, p. 1–34. ISBN 0-262-56097-6. Citado 2 vezes nas páginas 19 e 25.

FURNKRANZ, J.; GAMBERGER, D.; LAVRAC, N. *Foundations of Rule Learning*. [S.l.]: Springer-Verlag Berlin, 2012. Citado 4 vezes nas páginas 28, 29, 35 e 45.

HAN, J.; KAMBER, M. *Data Mining: Concepts and Techniques*. 3rd. ed. San Francisco, CA: Morgan Kaufmann, 2001. Citado 12 vezes nas páginas 9, 23, 24, 25, 27, 29, 30, 31, 36, 47, 48 e 50.

HAND, D. J.; SMYTH, P.; MANNILA, H. *Principles of Data Mining*. Cambridge, MA, USA: MIT Press, 2001. ISBN 0-262-08290-X, 9780262082907. Citado na página 24.

KAGGLE. *Titanic: Machine Learning from Disaster*. 2012. <https://www.kaggle.com/c/titanic-gettingStarted>. [Online; acessado pela última vez em 11 de novembro de 2014]. Citado 2 vezes nas páginas 63 e 92.

MITCHELL, T. *Machine Learning*. New York, NY: McGraw-Hill, 1997. ISBN 0-07-042807-7. Citado 8 vezes nas páginas 33, 40, 41, 42, 48, 50, 73 e 77.

OBSERVATORY, B. I. Big data: Artificial intelligence. Netherlands, p. 1 – 15, 2013. Citado 3 vezes nas páginas 19, 20 e 33.

REZENDE, S. et al. Sistemas inteligentes - fundamentos e aplicações. In: . 1st. ed. Barueri, SP, Brazil: Editora Manole Ltda, 2003. cap. Mineração de Dados, p. 307–336. Citado na página 24.

RUGGIERI, S. Yadt: Yet another decision tree builder. In: *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*. Washington, DC, USA: IEEE Computer Society, 2004. (ICTAI '04), p. 260–265. ISBN 0-7695-2236-X. Citado 2 vezes nas páginas 47 e 58.

SILVA, R. M.; YAMAKAMI, A.; ALMEIDA, T. A. An analysis of machine learning methods for spam host detection. In: *ICMLA (2)*. [S.l.]: IEEE, 2012. p. 227–232. ISBN 978-1-4673-4651-1. Citado na página 33.

TITANICA, E. *Titanic Passenger List : Comprehensive Titanic Passenger List and Biographies*. 2013. <http://www.encyclopedia-titanica.org/titanic-passenger-list/>. [Online; acessado pela última vez em 3 de novembro de 2014]. Citado 3 vezes nas páginas 67, 69 e 92.

WEISS, S. M.; INDURKHYA, N. Rule-based machine learning methods for functional prediction. *J. Artif. Int. Res.*, AI Access Foundation, USA, v. 3, n. 1, p. 383–403, dez. 1995. ISSN 1076-9757. Citado 2 vezes nas páginas 27 e 30.

WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. 2nd. ed. San Francisco, CA: Morgan Kaufmann, 2000. ISBN 0-12-088407-0. Citado 13 vezes nas páginas 20, 23, 36, 39, 40, 41, 44, 45, 50, 53, 58, 73 e 77.

ZHOU, Z. Three perspectives of data mining. *Artif. Intell.*, Elsevier Science Publishers Ltd., Essex, UK, v. 143, n. 1, p. 139–146, jan. 2003. ISSN 0004-3702. Citado 2 vezes nas páginas 20 e 27.