UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CLASSIFICAÇÃO DE *DATA STREAMS* UTILIZANDO ÁRVORE DE DECISÃO ESTATÍSTICA E A TEORIA DOS FRACTAIS NA ANÁLISE EVOLUTIVA DOS DADOS

MIRELA TEIXEIRA CAZZOLATO

ORIENTADORA: PROFA. DRA. MARCELA XAVIER RIBEIRO

São Carlos – SP

Julho/2014

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CLASSIFICAÇÃO DE *DATA STREAMS* UTILIZANDO ÁRVORE DE DECISÃO ESTATÍSTICA E A TEORIA DOS FRACTAIS NA ANÁLISE EVOLUTIVA DOS DADOS

MIRELA TEIXEIRA CAZZOLATO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software / Banco de Dados
Orientadora: Profa. Dra. Marcela Xavier Ribeiro

São Carlos – SP

Julho/2014

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

C386cd

Cazzolato, Mirela Teixeira.
Classificação de *data streams* utilizando árvore de decisão estatística e a teoria dos fractais na análise evolutiva dos dados / Mirela Teixeira Cazzolato. -- São Carlos : UFSCar, 2014.
71 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2014.

1. Ciência da computação. 2. Banco de dados. 3. Fluxo de dados. 4. Classificação. 5. *Data mining* (Mineração de dados). 6. Fractais. I. Título.

CDD: 004 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

"Classificação de Streams utilizando Árvore de Decisão Estatística e a Teoria dos Fractais na Análise Evolutiva dos Dados"

Mirela Teixeira Cazzolato

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Marcela Xavier Ribeiro

Profa. Dra. Marcela Xavier Ribeiro
(Orientadora - DC/UFSCar)

Eduardo A. do Valle Jr.

Prof. Dr. Eduardo Alves do Valle Junior
(FEE/UNICAMP)

Agma Traina

Profa. Dra. Agma Juci Machado Traina
(ICMC/USP)

São Carlos
Março/2014

A meus pais e meu irmão.

AGRADECIMENTOS

Primeiramente a Deus.

À minha orientadora, Prof[a] Dr[a] Marcela X. Ribeiro, pela confiança ao me orientar, pelas constantes lições, incentivando sempre a buscar algo a mais, pela paciência nas horas de discussão, pela excelente orientação e pelo exemplo de pessoa.

Aos meus pais, pelo exemplo de luta, persistência, coragem, competência e alegria. Ao meu irmão, exemplo de dignidade, honestidade e batalha. Aos meus avós e tios. E toda minha família, pelo apoio incondicional, tanto financeiro quanto emocional.

Aos amigos Amandia, André, Augusto Cesar, Bruno, Claudinei, Diego, Felipe, Isis, Juliana, Matheus, Odair, Pâmela, Rafael Tomé, Tatiana, Thiago Lacerda, Thiago Lima. Aos colegas de LaBDES e do café, Victor, Rafael, José Rafael, Marcos, Carlão. Aos colegas da turma de 2012 do PPGCC/DC.

Ao Grupo de Banco de Dados (GBD) pelas críticas, sugestões e discussões. Em especial, aos professores Renato Bueno, Marilde T. P. Santos e Ricardo Ciferri. Agradeço também ao Grupo de Banco de Dados e Imagens do ICMC-USP pelas valiosas sugestões, em especial aos professores Caetano Traina-Jr e Agma J. M. Traina.

Aos professores que me encorajaram, desde a graduação, a seguir meus estudos na pesquisa. Em especial aos professores Marina T. P. Vieira, Plínio Vilela, Anderson Belgamo, Ana Estela A. da Silva e Cecília S. A. Peixoto.

Aos funcionários do Departamento de Computação da UFSCar, por proporcionar serviços e infraestrutura adequada, de modo a possibilitar o desenvolvimento de nossos trabalhos.

À CAPES, pelo apoio financeiro.

A todos que, de alguma forma, contribuíram com a realização desse trabalho.

Obrigada!

A mente que se abre a uma nova ideia, jamais voltará ao seu tamanho original.

Albert Einstein

RESUMO

Um *data stream* é gerado de forma rápida, contínua, ordenada e em grande quantidade. Para o processamento de *data streams* deve-se considerar, dentre outros fatores, o uso limitado de memória, a necessidade de processamento em tempo real, a precisão dos resultados e o *concept drift* (que ocorre quando há uma mudança no conceito dos dados que estão sendo analisados). A árvore de decisão é uma popular forma de representação do modelo classificador, intuitiva, e rápida de construir, geralmente possuindo alta acurácia. As técnicas de árvores de decisão incrementais presentes na literatura geralmente apresentam um alto custo computacional para a construção e atualização do modelo, principalmente no que se refere ao cálculo para a decisão de divisão dos nós. Os métodos existentes possuem uma característica conservadora para lidar com quantidades de dados limitadas, tendendo a melhorar seus resultados conforme o número de exemplos aumenta. Outro problema é a geração dos dados com ruídos por muitas aplicações reais, pois as técnicas existentes possuem baixa tolerância a essas ocorrências. Este trabalho tem como objetivo o desenvolvimento de métodos de árvores de decisão para *data streams*, que suprem as deficiências do atual estado da arte. Além disso, outro objetivo deste projeto é o desenvolvimento de uma funcionalidade para detecção de *concept drift* utilizando a teoria dos fractais, corrigindo o modelo sempre que necessário, possibilitando a descrição correta dos acontecimentos mais recentes dos dados. Para atingir os objetivos foram desenvolvidos três algoritmos de árvore de decisão: o StARMiner Tree, o Automatic StARMiner Tree, e o Information Gain StARMiner Tree. Esses algoritmos utilizam um método estatístico como heurística de divisão de nós, que não é dependente do número de exemplos lidos e que é rápida. Os algoritmos obtiveram alta acurácia nos experimentos realizados, mostrando também um comportamento tolerante na classificação de dados ruidosos. Finalmente, foi proposto um método para a detecção de mudanças no comportamento dos dados baseado na teoria dos fractais, o Fractal Drift Detection Method. Ele detecta mudanças significativas na distribuição dos dados, fazendo com que o modelo seja atualizado sempre que o mesmo não descrever os dados atuais (se tornar obsoleto). O método obteve bons resultados na classificação de dados contendo *concept drift*, mostrando ser adequado para a análise evolutiva dos dados.

Palavras-chave: *Data Streams*, Classificação, Mineração de Dados, Árvore de Decisão, Algoritmo Incremental, *StARMiner Tree*, FDDM, Teoria dos Fractais

ABSTRACT

A data stream is generated in a fast way, continuously, ordered, and in large quantities. To process data streams there must be considered, among others factors, the limited use of memory, the need of real-time processing, the accuracy of the results and the concept drift (which occurs when there is a change in the concept of the data being analyzed). Decision tree is a popular form of representation of the classifier, that is intuitive and fast to build, generally obtaining high accuracy. The techniques of incremental decision trees present in the literature generally have high computational costs to construct and update the model, especially regarding the calculation to split the decision nodes. The existent methods have a conservative characteristic to deal with limited amounts of data, tending to improve their results as the number of examples increases. Another problem is that many real-world applications generate data with noise, and the existing techniques have a low tolerance to these events. This work aims to develop decision tree methods for data streams, that supply the deficiencies of the current state of the art. In addition, another objective is to develop a technique to detect concept drift using the fractal theory. This functionality should indicate when there is a need to correct the model, allowing the adequate description of most recent events. To achieve the objectives, three decision tree algorithms were developed: StARMiner Tree, Automatic StARMiner Tree, and Information Gain StARMiner Tree. These algorithms use a statistical method as heuristic to split the nodes, which is not dependent on the number of examples and is fast. In the experiments the algorithms achieved high accuracy, also showing a tolerant behavior in the classification of noisy data. Finally, a drift detection method was proposed to detect changes in the data distribution, based on the fractal theory. The method, called Fractal Detection Method, detects significant changes on the data distribution, causing the model to be updated when it does not describe the data (becoming obsolete). The method achieved good results in the classification of data containing concept drift, proving to be suitable for evolutionary analysis of data.

Keywords: Data Streams, Classification, Data Mining, Decision Tree, Incremental Algorithm, StARMiner Tree, FDDM, Fractal Theory

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Neste capítulo são apresentados a introdução, a motivação e o objetivo da realização deste trabalho.

1.1 Introdução

O Processo de KDD é definido por Fayyad, Piatetsky-Shapiro e Smyth (1996) como um processo não trivial de identificação de padrões contidos nos dados que sejam válidos, novos, potencialmente úteis e compreensíveis. A mineração de dados é considerada a principal etapa desse processo, nela são utilizados algoritmos computacionais para processar grandes volumes de dados em busca de padrões e conhecimento novos. Tradicionalmente, as tarefas de mineração têm sido desenvolvidas principalmente para a análise de dados históricos, armazenados em bases de dados. Atualmente a produção contínua de dados a partir de dispositivos como sensores, satélites ou radares, gera o desafio computacional de processar esses dados de forma incremental, produzindo conhecimento útil.

Diferentemente dos dados tradicionais, *data streams* (fluxos de dados) muitas vezes não podem ser analisadas somente com base em dados históricos, sua análise deve ser rápida e sensível a mudanças de tendência em tempo real. O comportamento de um *data stream* tende a sofrer alterações significativas ao longo do tempo, e o algoritmo deve ser capaz de identificar essas mudanças e alterar os resultados da mineração.

A classificação é uma tarefa de mineração de dados de caráter descritivo, e seu objetivo é produzir um modelo computacional para descrever o comportamento dos dados. O modelo construído pode ser utilizado, posteriormente, para a classificação de dados não rotulados. No contexto de *data streams* deve-se considerar o *concept drift*, que ocorre quando há a mudança

na distribuição dos dados que estão sendo gerados e processados, indicando que o modelo deve ser adaptado às novas tendências ao longo do tempo. Nesse sentido, alguns dos desafios relacionados ao desenvolvimento de métodos de classificação de *data streams* são:

- O modelo de aprendizado não pode ser estático e sim evolutivo, considerando as novas tendências dos dados;
- O classificador deve ser rápido para a tomada de decisão online;
- O modelo deve ser construído considerando memória limitada, processamento em tempo real e alta acurácia.

1.2 Motivação

A árvore de decisão é uma popular forma de representação do modelo classificador, frequentemente utilizada em áreas como marketing, finanças, e aplicações que são suporte à tomada de decisão. Trata-se de uma forma de representação de simples entendimento (autoexplicativa), rápida de construir, e que geralmente possui boa acurácia. As técnicas incrementais de árvores de decisão, em geral, apresentam um custo computacional alto para a construção e atualização do modelo, principalmente no que se refere ao cálculo efetuado para a decisão de divisão dos nós. Os métodos possuem uma característica conservadora a quantidades de dados limitadas (tendendo a melhorar conforme o número de exemplos aumenta). Além disso, em muitas aplicações reais os dados são gerados com ruídos, e as técnicas existentes possuem baixa tolerância a essas ocorrências, não se comportando bem na descrição dos dados.

1.3 Objetivo

Este projeto de pesquisa tem como objetivo o desenvolvimento de métodos de classificação utilizando árvores de decisão para *data streams* que suprem as deficiências do atual estado da arte, apresentando uma maior tolerância à presença de ruídos, com um heurística de divisão dos nós da árvore mais rápida e que não dependa do número de exemplos disponíveis. Além disso, outro objetivo deste projeto é o desenvolvimento de uma funcionalidade para detecção de *concept drift* utilizando a teoria dos fractais, corrigindo o modelo sempre que necessário, de forma a possibilitar a descrição correta dos acontecimentos mais recentes dos dados. As técnicas desenvolvidas dão suporte às principais necessidades existentes do contexto de *data streams*, como o processamento em tempo real e a apresentação de boa acurácia do modelo.

Outro objetivo deste projeto foi a validação das técnicas propostas por meio da realização de uma série de experimentos, considerando dados de diferentes contextos. Foram utilizadas bases reais e sintéticas, com variados tamanhos e características, possibilitando uma melhor análise do funcionamento dos algoritmos propostos, suas vantagens e limitações.

1.4 Organização do Trabalho

Este trabalho está organizado da seguinte maneira: No Capítulo 2 são apresentados a revisão bibliográfica do projeto e os trabalhos relacionados, descrevendo os principais conceitos relacionados à mineração de dados, classificação de *data streams* e ao problema do *concept drift*. Além disso, é dado um destaque maior à utilização de árvores de decisão, abordando com mais detalhes as técnicas incrementais de construção do modelo. Também são apresentados os trabalhos utilizados neste projeto, o algoritmo StARMiner, a teoria dos fractais, o algoritmo SID-Meter e a forma de comparação de métodos de classificação de *data streams*. No Capítulo 3 são apresentados os resultados obtidos neste projeto, com os algoritmos propostos ST, AST, IST, e FDDM. São apresentados também os resultados dos experimentos realizados, utilizando bases de dados de diferentes contextos e comparando os métodos propostos com algoritmos presentes na literatura. Finalmente, no Capítulo 4 são apresentadas as conclusões deste trabalho, com uma sumarização dos métodos propostos, suas características, pontos positivos e limitações. Também são listados os trabalhos futuros e as publicações realizadas durante o desenvolvimento do projeto.

Capítulo 2

CONCEITOS FUNDAMENTAIS E TRABALHOS RELACIONADOS

Neste capítulo são apresentados os principais conceitos relacionados ao projeto. São descritos o Processo de KDD, a mineração de dados e a tarefa de classificação, dando uma atenção maior para o uso de árvores de decisão, por ser o método utilizado neste trabalho de mestrado. Também são apresentados os principais conceitos e trabalhos relacionados ao tema deste projeto, descrevendo a construção de árvores de decisão incrementais e o problema do concept drift, o algoritmo StARMiner, a teoria dos fractais, o algoritmo SID-Meter e, finalmente, as abordagens de comparação dos métodos de classificação.

2.1 Considerações Iniciais

O Processo de Descoberta de Conhecimento em Bancos de Dados (KDD - *Knowledge Discovery in Databases*) é formalmente definido por Fayyad, Piatetsky-Shapiro e Smyth (1996) como um processo não trivial de identificação de padrões contidos nos dados que sejam válidos, novos, potencialmente úteis e compreensíveis. Conforme Han, Kamber e Pei (2011), trata-se de um processo que pode ser dividido em sete etapas, como mostrado na Figura 2.1: limpeza dos dados e integração (realizadas no pré-processamento), seleção, transformação, mineração, avaliação dos padrões e apresentação do conhecimento.

Na etapa de limpeza e integração são retirados possíveis ruídos e inconsistências que podem interferir no resultado final. Na seleção e transformação são escolhidos os dados considerados relevantes para análise e colocados em um formato apropriado para a mineração de dados. Após minerar os dados, os padrões encontrados são avaliados e apresentados de forma intuitiva para o entendimento do usuário. Como é possível observar na Figura 2.1, trata-se de um processo

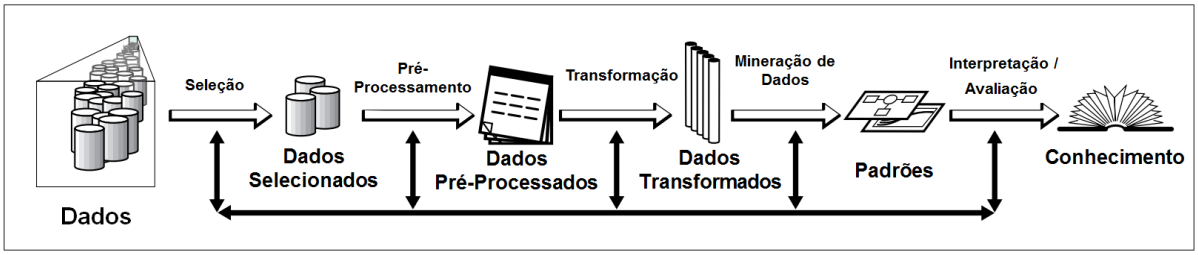


Figura 2.1: Processo de KDD (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996)

iterativo, ou seja, é possível avançar ou retornar nas etapas durante a execução do processo.

Neste trabalho será utilizado o termo "exemplo"como sinônimo de tupla ou registro, por se tratar de uma nomenclatura muito utilizada no contexto de *data streams*.

Utilizando árvores de decisão tradicionais, os dados analisados são relativamente estáticos, uma vez que é possível a leitura dos mesmos por diversas vezes. No que se refere a *data streams*, os dados são gerados em grande quantidade, a todo momento, e não é viável reconstruir o modelo para cada novo dado. Para lidar com *data streams* é utilizada uma técnica incremental, que não necessita reconstruir o modelo sempre que um novo exemplo é lido, e que constrói uma árvore de decisão com base em estatísticas suficientes extraídas dos dados lidos.

A seguir são apresentados os principais conceitos relacionados a esse projeto de mestrado, bem como os trabalhos relacionados. O capítulo está organizado da seguinte forma: na Seção 2.2 são descritas a etapa de mineração de dados e a tarefa de classificação; na Seção 2.3 são apresentados os conceitos relacionados a árvores de decisão; na Seção 2.4 são detalhadas as principais características da classificação incremental, na Seção 2.5 é descrito o problema do *concept drift* e na Seção 2.6 é descrita a utilização de árvores de decisão incrementais, bem como os principais algoritmos presentes na literatura; o algoritmo StARMiner é mostrado na Seção 2.7, a teoria dos fractais na Seção 2.8 e o algoritmo SID-Meter na Seção 2.9; na Seção 2.10 são apresentados métodos de comparação dos resultados da classificação e, finalmente, na Seção 2.11 são descritas as considerações finais do capítulo.

2.2 Mineração de Dados e Classificação

A mineração de dados, considerada a principal etapa do processo de KDD, é definida por Han, Kamber e Pei (2011) como o processo de descoberta de padrões interessantes e conhecimento a partir de grandes conjuntos de dados. Técnicas de mineração de dados podem ser utilizadas para extrair padrões que venham a ajudar os analistas na avaliação e otimização de processos de produção, negócios, prever o futuro comportamento dos dados (tendências), auxiliar na decisão

estratégica, dentre outros.

As tarefas de mineração diferem-se entre si devido à forma de lidar com os dados, buscando reconhecer possíveis padrões existentes. Dentre as principais tarefas existentes, pode-se citar as regras de associação, a classificação e o agrupamento. Neste trabalho a classificação é descrita com um destaque maior por ser a tarefa utilizada no desenvolvimento do projeto.

De acordo com Han, Kamber e Pei (2011), classificação é o processo de descoberta de um modelo (ou função) que descreve e distingue classes de dados ou conceitos. O resultado obtido tem várias aplicações, incluindo detecção de fraudes, produção e diagnóstico médico. No contexto da classificação, é utilizado o atributo classe (ou atributo alvo), que indica, para cada instância, sua classe correspondente. A classificação consiste em um processo de duas etapas: a de treinamento e a de teste.

Na fase de treinamento é construído um modelo que descreve um determinado conjunto de dados ou conceitos; isso é feito por meio da análise de um conjunto de treino, composto por tuplas do conjunto de dados e seus respectivos rótulos de classe. Como o rótulo da classe de cada tupla de treinamento é previamente conhecido, essa etapa também é conhecida como aprendizado supervisionado.

Na etapa de teste o modelo criado é usado para a classificação. Conforme Han, Kamber e Pei (2011) primeiramente é estimada a acurácia do classificador, utilizando um conjunto de dados de teste, que é independente dos dados de treinamento. Caso a precisão do classificador for considerada aceitável, o modelo pode ser usado para classificar futuras tuplas cujo rótulo de classe é desconhecido.

Como mencionado anteriormente, após a etapa de mineração ocorre a apresentação e avaliação dos resultados obtidos. Um modelo de classificação pode ser representado de diversas formas, como por meio de regras de classificação SE-ENTÃO e árvores de decisão.

Na Seção 2.3 é apresentada a classificação de dados utilizando árvores de decisão, que é a técnica utilizada neste projeto de mestrado, por ser uma forma de representação intuitiva, rápida de ser construída e geralmente possuir boa acurácia.

2.3 Árvores de Decisão

A árvore de decisão é uma forma de representação muito popular. Conforme com Rokach e Maimon (2008) na opinião de diversos pesquisadores, árvores de decisão são populares devido a sua simplicidade e transparência. Trata-se de uma representação autoexplicativa, não ha-

vendo a necessidade de ser um especialista em mineração de dados para entendê-la. De acordo com Han, Kamber e Pei (2011) árvores de decisão podem lidar com dados multidimensionais e sua representação é intuitiva e geralmente fácil de assimilar. As etapas de aprendizado e classificação de árvores de decisão são simples e rápidas, e em geral têm boa acurácia, embora seu uso bem sucedido possa depender dos dados disponíveis. A indução de algoritmos de árvore de decisão tem sido usada em diversas áreas de aplicação, como na medicina, manufatura e produção, análise financeira e astronomia. Uma árvore de decisão é composta por:

- **Nós:** cada nó interno representa um teste em um valor de atributo;
- **Ramos:** cada ramo representa os resultados de testes;
- **Folhas:** cada folha representa um resultado (rótulo) de classe.

Como exibido na Figura 2.2 os testes são realizados nos nós internos, utilizando um atributo (a). Cada nó folha representa um resultado de classe do atributo classe em questão (b). Considerando o exemplo exibido na árvore de decisão, o atributo classe indica se uma criança poderá brincar de acordo com o tempo. As possibilidades de rótulo são S (sim), N (não) e T (talvez). Dessa forma, por exemplo, se o tempo estiver chuvoso e houver relâmpagos, a árvore de decisão indica que a criança não poderá brincar.

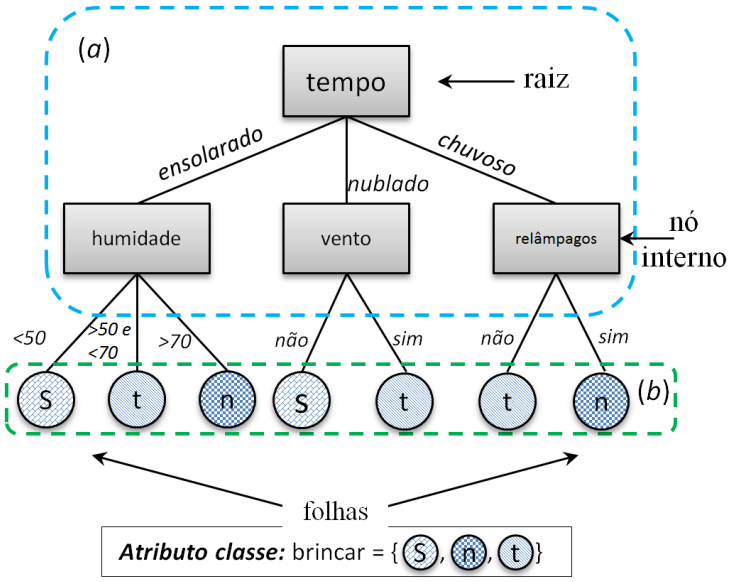


Figura 2.2: Exemplo de uma árvore de decisão e suas partes: (a) nós internos, em que são realizados os testes com base nos valores de um atributo; e (b) nós folha, cada um representando um rótulo (ou seja, um valor do atributo classe)

Na classificação tradicional, os dados são carregados para a memória em sua totalidade e são realizadas leituras e análises deles. Conforme Yang, Fong e Si (2012) esse processo

constrói um modelo de árvore estático de forma gulosa. Quando novos dados são adicionados ao conjunto analisado, todos os dados (os novos e os anteriormente existentes) são recarregados para a atualização do modelo.

O Algoritmo 1 é um algoritmo básico de árvore de decisão, extraído de (HAN; KAMBER; PEI, 2011) e descrito a seguir. Os parâmetros de entrada são:

- A partição de dados de treinamento *D*, que contém tuplas (registros) já rotuladas (ou seja, com classe conhecida). Inicialmente essa partição consiste no conjunto de dados inteiro;
- A lista de atributos dos dados;
- E o método (ou heurística) utilizado para determinar o critério de divisão que melhor particiona as tuplas em classes individuais. O critério de divisão deve retornar o atributo que melhor discrimina os dados de acordo com a classe e o ponto de divisão do nó. O Ganho de Informação e o Gini Index são exemplos de critérios de divisão muito utilizados (detalhado mais adiante).

A árvore inicia com um único nó, *N*, que representa todas as tuplas em *D* (linha 1). Se todas as tuplas em *D* pertencerem à mesma classe, fazer de *N* uma folha rotulada com a classe em questão (linhas 2 e 3). Se a lista de atributos está vazia (linha 4) não existem atributos sobrando para que as tuplas sejam particionadas, então é retornado *N* como nó folha rotulado com a classe majoritária (linha 5). Caso contrário (linha 6) é chamada a função *Metodo_selecao_atributo* para determinar o atributo que melhor particiona D em classes individuais. O nó *N* é então rotulado com o critério de divisão (linha 7), e para cada saída do critério de divisão, um ramo é criado a partir de *N*, particionando *D* em subconjuntos (linha 10 e 11). O algoritmo usa o mesmo processo recursivamente para formar uma árvore de decisão para as tuplas de cada partição resultante de *D*.

Um dos métodos de divisão mais utilizados é o Ganho de Informação. De acordo com Han, Kamber e Pei (2011): supondo que *N* seja um nó que contém tuplas de uma partição *D*; o atributo com o maior ganho de informação é escolhido como o atributo de divisão para o nó *N*. Esse atributo minimiza a informação necessária para classificar as tuplas nas partições resultantes, refletindo a menor aleatoriedade ou "impureza"nessas partições e minimizando o número esperado de testes necessários para classificar uma dada tupla, garantindo que uma árvore simples (mas não necessariamente a mais simples) seja encontrada.

A informação esperada necessária para classificar uma tupla em *D*, ou seja, a entropia de *D*, é calculada de acordo com a equação (2.1), em que p_i é a probabilidade de que uma tupla

Algoritmo 1: Algoritmo básico de classificação.

Entrada: Partição de dados D, que consiste em um conjunto de tuplas de treinamento e os rótulos de classe associados;
lista_atributos, o conjunto de atributos candidatos;
Metodo_selecao_atributo, um procedimento para determinar o critério de divisão que melhor particiona as tuplas de dados em classes individuais. Esse critério consiste em um *atributo_divisão* e, possivelmente, um *ponto_divisão* ou *subconjunto de divisão*.

Saída : Uma árvore de decisão

```
criar um nó N;
se tuplas em D são todas da mesma classe, C, então
    retorne N como um nó folha rotulado com a classe C;
se lista_atributos está vazia então
    retorne N como um nó folha rotulado com a classe majoritária em D
aplicar Metodo_selecao_atributo(D,lista_atributos) para encontrar o "melhor"critério de divisão
rotular o nó N com o critério_divisão
se atributo_divisão é discreto e divisão de múltiplos caminhos é permitida então
    lista_atributos ← lista_atributos – atributo_divisão // remove o atributo da lista
para cada saída j de criterio_divisão faça
    seja D_j o conjunto de tuplas de dados em D que satisfazem a saída j //uma partição
    se D_j está vazia então
        anexar uma folha rotulada com a classe majoritária em D ao nó N
    senão
        anexar o nó retornado por Gerar_árvore_decisão (D_j,lista_atributos) ao nó N;
retorne N
```

arbitrária D pertença à classe C_i, estimada por $|C_{(i,D)}|/|D|$.

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i) \tag{2.1}$$

Quando um nó é dividido utilizando um atributo A, cada ramo resultante da divisão corresponderá a uma partição de D que tem como valor de saída a_i de A. Para calcular a informação necessária (depois de particionar o nó) para atingir uma classificação exata é utilizada a equação (2.2), em que $Info(D_j)$ é a informação esperada necessária para classificar uma tupla j de D baseada no particionamento de A. Quanto menor a informação esperada necessária, maior a pureza das partições.

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j) \tag{2.2}$$

O ganho de informação (2.3) é definido como a diferença entre a informação original necessária (baseada no número de classes) e a nova informação necessária (obtida depois do particionamento de A).

$$Gain(A) = Info(D) - Info_A(D) \tag{2.3}$$

Dois algoritmos de classificação muito conhecidos são o ID3 (QUINLAN, 1986) e seu sucessor, o C4.5 (QUINLAN, 1993), ambos utilizam o ganho de informação como heurística de divisão dos nós.

No contexto de *data streams* são construídas árvores de decisão incrementais, que têm a capacidade de se adaptar conforme a chegada e o processamento de novos dados. As árvores de decisão geradas por meio da abordagem estática são consideradas ideais, pois são construídas utilizando de todo o conjunto de dado disponível. Ao que se refere a árvores incrementais, o modelo construído é estatístico, construído com base nos dados disponíveis no momento.

Na próxima seção são apresentadas as diferenças de processamento que devem ser consideradas na classificação de *data streams*, que é o foco deste trabalho.

2.4 Classificação de *Data Streams*

A mineração de dados é aplicada, tradicionalmente, em conjuntos de dados relativamente estáticos, em que os algoritmos podem ler os dados de entrada diversas vezes. Data streams (fluxos de dados) são gerados por diversas aplicações, como cotações financeiras, medidas de performance no monitoramento de redes e do tráfego, registros de log, dados de sensores, transações de cartão de crédito, dentre outras. Ao contrário dos dados convencionais, *data streams* são obtidos de forma contínua, gerando assim um grande volume de dados. Em muitos casos é inviável armazenar esses dados em sua totalidade para o posterior processamento. O armazenamento, consulta e extração de conhecimento de *data streams* diferem das técnicas utilizadas para dados convencionais.

Dentre os principais problemas enfrentados no processamento de *data streams* estão o uso limitado de memória e a necessidade de processamento em tempo real. Também há a necessidade de atualização do modelo de aprendizado, pois a distribuição dos dados pode mudar com o

tempo, ocorrendo o chamado *concept drift*, descrito com mais detalhes na Seção 2.5. De acordo com Babcock et al. (2002) assim como o processo de classificação tradicional, o processo de classificação de *data streams* é dividido em duas fases, a de construção (ou treinamento) e a de teste do modelo. Diferentemente do processo tradicional, que lida com dados relativamente estáticos, a classificação de *data streams* deve gerar um modelo de aprendizado evolutivo (ou seja, que considere as novas tendências dos dados) e ser rápida, de forma a auxiliar a tomada de decisões em tempo real.

Na Seção 2.5 é descrito o *concept drift*, um dos principais problemas a ser considerado na mineração de *data streams*.

2.5 *Concept Drift*

Para a construção do modelo, diversos algoritmos de classificação consideram que os dados de entrada são gerados por uma distribuição uniforme. No entanto, de acordo com Gama e Gaber (2007) a distribuição geradora dos exemplos de cada *stream*, na maior parte das vezes, muda com o tempo.

O *concept drift* ocorre quando a distribuição dos dados que estão sendo coletados se altera com o passar do tempo. Segundo Gama (2010) *concept* refere-se a variável alvo (atributo classe) que o modelo está tentando prever. Observações antigas, que refletem o comportamento dos dados no passado, se tornam irrelevantes no atual estado dos dados que está sendo descrito, e o modelo de aprendizado deve "esquecer"essa informação. De acordo com Gama, Rocha e Medas (2003), a teoria estatística garante que enquanto uma distribuição é estacionária a taxa erro vai diminuir. Quando a distribuição muda, o erro aumenta.

Existem diversas propostas de métodos para detectar a ocorrência de *concept drift* nos dados. Em geral, de acordo com Gama (2010), as abordagens de detecção de *concept drift* devem ser capazes de diferenciar a mudança da distribuição a possíveis ruídos nos dados. Alguns exemplos desses métodos são o teste CUSUM (*Cumulative Sum*), o DDM (*Drift Detection Method*) e o EDDM (*Early Drift Detection Method*), descritos de forma sucinta a seguir.

O teste CUSUM (*Cumulative Sum*) foi inicialmente proposto em (PAGE, 1954). Trata-se de um algoritmo de detecção de mudança que dispara um alarme quando a média dos dados de entrada é significantemente diferente de zero. De acordo com Bifet (2010), a entrada do algoritmo pode ser o erro de predição de um filtro Kalman, por exemplo). Considerando a entrada de um evento ϵ_t, o limiar h e o peso v, o teste CUSUM consiste em:

$$g_0 = 0 \tag{2.4}$$

$$g_t = max(0, g_{t-1} + \epsilon_t - v) \tag{2.5}$$

$$\text{se } g_t > h \text{ então dispara alarme e } g_t = 0 \tag{2.6}$$

O valor inicial do teste é zero (2.4). Depois da entrada de um evento ϵ_t, o teste assume um valor g_t maior ou igual a zero (2.5). Se g_t exceder o limiar h, uma mudança é detectada (2.6). A acurácia do teste CUSUM depende da escolha dos parâmetros v e h. Conforme Zeileis (2004) o teste CUSUM se preocupa em testar a hipótese $H_0 : g_i = g_0$ para $i = 1, ..., n$ contra a alternativa de que g_i varia com o tempo.

O método de detecção de *concept drift* DDM (*Drift Detection Method*) foi proposto em (GAMA et al., 2004) para controlar a taxa de erro do algoritmo, ao longo do tempo, utilizando duas janelas de tempo. Baseando-se na asserção de que o erro aumenta quando a distribuição muda, o algoritmo monitora o erro com o passar do tempo. Se o erro aumentar, indicando mudança na distribuição, o algoritmo entra no nível de aviso (*warning level*) no exemplo e_w, e logo após no nível indicando *concept drift* (*drift level*) no exemplo e_d. De acordo com os autores, essa é uma indicação de uma mudança na distribuição dos dados. Durante a execução do algoritmo, duas janelas são mantidas pelo método: a primeira contendo as estatísticas dos exemplos lidos até o momento, e a segunda contendo somente as estatísticas dos dados do começo até o número de erros aumentar. Conforme Baena-García et al. (2006) o número de erros em uma amostra de n exemplos é dado por uma distribuição binomial. Para cada ponto i na sequência da amostra, a taxa de erro é a probabilidade de erro de classificação (p_i), com desvio padrão dado por $s_i = \sqrt{p_i(1-p_i)/i}$. Assume-se que a taxa de erro p_i do algoritmo diminui enquanto o número de exemplos aumenta se a distribuição for estacionária. Um aumento significante no erro sugere que a distribuição de classe está mudando, e assim o modelo de decisão atual provavelmente está inadequado. Durante sua execução, o algoritmo armazena os valores de p_i e s_i quando $p_i + s_i$ atinge seu valor mínimo durante o processo (obtendo p_{min} e s_{min}), e quando as seguintes condições são verdadeiras:

- **Condição 1:** $p_i + s_i \geq p_{min} + 2s_{min}$ para o nível de aviso. Depois desse nível, os exemplos são armazenados devido à possibilidade de mudança no contexto;
- **Condição 2:** $p_i + s_i \geq p_{min} + 3s_{min}$ para o nível de *concept drift*. Depois desse nível, supõe-

se que ocorreu o *concept drift*, e então o modelo e os valores p_{min} e s_{min} são reiniciados. Um novo modelo é construído utilizando os exemplos armazenados desde a ativação do nível de aviso.

Considerando uma sequência de exemplos em que o erro do modelo atual aumenta atingindo o nível de aviso ao processar o exemplo e_w (Condição 1). É possível que o nível de aviso seja seguido por uma diminuição da taxa de erro. Segundo Gama et al. (2004) essa situação corresponde a um falso alarme. O método DDM tem um bom comportamento na detecção de mudanças abruptas e graduais (quando a mudança gradual não é muito lenta). No entanto, o DDM apresenta dificuldades quando a mudança é gradual e lenta, pois os exemplos serão armazenados por um longo tempo, o nível de *concept drift* pode levar muito tempo para disparar e o uso de memória pode aumentar muito.

O método EDDM (*Early Drift Detection Method*), apresentado em (BAENA-GARCÍA et al., 2006) foi proposto para melhorar a detecção de mudanças na presença de *concept drift* gradual e manter uma boa performance com *concept drift* abrupto. Ao contrário do DDM, que considera apenas a taxa de erro, o EDDM considera a distância entre dois erros de classificação. Conforme os exemplos são processados e o modelo atualizado, é esperado que a acurácia nas predições do modelo melhore, e que a distância entre dois erros aumente. O método calcula a distância média entre dois erros (p'_i) e seu desvio padrão (s'_i). Quando $p'_i + 2s'_i$ atinge o maior valor, p'_i e s'_i são armazenados como, respectivamente, p'_{max} e s'_{max}. O valor $p'_{max} + 2s'_{max}$ indica o ponto em que a distribuição das distâncias entre os erros é máximo. Esse ponto é atingido quando o modelo melhor se aproxima da distribuição dos dados de entrada. Assim como o DDM, o EDDM utiliza duas condições (com a definição de dois limiares):

- **Condição 1:** $(p'_i + 2s'_i)/(p'_{max} + 2s'_{max}) < \alpha$ para o nível de aviso. Depois desse nível, os exemplos são mantidos devido à possibilidade de mudança no contexto;
- **Condição 2:** $(p'_i + 2s'_i)/(p'_{max} + 2s'_{max}) < \beta$ para o nível de *concept drift*. O modelo e os valores p'_{max} e s'_{max} são reiniciados e um novo modelo é iniciado utilizando os exemplos armazenados desde a ativação do nível de aviso.

Caso a similaridade entre o valor atual de $p'_i + 2s'_i$ e o valor $p'_{max} + 2s_{max}$ aumentar ultrapassando o limiar de aviso, os exemplos armazenados são removidos e o método retorna a sua normalidade.

Além das técnicas supracitadas, existem diversas propostas de métodos de detecção de *concept drift* na literatura, como (PATIL; KULKARNI, 2013) (HAYAT; HASHEMI, 2010).

Os métodos DDM e EDDM foram utilizados na comparação de um dos algoritmos propostos neste projeto, o FDDM (Fractal Drift Detection Method), apresentado no capítulo 3. Na próxima seção são apresentadas as particularidades da classificação utilizando árvores de decisão considerando a abordagem incremental.

2.6 Árvores de Decisão Incrementais

De acordo com Gama, Rocha e Medas (2003) na abordagem incremental a técnica de indução de árvores de decisão mantém em cada nó de decisão um conjunto de estatísticas suficientes e somente divide um nó quando existe evidência estatística suficiente para tanto.

Em (DOMINGOS; HULTEN, 2000) foi apresentado o Hoeffding Tree (HT), um algoritmo básico de árvore de decisão para *data streams*. No mesmo trabalho foi proposto o VFDT (*Very Fast Decision Tree*), que é baseado no HT.

Para decidir quantos exemplos são necessários para a divisão de um nó, o VFDT utiliza o Hoeffding *bound*, apresentado na Equação 2.7.

Conforme Domingos e Hulten (2000): seja r uma variável de valor real aleatório cujo intervalo é R; seja n o número de observações independentes dessa variável cuja média é $\overline{r}$. O Hoeffding *bound* afirma que, com probabilidade $1-\delta$ (sendo δ a confiança) que a verdadeira média de r é ao menos $\overline{r}-\epsilon$, em que:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{2.7}$$

O VFDT permite o uso do Ganho de Informação ou Gini Index como medida de avaliação de atributo, ambas mencionadas aqui como critério G, e inclui os seguintes refinamentos, com relação ao HT:

- **Empate:** quando dois ou mais atributos tem o G muito similar, provavelmente muitos exemplos serão necessários para decidir entre eles com uma confiança alta. Neste caso, conforme (DOMINGOS; HULTEN, 2000), não faz muita diferença com relação a que atributo escolher. Dessa forma o VFDT pode opcionalmente decidir que existe um empate e realizar a divisão utilizando o melhor atributo atual se $\Delta\overline{G} < \epsilon < \tau$, em que τ é um limiar especificado pelo usuário;
- **Cálculo do G:** a parte mais significante no tempo de processamento de um exemplo é o recálculo de G. É ineficiente calcular G para cada novo exemplo, até porque ele tem pouca

influência na decisão de divisão. Assim, o VFDT permite a especificação de um número mínimo de exemplos n_{min} que deve ser acumulado em uma folha antes do recálculo de G;

- **Memória:** leva em consideração o fato de que a memória RAM disponível é limitada (finita). No VFDT, a memória é utilizada para manter os contadores de todos as folhas. De acordo com Domingos e Hulten (2000), se o máximo de memória é atingido, o VFDT desativa as folhas menos promissoras;

- **Atributos pobres:** quando atributos não parecem ser promissores, o uso de memória é minimizado descartando-os. Assim que a diferença entre o G de um atributo e o G do melhor atributo se tornar maior que ϵ, o atributo pode ser descartado e a memória usada para armazenar os contadores correspondentes é libertada;

- **Inicialização:** o VFDT pode receber como entrada uma árvore já inicializada;

- ***Rescans*:** o VFDT pode reler exemplos observados anteriormente. Essa opção pode ser ativada se os dados chegarem devagar o suficiente, fornecendo tempo para isso, ou se a base de dados é finita e pequena o suficiente, possibilitando varrê-la múltiplas vezes. Isso pode ajudar o algoritmo a construir uma árvore de decisão maior, que descreva os dados de forma mais eficiente.

Neste trabalho será dada uma atenção maior ao VFDT, por ser o algoritmo utilizado como base no desenvolvimento deste projeto. No Algoritmo 2 é apresentado um algoritmo básico de indução de árvore de decisão incremental baseado no VFDT, adaptado de (BIFET, 2010).

Para encontrar o melhor atributo para testar em um nó da árvore, pode ser suficiente considerar apenas um pequeno subconjunto de exemplos de treinamento que passam por aquele nó. Assim, dado um *stream* de exemplos, os primeiros serão usados para escolher o teste da raiz; uma vez o atributo da raiz é escolhido, os próximos exemplos passarão para baixo, para as folhas correspondentes e usadas para escolher os atributos apropriados de lá, e assim recursivamente.

O algoritmo inicia com um nó folha, a raiz da árvore (linha 1). Quando um novo exemplo chega, ele é ordenado até sua folha correspondente, são coletadas as estatísticas suficientes do dado e é incrementado n_l, que é o número de exemplos observados no nó l (linhas 3, 4 e 5).

Na linha 6 é verificado se foram observados exemplos suficientes no nó em questão para tentar realizar a divisão do mesmo. Isso é feito utilizando o n_{min} (número mínimo de exemplos que devem ser lidos para a tentativa de divisão) e também verificando se todos os dados do nó até aquele momento pertencem à mesma classe. Se sim, não há a necessidade de divisão.

Algoritmo 2: Algoritmo VFDT.

Entrada: n_{min}: número mínimo de exemplos do período de tolerância
τ : limiar de empate
Saída : Árvore de decisão *HT*.

```
1  Seja HT uma árvore com uma única folha (a raiz)
2  para cada exemplo de treinamento faça
3      Ordene o exemplo na folha l usando HT
4      Atualize as estatísticas em l
5      Incremente n_l, o número de exemplos vistos em l
6      se n_1 mod n_min = 0 e exemplos observados em l não pertencem à mesma classe
       então
7          Calcule Ḡ_l(X_i) para cada atributo
8          Seja X_a o atributo com maior Ḡ_l
9          Seja X_b o atributo com o segundo maior Ḡ_l
10         Calcule o Hoeffding bound ε = sqrt(R^2 ln(1/δ) / 2n_l)
11         se X_a ≠ X_∅ e (Ḡ_l(X_a) − (Ḡ_l X_b) > ε ou ε < τ) então
12             Substitua l por um nó interno que divide em X_a
13             para cada ramo resultante da divisão faça
14                 Adicione uma nova folha com as estatísticas iniciadas
```

Nas linhas 7, 8 e 9 é utilizada a heurística para a escolha dos dois melhores atributos a ser utilizados na divisão do nó. Segundo Domingos e Hulten (2000) as heurísticas disponíveis na VFDT são o Ganho de Informação (apresentado na Seção 2.3) e o Gini Index.

Como mencionado anteriormente, para resolver o problema de decidir exatamente quantos exemplos são necessários para cada nó, na linha 10 do Algoritmo 2 é utilizado o Hoeffding *bound*, apresentado na Equação (2.7).

Na linha 11 são verificadas as seguintes condições:

- $X_a \neq X_\emptyset$: se ao menos um atributo foi escolhido, ou seja, se o melhor atributo é diferente de nulo;
- $\bar{G}_l(X_a) - (\bar{G}_l X_b) > \epsilon$: se a diferença entre os dois melhores atributos é maior que ϵ. Essa condição é testada para evitar realizar a divisão de um nó quando dois ou mais atributos têm valores muito próximos, pois um atributo poderia se tornar o melhor nas próximas iterações;
- $\epsilon < \tau$: conforme n (número de exemplos observados no nó) aumenta, ϵ tende a diminuir. Caso os dois melhores atributos tenham valores muito próximos em diversas iterações, ϵ seria tão pequeno quando τ, que é um critério de empate.

Caso a condição da linha 11 for satisfeita, na linha 12 a então folha l torna-se um nó interno que divide utilizando o atributo X_a. Na linha 14 são iniciadas as estatísticas suficientes para cada folha resultante da divisão do nó.

Deve-se considerar o menor número possível de dados a serem armazenados na árvore, evitando assim um maior uso de memória. A decisão a respeito de quais estatísticas são suficientes e que devem ser armazenadas nas folhas da árvore depende muito do que é necessário para o funcionamento do algoritmo e da heurística de divisão dos nós. Muitas vezes utiliza-se de métodos incrementais para a obtenção de um valor aproximado para um cálculo, evitando o armazenamento demasiado de estatísticas nas folhas.

No Algoritmo 3 é apresentado um método incremental utilizado no desenvolvimento deste projeto. Trata-se de uma aproximação Gaussiana apresentada em (PFAHRINGER; HOLMES; KIRKBY, 2008), que é baseada no trabalho de (WELFORD, 1962) e que tem suas vantagens estudadas em (CHAN; LEWIS, 1979). De acordo com Pfahringer, Holmes e Kirkby (2008) trata-se de um método incremental robusto menos propenso a erros numéricos. Ele mantém três valores em memória, a soma dos pesos, a média e a soma das variâncias, mantendo-os de forma que seja menos vulnerável a erros de arredondamento. A saída do algoritmo, que é a média e a variância, pode ser acessada sempre que necessário.

Algoritmo 3: Algoritmo para o cálculo Gaussiano incremental.

Entrada: Um conjunto de pontos de dados.
Saída : *media*: centro da distribuição;
variancia: dispersão da distribuição.

1 $somaPeso = peso_{primeiro}$
2 $media = valor_{primeiro}$
3 $somaVariancia = 0$
4 **para cada** *ponto de dados* $(valor, peso)$ *depois do primeiro* **faça**
5 $somaPeso = somaPeso + peso$
6 $ultimaMedia = media$
7 $media = media + \frac{valor - ultimaMedia}{somaPeso}$
8 $somaVariancia = somaVariancia + (valor - ultimaMedia) \times (valor - media)$
9 $somaPeso = somaPeso + peso$
10 ***Saída disponível a qualquer momento:***
11 **retorne** $media = media$
12 **retorne** $variancia = \frac{somaVariancia}{somaPeso}$

Na literatura são apresentadas diversas adaptações do VFDT para contextos específicos; alguns exemplos descritos a seguir, de maneira sucinta.

Em (GAMA; ROCHA; MEDAS, 2003) foi proposto o VFDTc, uma extensão do VFDT

que utiliza o Ganho de Informação, lida com atributos numéricos e usa naïve Bayes nas folhas, considerada pelos autores uma técnica mais poderosa de classificação dos exemplos.

O OVFDT (*Optimized Very Fast Decision Tree*) foi proposto em (YANG; FONG, 2011) para controlar o tamanho da árvore mantendo uma boa acurácia. Isso é permitido usando um limiar de empate e poda incremental na indução da árvore.

Em (LI; ZHANG; LI, 2009) foi proposto o OcVFDT (*One-class Very Fast Decision Tree*) que é baseado no VFDT e no POSC4.5, e tem como objetivo a classificação *one-class*, que ocorre quando as classes dos exemplos não são conhecidas previamente.

Adicionalmente, em (REHMAN; LI; LI, 2012) os autores apresentam o algoritmo Empirical Bernstein Tree (EBT), que propõe a substituição do Hoeffding *bound*. Um dos problemas de se usar o HB, segundo os autores, é que o modelo construído pode se tornar mais conservador, necessitando de mais exemplos do que é realmente necessário.

O algoritmo VFDT original não trata do *concept drift*. Em (HULTEN; SPENCER; DOMINGOS, 2001) foi proposto o CVFDT, um algoritmo eficiente para mineração de árvores de decisão utilizando *data streams* gerados com mudanças constantes (com *concept drift*) e que é baseado no VFDT. O algoritmo cria subárvores alternativas, e sempre que o modelo atual se torna questionável, ele é substituído por uma subárvore alternativa com melhor acurácia.

Neste trabalho é proposto o StARMiner Tree, apresentado no Capítulo 3. Trata-se de um algoritmo de classificação de árvores de decisão para *data streams* baseado no VFDT, que utiliza estatísticas para decidir qual atributo usar na divisão de um nó por meio do algoritmo StARMiner (RIBEIRO, 2008), que é apresentado na Seção 2.7.

2.7 Algoritmo StARMiner

O algoritmo StARMiner (*Statistical Association Rule Miner*) foi proposto inicialmente em (RIBEIRO et al., 2005) e tem como objetivo minerar regras de associação estatísticas a partir de atributos com valores contínuos. O algoritmo encontra regras que selecionam um conjunto mínimo de características que preservam a habilidade de diferenciar os dados em categorias.

Conforme Ribeiro (2008), sendo x_j uma classe (categoria) e a_i um atributo (característica), as regras retornadas pelo algoritmo têm o formato geral apresentado na Equação (2.8).

$$x_j \rightarrow a_i \tag{2.8}$$

O antecessor da regra indica um subconjunto de dados que pertence à classe x_j, e o sucessor da regra representa um atributo a_i cujo comportamento é diferente e uniforme nos dados da classe x_j em relação às demais classes.

O StARMiner retorna as regras que satisfazem as seguintes condições:

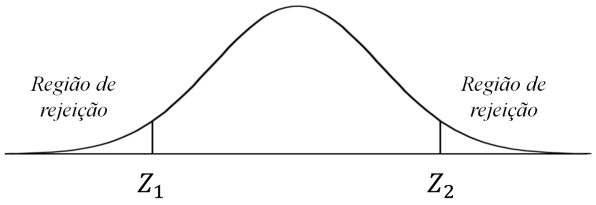
1. O atributo a_i deve ter um comportamento para a classe x_j diferente do seu comportamento nas demais classes dos dados;
2. O atributo a_i deve apresentar um comportamento uniforme nos dados da classe x_j.

Para atender as condições, o algoritmo utiliza restrições de interesse para a mineração das regras, que são:

- $\Delta\mu_{min}$: a mínima diferença entre as médias do atributo a_i entre os dados da classe x_j e os demais dados da base;
- σ_{max}: o desvio padrão máximo permitido do atributo a_i em dados da classe x_j;
- γ_{min}: a mínima confiança para rejeitar a hipótese H de que são iguais estatisticamente às médias dos valores de a_i nos conjuntos T_{x_j} (dados da classe x_j) e $T - T_{x_j}$ (dados das demais classes).

Usando o teste One-Sample Z-test, a hipótese H deve ser rejeitada com confiança igual ou maior que σ_{min}, e para isso são calculados os valores de Z (distância da média em relação ao desvio padrão da média) utilizando a Equação (2.9). Como mostra a Figura 2.3, H é rejeitada se a mesma estiver na região de rejeição, e os valores críticos Z_1 e Z_2 dependem de σ_{min}.

$$Z = \frac{\mu_{a_i}(T_{x_j}) - \mu_{a_i}(T - T_{x_j})}{\frac{\sigma_{a_i}(T_{x_j})}{\sqrt{(|T_x|)}}} \tag{2.9}$$



(a) Ilustração das regiões de rejeição

γ_{min}	0,9	0,95	0,99
Z_1	-1,64	-1,96	-2,58
Z_2	1,64	1,96	2,58

(b) Exemplos de valores de Z de acordo com o γ_{min}

Figura 2.3: Regiões de rejeição do teste de hipóteses

Uma regra $x_j \rightarrow a_i$ retornada pelo algoritmo relaciona um atributo a_i com uma classe x_j, onde os valores de a_i têm um comportamento estatisticamente diferente em dados da classe x_j. Esta propriedade indica que a_i é um atributo importante para distinguir dados da classe x_j dos demais dados (RIBEIRO, 2008).

No Algoritmo 4 é apresentado o algoritmo StARMiner, extraído de (RIBEIRO, 2008). O algoritmo recebe como entrada uma base de dados T da forma $(x_j, a_1, a_2, ..., a_n)$, em que x_j representa a classe da tupla e a_i um atributo dos dados, e também os limiares $\Delta\mu_{min}$, σ_{max} e γ_{min}.

Algoritmo 4: Algoritmo StARMiner.

Entrada: Base de dados T na forma $(x_j, a_1, a_2, ..., a_n)$;
Limiares $\Delta\mu_{min}$, σ_{max} e γ_{min}.
Saída : Regras mineradas.

```
1  percorra a base de dados T
2  para cada atributo a_i faça
3      para cada classe x_j faça
4          calcule μ_{a_i}(T_{x_j}) e μ_{a_i}(T − T_{x_j});
5  percorra a base de dados T
6  para cada atributo a_i faça
7      para cada classe x_j faça
8          calcule σ_{a_i}(T_{x_j}) e σ_{a_i}(T − T_{x_j})
9          calcule o valor de Z_{i_j}
10         se (μ_{a_i}(T_{x_j}) − μ_{a_i}(T − T_{x_j})) ≥ Δ_min e σ_{a_i}(T_{x_j}) ≤ σ_max e (Z_ij < Z_1 ou Z_{i_j} > Z_2
           então
11             escreva x_j → a_i, μ_{a_i}(T_{x_j}), μ_{a_i}(T − T_{x_j}), σ_{a_i}(T_{x_j}), σ_{a_i}(T − T_{x_j})
```

Os dados da base são lidos e os valores das médias de todos os atributos são calculados para cada classe (linhas 1 a 4). Na segunda varredura da base é calculado o desvio padrão e o valor de Z de cada atributo (linhas 5 a 9). A restrição de interesse é processada e a regra é retornada somente se a mesma satisfizer os limiares fornecidos como entrada, $\Delta\mu_{min}$, σ_{max} e γ_{min} (linhas 10 e 11).

Segundo Ribeiro (2008) a complexidade do algoritmo StARMiner é $\theta(ckN)$, em que N é o número de instâncias da base, k é o número de atributos e c é o número de classes.

As regras estatísticas encontradas pelo StARMiner revelam quais atributos têm o maior poder de diferenciar classes dos dados analisados. Isso ocorre porque o algoritmo gera regras que envolvem atributos que apresentam um comportamento uniforme e particular em dados da mesma classe. Uma terceira condição apresenta como é feito o processo de seleção de

características binária a partir das regras retornadas pelo algoritmo:

3. As características (ou atributos) presentes no conjunto de regras retornadas pelo algoritmo StARMiner são selecionadas como as mais relevantes.

Nesta seção foi apresentado o StARMiner, um dos algoritmos utilizados na implementação do algoritmo StARMiner Tree, que é apresentado no Capítulo 3. Neste trabalho é proposto o StARMiner Tree, um algoritmo de classificação de árvores de decisão baseado no VFDT e que usa StARMiner como critério para a divisão dos nós da árvore.

Um dos principais problemas relacionados à mineração de *data streams* é o chamado *concept drift*, descrito na Seção 2.5. Uma das propostas desse trabalho é o uso da teoria dos fractais, apresentada na Seção 2.8, para a detecção de mudanças no conceito dos dados.

2.8 Teoria dos Fractais

Conforme Traina-Jr. et al. (2010) um fractal é conhecido pela propriedade de auto similaridade. Um conjunto de dados tem as mesmas propriedades para uma grande variação em escala ou tamanho. Isso significa que partes de qualquer tamanho do fractal são exatamente ou estatisticamente similares ao fractal como um todo.

Na Figura 2.4 é apresentado o Triângulo de Sierpinsky como um exemplo de fractal. A partir de um triângulo equilátero é realizado um recorte triangular no centro do mesmo, gerando assim três novos triângulos menores. Segundo Traina-Jr. et al. (2010) esse processo é repetido infinitas vezes para cada um dos triângulos menores, gerando assim o Triângulo de Sierpinsky.

De acordo com Ribeiro (2008), a dimensão fractal é utilizada na geometria dos fractais para a interpretação estatística dos fractais, que indica o quanto um determinado fractal preenche o espaço em que ele está imerso. No caso do Triângulo de Sierpinski, mostrado na Figura 2.4, ele está imerso no espaço 2D, apesar de não ser considerado um objeto de duas dimensões, pois sua área tende a zero. No entanto, ele também não é considerado um objeto unidimensional, pois seu perímetro tende ao infinito.

Duas definições importantes, descritas em (TRAINA-JR. et al., 2010), são a dimensão de imersão e a dimensão intrínseca dos dados:

- A **dimensão de imersão** E de um conjunto de dados é a dimensão em que eles estão inseridos;

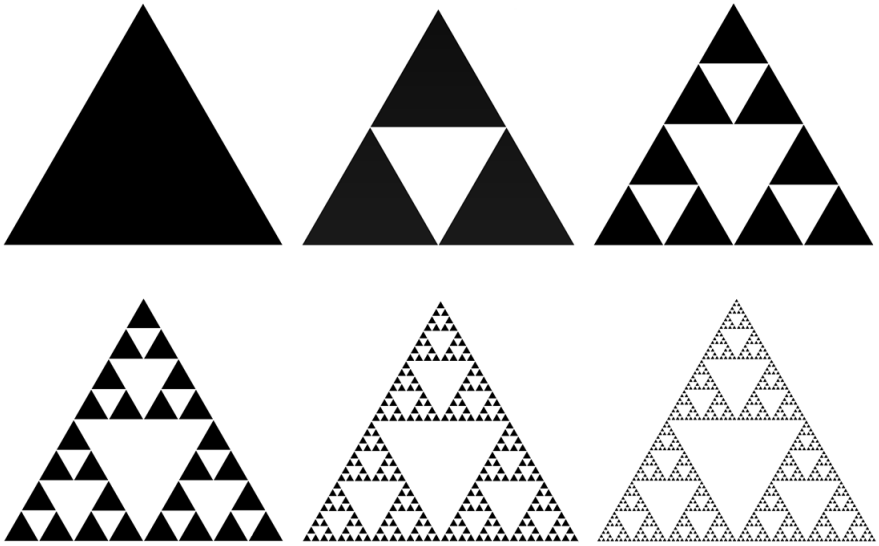


Figura 2.4: Triângulo de Sierpinski

- A **dimensão intrínseca** D de um conjunto de dados é a dimensão real do objeto, independentemente de sua dimensão de imersão.

De acordo com (RIBEIRO, 2008) um exemplo de dimensões de imersão e intrínseca, considerando um conjunto de pontos dispostos de maneira a formar um segmento de reta em um espaço tridimensional, é: a dimensão de imersão do conjunto é 3, mas sua dimensão intrínseca é 1. Ou seja, o conjunto pode ser perfeitamente transferido para o espaço unidimensional, conservando a distância entre os pontos.

Considerando um espaço multidimensional, um conjunto de dados pode ser representado por meio de características, que são os atributos ou as colunas de uma tabela, e objetos de dados, que são as linhas.

Segundo (TRAINA-JR. et al., 2010), conceitualmente se um conjunto de dados tem todas suas variáveis independentes entre si, sua dimensão intrínseca é a dimensão de imersão. No entanto, sempre que houver correlação entre duas ou mais variáveis a dimensão intrínseca é consequentemente diminuída. Normalmente a dimensão de imersão de um conjunto de dados esconde as reais características do conjunto de dados.

Em (SOUSA, 2006) são apresentadas algumas medidas para a dimensão fractal. A mais simples, segundo a autora, é voltada para fractais exatamente auto similares. Um fractal auto similar é composto por M réplicas em miniatura dele mesmo, em que cada réplica é uma versão em escala reduzida $1 : s$ do fractal original.

A seguir é apresentada a definição da dimensão fractal e exemplos de alguns métodos de cálculo, extraídos de (SOUSA, 2006).

Definição 3.1. Dimensão Fractal D: sendo M o número de réplicas e s o fator de escala segundo o qual cada réplica está reduzida, a Dimensão Fractal D de um fractal exatamente auto similar definido em um espaço E-dimensional é:

$$D \equiv \frac{\log M}{\log s} \tag{2.10}$$

Para fractais exatamente auto similares, a fórmula apresentada em 2.10 é adequada, pois apresenta regras de construção bem definidas. No entanto, fractais estatisticamente auto similares não estão associados a regras de construção bem definidas. A Definição 3.2 apresenta o método Box-Counting, apropriado para esse contexto.

Definição 3.2. Método Box-Counting: seja P um conjunto de pontos imerso em um espaço E-dimensional dividido por hiper-reticulado com células de lado r, e seja $N(r)$ o número de células que contêm um ou mais pontos do conjunto. A dimensão fractal D_0 para fractais infinitos, denominada Dimensão Fractal de Hausdorff, é definida por:

$$D_0 \equiv -\lim_{r \to 0} \frac{\log(N(r))}{\log(r)} \tag{2.11}$$

Para fractais com números de pontos finito a análise deve ficar restrita a um intervalo de escalas (r_1, r_2), para o qual o conjunto apresente, estatisticamente, auto similaridade (SOUSA, 2006). Nesse contexto é apropriado o uso da derivada equivalente ao $\lim_{r \to 0}$, permitindo o cálculo da dimensão fractal de Hausdorff do conjunto de dados em intervalos significativos, de acordo com a Definição 3.3.

Definição 3.3. Dimensão Fractal de Hausdorff D_0: para um conjunto finito de pontos P que apresenta a propriedade de auto similaridade em um intervalo de escalas (r_1, r_2), a dimensão de Hausdorff D_0 para este intervalo é definida por:

$$D_0 \equiv -\frac{\partial \log(N(r))}{\partial \log(r)} = constante \qquad r_1 < r < r_2 \tag{2.12}$$

De acordo com (RIBEIRO, 2008), uma das formas mais simples de cálculo da dimensão fractal é utilizando a dimensão de correlação D_2, descrita na Definição 3.4.

Definição 3.4. Dimensão de Correlação D_2: dado um hiper-reticulado de células i de lado r, cuja contagem de ocupação $C^2_{r,i}$ corresponde à quantidade de pontos que incidem na i-ésima célula, a dimensão fractal de correlação D_2 é dada por:

$$D_2 \equiv \frac{\partial \log(\sum_i C_{r,i}^2)}{\partial \log(r)} \qquad r \in [r_1, r_2] \tag{2.13}$$

onde r_1 e r_2 correspondem a mínima e máxima distâncias entre dois pontos quaisquer.

As formas de cálculo citadas acima são alguns exemplos de como a dimensão fractal pode ser calculada. Existem diversas outras formas não descritas aqui que podem ser utilizadas em diferentes contextos, de acordo com os dados disponíveis. Neste projeto foi utilizado o algoritmo SID-Meter, proposto em (SOUSA et al., 2006), como método de detecção de *concept drift* na árvore de decisão gerada pelo classificador. Na próxima seção é descrito o algoritmo utilizado, SID-Meter.

2.9 Algoritmo SID-Meter

Nesta seção será apresentado o algoritmo SID-Meter (*data Stream Intrinsec Dimension meter*), proposto em (SOUSA et al., 2006) para medir a dimensão intrínseca D dos dados por meio da dimensão de correlação fractal D_2 entre atributos. O algoritmo é baseado na abordagem Box-Occupancy Counting, foi desenvolvido para lidar com *data streams* e considera os seguintes requisitos:

- O valor atualizado de D pode ser obtido a qualquer momento;
- Cada exemplo é processado de acordo com sua chegada, e então descartado;
- O espaço utilizado pelos exemplos nas janelas pode ser modificado durante o tempo, conforme novos exemplos chegam e os mais antigos são descartados;
- D pode mudar lentamente conforme novos exemplos chegam e os antigos passam a ser desconsiderados;
- Uma sequência de exemplo pode exceder a memória;
- Cada exemplo apresenta muito pouca informação individualmente. Assim, uma mudança significativa no comportamento dos dados requer uma sequência de eventos.

Para processar os exemplos, o algoritmo utiliza uma janela deslizante (*sliding window*) dividida em n_c períodos de contagem, e cada período é definido por um número predeterminado de exemplos e_i. Conforme exibido na Figura 2.5, quando n_i exemplos são gerados e processados, os n_i exemplos mais antigos são descartados.

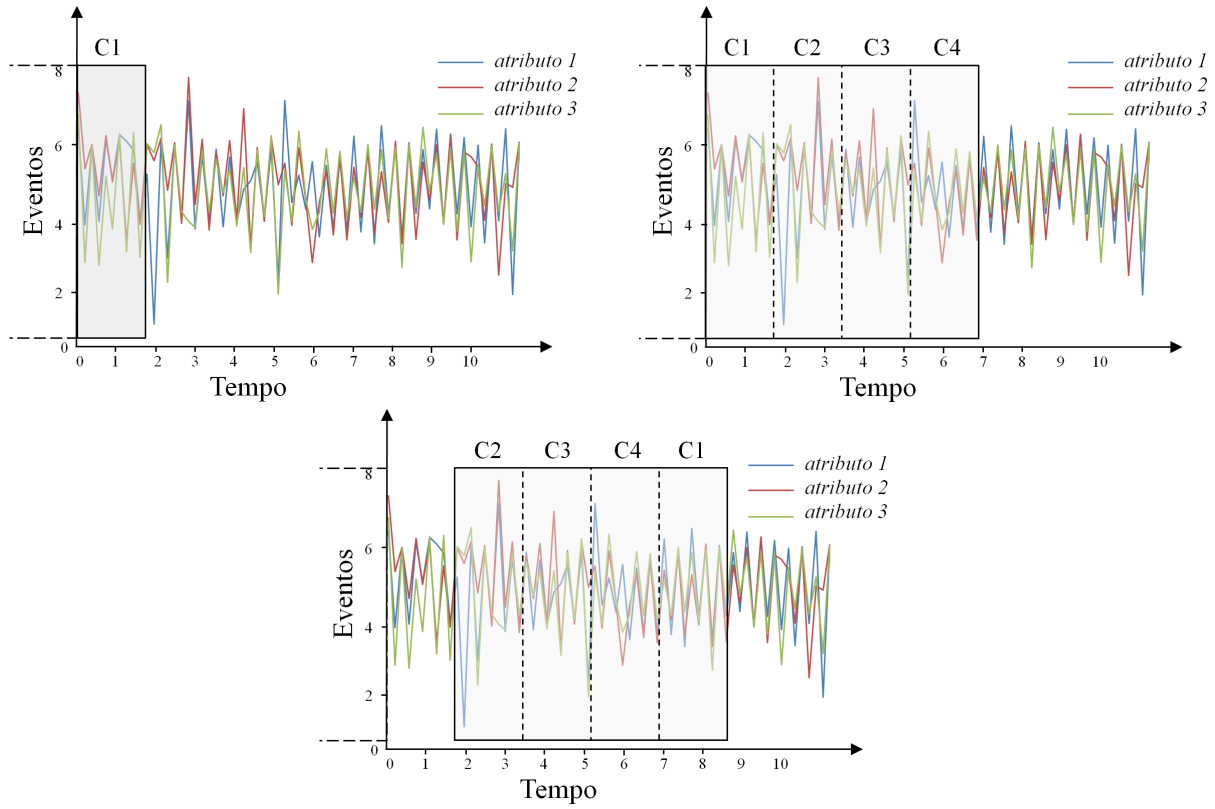


Figura 2.5: Exemplo de períodos de contagem em uma janela deslizante sobre um *data stream* tridimensional, extraído de (SOUSA; RIBEIRO; TRAINA, 2006)

Como descrito em (SOUSA et al., 2006), o algoritmo funciona da seguinte forma. Primeiramente é definido o tamanho inicial do hipercubo, utilizando os primeiros exemplos assim que o processo é iniciado. Dessa forma, depois de n_i exemplos recebidos é possível determinar o hipercubo inicial. Considerando rl_i como o menor valor do atributo a_i e rh_i o maior valor do atributo a_i, a distância (*range*) $r_0 = max(rh_i - rl_i)$ determina o tamanho do lado do hipercubo inicial. A estrutura de grades (*grids*) é criada gerando até R grades E-dimensionais sucessivas de células de tamanho $r_j = r_{j-1}/2$, em que R determina o número de pontos da plotagem do Box-Counting. Para cada célula no nível j, 2^E células são geradas no próximo nível, $j+1$.

O algoritmo SID-Meter utiliza a estrutura de árvore *counting tree* (que é mantida na memória) para o armazenamento dos dados nas janelas. Cada nível j corresponde a uma grade de células de tamanho lateral r_j (com r_0 correspondendo ao nível folha), e cada nó corresponde a uma célula. A Figura 2.6 apresenta um exemplo de *counting tree* com duas dimensões, extraída de (SOUSA, 2006). A identificação de uma célula é feita por meio do identificador $[b_1 b_2 ... b_e]$ como parte de uma célula no nível imediatamente superior, tal que $b_i = 0$ para células na metade inferior da dimensão i, e $b_i = 1$ na metade superior. Um nó da árvore é criado somente quando existe pelo menos um exemplo na célula correspondente da grade. Dessa forma, o número total de nós em cada nível é no máximo o número de exemplos dentro da janela deslizante.

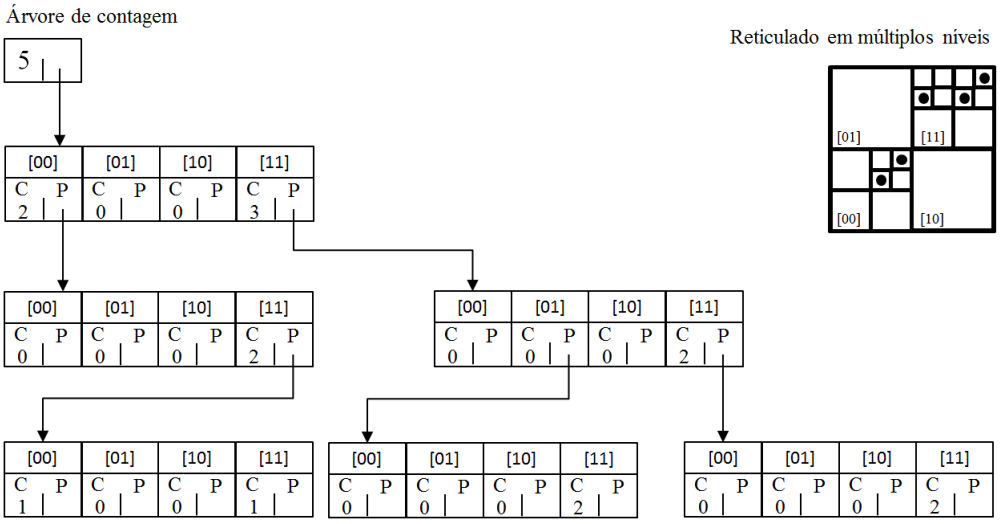


Figura 2.6: Exemplo de uma árvore de contagem de quatro níveis no espaço bidimensional, extraído de (SOUSA, 2006)

Em cada nó da árvore os períodos de contagem (*counting periods*) são representados por um *array* $C[]$ e n_c contadores, um para cada período k. $C[]$ é utilizado como uma lista circular, tal que um contador atual calcula a ocupância (número de eventos) dos exemplos lidos no período de contagem atual correspondente. Quando um período expira, o próximo contador é zerado e usado para contar os exemplos do período seguinte. Assim, quando n_c períodos são completados, os exemplos mais antigos são descartados e o próximo contador é utilizado para calcular a ocupação dos novos exemplos, permitindo que o valor de D seja calculado utilizando os dados mais recentes.

Os n_i primeiros exemplos utilizados para definir o tamanho inicial do hipercubo pertencem ao primeiro período de contagem, e uma vez completo, a dimensão intrínseca D é calculada por meio da Equação 2.13. O valor de $C_{r_j,i}$ em cada nó i é a soma $C_{r_j,i} = \sum_{k=0}^{k<n_c} C_i[k]$ para cada nível j da árvore (lembrando que um nível j corresponde a uma grade de célula tamanho r_j). A plotagem Box-Counting é construída realizando uma navegação completa pela árvore e plotando $< log(\sum_{i=0}^{i<2^E} C_{r_j}^2, i), log(r_j) >$ para cada valor r_j. O ângulo da linha que melhor define o gráfico fornece uma estimativa da dimensão intrínseca D para os exemplos presentes na janela deslizante. A memória ocupada pelos eventos é então liberada para ser utilizada pela árvore de contagem.

Nos trabalhos (SOUSA et al., 2006), (SOUSA, 2006) e (SOUSA; RIBEIRO; TRAINA, 2006) o algoritmo é apresentado com mais detalhes. O algoritmo SID-Meter, apresentado nesta seção, foi utilizado como método de detecção de mudança de tendência nos dados dentro do classificador proposto. No Capítulo 3 seu uso neste trabalho é descrito detalhadamente.

Na próxima seção são apresentados métodos utilizados na comparação dos resultados da classificação, de forma a avaliar, comparar e validar os resultados obtidos.

2.10 Comparação de Métodos de Classificação

Assim como ocorre para a classificação convencional, para comparar os resultados obtidos com a classificação de *data streams* são utilizados métodos para definir quais exemplos serão utilizados para o treinamento do algoritmo e quais serão utilizados para testar o modelo construído. Na abordagem não incremental um método muito usado é a validação cruzada (*cross-validation*). De acordo com Han, Kamber e Pei (2011) uma das abordagens da validação cruzada (a *k-fold cross-validation*) particiona os dados em k subconjuntos mutuamente exclusivos, de tamanho aproximadamente igual. Assim, o treino e o teste são realizados k vezes.

De acordo com Bifet (2010) no contexto de *data streams* uma das principais preocupações é como representar o estado atual da classificação no tempo. Duas abordagens muito conhecidas são:

Holdout: com dados em grandes quantidades, a validação cruzada consome muito tempo de processamento. Assim, utiliza-se o *holdout*, que é útil principalmente quando a divisão do conjunto entre dados de treino e de teste foi pré-definida. Com o *holdout* é realizada um único teste periodicamente (em um intervalo de exemplos predefinido).

Prequential ou Interleaved Test-Then-Train: trata-se do teste e treino intercalados, em que cada exemplo individual pode ser utilizado para testar o modelo antes de ser utilizado para o treinamento, fazendo com que a acurácia possa ser calculada de forma incremental. Dessa forma o modelo sempre será testado por exemplos que não observados até o momento, e depois esses exemplos serão utilizados para treinar o modelo.

Ainda em (BIFET, 2010) o autor afirma que as avaliações experimentais devem ser feitas considerando grandes conjuntos de dados. Na prática, algoritmos de classificação de *data streams* devem ser capazes de lidar com quantidades muito grandes (potencialmente infinitas) de exemplos, e a demonstração de sistemas somente em pequenas quantidades de dados não retratam um cenário convincente de sua capacidade para lidar com aplicações que demandam *data streams*.

2.11 Considerações Finais

Nesta seção foram apresentados os conceitos e trabalhos relacionados à proposta deste projeto. Foram descritos com mais detalhes a construção e funcionamento das árvores de decisão incrementais, utilizadas no contexto da classificação de *data streams*. Além disso, o algoritmo StARMiner foi apresentado, por ser um dos algoritmos utilizados na proposta deste trabalho. Por fim foram descritos os principais conceitos com relação à teoria dos fractais, por ser uma técnica a ser utilizada para a detecção de *concept drift* neste trabalho.

No Capítulo 4 são apresentados os métodos propostos e os resultados deste trabalho de mestrado.

Capítulo 3

TRABALHO DESENVOLVIDO

Neste capítulo é apresentado o trabalho desenvolvido durante este projeto de mestrado. São descritos os algoritmos de árvores de decisão propostos StARMiner Tree, Automatic StARMiner Tree e Information Gain StARMiner Tree. Também é apresentado o método de detecção de concept drift desenvolvido, o Fractal Drift Detection Method. Além disso, são descritos diversos experimentos, realizados com os algoritmos propostos.

3.1 Considerações Iniciais

Árvore de decisão é uma forma de representação do modelo classificador popular e intuitiva, rápida de construir, e que geralmente possui alta acurácia. No contexto de *data streams*, as técnicas de árvores de decisão incrementais, em geral, apresentam um custo computacional alto para a construção e atualização do modelo. Esse custo ocorre, principalmente, no que se refere ao cálculo efetuado para a decisão de divisão dos nós. Os métodos existentes também possuem uma característica conservadora a quantidades de dados limitadas (e tendem a melhorar conforme o número de exemplos aumenta). Além disso, em muitas aplicações reais os dados são gerados com ruídos, e as técnicas existentes possuem baixa tolerância a essas ocorrências, não se comportando bem na descrição dos dados.

Neste capítulo são apresentados os métodos desenvolvidos durante este projeto de mestrado. São descritos os algoritmos de árvores de decisão propostos, ST, AST e IST. Esses algoritmos utilizam uma heurística de divisão dos nós baseada em estatísticas, que não é dependente do número de exemplos lidos, mantém alta acurácia na classificação, e que tem um tempo de execução baixo, para dados numéricos (reais). Os algoritmos também apresentam um comportamento tolerante ao lidar com dados ruidosos. Além disso, também é apresentado um método de detecção de *concept drift* baseado na teoria dos fractais, o FDDM, que pode ser

utilizado com qualquer algoritmo de classificação utilizando árvores de decisão.

O capítulo está organizado da seguinte forma: na Seção 3.2 é apresentado o algoritmo StARMiner Tree (ST); na Seção 3.3 são apresentadas as extensões do ST, que são os algoritmos Automatic StARMiner Tree (AST) e Information Gain StARMiner Tree (IST); na Seção 3.4 é descrito o método de detecção de *concept drift* proposto, o Fractal Drift Detection Method; na Seção 3.5 são descritos os experimentos realizados utilizando os algoritmos ST, AST e FDDM; finalmente, na Seção 3.6 são apresentadas as considerações finais deste capítulo.

3.2 Algoritmo StARMiner Tree (ST)

O algoritmo StARMiner Tree (ST) foi desenvolvido considerando a estrutura geral do VFDT, e utiliza o princípio do StARMiner para decidir quando efetuar a divisão de um nó da árvore e qual atributo utilizar para isso. O VFDT utiliza o ganho de informação e o *Hoeffding bound* (HB) para decidir o melhor atributo a ser utilizado na divisão de um nó folha da árvore. No ST foram aplicados os princípios estatísticos do StARMiner para decidir qual o melhor atributo (que descreve melhor os dados). Resumidamente, para cada novo exemplo, o modelo que está sendo criado é utilizado para classificá-lo para sua folha correspondente. Com o passar do tempo há a necessidade de evoluir o modelo, de forma a descrever os dados adequadamente. Assim, uma heurística é utilizada para decidir se uma determinada folha deve ser dividida, e se deve, qual atributo usar no teste daquele nó. No Algoritmo 5 é apresentado o StARMiner Tree.

O algoritmo inicia com a árvore de decisão *ST*, composta de um nó folha, a raiz da árvore (linha 1). Cada exemplo e_i é da forma $e_i = (v, c_x)$, em que C é o conjunto de k valores de classe $C = \{c_1, c_2, ..., c_k\} | 0 < x \leq k, c_x \in C$, e v é um vetor de j atributos (ou características) $v = (a_1, a_2, ..., a_j) | j > 0$. Quando um novo exemplo chega, ele é ordenado até sua folha f correspondente, são coletadas as estatísticas suficientes e é incrementado n_f, que é o número de exemplos observados na folha f (linhas 3, 4 e 5).

As estatísticas suficientes utilizadas pelo ST, para cada atributo $a_m | m \leq j$, e cada valor de classe $c_x | x \leq k$, são a média (Equação 3.1) e a variância incrementais (Equações 3.2 e 3.3), e o número de exemplos lidos em um nó folha n_f (Equação 3.4) para cada valor de classe c_x. Esses dados são mantidos para cada classe e em cada folha f da árvore, como é exemplificado na Figura 3.1.

$$\mu_{n_f a_m}(i, x) = \left\{ \begin{array}{l} 0, \text{ se } n_f = 0 \\ \mu_{n_{f-1}}(i, x) + \frac{valor_{n_f} - \mu_{n_{f-1}}(i,x)}{n_f} \end{array} \right\} \tag{3.1}$$

Algoritmo 5: Algoritmo StARMiner Tree - ST

Entrada: $\Delta\mu_{min}$, a mínima diferença permitida entre as médias
σ_{max}, o desvio padrão máximo permitido
γ_{min}, a confiança mínima
Saída : A árvore de decisão ST.

```
1  Seja ST uma árvore com uma única folha (a raiz)
2  para cada exemplo de treinamento e_i faça
3      Ordene o exemplo na folha f usando ST
4      Atualize as estatísticas em f (μ_{n_f a_m}, σ_{n_f a_m} e n_f c_x para cada valor de classe x)
5      Incremente n_f, o número de exemplos observados em f
6      se n_f mod n_min = 0 e exemplos observados em f não pertencem à mesma classe então
7          L = SelecioneMelhoresAtributos(Δμ_min, σ_max, γ_min)
8          se L ≠ ∅ então
9              Seja X_a o melhor atributo de L
10             Substitua a folha f por um nó interno que divide em X_a
11             para cada ramo gerado pela divisão faça
12                 Adicione uma folha com as estatísticas suficientes inicializadas
```

$$\sigma^2_{n_f}(i,x) = \frac{\omega_{n_f}}{n_f - 1} \tag{3.2}$$

$$\omega_{n_f}(i,x) = \left\{ \begin{array}{l} 0, \text{ se } n_f = 0 \\ \omega_{n_{f-1}}(i,x) + (valor_{n_f}(i,x) - \mu_{n_{f-1}}(i,x)) \times (valor_{n_f}(i,x) - \mu_{n_f}(i,x)) \end{array} \right\} \tag{3.3}$$

$$n_f c_x \tag{3.4}$$

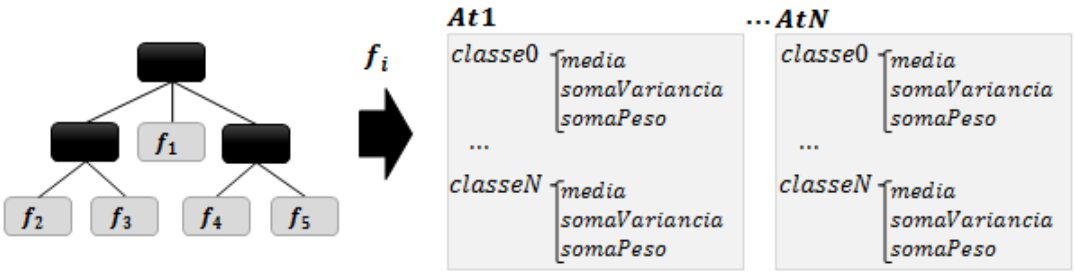


Figura 3.1: Representação das estatísticas suficientes armazenadas em cada folha da árvore

O algoritmo StARMiner original foi modificado para lidar com *data streams* de forma incremental, lendo os dados apenas uma vez para extrair as informações necessárias, que no caso são a média, a variância e o n. Para isso foi utilizado o método de aproximação gaussiano

apresentado no Capítulo 2, no Algoritmo 3.

Na linha 6 é verificado se foram observados exemplos suficientes no nó em questão para a tentativa de divisão do mesmo. Isso é realizado utilizando o parâmetro n_{min} (número mínimo de exemplos que devem ser lidos para a tentativa de divisão). Esse parâmetro funciona como um período de tolerância (*grace period*), pois é computacionalmente custoso realizar a tentativa de divisão de um nó sempre que um exemplo é processado, e um exemplo tem pouca significância na decisão de divisão de um nó, então espera-se até n_{min} exemplos sejam lidos. Também é verificado se todos os dados observados na folha, até aquele momento, pertencem à mesma classe, pois nesse caso não há a necessidade de divisão.

Quando o número mínimo de exemplos é observado, o ST escolhe os melhores atributos para a divisão do nó por meio da Função SelecioneMelhoresAtributos. A função recebe como entrada os três parâmetros $\Delta\mu_{min}$, σ_{max} e γ_{min}. Nas linhas 5 e 6 são selecionados os atributos que satisfazem as seguintes condições:

Função SelecioneMelhoresAtributos($\Delta\mu_{min}$, σ_{max}, γ_{min})

```
1  Seja L a lista de atributos L = {a_1, ..., a_i}
2  Seja L_1 uma lista vazia de atributos
3  para cada atributo a_m ∈ L faça
4      para cada classe c_x ∈ X faça
5          se (μ_{a_m}(T_{c_x}) − μ_{a_m}(T − T_{c_x})) ≥ Δμ_min então
6              se σ_{a_m}(T_{c_x}) ≤ σ_max então
7                  Calcule Z_{m_x} = (μ_{a_m}(T_{c_x}) − μ_{a_m}(T − T_{c_x})) / (σ_{a_m}(T_{c_x}) / √(|T_c|))
8                  Obtenha os valores Z_1 e Z_2
9                  se Z_{m_x} < Z_1 ou Z_{m_x} > Z_2 então
10                     Adicione a_m a L_1
   Retorne: L_1
```

- O atributo a_m deve apresentar um comportamento na classe c_x diferente de seu comportamento nas outras classes;
- O atributo a_m deve apresentar um comportamento uniforme em exemplos da classe c_x.

Para satisfazer essas condições são utilizadas as restrições de interesse do algoritmo StARMiner $\Delta\mu_{min}$, σ_{max}. Nas linhas 7 e 8 são verificadas as regiões de rejeição da hipótese H utilizando o parâmetro γ_{min}, como foi apresentado na Figura 2.3. Se H é rejeitada (linha 9), o atributo a_m é adicionado à lista L_1.

No Algoritmo 5 é verificado se ao menos um atributo foi selecionado, ou seja, se $L \neq \emptyset$ (linha 8). O melhor atributo X_a de L é escolhido como aquele que identifica, respectivamente, mais classes, tem maior $\mu_{a_m}(T - T_{x_j})$ e menor $\sigma_{a_m}(T_{x_j})$. Isso é justificado da seguinte forma: quanto maior o número de classes que o atributo representa, melhor ele divide o espaço em faixas de valores; quando mais distante a média de uma classe, melhor ela se destaca das demais classes; e quanto menor o desvio padrão nos dados da classe, mais uniforme é seu comportamento. O atributo X_a é então utilizado para dividir o nó (linhas 9 e 10). Depois de selecionar o atributo a_m a ser utilizado, é feito o cálculo do ganho de informação para decidir os ramos resultantes da divisão do nó. Para isso são testadas possíveis divisões do nó para a_m com base no menor e maior valores observados, e o número máximo de divisões para teste, que pode ser informado pelo usuário. Por exemplo: supondo que o número máximo de divisões definido seja dez, o algoritmo cria dez pontos de corte entre os valores mínimo e máximo observados, e calcula a entropia e consequentemente o ganho de cada teste por meio das equações 2.1, 2.2 e 2.3. A divisão com maior ganho de informação é escolhido para dividir os ramos daquele nó (linhas 11 e 12).

3.3 Extensões do ST

Durante o desenvolvimento deste projeto de mestrado foram propostas extensões ao algoritmo original, o StARMiner Tree (ST). Nas próximas seções são apresentadas as principais modificações que foram inclusas, e na Seção 3.5 são apresentados experimentos utilizando as variações do ST.

3.3.1 Algoritmo Automatic StARMiner Tree (AST)

Neste trabalho foram adaptadas as fórmulas de parametrização automática para o cálculo dos parâmetros $\Delta\mu_{min}$ e σ_{max}, do algoritmo StARMiner, propostas em (WATANABE et al., 2012). A mínima diferença entre as médias $\Delta\mu_{min}$ é calculada de acordo com a Equação 3.5. O expoente foi modificado para 2,5, por mostrar resultados melhores durante os experimentos realizados. O máximo desvio padrão permitido é calculado conforme a Equação 3.6.

$$\Delta\mu_{min} = min(b_{a_m}) + (max(b_{a_m}) - min(b_{a_m}))^{2,5},$$
em que
$$d_{a_m} = min(||\mu_{a_m}(T_{c_x}) - \mu_{a_m}(T - T_{c_x})| - |\sigma_{a_m}(T_{x_j}) - \mu_{a_m}(T_{c_x})| - |\mu_{a_m}(T - T_{c_x}) - \sigma_{a_m}(T - T_{c_x})||). \quad (3.5)$$

$$\sigma_{max} = \gamma_{min} * max(\sigma_{a_i}(T_{x_j})) \tag{3.6}$$

Dessa forma, foi proposto o Automatic StARMiner Tree (AST), com o cálculo automático dos parâmetros, como mostrado no Algoritmo 6. O algoritmo é basicamente o mesmo que o ST, mas recebe como entrada apenas a confiança mínima γ_{min}. Na linha 7 são calculadas as medidas de interesse de acordo com as Equações 3.5 e 3.6.

Algoritmo 6: Algoritmo Automatic StARMiner Tree - AST

Entrada: γ_{min}, a confiança mínima
Saída : A árvore de decisão AST.

```
1  Seja AST uma árvore com uma única folha (a raiz)
2  para cada exemplo de treinamento e_i faça
3      Ordene o exemplo na folha f usando AST
4      Atualize as estatísticas em f (μ_{n_f a_m}, σ_{n_f a_m} e n_f c_x para cada valor de classe x)
5      Incremente n_f, o número de exemplos observados em f
6      se n_f mod n_min = 0 e exemplos observados em f não pertencem à mesma classe
       então
7          Calcule m_{a_m}, Δμ_min e σ_max
8          L = SelecioneMelhoresAtributos(Δμ_min, σ_max, γ_min)
9          se L ≠ ∅ então
10             Seja X_a o melhor atributo de L
11             Substitua a folha f por um nó interno que divide em X_a
12             para cada ramo gerado pela divisão faça
13                 Adicione uma folha com as estatísticas suficientes inicializadas
```

3.3.2 Algoritmo Information Gain StARMiner Tree (IST)

O algoritmo Information Gain StARMiner Tree (IST) foi desenvolvido para lidar, além dos valores numéricos (reais), com dados categóricos. O algoritmo utiliza dois observadores (módulos) de extração das informações suficientes: o observador StARMiner para dados numéricos (Figura 3.2(a)) e o observador Ganho de Informação (para dados categóricos) (Figura 3.2(b)). Como mostrado na Figura 3.2, para cada exemplo de entrada (c) o algoritmo verifica o tipo de cada atributo (d), e utiliza o observador correspondente para extrair as informações. Ao realizar a divisão do nó (e), o melhor atributo numérico A_1 é escolhido de acordo com os princípios do StARMiner, como acontece nos algoritmos ST e AST. O melhor atributo categórico A_2 é aquele que apresenta maior ganho de informação. Finalmente, para a divisão do nó é escolhido o atributo com maior ganho de informação entre A_1 e A_2 (f).

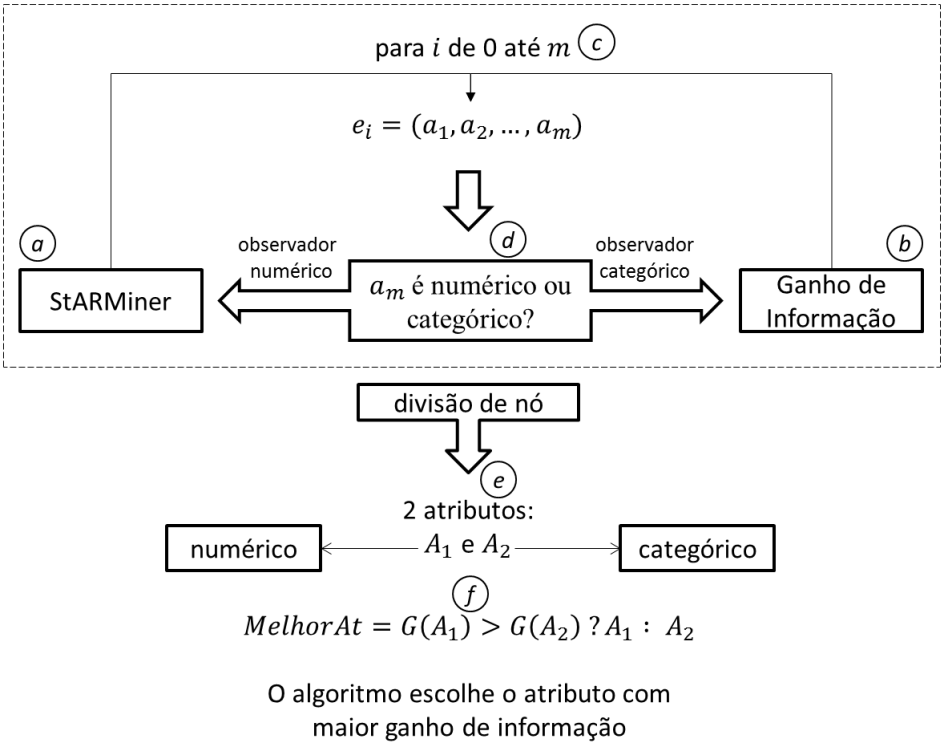


Figura 3.2: Extração de características de acordo com o tipo de cada atributo no algoritmo IST

No algoritmo IST, o StARMiner pode ser utilizado tanto com os valores de entrada informados pelo usuário utilizados no ST, como com a parametrização automática utilizada no AST.

3.4 Fractal Drift Detection Method (FDDM)

Neste trabalho de mestrado foi utilizado o algoritmo SID-Meter (SOUSA et al., 2006) como base para o desenvolvimento do método de detecção de *concept drif* utilizando a teoria de fractais (chamado aqui de *Fractal Drift Detection Method* - FDDM). A Figura 3.3 mostra a estrutura geral do algoritmo.

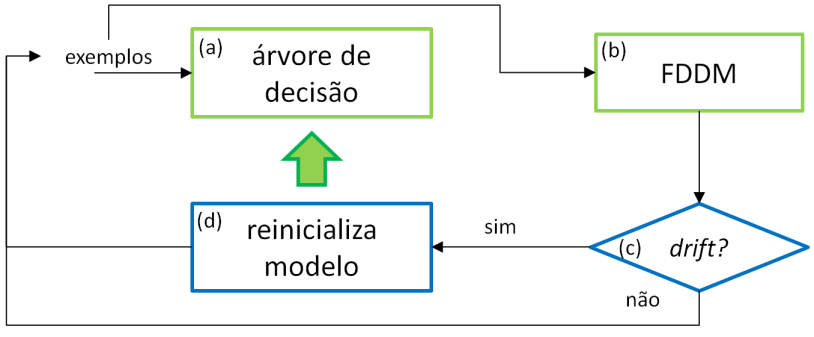


Figura 3.3: Estrutura geral da detecção de mudanças utilizando o FDDM

O método é independente do algoritmo de árvore de decisão, ou seja, ele pode ser utilizado juntamente com o ST, AST ou VFDT, por exemplo. A cada exemplo e_i lido, o algoritmo incrementa o modelo (Figura 3.3(a)) e o exemplo é processado pelo FDDM (Figura 3.3(b)). O FDDM utiliza uma *counting tree* dividida em n_c períodos de contagem, que suportam n exemplos. A estrutura do algoritmo funciona como descrito na Seção 2.9, na explicação do SID-Meter. Na implementação, o algoritmo original (que está em C++) foi passado para Java, e seu funcionamento foi modificado para evitar laços de repetição e o armazenamento de exemplos (em sua totalidade) na memória: o algoritmo original esperava a leitura de n de exemplos antes de atualizar a árvore de contagem (armazenando-os em um *buffer*), e neste trabalho o algoritmo faz isso incrementalmente, sempre que um novo exemplo entra, sem alterar o resultado do cálculo.

Conforme os exemplos são processados pelo FDDM, o algoritmo periodicamente fornece estimativas de D, para os exemplos mais recentes lidos. Considerando a dimensão intrínseca medida no fim de um período p, denotada por D_p, e a dimensão intrínseca no período $p-1$ denotada por D_{p-1}: os valores D_{p-1} e D_p são comparados continuadamente, de modo que quando ocorre uma diferença significativa entre eles, um processo de monitoramento é iniciado (SOUSA; RIBEIRO; TRAINA, 2006). No contexto da classificação, esse evento é chamado de período de aviso (*warning period*). Isso ocorre porque o valor $\lceil D \rceil$ é o menor número de atributos necessários para caracterizar um conjunto de dados multidimensional (TRAINA-JR. et al., 2010). Assim, a ocorrência de $|\lceil D_{p-1} \rceil - \lceil D_p \rceil| > \epsilon$, para $\epsilon > 0$ é considerada uma diferença significante. Na Figura 3.4 é mostrado um exemplo de ocorrência de uma mudança significativa na distribuição (entre os tempos t_3 e t_4). Assim que a mudança foi detectada, é disparado um alarme e o método entra no **período de aviso**. Quanto menor ϵ, mais sensível fica o processo de monitoramento.

Considerando o exemplo da Figura 3.4, o $D_{inicial}$ é a média das medidas D de t_1 até t_4, onde foi detectada a mudança. Ou seja, $D_{inicial} = (D_1 + D_2 + D_3 + D_4)/4$, considerando $n_c = 4$. Assim, como mostra a Figura 3.5, o algoritmo calcula $D_{atualizado}$ com base nas últimas n_c janelas de dados, enquanto $D_{inicial}$ se mantém. Dessa forma, a cada novo período de contagem, o algoritmo verifica se a mudança nas médias foi significativa: $|D_{inicial} - D_{atualizado}| > \alpha$. Em resumo, o período de aviso é iniciado em p, e compreende os seguintes passos:

1. Medindo $\overline{D}_{inicial}$: a média da dimensão intrínseca na janela atual, calculada considerando as últimas n_c medidas de D;

2. Medindo $\overline{D}_{atualizado}$ (a partir do período p): a média atualizada da dimensão intrínseca das últimas n_c medidas de D.

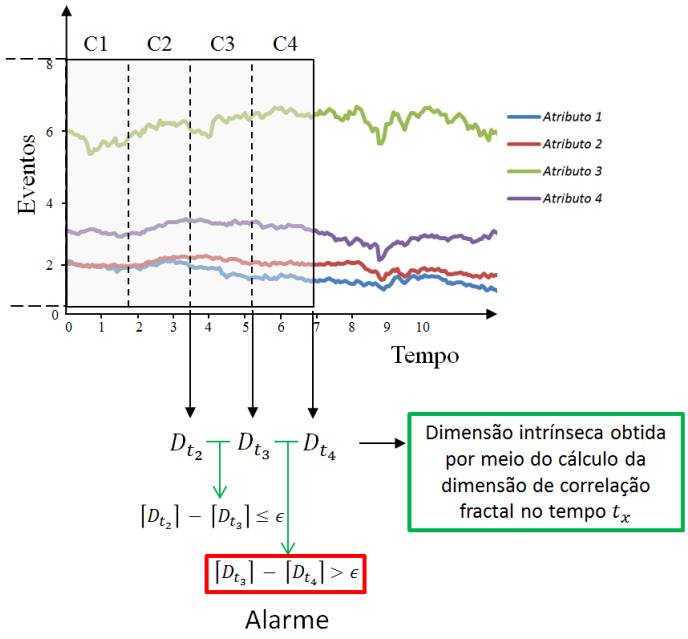


Figura 3.4: Exemplo de alarme ao ocorrer uma mudança significativa na distribuição dos dados

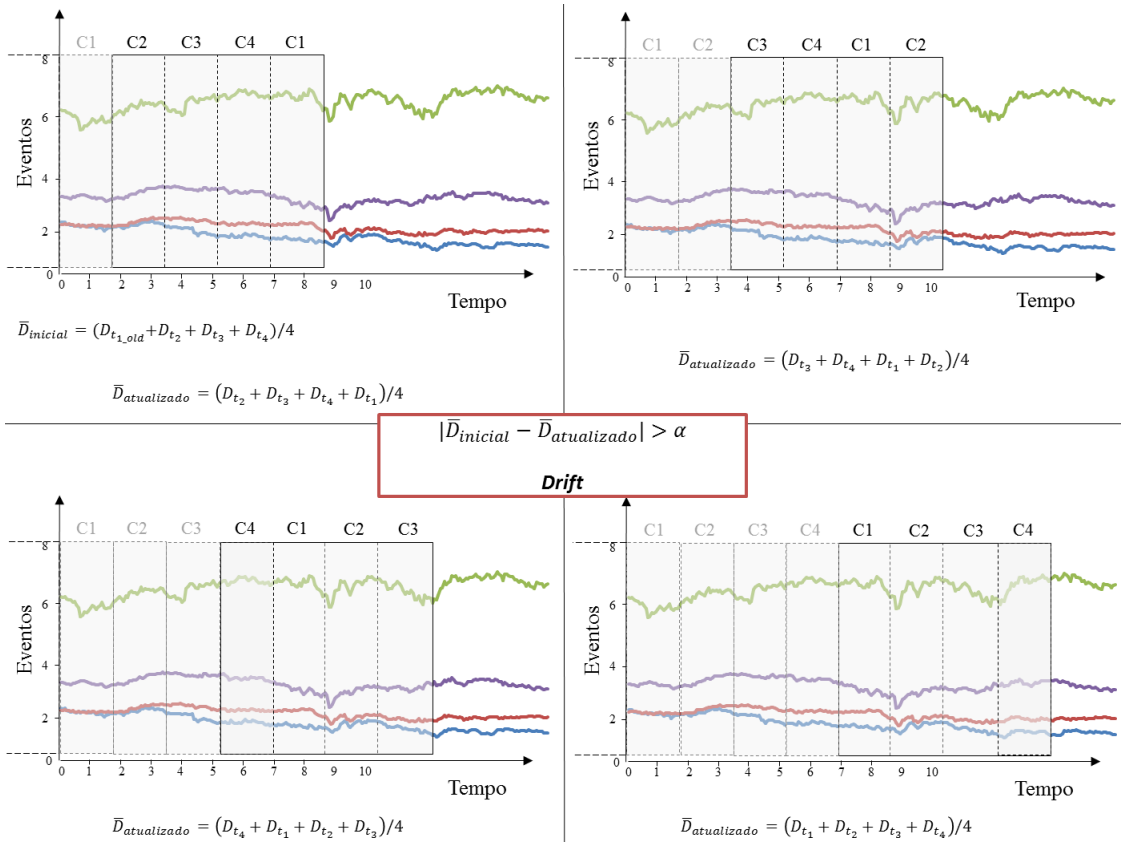


Figura 3.5: Detecção de mudança na distribuição, durante o período de aviso

Se a mudança for significativa, o algoritmo detecta a mudança na distribuição (Figura 3.3(c)), e entende que ocorreu o *concept drift*, continuando a monitorar os dados a partir da janela atual. O modelo que estava sendo construído até então é descartado, e uma nova árvore

de decisão é iniciada, com base nos dados mais recentes (Figura 3.3(d)).

Se depois de n_c períodos de contagem a diferença entre $D_{inicial}$ e $D_{atualizado}$ não for significativa, o algoritmo entende que foi um alarme falso, volta a seu estado normal (cancelando o período de aviso), e continua monitorando os dados.

Os valores ϵ e α são passados como parâmetros para o algoritmo, bem como o número de períodos de contagem n_c, e o número de pontos (exemplos) por período de contagem.

3.5 Análise Experimental

Nesta seção serão apresentados experimentos utilizando os métodos propostos neste trabalho de mestrado: o ST, o AST, o IST e o FDDM. Para os experimentos foram utilizadas diferentes bases de dados, com diferentes configurações, com ou sem ruídos, comparando os resultados com os algoritmos presentes na literatura VFDT, VFDTcNB e J48. O método FDDM foi comparado com os métodos apresentados na Seção 2.5, DDM e EDDM. Os nomes das bases de dados foram mantidos como foram originalmente obtidas, bem como o nome de seus atributos.

3.5.1 Bases de Dados e Ambiente de Trabalho

Os algoritmos propostos neste trabalho de mestrado foram implementados utilizando a linguagem de programação Java. Todas as implementações foram realizadas na ferramenta MOA (Massive Online Analisys)[1](BIFET, 2010), que possui uma série de algoritmos disponíveis para a análise de dados incrementais. Como medidas de interesse foram utilizadas a acurácia (média e final) , o tamanho da árvore gerada, o tempo de execução, a estatística Kappa, e o teste *t-Student*. Para determinar o resultado de cada valor da Kappa obtido, foi adotada a escala de Landis e Koch, apresentada em (EMAM, 1999) e exibida na Tabela 3.1.

Estatística Kappa	Nível de concordância
$< 0,00$	Pobre
$0,01 - 0,20$	Fraco
$0,21 - 0,40$	Razoável
$0,41 - 0,60$	Moderado
$0,61 - 0,80$	Substancial
$0,81 - 1,00$	Quase Perfeito

Tabela 3.1: Classificação dos resultados da Kappa de Landis e Koch, extraída de (EMAM, 1999)

Na fase de teste foi utilizada a validação prequential, também chamada de teste e treino

[1]Massive Online Analisys: http://moa.cms.waikato.ac.nz/

intercalados (PATIL; ATTAR, 2011), conforme descrito anteriormente na Seção 2.10. Todos os experimentos foram realizados em um computador de processador Intel i7 / 2.8GHz, 8GB de memória, considerando o parâmetro $\gamma = 0.9$ para todos os experimentos. As configurações não citadas na descrição dos experimentos foram utilizadas conforme o padrão disponível na MOA.

3.5.2 Experimentos com o ST e AST

Os algoritmos ST e AST foram aplicados em diversos conjuntos de dados, ao longo do desenvolvimento desse projeto de mestrado. A seguir são descritos quatro experimentos, sendo três com bases de dados reais e um com dados sintéticos. O conjunto de dados sintético *SEA* foi gerado utilizando a ferramenta MOA. Os conjuntos de dados reais utilizados foram o *Electricity*, *Sick Euthyroid* e o *Skin Segmentation*. Os resultados foram comparados com os dos algoritmos VFDT e VFDTcNB.

De acordo com (HULTEN; SPENCER; DOMINGOS, 2001) o conjunto SEA contém *concept drift* abrupto, três atributos, em que somente dois são relevantes, e duas classes (mais detalhes sobre essa base podem ser encontrados em (STREET; KIM, 2001)). Para os experimentos foram gerados 1 milhão de exemplos, e inseridos níveis de ruído nos dados utilizando a ferramenta MOA. O conjunto de dados *Electricity* foi obtido no site da MOA. Ele contém 45.312 exemplos coletados do mercado australiano de eletricidade de Nova Gales do Sul (Australian New South Wales Electricity Market), em que os preços não são fixos, e são afetados pela oferta e demanda do mercado. Os atributos *date* e *day* do conjunto original foram excluídos, e os dados do conjunto foram normalizados. O conjunto *Electricity*, após processado, contém 6 atributos numéricos com dois valores de classe, “up”e “down”. Para realizar os experimentos, foram inseridos níveis de ruído no atributo classe utilizando a ferramenta WEKA (WITTEN; FRANK; HALL, 2011). O conjunto *Sick Euthyroid* foi obtido no repositório da UCI. Para o experimento descrito foram considerados apenas os atributos numéricos *TSH*, *T3*, *TT4*, *T4U*, *FTI* e *TBG*. O conjunto de dados contém 3.163 exemplos com dois valores de classe, “sick-euthyroid”e “negative”. O conjunto *Skin Segmentation* também foi obtido no repositório da UCI e contém 245.057 exemplos, três atributos e dois valores de classe, “skin”e “non-skin”.

Nos experimentos utilizando o gerador *SEA*, o modelo foi testado a cada 50 mil exemplos. Utilizando a base Electricity, o modelo foi testado a cada 300 exemplos. Usando o conjunto de dados Sick Euthyroid, os algoritmos leram os exemplos 5 vezes, pois o conjunto de dados é relativamente menor, e o modelo foi testado a cada 150 exemplos, considerando o $n_{min} = 200$. As configurações do ST utilizadas são mostradas na Tabela 3.2. Esses valores foram obtidos após a realização de uma série de experimentos, com diferentes valores de $\Delta\mu_{min}$ e σ_{max}.

Parâmetros do ST		
Conjuntos de dados	$\Delta\mu_{min}$	σ_{max}
Electricity	0,014	0,36
SEA	0,014	0,36
Skin Segmentation	0,01	0,04
Sick Euthyroid	0,014	0,036

Tabela 3.2: Parâmetros do ST utilizados nos experimentos

As acurácias média e final, obtidas utilizando os conjuntos de dados Skin Segmentation e Sick Euthyroid, foram resumidas na Tabela 3.3. Na Figura 3.6 são apresentados as variações de acurácia obtidas com os dois conjuntos de dados. Em ambos os casos, os melhores resultados de acurácia foram alcançados pelo ST e AST, seguidos por, respectivamente, VFDT e VFDTcNB. Com o conjunto Skin Segmentation, o ST obteve a melhor variação de acurácia em praticamente todo o processo de classificação. O AST também descreveu bem os dados desde os primeiros exemplos, finalizando o processo com a segunda melhor acurácia. Embora o VFDT e VFDTcNB não tenham começado bem a classificação, depois dos primeiros 100 mil exemplos eles alcançaram resultados muito próximos aos do AST, finalizando a classificação quase iguais. Os resultados da estatística kappa foram quase perfeitos com todos os algoritmos. Os melhores resultados obtidos no início da classificação pelo ST e AST podem ser justificados devido ao fato de ambos utilizarem medidas estatísticas não dependentes do número de exemplos lidos, ao contrario dos outros algoritmos, VFDT e VFDTcNB. Por outro lado, usando a base Sick Euthyroid é visível a diferença entre os resultados do ST e AST (que não são dependentes do número de exemplos) com relação ao VFDT e VFDTcNB. Os valores de kappa obtidos pelo ST e AST foram quase perfeitos, enquanto os resultados do VFDT e VFDTcNB foram substanciais.

Resultados dos experimentos					
Conjuntos de dados		Ac. Média	Ac. Final	Variância	Kappa Final
Skin Segmentation	ST	99,68	99,9	0,235	99,69
	AST	99,07	99,4	0,193	98,13
	VFDT	98,08	99,3	4,03	97,83
	VFDTcNB	98,8	99,2	1,067	97,49
Sick Euthyroid	ST	98,22	98,1	0,103	89,67
	AST	97,85	97,9	0,379	88,97
	VFDT	95,57	95,8	0,52	79,38
	VFDTcNB	93,57	93,1	0,48	68,45

Tabela 3.3: Resultados dos experimentos utilizando os conjuntos de dados Skin Segmentation e Sick Euthyroid

Foi realizado um teste de significância para rejeitar a hipótese H_0 que a média das acurácias

dos algoritmos são estatisticamente iguais, com 95% de confiança. Na Tabela 3.4 são apresentados os resultados da execução do teste *t-Student* bicaudal entre as acurácias médias dos classificadores, utilizando as bases Skin Segmentation e Sick Euthyroid. O valor "1"indica que o algoritmo da linha tem uma acurácia estatisticamente superior ao da coluna, rejeitando H_0; "-1"indica que o algoritmo da linha tem um valor de acurácia estatisticamente inferior, em comparação ao da coluna, também rejeitando H_0; o valor "0"indica que os resultados dos dois algoritmos são estatisticamente iguais, e nesse caso H_0 não é rejeitado. Os resultados mostram que o ST obteve os melhores resultados, em comparação com os demais. Usando a base Skin Segmentation o AST obteve um resultado melhor que o VFDT, e um resultado estatisticamente igual ao do VFDTcNB. Finalmente, com a base Sick Euthyroid o AST alcançou resultados melhores estatisticamente que o VFDT e o VFDTcNB.

Teste de significância das acurácias usando o teste t-Student								
	Skin Segmentation				*Sick Euthyroid*			
	ST	AST	VFDT	VFDTcNB	ST	AST	VFDT	VFDTcNB
ST	0	1	1	1	0	1	1	1
AST	-1	0	1	0	-1	0	1	1
VFDT	-1	-1	0	0	-1	-1	0	1
VFDTcNB	-1	0	0	0	-1	-1	-1	0

Tabela 3.4: Resultados do teste de significância usando as bases Skin Segmentation e Sick Euthyroid datasets

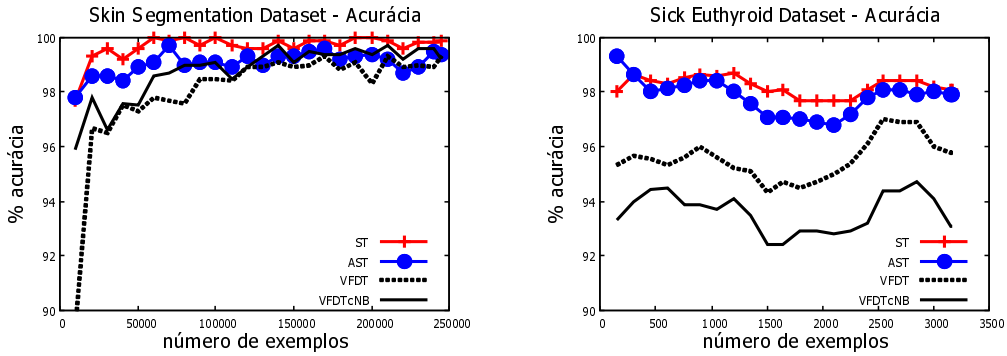


Figura 3.6: Resultados de acurácia utilizando os conjuntos de dados Skin Segmentation e Sick Euthyroid

Na Figura 3.7 são mostrados os tamanhos das árvores geradas pelos modelos utilizando as bases Skin Segmentation e Sick Euthyroid. Como é ilustrado, o ST e o AST construíram árvores maiores em ambos os experimentos. No entanto, o gráfico apresentado na Figura 3.8 mostra que o tempo de execução do ST e AST, utilizando essas duas bases foram próximos (quando não menores), que os tempos do VFDT e do VFDTcNB. Isso ocorreu porque o cálculo do ganho de informação utilizado pelos algoritmos VFDT e VFDTcNB, é mais computacionalmente

custoso do que a execução do critério de divisão do StARMiner. Assim, mesmo criando árvores maiores, a perda no tempo de execução não é tão grande, e o ganho na acurácia é notável.

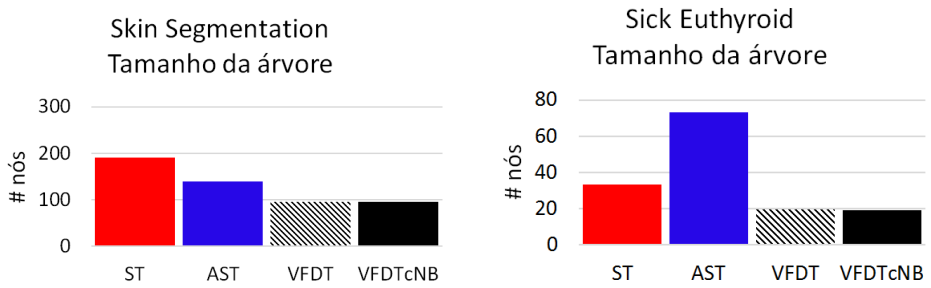


Figura 3.7: Tamanho das árvores geradas utilizando os conjuntos de dados Skin Segmentation e Sick Euthyroid

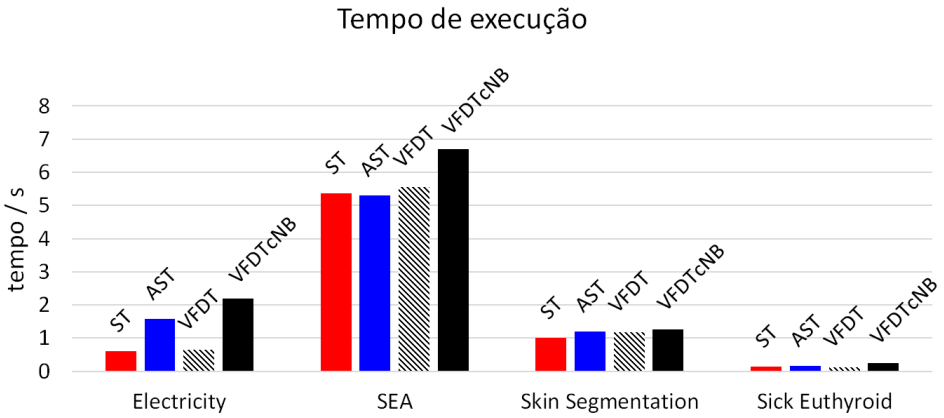


Figura 3.8: Tempo de execução dos algoritmos nos experimentos

Os resultados de acurácia obtidos pelos experimentos utilizando o gerador SEA e o conjunto Electricity são exibidos na Tabela 3.5 e na Figura 3.9, de acordo com o nível de ruído, em termos de acurácia média e final. Os resultados obtidos utilizando o gerador SEA mostram que o ST e AST atingiram a melhor acurácia nos dados contendo até 15% de ruído, e o VFDT obteve valores próximos ao ST e AST, com mais de 15% of ruído. Como é possível observar o AST obteve a melhor acurácia final e média na maior parte dos experimentos, independentemente do nível de ruído. O tempo médio de execução obtido por cada algoritmo nos experimentos são apresentados na Figura 3.8. Nos experimentos o algoritmo ST construiu árvores de decisão com média de 317 nós, o AST com 220 nós, o VFDT com 37 nós e o VFDTcNB com 38 nós. A criação de árvores maiores fez com que os métodos ST e AST obtivessem um maior tempo de execução, no entanto, o ganho na acurácia foi grande, em comparação aos demais algoritmos.

Utilizando a base Electricity, o AST obteve a melhor variação de acurácia. O VFDT e AST vieram em seguida, com o VFDT obtendo um melhor resultado nos níveis intermediários de ruído, e finalizando atrás do ST. Para essa base o ST criou árvores com média de 169 nós, seguido pelo AST (134 nós), e VFDT e VFDTcNB, com média 698 nós. O tempo de execução

dos algoritmos VFDT e VFDTcNB foram mais altos, pois criaram árvores de decisão maiores para essa base. As árvores do VFDT e VFDTcNB foram maiores devido à presença de *concept drift* (inserida nos dados pelo gerador da MOA).

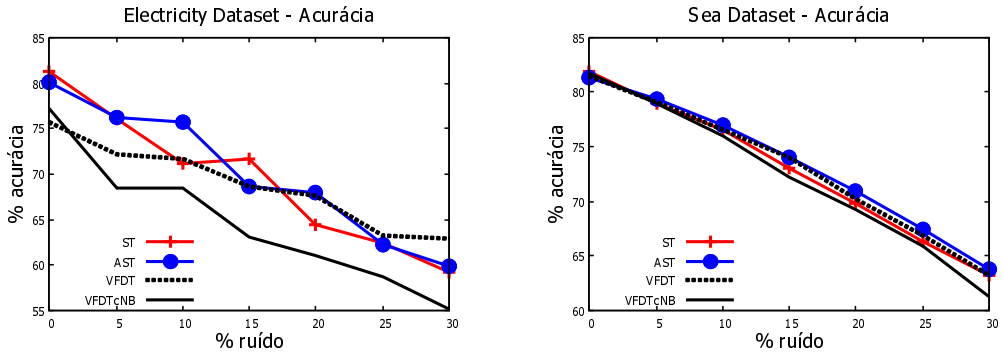
De forma geral, utilizando bases com ruídos é possível observar que a utilização do método estatístico pelos algoritmos ST e AST possibilitou a obtenção de melhores resultados em acurácia. A perda dos algoritmos propostos não foi tão atenuada, em comparação com o VFDT e o VFDTcNB, porque as medidas estatísticas utilizadas (com a média e variância) não foram tão afetadas, mostrando um comportamento tolerante a ruídos.

Resultados dos experimentos									
Base		Ac	***Ruído / %***						
			0	5	10	15	20	25	30
SEA	ST	M	81,26	78,24	75,19	71,99	68,99	65,86	62,68
		F	81,9	79,0	76,6	73,1	69,9	66,3	63,3
	AST	M	81,16	78,14	75,18	71,85	68,86	65,54	62,52
		F	81,4	79,4	77,0	74,0	71,0	67,5	63,8
	VFDT	M	80,99	78,1	75,16	71,94	68,8	65,71	62,7
		F	81,5	79,1	76,5	74,0	70,2	66,9	63,2
	VFDTcNB	M	80,68	77,76	74,62	71,26	68,41	64,93	61,73
		F	81,7	79,0	76	72,2	69,3	65,9	61,3
Electricity	ST	M	78,93	75,21	71,36	68,68	65,61	62,06	59,05
		F	81,3	76,0	71,2	71,6	64,5	62,4	59,3
	AST	M	79,15	75,71	72,2	69,54	66,06	63,23	60,0
		F	80,1	76,2	75,8	68,7	68,0	62,3	59,9
	VFDT	M	74,64	72,02	70,45	67,42	64,19	61,75	58,7
		F	75,8	72,2	71,06	68,7	67,6	63,2	62,9
	VFDTcNB	M	77,16	72,07	69,92	67,08	64,49	61,29	58,78
		F	77,3	68,4	64,1	63,1	61,1	58,8	55,2

Tabela 3.5: Experimentos utilizando o gerador SEA e o conjunto de dados Electricity, onde M é a acurácia média e F é a acurácia final

3.5.3 Experimentos com o IST

A ferramenta Weka (WITTEN; FRANK; HALL, 2011) foi utilizada para inserir níveis de ruído (de 0 a 40 porcento) nos valores do atributo classe do conjuntos de dados originais. O algoritmo IST foi aplicado em três conjuntos de dados, contendo atributos categóricos e numéricos (reais). Os parâmetros utilizados são apresentados na Tabela 3.6. Para a comparação dos resultados foram utilizados os algoritmos VFDT (DOMINGOS; HULTEN, 2000), VFDTcNB (GAMA; ROCHA; MEDAS, 2003) e J48 (uma versão de código aberto do C4.5 (QUINLAN, 1993)). As bases de dados utilizadas (*Mammography Mass*, *Indian Liver Patient Dataset* - ILPD e *Pima*



(a) (b)

Figura 3.9: Resultados de acurácia utilizando o conjunto de dados Electricity (a) e o gerador SEA (b)

Indians Diabetes) foram obtidas no repositório da UCI[2]. Para esses conjuntos de dados, com número de exemplos relativamente reduzido, os algoritmos foram aplicados mais de uma vez sobre os dados (número de passos), devido à característica conservadora do VFDT e VFDTcNB (com o uso do Hoeffding *bound*), que não construíram árvores com um único passo.

Conjunto de dados	n_{min}	Intervalo de teste	nº de passos
ILTP	30	30	2
Mammography mass	90	90	5
Pima Indians Diabetes	50	50	5

Tabela 3.6: Parâmetros utilizados nos experimentos utilizando as bases ILPD, Mammography e Pima Indians Diabetes

O conjunto de dados *Pima Indians Diabetes* contém 768 exemplos de pacientes do sexo feminino de pelo menos 21 anos da Pima Indian heritage. O conjunto inicial contém 9 atributos numéricos. O atributo "age"foi modificado, gerando um atributo categórico com faixa de valores "21_to_40_years"(com 574 exemplos), "41_to_60_years"(com 167 exemplos) e a faixa "81_to_90_years"(com 27 valores). Os demais atributos são: *number of times pregnant*, *plasma glucose concentration a 2 hours in an oral glucose tolerance test*, *diastolic blood pressure (mm Hg)*, *triceps skin fold thickness (mm)*, *2-hour serum insulin (mu U/ml)*, *body mass index (weight in kg/(height in m^2)*, e *diabetes pedigree function*. O atributo classe contém dois valores, "1"(com 500 exemplos) e "2"(com 268 exemplos).

Como é possível de observar na Figura 3.10, os resultados de acurácia final do IST foram substancialmente melhores que os demais, começando com 83,98% contra 75,91% do VFDTcNB, 75,65% do J48 e 74,83% of VFDT. Na Figura 3.10(b) são apresentados os resultados da kappa, que são similares aos resultados de acurácia. O IST começou com um resultado

[2] *UCI - Machine Learning Repository:* http://archieve.ics.uci.edu/ml

substancial de 64,29%, seguido por valores moderados do J48 com 44,44%, VFDTcNB com 43,41%, e VFDT com 39,51%. Com 20% de ruído o IST alcançou um resultado razoável de 39,74%, seguido por valores razoáveis do VFDTcNB com 20,68% e VFDT com 20,27%. O J48 obteve um valor fraco de kappa de 16,72%. Finalmente, o IST finalizou com um resultado razoável de 31,14%, contra valores fracos do VFDTcNB (6,8%) e VFDT (4,8%), e o resultado pobre do J48, com -3,85%. Conforme a Figura 3.10(c) a maior árvore foi criada pelo algoritmo IST, começando com 126 nós contra 110 do J48, e 5 do VFDT e VFDTcNB. Com 20% de ruído o IST criou um modelo com 133 nodes, J48 com 36, e VFDT e VFDTcNB com 5 nós. O IST finalizou com 131 nós, contra 16 do J48, e 3 do VFDT e VFDTcNB. A perda de acurácia do IST não foi tão acentuada como a dos demais algoritmos. Isso pode ser justificado devido ao fato de que o uso do observador StARMiner (contra o Ganho de Informação utilizado pelos demais métodos) não foi tão afetado pelos níveis de ruído.

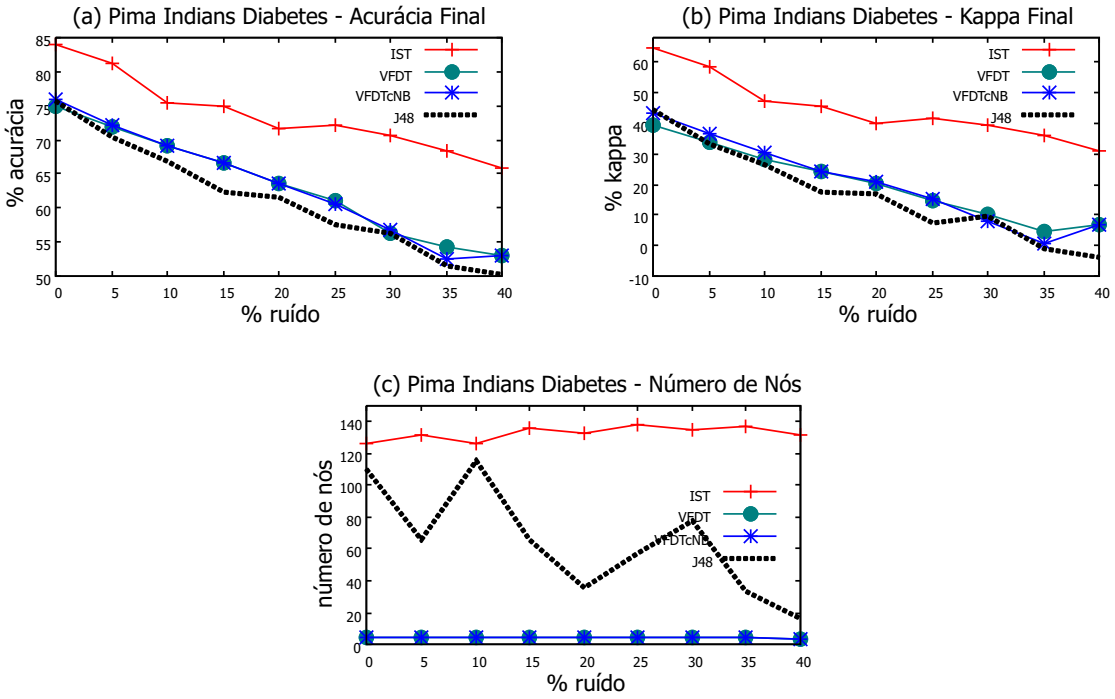


Figura 3.10: Resultados utilizando a base de dados Pima Indians Diabetes

O conjunto de dados Indian Liver Patient Dataset foi coletado do noroeste de Andhra Pradesh, Índia. O conjunto contém 583 exemplos, 11 atributos numéricos (*age*, *total bilirubin*, *direct bilirubin*, *alkaline phosphatase*, *alamine aminotransferase*, *aspartate aminotransferase*, *total proteins*, *albumin and albumin*, e *globulin ratio*), e 1 categórico (*gender of the patient*). O atributo classe contém 2 valores (416 "*liver patient records*"e 167 "*non liver patient records*").

Como é possível de observar na Figura 3.11 o IST obteve os melhores resultados de acurácia com a maioria dos níveis de ruído, alcançando 76,16% com 0% de ruído, 65,01% com 20%,

e 62,09% com 40%. O VFDT e VFDTcNB obtiveram resultados similares entre si, com o primeiro começando melhor com 71,36% contra 70,5%, empatando em 62,78% com 20% de ruído, e finalizando com 55,75% e 54,37% com 40% de ruído com o VFDT e VFDTcNB, respectivamente. Como é mostrado na Figura 3.11(b), os valores de kappa do IST, embora razoáveis, foram os melhores obtidos com todos os níveis de ruído, alcançando 38,88%, 21,75% e 22,74% com respectivamente 0, 20 e 40 porcento de ruído. O J48 começou com um valor fraco (16,32%), alcançando 17,42% com 20% de ruído e finalizando com um resultado pobre, com -5,65%. O VFDT e VFDTcNB obtiveram resultados razoáveis em todos os níveis de ruído na classificação.

Na Figura 3.11(c) é possível observar que o J48 iniciou com a maior árvore (99 nós), seguido pelo IST com 79 nós. O número de nós criado pelo IST continuou grande em todos os níveis de ruído, finalizando com 85 nós. O J48 criou árvores de tamanhos variados, finalizando com 17 nós. O VFDT e VFDTcNB criaram árvores para os níveis de 0 e 5 porcento de ruído (com 3 e 5 nós respectivamente). Nos demais níveis os algoritmos finalizaram o processo de classificação com somente 1 nó na árvore de decisão. Isso indica que ambos algoritmos não descreveram os dados de forma adequada.

A menor perda do IST pode ser justificada devido ao uso do observador estatístico StAR-Miner (para os atributos numéricos), baseado nos valores de média e variância dos exemplos observados, que não foi afetado tão severamente como ocorreu com o observador do Ganho de Informação.

O conjunto de dados Mammography Mass (ELTER; SCHULZ-WENDTLAND; WITTENBERG, 2007) é composto por 5 atributos (1 numérico e 4 nominais), e pode ser usado para predizer a gravidade (benigno ou malígno) de uma lesão de massa de mamografia dos atributos BI-RADIS e da idade do paciente. A base contém os seguintes atributos: *BI-RADS assessment*, *age*, e três atributos BI-RADS (*shape*, *margin* e *density*). O atribuo classe é a gravidade, que pode ser massa “benign”(com 516 exemplos) ou “malignant”(445 exemplos).

Na Figura 3.12(a) é possível verificar que todos os algoritmos apresentaram comportamentos similares em termos de acurácia e kappa. A melhor acurácia foi obtida pelo IST, seguido por VFDTcNB. O VFDT e J48 alternaram seus resultados, com o J48 finalizando com um melhor valor (56,29% contra 52,45% do VFDT). Os valores da kappa mostraram um comportamento similar aos resultados da acurácia, como é mostrado na Fig. 3.12(b). O IST e VFDTcNB iniciaram com valores substanciais, seguidos respectivamente pelo VFDT e J48 com resultados moderados. Com 20% de ruído o IST atingiu o único valor moderado (44,47%), seguido por resultados razoáveis do VFDTcNB (38,19%), J48 (34,34%) e VFDT (38,95%). No último nível

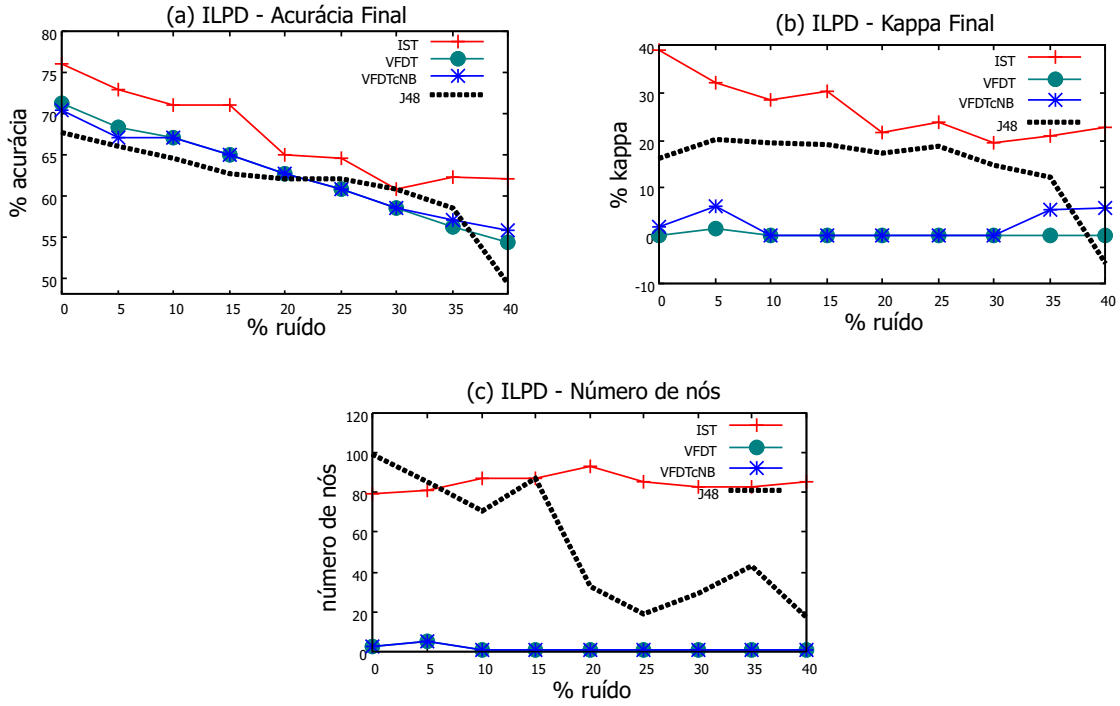


Figura 3.11: Resultados utilizando a base de dados ILPD

de ruído o IST foi melhor em termos de kappa, com um valor razoável de 23,39%, contra valores fracos do VFDTcNB (16,97%), J48 (12,58%) e VFDT (4,4%). De acordo com a Fig. 3.12(c) a maior árvore foi criada pelo J48 em quase todos os níveis de ruído. Com 0% o J48 criou 58 nós, o IST 55, e o VFDT e VFDTc criaram 10. Com 20% o J48 criou uma árvore com 56 nós, o IST com 48, e o VFDT e VFDTcNB com 10. No final, com 40% de ruído a maior árvore foi criada pelo IST (58 nós), seguido pelo J48 (52 nós), e VFDT e VFDTcNB (ambos com 6 nós). Comparando a diferença entre a acurácia do IST e dos demais algoritmos no início (com 0% de ruído) e no fim (com 40% de ruído), é possível observar que o uso do observador StARMiner ajudou o IST a alcançar os melhores resultados.

O IST obteve os melhores resultados nos três experimentos. No entanto, na maior parte dos cenários os algoritmo criou árvores de decisões maiores. A Figura 3.13 mostra o tempo de execução médio dos três algoritmos incrementais utilizados (IST, VFDT e VFDTcNB). O tempo de execução do J48 não foi incluído porque o algoritmo utilizado está implementado na ferramenta WEKA, enquanto os demais foram testados utilizando a ferramenta MOA. Assim, a diferença em implementação pode interferir no tempo de execução, impossibilitando uma comparação justa.

No experimento utilizando o conjunto de dados ILPD, o IST levou o maior tempo de execução, com 0,06 segundos contra 0,03 segundos do VFDTcNB e 0,02 segundos do VFDT.

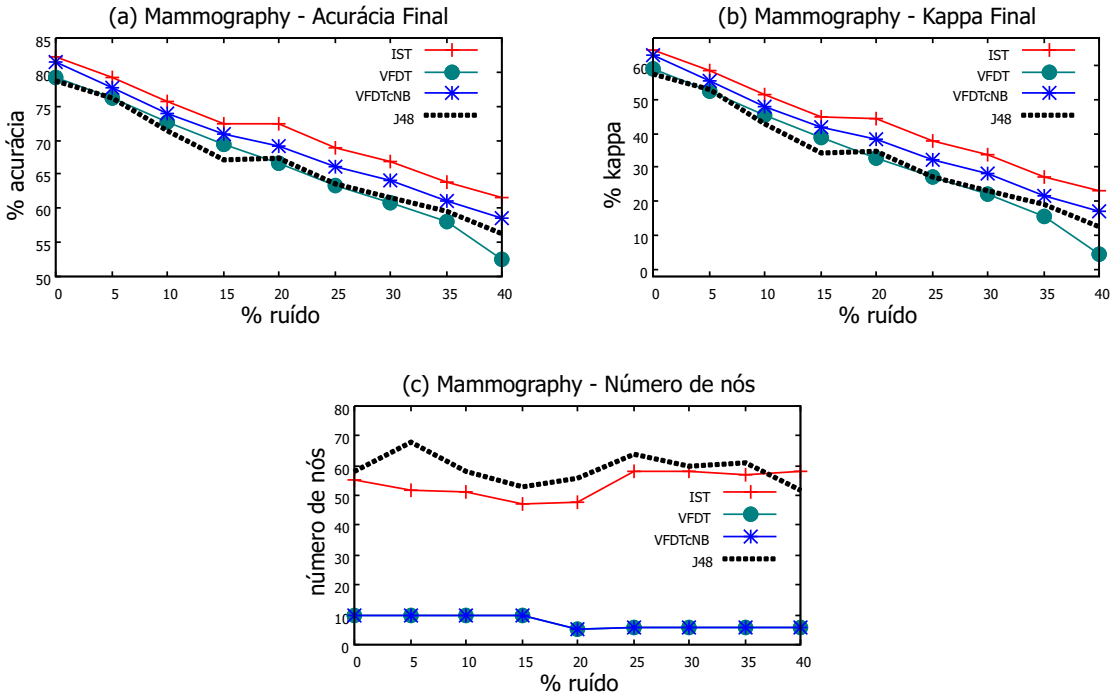


Figura 3.12: Resultados utilizando a base de dados Mammography

Essa ocorrência é justificada pelo tamanho das árvores construídas, com o IST obtendo mais nós que o VFDT e VFDTcNB. O VFDTcNB obteve maior tempo de execução que o VFDT devido aos testes extra realizados nas folhas (utilizando Naïve Bayes). No segundo experimento, utilizando o conjunto de dados Mammography Mass, o tempo de execução do IST foi o mesmo que o do VFDTcNB, com 0,02 segundos contra 0,009 do VFDT. Novamente, a diferença ocorreu porque IST criou uma árvore de decisão maior (no entanto, dessa vez a diferença no número de nós foi menor). Também é importante destacar que, apesar da árvore de decisão do IST ter sido maior, o VFDTcNB levou o mesmo tempo para criar sua árvore. Finalmente, no experimento utilizando o conjunto de dados Pima Indian Diabetes, o maior tempo de execução foi obtido pelo IST (0,03 segundos), seguido respectivamente pelo VFDTcNB (0,02 segundos) e o VFDT (0,01 segundo).

Ao que refere aos resultados obtidos de acordo com o nível de ruído, nos experimentos é possível observar que a perda de acurácia e kappa do IST foi, em geral, menor que os outros três algoritmos. Esse fato mostra uma característica tolerante a ruídos em comparação aos demais algoritmos.

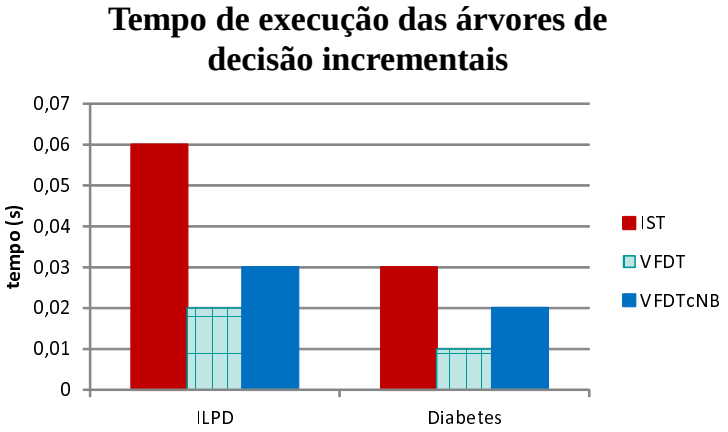


Figura 3.13: Tempo médio de execução dos algoritmos incrementais nos experimentos utilizando as bases ILPD, Mammography e Pima Indians Diabetes

3.5.4 Experimentos com FDDM

O algoritmo FDDM foi proposto para detectar mudanças significativas na distribuição dos dados. A seguir são apresentados dois experimentos utilizando o FDDM, comparando os resultados com os algoritmos da literatura DDM e EDDM, e com o desempenho do classificador sem detector. Os experimentos foram realizados considerando *concept drift* simulado, e com diferentes algoritmos de árvore de decisão. Na Tabela 3.7 são apresentados os parâmetros utilizados pelo FDDM, que foram obtidos por apresentar os melhores resultados em testes anteriores.

	Parâmetros do FDDM			
Experimento	α	ϵ	períodos de contagem	pontos por período de contagem
1	0,8	1,2	4	800
2	0,5	0,35	4	250

Tabela 3.7: Parâmetros utilizados pelo FDDM nos experimentos com *concept drift*

No primeiro experimento foram utilizados os geradores de dados Random Tree e Random RBF, ambos da ferramenta MOA, gerando dados com 5 atributos numéricos e 2 valores de classe. Foram gerados 1 milhão de exemplos, e o modelo foi testado a cada 10 mil exemplos. Inicialmente o classificador foi aplicado nos dados do Random Tree. A mudança na distribuição foi realizada da seguinte forma: ao atingir 480 mil exemplos, os dados do gerador Random RBF foram inseridos de forma gradual nos exemplos do Random Tree. Durante um intervalo de 20 mil exemplos (ou seja, dos exemplos 480 mil a 500 mil) os dados continham exemplos de ambos os geradores. Depois dos 500 mil, os dados consistiam apenas de exemplos do Random RBF. O VFDT foi o algoritmo de árvore de decisão aplicado nos dados.

Na Figura 3.14 são apresentados os resultados de acurácia obtidos aplicando o VFDT sem

detector de *concept drift*, com o FDDM, com o EDDM e com o DDM. Por utilizar o mesmo algoritmo para a construção do modelo, as variações de acurácia são muito próximas.

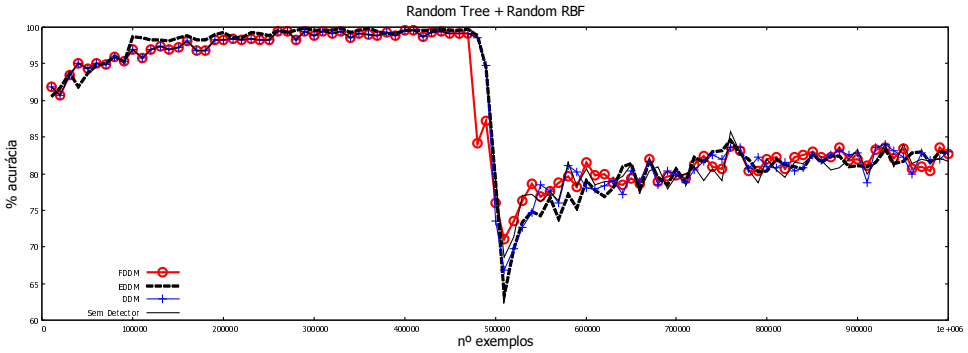


Figura 3.14: Resultado de acurácia utilizando os algoritmos de detecção de *concept drift* no primeiro experimento

Na Figura 3.15 é exibido com destaque o período em que a mudança na distribuição ocorre e quando os algoritmos detectaram a mudança (indicados pelas linhas verticais). Como é possível observar, o FDDM foi o primeiro algoritmo a detectar a ocorrência da mudança, aos 480 mil exemplos. O DDM detectou o *concept drift* primeiramente aos 500 mil exemplos. Aos 510 mil exemplos ele detectou novamente uma mudança, juntamente com o EDDM.

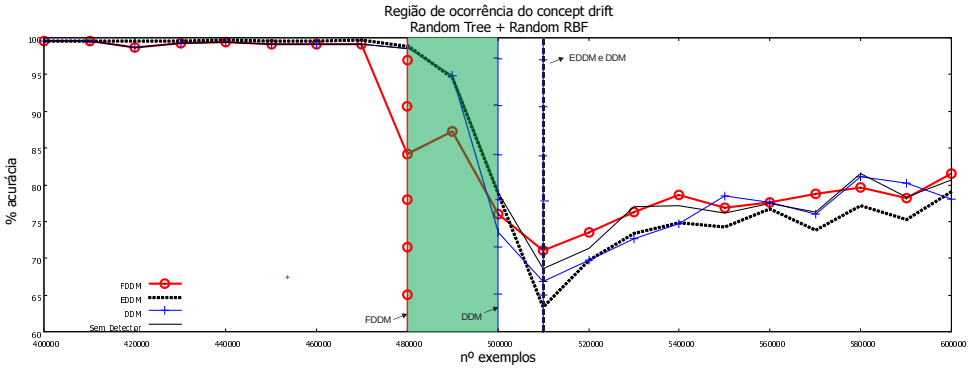


Figura 3.15: Região de ocorrência e detecções do *concept drift* no primeiro experimento

No segundo experimento foi utilizada a base de dados Skin Segmentation (descrita anteriormente) juntamente com o gerador de *streams* Random Tree (da ferramenta MOA, configurado para gerar 3 atributos e 2 valores de classe). Foram gerados 500 mil exemplos, e o modelo foi testado a cada 500 exemplos. A primeira parte da classificação foi realizada com dados da base Skin Segmentation, e a mudança na distribuição foi realizada da seguinte forma: ao atingir 240 mil exemplos, os dados do gerador Random Tree foram inseridos de forma abrupta nos exemplos da base Skin Segmentation. Durante um intervalo de 2.000 exemplos (ou seja, entre os exemplos 238 mil a 240 mil) os dados continham exemplos de ambos os geradores. Depois

dos 240 mil, os dados consistiam apenas de exemplos do Random Tree. O AST foi o algoritmo de árvore de decisão aplicado nos dados.

Na Figura 3.16 são exibidas as acurácias obtidas aplicando o AST sem detector de *concept drift*, com o FDDM, com o EDDM e com o DDM. Por utilizar o mesmo algoritmo para a construção do modelo, as variações de acurácia obtidas antes da mudança de distribuição foram muito próximas. O EDDM detectou *concept drift* de forma precipitada 9 vezes no início da classificação, nas marcas de 500, 1.000, 1.500, 2.000, 2.500, 3.000, 4.000, 4.500 e 5.000.

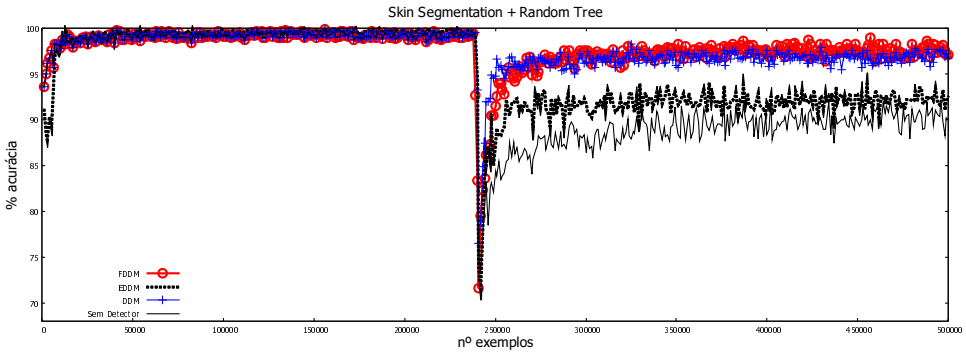


Figura 3.16: Resultado de acurácia utilizando os algoritmos de detecção de *concept drift* no segundo experimento

Na Figura 3.17 é exibido com destaque o período em que a mudança na distribuição ocorre e quando os algoritmos detectaram a mudança (indicados pelas linhas verticais). Como é possível observar, novamente o FDDM foi o primeiro algoritmo a detectar a ocorrência da mudança, ao atingir 240 mil exemplos. Logo em seguida, o FDDM detectou novamente uma mudança, juntamente com DDM, ao atingir 240.500 exemplos. Por último, o EDDM detectou uma mudança significativa na distribuição dos dados apenas no exemplo 241.500.

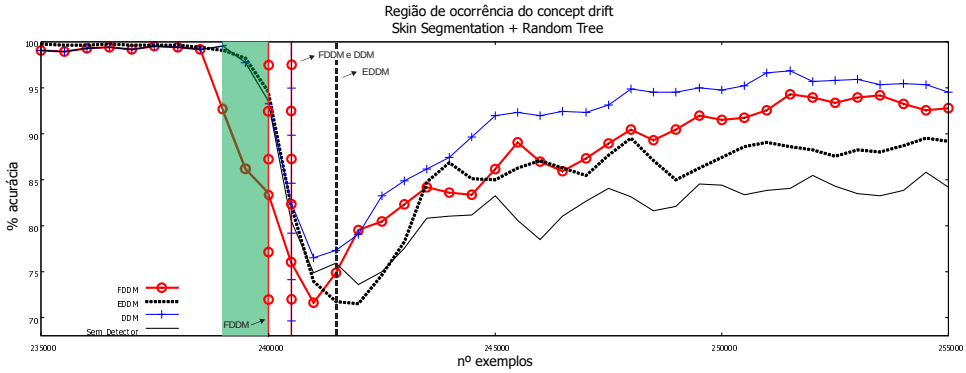
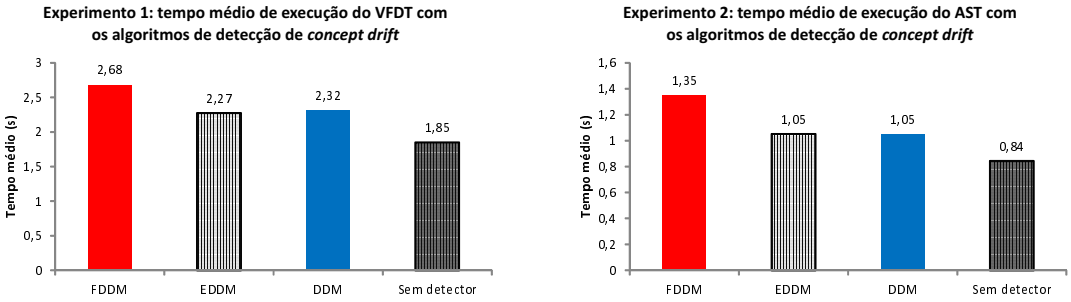


Figura 3.17: Região de ocorrência e detecções do *concept drift* no segundo experimento

Nos dois experimentos apresentados, o FDDM detectou a mudança na distribuição dos dados primeiro, comparando sua performance com os algoritmos EDDM e o DDM. Considerando

as mudanças gradual (experimento 1) e abrupta (experimento 2) na distribuição dos dados, o método proposto conseguiu se comportar bem nos dois cenários. Nesse caso, a utilização dos parâmetros do FDDM pode auxiliar na adequação do modelo classificador de acordo com a evolução dos dados. A utilização de um princípio baseado na teoria dos fractais permitiu que fosse realizada uma análise evolutiva dos dados, monitorando comportamento dos atributos ao longo do tempo e detectando quando houve uma mudança significativa na distribuição.

Na Figura 3.18 são exibidos o tempo médio de execução (em segundos) dos algoritmos. O tempo de execução do FDDM foi maior em ambos experimentos. Isso ocorre porque o algoritmo mantém uma janela com n_c contadores, cada um contendo n exemplos, fazendo com que o FDDM tenha que controlar esses contadores, atualizando o cálculo da dimensão fractal periodicamente e monitorando a diferença desses valores com o passar do tempo. O EDDM e o DDM obtiveram tempos de execução mais rápidos por atualizarem valores em memória (a taxa de erro e a distância de dois erros de classificação), o que é computacionalmente menos custoso, se comparado aos cálculos realizados pelo FDDM. Por fim, o método aplicado sem nenhum detector obteve o menor tempo de execução.



(a) (b)

Figura 3.18: Tempo médio de execução dos algoritmos no primeiro (a) e segundo (b) experimentos com *concept drift*

3.6 Considerações Finais

Neste capítulo foram apresentados os algoritmos de árvore de decisão ST, AST e IST, propostos neste projeto. Também foi descrito o método FDDM, proposto para detectar mudanças significativas na distribuição dos dados utilizando a teoria dos fractais, permitindo a adaptação do modelo sempre que preciso. Por fim, foram apresentados os experimentos realizados para validar os algoritmos propostos, comparando-os com algoritmos presentes na literatura.

Capítulo 4

CONCLUSÕES

Neste capítulo são apresentadas as conclusões finais deste trabalho de mestrado. As características dos métodos propostos são sumarizadas, descrevendo seus pontos positivos e limitações. Os trabalhos futuros são listados, e por fim é apresentada a conclusão geral do trabalho, apresentando também as publicações realizadas durante o desenvolvimento do projeto.

4.1 Pontos positivos e Limitações

As características dos algoritmos de árvore de decisão StARMiner Tree, Automatic StARMiner Tree e Information Gain StARMiner Tree, desenvolvidos neste trabalho, estão sumarizadas na Figura 4.1. De forma geral, os algoritmos propostos apresentaram resultados melhores termos de acurácia em comparação com algoritmos presentes na literatura. O tempo de execução foi menor na construção da árvore de decisão, pois o cálculo da heurística de divisão do nó proposta (baseada em estatísticas) é mais rápida em comparação com o Ganho de Informação (utilizados pelo VFDT e VFDTcNB). No entanto, por criar árvores maiores, em alguns casos o algoritmo gastou um tempo maior para a classificação dos exemplos, em comparação com o VFDT e VFDTcNB. Outra limitação dos métodos propostos é que eles não possuem garantias teóricas, ao contrário dos algoritmos que utilizam o Hoeffding *bound* para decidir quando dividir um nó folha da árvore.

Na presença de ruído nos dados, a perda de acurácia dos classificadores propostos não foi tão acentuada como ocorreu com os demais algoritmos utilizados na comparação. Isso pode ser justificado porque as medidas estatísticas utilizadas (relacionados à média e variância) não foram tão afetadas, em comparação com os princípios utilizados pelos outros algoritmos, que utilizam a entropia, o Hoeffding *bound* e outras medidas.

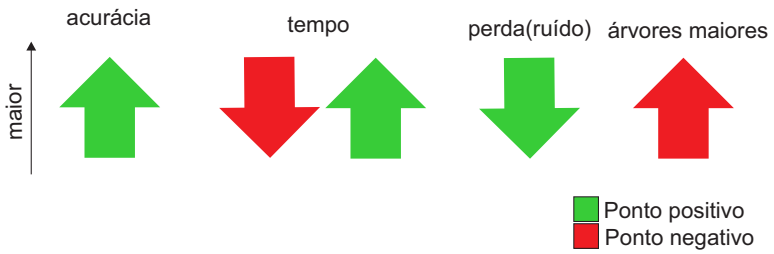


Figura 4.1: Sumarização das características dos algoritmos de árvore de decisão propostos

O método FDDM (Fractal Drift Detection Method), proposto para a detecção de *concept drift* nos dados, tem como característica detectar mudanças graduais e abruptas na distribuição dos dados, considerando a definição dos valores dos parâmetros de entrada do algoritmo. Os experimentos apresentados no Capítulo 3 mostraram que o FDDM consegue detectar as mudanças adequadamente, utilizando a dimensão de correlação fractal entre os atributos, de forma não supervisionada. As limitações do método diz respeito ao uso da memória e ao tempo de execução, que são maiores em comparação aos métodos DDM e EDDM. Isso ocorre porque o algoritmo mantém uma janela de dados com os contadores, de modo a monitorar a dimensão fractal dos últimos exemplos lidos.

4.2 Trabalhos Futuros

No decorrer do desenvolvimento deste projeto de mestrado foram observadas, dentre outras, as seguintes oportunidades para o desenvolvimento de trabalhos futuros:

- Inclusão de um método para a poda da árvore de decisão, sempre que os nós ficarem obsoletos ou irrelevantes no modelo.
- Trabalhar com técnicas de classificação espaço-temporal, utilizando técnicas de árvores de decisão incrementais aplicadas ao contexto de *data streams*.
- Estudo do resultado semântico da árvore de decisão gerada dentro de um contexto.
- Aplicação dos métodos de árvore de decisão em um contexto específico, de forma a identificar pontos positivos do mesmo, e características a serem otimizadas para ajudar na tomada de decisão.
- Estudar e propor o cálculo automático dos parâmetros ϵ e α do algoritmo FDDM, além do controle automático do tamanho da janela.
- Estudar e propor uma versão supervisionada do FDDM, analisando a dimensão fractal de cada classe separadamente.

4.3 Conclusão Geral

A árvore de decisão foi utilizada neste projeto por ser uma popular forma de representação do modelo classificador, frequentemente utilizada em diversas áreas, autoexplicativa, rápida de construir, e que geralmente possui alta acurácia. Durante este projeto de mestrado, os métodos e algoritmos apresentados no Capítulo 3 foram desenvolvidos no intuito de suprir limitações dos trabalhos descritos na literatura.

Na abordagem incremental, as técnicas existentes geralmente apresentam um custo computacional alto para a construção e atualização do modelo, principalmente no que se refere ao cálculo efetuado para a decisão de divisão dos nós. Os métodos possuem uma característica conservadora a quantidades de dados limitadas (tendendo a melhorar conforme o número de exemplos aumenta). Além disso, em muitas aplicações reais, os dados são gerados com ruídos, e as técnicas existentes possuem baixa tolerância a essas ocorrências, não se comportando bem na descrição dos dados. Neste trabalho, foram propostos algoritmos que utilizam uma heurística de divisão dos nós baseada em estatísticas, que mostrou-se rápida, e que não é dependente do número de exemplos lidos. Os métodos propostos mostraram um comportamento tolerante na classificação de dados ruidosos. Finalmente, foi proposto um método para a detecção de mudanças no comportamento dos dados baseado na teoria dos fractais, fazendo com que o modelo seja atualizado sempre que o mesmo não descrever os dados atuais (se tornar obsoleto).

A seguir são descritos os métodos propostos neste trabalho, de forma sucinta:

- StARMiner Tree (ST): Algoritmo de árvore de decisão que recebe como entrada três parâmetros ($\Delta\mu_{min}$, σ_{max} e γ_{min}). O ST apresenta ganhos de resultado em termos de acurácia, além de apresentar uma heurística de divisão dos nós mais rápida que o Ganho de Informação;

- Automatic StARMiner Tree (AST): Algoritmo de árvore de decisão com o cálculo automático dos parâmetros de entrada. O AST apresenta as mesmas características do ST, porém com a parametrização automática, o que facilita a utilização do algoritmo pelos usuários.

- Information Gain StARMiner Tree (IST): Algoritmo de árvore de decisão com um módulo adicional, capaz de descrever dados numéricos (reais) e categóricos. O IST possui dois observadores, um para dados numéricos (observador StARMiner) e um para dados categóricos (observador Ganho de Informação). O observador StARMiner pode ser utilizado com ou sem a parametrização automática. O IST apresenta ganho em termos de

acurácia, em comparação com algoritmos presentes na literatura, e possui uma maior tolerância a dados com ruído;

- Fractal Drift Detection Method (FDDM): Método de detecção de *drift* nos dados por meio da teoria dos fractais. O FDDM pode ser utilizado independentemente do algoritmo de árvore de decisão (funcionando com o ST, AST e VFDT, por exemplo), e apresenta resultados significativos na detecção de *concept drift* (abrupto e gradual) nos dados.

Durante o desenvolvimento deste projeto os resultados obtidos foram divulgados nos seguintes trabalhos:

- CAZZOLATO, M.T.; RIBEIRO, M.X.. **A Statistical Decision Tree Algorithm for Medical Data Stream Mining.** In: 26th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2013), 2013, Porto, Portugal. Los Alamitos: IEEE press, 2013, p.1-4.

- CAZZOLATO, M.T.; RIBEIRO, M.X. YAGUINUMA, C.A.; SANTOS, M.T.P.. **A Statistical Decision Tree Algorithm for Data Stream Classification.** In: 15th International Conference on Enterprise Information Systems (ICEIS 2013), 2013, Angers Loire Valey, France. Berlin: Springer, 2013. p.1-7.

- CAZZOLATO, M.T.; RIBEIRO, M.X. **A Statistical Decision Tree Algorithm Applied on Noisy Data Streams.** In: Proceedings of KDMiLe - Symposium on Knowledge Discovery, Mining and Learning", ISSN 2318-1060. (KDMiLe 2013), 2013, São Carlos, Brazil. p.1-8.

- CAZZOLATO, M.T.; RIBEIRO, M.X.. **Classifying High-Speed Data Streams Using Statistical Decision Trees.** In: Journal of Information and Data Management (JIDM) - Special Issue KDMiLe. Submetido em Nov/2013, aceito para publicação em Maio/2014. Brazil, 2014. p.1-10.

REFERÊNCIAS BIBLIOGRÁFICAS

BABCOCK, B. et al. Models and issues in data stream systems. In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM, 2002. (PODS '02), p. 1–16. ISBN 1-58113-507-6. Disponível em: <http://doi.acm.org/10.1145/543613.543615>.

BAENA-GARCÍA, M. et al. Early drift detection method. In: *In Fourth International Workshop on Knowledge Discovery from Data Streams*. [S.l.: s.n.], 2006.

BIFET, A. Adaptive stream mining: Pattern learning and mining from evolving data streams. In: *Proceedings of the 2010 conference on Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2010. p. 1–212. ISBN 978-1-60750-090-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1735125.1735127>.

CHAN, T. F.; LEWIS, J. G. Computing standard deviations: accuracy. *Commun. ACM*, ACM, New York, NY, USA, v. 22, n. 9, p. 526–531, set. 1979. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/359146.359152>.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2000. (KDD '00), p. 71–80. ISBN 1-58113-233-6. Disponível em: <http://doi.acm.org/10.1145/347090.347107>.

ELTER, M.; SCHULZ-WENDTLAND, R.; WITTENBERG, T. The prediction of breast cancer biopsy outcomes using two cad approaches that both emphasize an intelligible decision process. *Medical Physics*, v. 34, n. 11, p. 4164–72, 2007.

EMAM, K. E. Benchmarking kappa: Interrater agreement in software processassessments. *Empirical Softw. Engg.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 4, n. 2, p. 113–133, jun. 1999. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1023/A:1009820201126>.

FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P. (Ed.). *From data mining to knowledge discovery: an overview*. Menlo Park, CA, USA: Advances in knowledge discovery and data mining: American Association for Artificial Intelligence, 1996. ISBN 0-262-56097-6.

GAMA, J. *Knowledge Discovery from Data Streams*. 1st. ed. [S.l.]: Chapman & Hall/CRC, 2010. ISBN 1439826110, 9781439826119.

GAMA, J.; GABER, M. *Learning from Data Streams: Processing Techniques in Sensor Networks*. [S.l.]: Springer, 2007. (New generation computing). ISBN 9783540736783.

GAMA, J. et al. Learning with drift detection. In: BAZZAN, A.; LABIDI, S. (Ed.). *Advances in Artificial Intelligence SBIA 2004*. Springer Berlin Heidelberg, 2004, (Lecture Notes in Computer Science, v. 3171). p. 286–295. ISBN 978-3-540-23237-7. Disponível em: <http://dx.doi.org/10.1007/978-3-540-28645-5 29>.

GAMA, J. a.; ROCHA, R.; MEDAS, P. Accurate decision trees for mining high-speed data streams. In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2003. (KDD '03), p. 523–528. ISBN 1-58113-737-0. Disponível em: <http://doi.acm.org/10.1145/956750.956813>.

HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques: Concepts and Techniques*. [S.l.]: Elsevier Science, 2011. (The Morgan Kaufmann Series in Data Management Systems). ISBN 9780123814807.

HAYAT, M.; HASHEMI, M. A dct based approach for detecting novelty and concept drift in data streams. In: *Soft Computing and Pattern Recognition (SoCPaR), 2010 International Conference of*. [S.l.: s.n.], 2010. p. 373–378.

HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2001. (KDD '01), p. 97–106. ISBN 1-58113-391-X. Disponível em: <http://doi.acm.org/10.1145/502512.502529>.

LI, C.; ZHANG, Y.; LI, X. Ocvfdt: one-class very fast decision tree for one-class classification of data streams. In: *Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data*. New York, NY, USA: ACM, 2009. (SensorKDD '09), p. 79–86. ISBN 978-1-60558-668-7. Disponível em: <http://doi.acm.org/10.1145/1601966.1601981>.

PAGE, E. S. Continuous inspection schemes. *Biometrika*, v. 41, n. 1-2, p. 100–115, 1954.

PATIL, A.; ATTAR, V. Framework for performance comparison of classifiers. In: DEEP, K. et al. (Ed.). *Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011) December 20-22, 2011*. Springer India, 2011, (Advances in Intelligent and Soft Computing, v. 131). p. 681–689. ISBN 978-81-322-0490-9. Disponível em: <http://dx.doi.org/10.1007/978-81-322-0491-6 62>.

PATIL, P. D.; KULKARNI, P. Adaptive supervised learning model for training set selection under concept drift data streams. In: *Cloud Ubiquitous Computing Emerging Technologies (CUBE), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 36–41.

PFAHRINGER, B.; HOLMES, G.; KIRKBY, R. Handling numeric attributes in hoeffding trees. In: *Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining*. Berlin, Heidelberg: Springer-Verlag, 2008. (PAKDD'08), p. 296–307. ISBN 3-540-68124-8, 978-3-540-68124-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1786574.1786604>.

QUINLAN, J. R. Induction of decision trees. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 1, n. 1, p. 81–106, mar. 1986. ISSN 0885-6125. Disponível em: <http://dx.doi.org/10.1023/A:1022643204877>.

QUINLAN, J. R. *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN 1-55860-238-0.

REHMAN, M. Z.-U.; LI, T.-r.; LI, T. Exploiting empirical variance for data stream classification. *Journal of Shanghai Jiaotong University (Science)*, Shanghai Jiaotong University Press, v. 17, p. 245–250, 2012. ISSN 1007-1172. Disponível em: <http://dx.doi.org/10.1007/s12204-012-1261-5>.

RIBEIRO, M. X. *Suporte a Sistemas de Auxílio ao Diagnóstico e de Recuperação de Imagens por Conteúdo Usando Mineração de Regras de Associação*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação – ICMC, USP, São Carlos, 2008.

RIBEIRO, M. X. et al. Mining statistical association rules to select the most relevant medical image features. IEEE Computer Society - First International Workshop on Mining Complex Data (IEEE MCD'05), Houston, USA, p. 91–98, 2005.

ROKACH, L.; MAIMON, O. *Data Mining with Decision Trees: Theory and Applications*. [S.l.]: World Scientific Publishing Company, Incorporated, 2008. (Series in Machine Perception and Artificial Intelligence). ISBN 9789812771728.

SOUSA, E. P. M. D.; RIBEIRO, M. X.; TRAINA, A. J. M. Tracking the intrinsic dimension of evolving data streams to update association rules. In: *3rd International Workshop on Knowledge Discovery from Data Streams, part of 23th International Conference on Machine Learning (ICML06)*. Pittsburgh,PA: [s.n.], 2006.

SOUSA, E. P. M. de. *Identificação de Correlações Usando a Teoria dos Fractais*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação – ICMC, USP, São Carlos, 2006.

SOUSA, E. P. M. de et al. Evaluating the intrinsic dimension of evolving data streams. In: *Proceedings of the 2006 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2006. (SAC '06), p. 643–648. ISBN 1-59593-108-2. Disponível em: <http://doi.acm.org/10.1145/1141277.1141426>.

STREET, W. N.; KIM, Y. A streaming ensemble algorithm (sea) for large-scale classification. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2001. (KDD '01), p. 377–382. ISBN 1-58113-391-X. Disponível em: <http://doi.acm.org/10.1145/502512.502568>.

TRAINA-JR., C. et al. Fast feature selection using fractal dimension. In: *Journal of Information and Data Management - JIDM*. [S.l.: s.n.], 2010. p. 3–16.

WATANABE, C. et al. A statistical associative classifier with automatic estimation of parameters on computer aided diagnosis. In: *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*. [S.l.: s.n.], 2012. v. 1, p. 564–567.

WELFORD, B. P. Note on a method for calculating corrected sums of squares and products. *Technometrics*, v. 4, n. 3, p. 419–420, 1962. Disponível em: <http://www.tandfonline.com/doi/abs/10.1080/00401706.1962.10490022>.

WITTEN, I.; FRANK, E.; HALL, M. *Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques*. [S.l.]: Elsevier Science, 2011. (The Morgan Kaufmann Series in Data Management Systems). ISBN 9780080890364.

YANG, H.; FONG, S. Optimized very fast decision tree with balanced classification accuracy and compact tree size. In: *Data Mining and Intelligent Information Technology Applications (ICMiA), 2011 3rd International Conference on*. [S.l.: s.n.], 2011. p. 57 –64.

YANG, H.; FONG, S.; SI, Y.-W. Multi-objective optimization for incremental decision tree learning. In: *Proceedings of the 14th international conference on Data Warehousing and Knowledge Discovery*. Berlin, Heidelberg: Springer-Verlag, 2012. (DaWaK'12), p. 217–228. ISBN 978-3-642-32583-0. Disponível em: <http://dx.doi.org/10.1007/978-3-642-32584-718>.

ZEILEIS, A. Alternative boundaries for cusum tests. *Statistical Papers*, Springer-Verlag, v. 45, n. 1, p. 123–131, 2004. ISSN 0932-5026. Disponível em: <http://dx.doi.org/10.1007/BF02778274>.

GLOSSÁRIO

AST – *Automatic StARMiner Tree*

CUSUM – *Cumulative Sum*

DDM – *Drift Detection Method*

EDDM – *Early Drift Detection Method*

FDDM – *Fractal Drift Detection Method*

HB – *Hoeffding bound*

IST – *Information Gain StARMiner Tree*

KDD – *Knowledge Discovery in Database*

MOA – *Massive Online Analysis*

SID-Meter – *data Stream Intrinsec Dimension meter*

ST – *StARMiner Tree*

StARMiner – *Statistical Association Rule Miner*

VFDT – *Very Fast Decision Tree*