UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MINERAÇÃO DE PREFERÊNCIAS CONTEXTUAIS EM *DATA STREAMS*

JAQUELINE APARECIDA JORGE PAPINI

Uberlândia - Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JAQUELINE APARECIDA JORGE PAPINI

MINERAÇÃO DE PREFERÊNCIAS CONTEXTUAIS EM *DATA STREAMS*

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Banco de Dados.

Orientadora:
Prof[a]. Dr[a]. Sandra Aparecida de Amo

Uberlândia, Minas Gerais
2014

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU

P217m Papini, Jaqueline Aparecida Jorge, 1986-
Mineração de preferências contextuais em data streams / Jaqueline Aparecida Jorge Papini. - 2014.
123 p. : il.

Orientadora: Sandra Aparecida de Amo.
Dissertação (mestrado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Mineração de dados (Computação) - Teses. 3. Algoritmos de computador - Teses. I. Amo, Sandra Aparecida de. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Mineração de Preferências Contextuais em *Data Streams***" por **Jaqueline Aparecida Jorge Papini** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 27 de Fevereiro de 2014

Orientadora:

Profª. Drª. Sandra Aparecida de Amo
Universidade Federal de Uberlândia

Banca Examinadora:

Profª. Drª. Maria Camila Nardini Barioni
Universidade Federal de Uberlândia

Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho
Universidade de São Paulo

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Fevereiro de 2014

Autor:	**Jaqueline Aparecida Jorge Papini**
Título:	**Mineração de Preferências Contextuais em *Data Streams***
Faculdade:	**Faculdade de Computação**
Grau:	**Mestrado**

Fica garantido à Universidade Federal de Uberlândia o direito de circulação e impressão de cópias deste documento para propósitos exclusivamente acadêmicos, desde que o autor seja devidamente informado.

_____

Autor

Dedicatória

Aos meus pais, Latif e William, e ao meu irmão Juninho.

Ao meu amado marido Allan.

E, principalmente, à minha querida avó Nair, que apesar de já ter ido morar no céu, nunca deixará de estar presente em minha vida, abençoando-me e ajudando-me em tudo o que eu precisar – sempre, a minha amada vovó, grande amiga e segunda mãe.

Agradecimentos

A Deus,
por sempre ter me dado força nos momentos mais difíceis.

À Profª. Sandra,
minha orientadora e conselheira, pelas oportunidades e desafios que me proporcionou, pelos seus ensinamentos, pela paciência, pela amizade, por seu incentivo e, principalmente, por sua confiança. Pode ter certeza, Sandra, que levarei seus ensinamentos pela minha vida.

Ao meu amado marido Allan,
pelo seu amor e apoio incondicional durante todo o meu mestrado. Você esteve presente todos os dias, nos meus altos e baixos, sempre clareando os meus pensamentos, me ajudando nas coisas mais difíceis e me dando força para seguir em frente. Nunca vou esquecer o que você fez por mim durante todo este período. Uma verdadeira prova de amor!

Aos meus pais, Latif e William,
pelo carinho e amor em todos os momentos da minha vida. Sem dúvida vocês são pessoas que enchem meu coração de orgulho pelo caráter, força e hombridade com que permeiam suas vidas. Vocês são verdadeiros exemplos para mim.

Ao meu amado irmão William Júnior,
por sempre estar presente em minha vida, tanto nos momentos alegres quanto nos mais difíceis. Aprendi muito com você durante o tempo que moramos juntos em Uberlândia. Você é um irmão maravilhoso. Sempre serei agradecida a Deus por Ele ter me dado você.

A toda a minha família e à família do Allan,
pelo carinho e pela compreensão das muitas viagens não ocorridas a São José do Rio Preto e Itumbiara. Prometo que agora vou reservar um tempo sagrado para fazer isso.

Aos meus amigos de laboratório,
pela excelente convivência, pelos conselhos, pela prezada amizade, e pelos descontraídos cafés da tarde. Agradeço a todos de coração. Em especial, à Juliete, ao Guilherme, ao Lucas, ao Marcos, à Joicy e à Cleiane, por participarem intensamente dos meus dias de mestrado, e por torná-los mais alegres e felizes. Sempre serei grata pela força que vocês me deram durante todo o mestrado.

"O futuro pertence àqueles que acreditam na beleza de seus sonhos."
(Eleanor Roosevelt)

Resumo

O cenário tradicional de mineração de preferências, referido aqui como cenário *batch*, tem sido amplamente estudado na literatura nos últimos anos. Entretanto, a natureza dinâmica do problema de mineração de preferências cada vez mais requer soluções que rapidamente se adaptam a mudanças. A principal razão para isto é que normalmente as preferências do usuário não são estáticas e podem evoluir sobre o tempo. No trabalho descrito nesta dissertação, é abordado o problema de mineração de *preferências contextuais* no cenário de *data stream*. Preferências Contextuais têm sido recentemente tratadas na literatura e alguns métodos para minerar este tipo especial de preferências têm sido propostos no cenário *batch*. Como principais contribuições do trabalho descrito nesta dissertação, o problema de mineração de preferências contextuais no cenário de *data stream* é formalizado e são propostos três algoritmos para resolver este problema. Em adicional, também foi proposto um formalismo sobre *concept drift* em preferências contextuais. Dois dos algoritmos propostos foram implementados e a eficiência destes foi mostrada através de um conjunto extenso de experimentos sobre dados reais e sintéticos (com e sem a introdução de *concept drift*).

Palavras chave: mineração de preferências, *data streams*, algoritmos incrementais, redes bayesianas, *concept drift*, ciência de contexto.

Abstract

The traditional preference mining setting, referred to here as the *batch setting*, has been widely studied in the literature in recent years. However, the dynamic nature of the problem of mining preferences increasingly requires solutions that quickly adapt to change. The main reason for this is that frequently user's preferences are not static and can evolve over time. In the work described in this master's thesis, we address the problem of mining *contextual preferences* in the data stream setting. Contextual Preferences have been recently treated in the literature and some methods for mining this special kind of preferences have been proposed in the batch setting. As main contributions of the work described in this master's thesis, we formalize the contextual preference mining problem in the data stream setting and propose three algorithms for solving this problem. In addition, we also propose a formalism about concept drift in contextual preferences. We have implemented two of the proposed algorithms and showed their efficiency through an extensive set of experiments over real and synthetic data (with and without concept drift).

Keywords: preference mining, data streams, incremental algorithms, bayesian networks, concept drift, context-awareness.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

acc	Acurácia
cp-rules	*Contextual Preference Rules*
FPSMining	*Fast Preference Stream Mining*
H0	Hipótese Nula
HA	Hipótese Alternativa
HT	*Hoeffding Tree*
IGA	*Incremental Genetic Algorithm*
IGAPSMining	*Incremental Genetic Algorithm for Preference Stream Mining*
IncFPSMining	*Incremental Fast Preference Stream Mining*
k	mil
m	milhões
MOA	*Massive Online Analysis*
NB	*Naive Bayes*
prec	Precisão
Prequential	*Predictive Sequential*
RBP	Rede Bayesiana de Preferências
s	segundos
TC	Taxa de Comparabilidade

Capítulo 1

Introdução

A cada dia, descobrir os gostos e as preferências dos usuários tem se tornado mais desafiador. Antigamente, o usuário não tinha muita possibilidade de escolha e precisava se contentar com o pouco que lhe era oferecido pelas aplicações. Enquanto isso, na sociedade atual, a flexibilidade de escolha e customização é muito valorizada nas mais diversas áreas. Além disso, a diversidade de informações com as quais o usuário tem acesso diariamente contribui para que suas preferências sofram várias mudanças e evoluções com o passar do tempo. Isso evidencia a dinamicidade das preferências.

Na sociedade atual, as aplicações modernas permitem com que seus usuários acessem e compartilhem dados a qualquer hora e em qualquer lugar. Estes dados podem conter informações interessantes e úteis sobre as preferências dos usuários. Aplicações que conseguem extrair conhecimento de preferências a partir destes dados, com o intuito de facilitar o dia a dia das pessoas, são de extrema importância frente ao volume diário de informações com as quais estas são confrontadas. Mas como extrair continuamente essas informações de preferências dos usuários sem ser inoportuno e intrusivo? As propostas desta dissertação visam responder a esta pergunta.

Sob outro ângulo de análise, os dados de hoje em dia se diferem dos dados de décadas passadas em um ponto crucial de amplo impacto nos mais diversos sistemas: seu volume. Esse enorme aumento do volume de dados digitais presenciado nos últimos anos foi parcialmente ocasionado por uma nova classe de aplicações emergentes - aplicações em que os dados são gerados a taxas muito elevadas, na forma de *data streams*. De uma forma geral, um *data stream* pode ser visto como uma sequência de tuplas relacionais que chegam continuamente em tempo variável. Alguns dos típicos domínios de aplicação de *data streams* são: aplicações web, dados do mercado financeiro, fluxo de transações de cartão de crédito, monitoramento de rede e dados de sensores. Abordagens tradicionais de mineração de dados não conseguem processar com êxito os *data streams*, principalmente devido ao seu volume de dados potencialmente infinito e a sua evolução sobre o tempo. Com isso, várias técnicas de mineração de *data streams* surgiram para lidar apropriadamente com este novo formato dos dados [Domingos e Hulten 2000, Oza e Russell 2001, Gaber et al.

2005, Bifet et al. 2009, Bifet et al. 2010, Bifet et al. 2011, Gama 2010, Vivekanandan e Nedunchezhian 2011, Yong et al. 2012].

Ainda assim, grande parte das pesquisas na área de mineração de dados aborda o cenário *batch* [1]. Em particular, a maioria das pesquisas na área de mineração de preferências tem se concentrado nesse cenário, onde o algoritmo de mineração tem a sua disposição um conjunto de informações estáticas de preferências do usuário [Freund et al. 2003, Holland et al. 2003, Jiang et al. 2008, de Amo et al. 2012a, de Amo et al. 2012b]. Entretanto, devido à dinamicidade das preferências dos usuários, essa parece não ser a melhor solução. Seguindo esse raciocínio, esta dissertação propõe algoritmos capazes de minerar as preferências do usuário a partir de *data streams*.

As questões mais importantes que tornam o processo de mineração de *data streams* muito mais desafiador do que no cenário *batch* são as seguintes: (1) os dados não são armazenados e não estão disponíveis sempre que necessário; cada tupla deve ser aceita conforme ela chega e uma vez inspecionada ou ignorada, ela é descartada, sem nenhuma possibilidade de ser inspecionada novamente; (2) o processo de mineração precisa lidar com limitações de ambos, memória e tempo; (3) o algoritmo de mineração deve ser capaz de produzir o melhor modelo em qualquer instante que é solicitado, usando apenas os dados de treinamento que foram observados até o momento. Mais detalhes sobre estes desafios são encontrados em [Rajaraman e Ullman 2011].

O problema abordado pelo trabalho descrito nesta dissertação está inserido dentro desse cenário. Mais especificamente, o problema proposto consiste em extrair um *modelo de preferências* a partir de um *stream* de preferências, que contém informações implícitas de preferências do usuário ao longo do tempo. Os algoritmos propostos nesta dissertação extraem um conjunto reduzido de estatísticas suficientes a partir do *stream* de preferências do usuário, e utilizam as mesmas para evoluir o modelo de preferências ao longo do tempo. No trabalho descrito nesta dissertação, esse modelo de preferências é representado por uma *Rede Bayesiana de Preferências (RBP)*, capaz de efetuar predições sobre as preferências do usuário. Além disso, o trabalho descrito nesta dissertação foca em um tipo particular de preferência, as *preferências contextuais*. Esse tipo de preferência possui um alto poder expressivo. Uma preferência contextual permite especificar que alguns valores de um atributo particular são preferíveis a outros *em um dado contexto*. Por exemplo, um usuário pode preferir viajar para praia a viajar para lugares históricos *se estiver com os amigos*, porém dá preferência às viagens a lugares históricos *se estiver sozinho*.

Após essa introdução sobre o contexto no qual o trabalho descrito nesta dissertação está inserido, será apresentado a seguir um exemplo de motivação com o intuito de proporcionar um melhor entendimento sobre o problema investigado.

[1] Assim como em vários trabalhos que lidam com *data streams*, neste texto o aprendizado de máquina convencional será referido apenas como *batch*.

1.1 Motivação

Como motivação da pesquisa envolvida nesta dissertação, o Exemplo 1.1 mostra um típico cenário no qual os algoritmos propostos de mineração de preferências em *data streams* podem ser empregados em benefício dos usuários.

Exemplo 1.1. Ler notícias é um hábito corriqueiro e diário para muitas pessoas. Sendo assim, considere um cenário de aplicação onde um site de notícias *online* deseja aprender as preferências de um usuário e, com base nisso, efetuar recomendações personalizadas de notícias. Neste cenário, considere que as notícias são representadas de acordo com o seguinte esquema relacional: Notícias (*Mídia, Categoria, Subcategoria, Tópico, Autor, País, Cidade, Idioma*). Considere ainda que o usuário U_1 possui inúmeras preferências sobre notícias. Abaixo é listada uma pequena amostra de suas preferências:

- Para mim, a *mídia* da notícia é totalmente influenciada pela *categoria* da notícia. Sendo assim, algumas das minhas preferências em relação à mídia da notícia são:
  - Se a categoria da notícia for *humor*, eu prefiro a mídia *vídeo* à mídia *foto*;
  - Ainda dentro da categoria *humor*, eu prefiro a mídia *foto* à mídia *texto*;
  - Por outro lado, se a categoria for *política*, eu prefiro a mídia *texto* a todas as outras;
- As *categorias* das notícias são auto-suficientes na expressão das minhas preferências. Dessa forma, algumas das minhas preferências em relação à categoria da notícia são:
  - Eu prefiro a categoria *esporte* à categoria *política*;
  - Também prefiro a categoria *educação* à categoria *economia*;
- Quando o tópico é *corrupção*, não tenho dúvidas, prefiro as notícias escritas pelo *Janio de Freitas*.

Existem várias maneiras do site de notícias *online* obter as preferências do usuário U_1. Uma das maneiras possíveis é pedindo para ele informar uma lista com todas as suas preferências. Isso é impraticável para usuários que possuam muitas preferências. Outra opção é solicitar uma amostra reduzida de suas preferências. Mas como informar de forma estática algo que é dinâmico? O exemplo de notícias ilustra bem esse cenário. A natureza dinâmica da ocorrência de notícias faz com que a cada dia surja algo novo, algo que pode despertar o interesse das pessoas ou até mesmo causar repudia. Além disso, as preferências das pessoas estão em constante evolução, seja pela evolução de seu estágio educacional, seja por mudanças na sua vida financeira, ou ainda pela elevação de seu nível de maturidade. Dessa forma, extrair automaticamente e constantemente as informações de preferências dos usuários através de um *stream* de preferências parece ser interessante. A seção 4.2 apresenta uma proposta básica de como converter o *stream* de interações do

usuário com um site em um *stream de preferências*. Nesses casos, algoritmos que consigam aprender um modelo expressivo das preferências do usuário a partir de um *stream* parecem ser adequados para esse tipo de problema. Esses algoritmos precisam ainda ser capazes de se adaptar rapidamente às mudanças de preferências dos usuários, de forma a não efetuar predições de acordo com conceitos antigos, que não são mais válidos para o cenário atual.

É dentro desse contexto que estão inseridos os algoritmos propostos nesta dissertação. Observe que as preferências do usuário U_1 podem ser expressas através de preferências contextuais, que é o foco de interesse deste trabalho. Por exemplo, a primeira preferência listada pode ser modelada da seguinte forma: $Mídia = vídeo > Mídia = foto \mid Categoria = humor$. O símbolo $>$ indica uma ordem entre os dois tipos de mídia citados. Como mencionado anteriormente, os algoritmos propostos nesta dissertação têm como objetivo extrair uma Rede Bayesiana de Preferências (RBP) a partir de um *stream* de preferências. Um dos componentes de uma RBP, como será visto posteriormente, é um grafo acíclico que representa as dependências entre os atributos do esquema relacional. Neste exemplo, os atributos *Mídia* e *Categoria* constituem nós deste grafo. A nítida influência da categoria da notícia para a determinação de sua mídia é representada por uma aresta orientada do atributo *Categoria* para o atributo *Mídia*. Neste mesmo exemplo, o atributo *Categoria* não possui nós pais no grafo e, possivelmente, o atributo *Tópico* influencia nas preferências sobre o atributo *Autor*, implicando na existência de uma aresta orientada entre esses atributos no grafo da RBP. A ideia é que a extração desse tipo de conhecimento sobre as preferências do usuário seja feita automaticamente, sem que o usuário precise informar uma lista com as suas preferências.

Todas as ideias e conceitos apresentados neste exemplo são melhor discutidos e detalhados ao longo dos capítulos desta dissertação.

1.2 Objetivos

Esta seção apresenta os objetivos gerais e os objetivos específicos que foram propostos e alcançados com o trabalho descrito nesta dissertação.

1.2.1 Objetivos Gerais

Os dois objetivos principais do trabalho descrito nesta dissertação são:

1. Formalizar o problema intitulado "Mineração de Preferências Contextuais do Usuário no cenário de *Data Stream*", tratado no trabalho descrito nesta dissertação;
2. Propor algoritmos eficientes para solucionar este problema.

1.2.2 Objetivos Específicos

Os principais objetivos específicos referentes ao primeiro objetivo geral são:

1. Introduzir o conceito de *stream de preferências* como uma forma de representação das informações de preferências do usuário ao longo do tempo;
2. Extrair um modelo de preferência a partir de um *stream de preferências*, capaz de se adaptar ao longo do tempo e predizer acuradamente as preferências do usuário;
3. Introduzir a noção de *concept drift* no cenário de preferências contextuais.

Os principais objetivos específicos referentes ao segundo objetivo geral são:

1. Analisar os requisitos que um algoritmo deve obedecer para ser considerado adequado para o ambiente de *data stream*;
2. Avaliar a eficiência dos algoritmos propostos através de uma análise de complexidade em relação ao tempo de execução dos mesmos;
3. Avaliar a eficiência dos algoritmos propostos segundo critérios de qualidade a serem adaptados para *streams*;
4. Projetar e implementar um gerador de *stream* de dados sintéticos de preferências para a realização de experimentos nos algoritmos propostos;
5. Avaliar os algoritmos propostos sobre dados reais.

1.3 Contribuições

As principais contribuições do trabalho descrito nesta dissertação estão enumeradas abaixo:

1. Identificação e formalização sistemática de um novo problema na área de mineração de preferências: "A partir do *stream de preferências* do usuário, é possível realizar o aprendizado de suas *preferências contextuais* de forma eficiente?";
2. Proposta de algoritmos de mineração de preferências contextuais do usuário no ambiente de *data stream*;
3. Introdução da noção de *concept drift* para o cenário de preferências contextuais;
4. Desenvolvimento de um gerador de *stream* de dados sintéticos para preferências contextuais.

Cada uma destas contribuições será discorrida em mais detalhes ao longo dos capítulos desta dissertação.

1.4 Organização da Dissertação

Esta dissertação está organizada da maneira descrita abaixo. A Figura 1.1 exibe o diagrama de influência entre os capítulos deste trabalho. Nesta figura, os capítulos destacados com cor mais escura representam as principais contribuições do trabalho descrito nesta dissertação.

Capítulo 2 – Background. Apresenta uma visão geral sobre mineração de preferências e mineração de *data streams*, de forma a facilitar o entendimento do leitor sobre as novas definições apresentadas posteriormente nesta dissertação.

Capítulo 3 – Trabalhos Correlatos. Discute os principais trabalhos relacionados com a proposta desta dissertação.

Capítulo 4 – Formalização do Problema. Apresenta a formalização do problema tratado no trabalho descrito nesta dissertação, intitulado "Mineração de Preferências Contextuais do Usuário no Cenário de *Data Stream*".

Capítulo 5 – Algoritmo FPSMining. Apresenta o primeiro algoritmo proposto para solucionar o problema abordado pelo trabalho descrito nesta dissertação. Este algoritmo é não incremental e foi nomeado como FPSMining (***F**ast **P**reference **S**tream **Mining***), principalmente por ser eficiente em termos de tempo de execução. Este capítulo também apresenta a análise de complexidade desse algoritmo.

Capítulo 6 – Algoritmo IncFPSMining. Apresenta o algoritmo IncFPSMining (***Inc**remental **FPSMining***), que é uma versão incremental do algoritmo FPSMining. Este algoritmo utiliza a teoria do *Hoeffding Bound* para atualizar incrementalmente o modelo de preferências. Este capítulo também apresenta a análise de complexidade desse algoritmo.

Capítulo 7 – Algoritmo IGAPSMining. Apresenta o terceiro algoritmo proposto nesta dissertação, nomeado IGAPSMining (***I**ncremental **G**enetic **A**lgorithm for **P**reference **S**tream **Mining***), que é baseado em uma técnica heurística bastante promissora em termos de qualidade e tempo de execução. Este capítulo também apresenta a análise de complexidade desse algoritmo.

Capítulo 8 – Concept Drift em Preferências Contextuais. Apresenta um formalismo sobre os possíveis tipos de *concept drifts* em preferências contextuais. Neste capítulo, também é apresentado o gerador de *stream* sintético de preferências, com e sem a introdução de *concept drift*, proposto nesta dissertação. Por fim, neste capítulo também são discutidas as técnicas que foram incorporadas aos algoritmos propostos para o tratamento desse fenômeno.

Capítulo 9 – Resultados Experimentais. Discute e apresenta os resultados experimentais realizados sobre dados reais e dados sintéticos (com e sem a introdução de *concept drifts*).

Capítulo 10 – Conclusão. Apresenta as principais conclusões obtidas com este trabalho e discute alguns trabalhos futuros a curto e médio prazo.

Referências Bibliográficas.

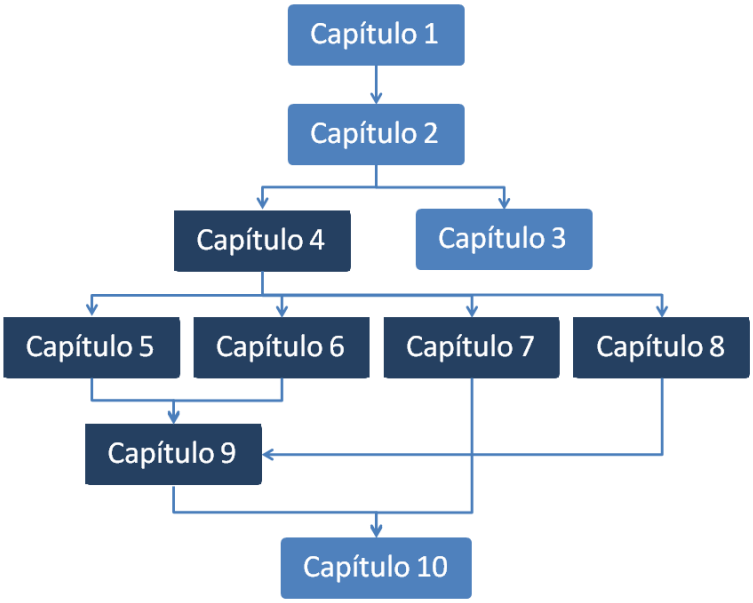


Figura 1.1: Diagrama de influência entre os capítulos desta dissertação

Capítulo 2

Background

Diversos são os tópicos envolvidos na temática abordada pelo trabalho descrito nesta dissertação. A seguir será dada uma visão geral sobre os dois principais assuntos tratados neste trabalho: mineração de *data streams* e mineração de preferências.

Este capítulo serve como base para o entendimento de todos os capítulos posteriores.

2.1 Mineração de Data Streams

A mineração de *data streams* é um dos assuntos mais interessantes de mineração de dados nos últimos anos. Este tipo particular de mineração está preocupado com a extração de estruturas de conhecimento, representadas em modelos e padrões, a partir de *streams* de informações potencialmente infinitos. Recentemente, a pesquisa em mineração de *data streams* tem sido foco de grande atenção, devido, principalmente, à crescente geração de informações em *streams* a partir das mais diversas aplicações.

Muitos são os trabalhos na literatura que exploram o tema de mineração de *data streams* [Bifet e Gavaldà 2007, Bifet et al. 2009, Bifet et al. 2010, Bifet et al. 2011, Domingos e Hulten 2000, Oza e Russell 2001, Gaber et al. 2005, Vivekanandan e Nedunchezhian 2011, Yong et al. 2012, Brzezinski 2010, Gama 2010, Rajaraman e Ullman 2011]. Nestes trabalhos, diferentes técnicas e algoritmos são propostos, sendo que a grande maioria aborda o problema de classificação em *streams*. Dentro do cenário de *data stream*, o problema que o trabalho descrito nesta dissertação aborda é o de *mineração de preferências contextuais*, que de acordo com o nosso conhecimento, não foi abordado por nenhum outro autor até o presente momento.

Nesta seção são apresentados os principais conceitos necessários para o entendimento do processo de mineração de *data streams*. Para tanto, primeiramente é feita uma introdução sobre o conceito de *data stream* na literatura. Em seguida, são apresentados os principais requisitos que um algoritmo deve obedecer para ser considerado adequado para trabalhar com *streams*. Após isso, é apresentada uma explicação sucinta sobre *concept drift*, que é um assunto amplamente pesquisado na área de *data streams*. Na sequência, são

apresentados dois protocolos típicos de amostragem em *data streams*. Por fim, referencia-se brevemente um *framework* de código aberto bastante conhecido para mineração de *data streams*.

2.1.1 Conceito de Data Stream

Antes de entender o processo de mineração em *data streams*, é preciso conhecer melhor este formato de dados. A seguir é apresentada uma definição formal de *data stream*, retirada de [Brzezinski 2010].

Definição 2.1. (*Data Stream*) *Data Stream* é uma sequência (possivelmente infinita) ordenada de instâncias $(x_1, x_2, x_3, ..., x_t, ...)$ que chegam de forma contínua a uma taxa em que não é possível armazená-las de forma permanente.

Para ilustrar melhor este conceito, é apresentado a seguir um exemplo de *data stream* retirado de [Rajaraman e Ullman 2011].

Exemplo 2.1. (Fonte de *Stream* - Dados de Sensor) Suponha um sensor de temperatura boiando no oceano, enviando para uma estação a leitura da temperatura da superfície a cada hora. Os dados produzidos por este sensor é um *stream* de números reais cuja taxa de dados é muito baixa. Agora, considere um sensor com um GPS, e deixe-o relatar a altura da superfície ao invés da temperatura. A altura da superfície varia muito mais rapidamente do que a temperatura, de modo que é possível ter o sensor enviando uma leitura a cada décimo de segundo. Se a cada vez o sensor enviar um número real de 4 bytes, então ele produz 3,5 megabytes/dia. Isso ainda vai levar algum tempo para encher a memória principal, quem dirá um disco. Mas para aprender algo sobre o comportamento de um oceano, podemos implantar um milhão de sensores, cada um enviando um *data stream*, a uma taxa de dez dados/segundo. Um milhão de sensores não é muito: um para cada 150 Km^2 de oceano. Agora, tem-se 3,5 terabytes/dia, e definitivamente será inviável armazenar os dados destes sensores por muito tempo.

2.1.2 Requisitos para os Algoritmos

O artigo [Bifet et al. 2011] é um verdadeiro estado da arte para mineração em *data streams*. De acordo com [Bifet et al. 2011], um algoritmo deve ter quatro requisitos principais para ser considerado adequado para a mineração de *data streams*. Um ciclo de mineração de *data streams* também é proposto por [Bifet et al. 2011]. A seguir é dada uma breve síntese destes conceitos.

Requisito 1: Processar um elemento por vez, e inspecioná-lo no máximo uma vez. A característica chave de um *data stream* é que os dados fluem um elemento depois do outro. Não existe permissão para acesso randômico nos dados do *stream*. Cada elemento

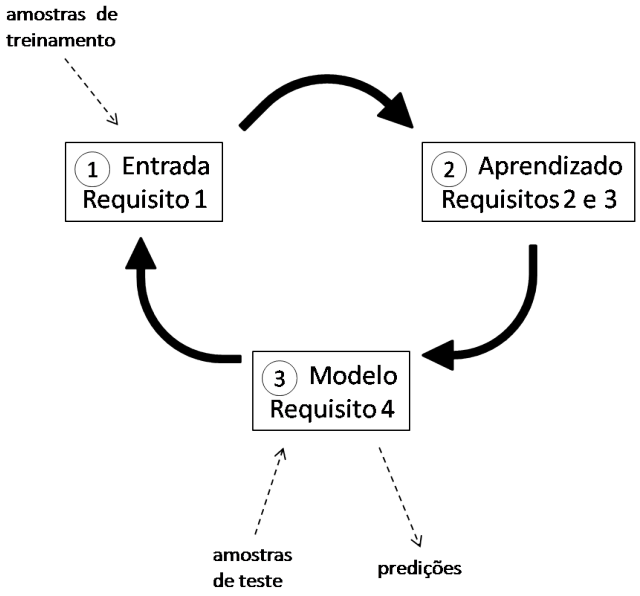


Figura 2.1: Ciclo de Mineração de *Data Stream*[1]

deve ser aceito conforme ele chega, na ordem em que chega. Uma vez inspecionado ou ignorado, o elemento é descartado.

Requisito 2: Usar uma quantidade limitada de memória. A principal motivação para empregar o modelo de *data stream* é que ele permite o processamento de dados que são muitas vezes maiores do que a memória de trabalho disponível.

Requisito 3: Trabalhar em uma quantidade limitada de tempo. A complexidade de execução do algoritmo deve ser linear sobre o número de elementos processados.

Requisito 4: Estar pronto para predizer a qualquer momento. A partir de um número qualquer de elementos processados do *stream*, o algoritmo ideal deve ser capaz de produzir o melhor modelo que é possível construir com destes dados.

Ciclo de Mineração de Data Stream

O modelo geral de mineração de *data streams* segue três passos em um ciclo de repetição, apresentados a seguir.

1. *Entrada:* o algoritmo recebe o próximo elemento disponível do *stream* (Requisito 1).
2. *Aprendizado:* o algoritmo processa o elemento, atualizando suas estruturas de dados. Isso é feito sem exceder os limites de memória (Requisito 2), e tão rápido quanto possível (Requisito 3).

[1]Figura adaptada do trabalho de [Bifet et al. 2011].

3. *Modelo:* o algoritmo está pronto para aceitar o próximo elemento. Mesmo sob a solicitação de várias requisições, o algoritmo deve ser capaz de atualizar sua estrutura de dados (Requisito 4).

A Figura 2.1 ilustra o ciclo de mineração de *data stream* descrito anteriormente.

2.1.3 Concept Drift

Existem vários trabalhos na literatura específicos sobre o tema de *concept drift* [Bach e Maloof 2010, Tsymbal 2004], ou ainda que possuam seções inteiras abordando este assunto [Brzezinski 2010]. Assim, este tema é amplamente discutido dentro da área de mineração de *data streams*.

Os conceitos do mundo real geralmente não são estáveis, e mudam com o passar do tempo. Muitas vezes, essas mudanças tornam o modelo construído sobre dados antigos incompatível com os novos dados. Desta forma, é necessária a atualização regular do modelo. Este problema, conhecido na literatura como *concept drift*, dificulta a tarefa de atualização do modelo e exige alguns cuidados especiais [Tsymbal 2004].

Mais especificamente, o termo *concept drift* significa que as propriedades estatísticas da variável alvo que o modelo está tentando prever podem mudar com o tempo de maneiras imprevisíveis. A palavra *concept* geralmente se refere à variável alvo, enquanto que a palavra *drift* se refere aos desvios que a variável alvo tem com o passar do tempo. Esse fenômeno faz o *stream* evoluir com o tempo.

Um exemplo simples para apresentar o problema de *concept drift* é a tarefa de detectar e filtrar *spams* em e-mails. A distinção entre e-mails indesejados e desejados é específica por usuário e evolui com o tempo. As vezes um determinado tipo de e-mail não é considerado *spam* para o usuário no presente momento, mas dentro de algum tempo pode vir a ser, e, desta forma, este tipo de e-mail não deverá poluir a sua caixa de entrada. Algoritmos que lidam com *data streams* com *concept drift* devem ser capazes de detectar tais mudanças de conceito e readequar seus modelos.

A seguir é apresentada uma definição formal de *concept drift*, retirada de [Brzezinski 2010].

Definição 2.2. (*Concept Drift*) *Concept Drift* é uma imprevista substituição de uma fonte de dados S_1 (com uma distribuição de probabilidade $\prod_{S_1}$), por outra fonte S_2 (com distribuição $\prod_{S_2}$).

Dessa forma, a distribuição que gera os elementos de um *stream* pode mudar ao longo do tempo. O capítulo 8 apresenta como este tema é abordado pelo trabalho descrito nesta dissertação.

2.1.4 Protocolos de Amostragem Típicos

Nesta subseção é feita uma breve introdução sobre dois dos principais protocolos de amostragem para *data streams* disponíveis na literatura: *Holdout* e *Prequential*. Estes dois protocolos foram utilizados nos experimentos realizados no trabalho descrito nesta dissertação (capítulo 9).

Holdout

A seguir é feita uma apresentação do protocolo *Holdout*, tanto no cenário *batch* quanto no cenário de *data stream*.

Holdout - Batch: Esta técnica divide os dados disponíveis em dois subconjuntos mutuamente exclusivos. Um dos conjuntos é utilizado para treinamento do modelo, e é chamado de conjunto de treinamento, enquanto que os dados restantes são utilizados para teste e compõem o conjunto de teste ou *holdout*. Manter estes conjuntos separados garante que o desempenho de generalização está sendo medido.

Holdout - Data Stream: Para rastrear o desempenho do modelo sobre o tempo, este pode ser avaliado periodicamente. O *Holdout* usa porções do *stream* diferentes para treinar e testar o modelo, isto é, os primeiros n_{train} elementos do *stream* são usados para treinar o modelo, os próximos n_{test} elementos do *stream* são utilizados para testar o modelo, e assim por diante.

O protocolo *Holdout* para *data stream* constitui a base do *framework* experimental descrito em [Bifet et al. 2011]. Este *framework* tem o propósito de avaliar o desempenho de classificadores sobre *data streams*. O Algoritmo 1 apresenta o pseudo-código deste *framework*, retirado de [Bifet et al. 2011]. Para parte dos experimentos realizados no trabalho descrito nesta dissertação, adaptou-se este *framework* para a avaliação dos algoritmos propostos no cenário de mineração de preferências. Para maiores detalhes, veja o capítulo 9.

Uma pesquisa sobre classificação em *data streams* foi feita por [Bifet et al. 2011] para levantar as práticas de amostragem típicas. Dos resultados dessa pesquisa, tem-se que o *Holdout* é o protocolo mais utilizado para se obter estimativas de acurácia em *data streams*.

Entretanto, para a avaliação de *streams* com a ocorrência de *concept drift*, o protocolo *Holdout* parece não ser o mais adequado. Suponha que a ocorrência de um *concept drift* coincida com um conjunto de treinamento do *Holdout*. Neste caso, existe a possibilidade do *concept drift* passar praticamente despercebido pelo modelo. Por outro lado, se a ocorrência de um *concept drift* coincidir com um conjunto de testes do *Holdout*, esse fenômeno pode afetar drasticamente a taxa de acerto do modelo.

Algoritmo 1: Procedimento de Avaliação baseado no Holdout

```
1  Fixe m_bound, que é a quantidade máxima de memória alocada para o modelo
2  Mantenha n_test elementos para teste
3  enquanto avaliações adicionais são desejadas faça
4      Inicie o temporizador de treinamento
5      para i = 1 até n_train faça
6          Obtenha o próximo elemento e_train a partir do stream de treinamento
7          Treine e atualize o modelo com e_train, garantindo que m_bound é obedecido
8      Pare o temporizador de treinamento e grave o tempo de treinamento
9      Inicie o temporizador de teste
10     para i = 1 até n_test faça
11         Obtenha o próximo elemento e_test a partir do stream de teste
12         Teste o modelo com e_test e atualize a acurácia
13     Pare o temporizador de teste e grave o tempo de teste
14     Grave as estatísticas do modelo (acurária, tamanho, etc.)
```

Seguindo este raciocínio, nos experimentos realizados no trabalho descrito nesta dissertação (capítulo 9), utilizou-se o protocolo *Holdout* sempre que os dados não tinham a introdução explícita de *concept drift*. Dessa forma, foram realizados experimentos com esse protocolo tanto com dados sintéticos sem a introdução de *concept drift* quanto com dados reais.

Prequential

Outro protocolo de amostragem bastante utilizado para avaliar algoritmos em *data streams* é o *Prequential (Predictive Sequential)* [Dawid 1984]. Neste protocolo, cada elemento individual do *stream* é usado para testar o modelo antes de ser usado para treinar o mesmo. Com isso, a acurácia pode ser incrementalmente atualizada. Quando o processo ocorre nesta ordem (primeiro teste, depois treinamento), o modelo sempre é testado sobre elementos não vistos. A principal vantagem deste protocolo é que não é necessário separar nenhum conjunto apenas para testes, o que maximiza o uso dos dados disponíveis.

Apesar disso, é sabido que o protocolo *Prequential* é mais pessimista. A taxa de erro estimada no *Prequential* sobre todo o *stream* pode ser fortemente influenciada pela primeira sequência de erros, quando o modelo praticamente não havia sido treinado. Esta observação conduz de forma intuitiva à seguinte estratégia: calcular o erro no *Prequential* usando um mecanismo de esquecimento. Isto pode ser feito ou usando uma janela temporal dos mais recentes erros observados, ou usando um fator de decaimento sobre a taxa de erro do modelo [Gama 2010]. A utilização de um fator de decaimento faz com que os erros passados do modelo tenham seu peso diminuído com o passar do tempo, aproximando a avaliação da capacidade atual do modelo.

Nos experimentos realizados no trabalho descrito nesta dissertação (capítulo 9), utilizou-se o protocolo *Prequential* sempre que os dados tinham introdução explícita de *concept*

drift. O protocolo *Prequential* é bastante adequado para avaliar *streams* que possuem a ocorrência de *concept drift*, visto que ele é capaz de avaliar o comportamento do modelo em cada etapa da evolução do *drift*.

2.1.5 MOA

MOA (*Massive Online Analysis*) [Bifet et al. 2010] é um *framework* de código aberto para mineração de *data streams*. Este *framework* inclui uma coleção de algoritmos de aprendizado de máquina (classificação, regressão e clusterização) e ferramentas para a sua avaliação. Dentre as ferramentas para a avaliação dos métodos, o MOA oferece uma coleção de geradores de dados sintéticos, com e sem a introdução de *concept drift*.

Este *framework* é bastante utilizado na área de mineração de *data streams*, e seu código fonte está publicamente disponível em: `http://sourceforge.net/projects/moa-datastream`.

Parte dos experimentos realizados no trabalho descrito nesta dissertação (capítulo 9) utilizam algoritmos implementados no MOA como *baselines* para os algoritmos propostos nos capítulos 5 e 6.

2.2 Mineração de Preferências

Nos últimos anos, o aprendizado de preferências tem atraído muita atenção dentro das comunidades de Mineração de Dados e Aprendizado de Máquina. Isso se deve, principalmente, à demanda crescente por este tipo de serviço dentro das aplicações emergentes. Dada a velocidade com que o aprendizado de preferências tem ganhado importância atualmente, a tendência com o passar do tempo é que isso deixe de ser um diferencial nas aplicações modernas, e passe a ser um requisito básico para a garantia de satisfação do cliente.

Existe uma vasta literatura disponível com relação ao estudo de preferências. A grande maioria dos trabalhos atuam no cenário *batch* e propõe um algoritmo de mineração de preferências [Holland et al. 2003, Jiang et al. 2008, de Amo et al. 2012a, de Amo et al. 2012b], um método para elicitação das preferências do usuário [Holland et al. 2003], ou ainda um formalismo sobre preferências [Wilson 2004, Fürnkranz e Hüllermeier 2010]. O artigo [Jiang et al. 2008], por exemplo, apresenta um algoritmo capaz de aprender preferências a partir de um conjunto de amostras *superiores* e *inferiores* providas pelo usuário. Enquanto isso, o artigo [de Amo et al. 2012a] apresenta a formalização do problema de mineração de preferências contextuais no cenário *batch* (a subseção 2.2.2 fornece maiores detalhes sobre esse problema).

2.2.1 Categorização dos Métodos

Métodos para o aprendizado de preferências podem ser categorizados segundo diferentes critérios, tais como o tipo de problema abordado (*label ranking* ou *object ranking*), o tipo de especificação do modelo de preferência (quantitativo ou qualitativo), a forma de representação das preferências (pareto, contextual, etc.), o tipo de modelo a ser inferido (função *score*, função de *ranking*, relação de preferência, etc.), dentre outros. A seguir será explicado cada um desses critérios em mais detalhes.

Um texto abrangente apresentando diferentes abordagens teóricas e técnicas para o aprendizado de preferência pode ser encontrado em [Fürnkranz e Hüllermeier 2010]. De uma forma geral, o aprendizado de preferências pode ser dividido em dois problemas distintos: *label ranking* (foca nos usuários) e *object ranking* (foca nos objetos). O problema de *label ranking* visa predizer uma ordem sobre um conjunto de *labels* para um dado objeto [De Sá et al. 2011, Hüllermeier et al. 2008]. De acordo com [de Amo et al. 2012a], os objetos podem ser vistos como tuplas contendo informações sobre os usuários, tais como: idade, formação, situação financeira, cidade, profissão, etc. E os *labels* representam objetos a serem ordenados. O problema de *label ranking* consiste em descobrir regras relacionando as informações pessoais dos usuários com a maneira como os mesmos ordenam *labels*. Uma pergunta de interesse neste problema é: "Quais características do usuário U fazem ele ordenar um conjunto de objetos O como $[o_1, o_2, ...]$ (por ordem de preferência)?". Geralmente o objetivo deste problema consiste na descoberta de perfis de usuários. Por outro lado, o problema de *object ranking* visa predizer qual é o objeto preferido entre dois dados objetos. Uma pergunta de interesse neste problema é: "Quais características dos objetos o_1 e o_2 tornam o_1 preferível a o_2 pelo usuário U?". O objetivo deste problema consiste na descoberta de preferências específicas de um usuário. O trabalho de [Hüllermeier et al. 2008] discute as diferenças entre esses dois problemas. O trabalho descrito nesta dissertação atua sobre o problema de *object ranking*.

Quanto aos modelos de preferências, estes podem ser especificados sobre um *framework quantitativo* [Crammer e Singer 2001] ou *qualitativo* [de Amo et al. 2012a]. Na formulação quantitativa, preferências sobre filmes, por exemplo, podem ser elicitadas por pedir ao usuário para dar uma nota para cada filme presente em uma base de dados. Note que este processo pode ser muito custoso, e muitas vezes impraticável. Nesses trabalhos, as preferências são especificadas através de uma função de *score* e o objetivo principal é encontrar uma regra de predição que atribui um *score* para cada tupla do banco de dados. Por outro lado, na formulação qualitativa o modelo de preferência consiste em um conjunto de regras especificadas em um dado formalismo matemático, capaz de expressar as preferências do usuário. O trabalho descrito nesta dissertação utiliza a abordagem qualitativa para a especificação do modelo de preferência. Para isso, este trabalho considerou as *regras de preferências contextuais* (*contextual preference rules* ou *cp-rules*) introduzidas

por [Wilson 2004]. Uma *cp-rule* permite especificar que alguns valores de um atributo particular são preferíveis a outros *em um dado contexto*. Por exemplo, um usuário pode preferir comédias a dramas *se o diretor é Woody Allen*. Outra forma bastante conhecida de representação das preferências do usuário é a pareto. Nesta forma, as preferências não são condicionais ou contextuais, ou seja, as preferências sobre os valores de um atributo não dependem de valores de outros atributos. Um exemplo de preferência pareto é dado por: "Prefiro os hotéis *mais baratos*, ou seja, cujo preço da diária seja o menor possível". Note que o modelo de preferências contextuais é mais expressivo do que o modelo pareto.

Por fim, os tipos mais comuns de modelos que podem ser inferidos são: (1) função *score*, que avalia alternativas individuais; (2) função de *ranking*, que avalia um conjunto de alternativas; (3) relação de preferência, que compara pares de alternativas concorrentes e pode ser facilmente obtida a partir de funções de *score* ou *ranking*. No trabalho descrito nesta dissertação, o modelo a ser inferido é uma relação de preferência (veja mais detalhes no capítulo 4).

O capítulo 3 apresenta uma visão geral sobre os principais trabalhos de mineração de preferências disponíveis na literatura.

2.2.2 Mineração de Preferências Contextuais no Cenário Batch

Nesta subseção, é brevemente introduzido o problema de mineração de preferências contextuais no cenário *batch*. Maiores detalhes sobre este problema são encontrados em [de Amo et al. 2012a]. Esse problema serve como base para o entendimento do problema abordado pelo trabalho descrito nesta dissertação (capítulo 4).

Neste problema, uma *relação de preferência* sobre um conjunto finito de objetos $A = \{a_1, a_2, ..., a_k\}$ é uma ordem parcial estrita sobre A, que é uma relação binária $R \subseteq A \times A$, satisfazendo as propriedades irreflexiva e transitiva. Tipicamente, uma ordem parcial estrita é representada pelo símbolo $>$. Assim, se $>$ é uma relação de preferência, denota-se por $a_1 > a_2$ o fato de que a_1 é preferível a a_2.

Definição 2.3. (Banco de Dados de Preferências) Seja $R(A_1, A_2, ..., A_n)$ um esquema relacional. Seja $\text{Tup}(R)$ o conjunto de todas as tuplas sobre R. Um *banco de dados de preferências* sobre R é um conjunto finito $\mathcal{P} \subseteq \text{Tup}(R) \times \text{Tup}(R)$, que é *consistente*, ou seja, se $(u, v) \in \mathcal{P}$, então $(v, u) \notin \mathcal{P}$. O par (u, v), normalmente chamado de *bitupla*, representa o fato de que o usuário prefere *a tupla u à tupla v*.

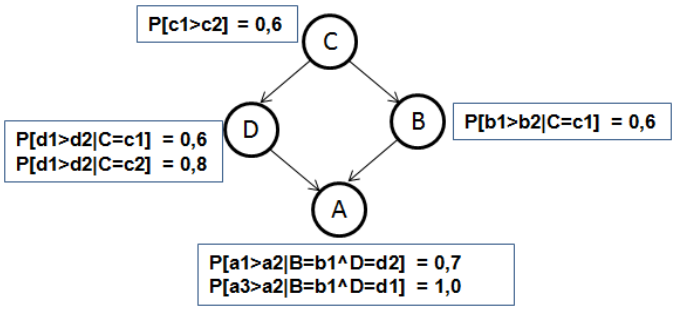
Exemplo 2.2. Seja $R(A, B, C, D)$ um esquema relacional, com os domínios dos atributos dados por $\mathbf{dom}(A) = \{a_1, a_2, a_3\}$, $\mathbf{dom}(B) = \{b_1, b_2\}$, $\mathbf{dom}(C) = \{c_1, c_2\}$ e $\mathbf{dom}(D) = \{d_1, d_2\}$. Seja I uma instância sobre R, como mostrado na Figura 2.2(a). A Figura 2.2(b) ilustra um banco de dados de preferências sobre R, representando uma amostra fornecida pelo usuário sobre as suas preferências em relação às tuplas de I.

Id	A	B	C	D
id_1	a_1	b_1	c_1	d_1
id_2	a_1	b_1	c_1	d_2
id_3	a_2	b_1	c_1	d_2
id_4	a_1	b_2	c_1	d_2
id_5	a_2	b_1	c_2	d_1
id_6	a_3	b_1	c_1	d_1

(a)

(id_1,id_2)
(id_1,id_3)
(id_4,id_5)
(id_4,id_2)
(id_5,id_6)
(id_3,id_5)
(id_4,id_1)

(b)



(c)

Figura 2.2: (a) Uma instância I. (b) Um Banco de Dados de Preferências $\mathcal{P}$ sobre I. (c) Rede Bayesiana de Preferência $\mathbf{PNet}_1$.

O problema de mineração de preferências contextuais no cenário *batch* consiste em extrair um *modelo de preferência* de um banco de dados de preferências fornecido pelo usuário. O modelo de preferência é especificado por uma *Rede Bayesiana de Preferência*, definida a seguir.

Definição 2.4. (Rede Bayesiana de Preferência (RBP)) Uma *Rede Bayesiana de Preferência* sobre um esquema relacional $R(A_1, ..., A_n)$ é um par (G, θ), onde: (1) G é um grafo direcionado e acíclico, cujos nós são atributos em $\{A_1, ..., A_n\}$ e as arestas representam a dependência entre os atributos; (2) θ é um mapeamento que associa, para cada nó de G, uma *tabela de probabilidade condicional de preferências*, ou seja, um conjunto finito de probabilidades condicionais da forma $P[E_2|E_1]$, onde: (1) E_1 é um evento dado pela fórmula $(A_{i_1} = a_{i_1}) \wedge \ldots \wedge (A_{i_k} = a_{i_k})$, de tal modo que $\forall j \in \{1, ..., k\}$, $a_{i_j} \in \mathbf{dom}(A_{i_j})$, e (2) E_2 é um evento da forma "$(B = b_1)$ é preferível a $(B = b_2)$", onde B é um atributo de R, $B \neq A_{i_j}$ $\forall j \in \{1, ..., k\}$ e $b_1, b_2 \in \mathbf{dom}(B)$, $b_1 \neq b_2$.

Exemplo 2.3. A Figura 2.2(c) ilustra uma Rede Bayesiana de Preferência $\mathbf{PNet}_1$ sobre o esquema relacional $R(A, B, C, D)$.

Cada probabilidade condicional $P[E_2|E_1]$ em uma tabela da RBP representa uma *cp-rule probabilística*, onde o evento condicional E_1 é o *contexto* e o evento E_2 é a *preferência*. Uma *cp-rule* probabilística, associada a um nó X do grafo G, representa um grau de confiança em preferir alguns valores de X em detrimento de outros, dependendo dos valores assumidos pelos seus nós pais no grafo. Por exemplo, $P[D = d_1 > D = d_2|C = c_1] = 0,6$ significa que a probabilidade de $D = d_1$ ser preferível a $D = d_2$, dado que $C = c_1$, é 60%.

Em tarefas de classificação, Redes Bayesianas são usadas como uma ferramenta para *classificar* tuplas. Neste cenário de mineração de preferências, RBPs são usadas para *comparar* pares de tuplas. Antes de definir a acurácia, a taxa de comparabilidade e a precisão de uma RBP, que são as medidas de interesse utilizadas no trabalho descrito nesta dissertação, é preciso entender a *ordem parcial estrita* inferida pela RBP.

Exemplo 2.4. Considere a RBP $\mathbf{PNet}_1$ descrita na Figura 2.2(c). Esta RBP permite inferir uma ordem de preferência no conjunto de tuplas possíveis sobre $R(A, B, C, D)$. Considere as tuplas $u_1 = (a_1, b_1, c_1, d_1)$ e $u_2 = (a_2, b_2, c_1, d_2)$. De acordo com esta ordenação, a tupla u_1 é preferível à tupla u_2. Para concluir isso, foram executados os seguintes passos: (1) Seja $\Delta(u_1, u_2)$ o conjunto de atributos em que u_1 e u_2 diferem. Neste exemplo, $\Delta(u_1, u_2) = \{A, B, D\}$; (2) Seja $\min(\Delta(u_1, u_2)) \subseteq \Delta(u_1, u_2)$, de tal modo que os atributos em $\min(\Delta)$ não possuam ancestrais em Δ (de acordo com o grafo G da RBP $\mathbf{PNet}_1$). Neste exemplo, $\min(\Delta(u_1, u_2)) = \{D, B\}$. Para que u_1 seja preferível a u_2 é necessário e suficiente que $u_1[D] > u_2[D]$ e $u_1[B] > u_2[B]$; (3) Calcule as seguintes probabilidades: p_1 = probabilidade de $u_1 > u_2 = P[d_1 > d_2 | C = c_1] * P[b_1 > b_2 | C = c_1] = 0{,}6 * 0{,}6 = 0{,}36$; p_2 = probabilidade de $u_2 > u_1 = P[d_2 > d_1 | C = c_1] * P[b_2 > b_1 | C = c_1] = 0{,}4 * 0{,}4 = 0{,}16$. Para comparar u_1 e u_2, seleciona-se o maior valor entre p_1 e p_2. Neste exemplo, $p_1 > p_2$ e por isso é possível inferir que u_1 é preferível a u_2. Se $p_1 = p_2$, conclui-se que u_1 e u_2 são incomparáveis.

Definição 2.5. (Acurácia, Taxa de Comparabilidade e Precisão) Seja **PNet** uma RBP sobre um esquema relacional R. Seja $\mathcal{P}$ um banco de dados de preferências de *teste* sobre R. A *acurácia (acc)* da **PNet** em relação a $\mathcal{P}$ é definida por $\mathbf{acc}(\mathbf{PNet}, \mathcal{P}) = \frac{N}{M}$, onde M é o número de bituplas em $\mathcal{P}$ e N é a quantidade de bituplas $(u_1, u_2) \in \mathcal{P}$ compatível com a ordem de preferência inferida pela **PNet** sobre as tuplas u_1 e u_2. A *taxa de comparabilidade (TC)* é definida por $\mathbf{TC}(\mathbf{PNet}, \mathcal{P}) = \frac{F}{M}$, onde F é o número de elementos de $\mathcal{P}$ que são comparáveis pela **PNet**. A *precisão (prec)* é definida por $\mathbf{prec}(\mathbf{PNet}, \mathcal{P}) = \frac{acc}{TC} = \frac{N}{F}$. Observe que cada uma das três medidas pode ser derivada das outras duas.

Dessa forma, o problema de minerar preferências contextuais no cenário *batch* é declarado como: "Dado um banco de dados de preferências de treinamento T_1 sobre um esquema relacional R e um banco de dados de preferências de teste T_2 sobre R, encontre uma RBP sobre R, tendo boa acurácia, taxa de comparabilidade e precisão em relação a T_2".

2.3 Considerações Finais

Neste capítulo foi apresentada uma visão geral sobre mineração de *data streams* e mineração de preferências. Dentre os temas abordados, foram discutidos conceitos como: quais são os requisitos necessários para os algoritmos atuarem no cenário de *data stream*, a noção de *concept drift*, a explicação de dois protocolos de amostragem comumente utilizados em *data streams*, uma visão geral de como os métodos de mineração de preferências podem ser categorizados e a explicação do problema de mineração de preferências contextuais no cenário *batch*. Todos esses conceitos servirão como base para o entendimento dos capítulos posteriores.

Capítulo 3

Trabalhos Correlatos

Com o intuito de facilitar o entendimento e prover uma análise comparativa, este capítulo apresenta uma síntese sobre os principais trabalhos de mineração de preferências, tanto no cenário *batch* quanto no cenário de *data stream*.

Os tópicos mais relevantes da síntese realizada são apresentados na Tabela 3.1, que dispõe os trabalhos da literatura em ordem cronológica decrescente. Essa tabela também contempla as características principais dos algoritmos propostos nesta dissertação. A seguir é dada uma breve explicação acerca de algumas nomenclaturas utilizadas nas colunas da Tabela 3.1. Muitos dos conceitos necessários para uma melhor compreensão dessa síntese foram introduzidos anteriormente no capítulo 2.

Nomenclatura das colunas da Tabela 3.1:

- LR/OR (*Label Ranking*/*Object Ranking*): informa o tipo do problema de mineração de preferências;
- QT/QL (Quantitativo/Qualitativo): informa se a elicitação de preferências é formalizada sobre um *framework* quantitativo ou qualitativo;
- U/A (Usuário/Automática): informa se durante a elicitação de preferências foi solicitado ao usuário informações sobre suas preferências, ou se isso foi extraído de maneira automática;
- DS/DR (Dados Sintéticos/Dados Reais): informa se o algoritmo de mineração de preferências foi avaliado sobre dados sintéticos e/ou dados reais;
- Tam. DS/Tam. DR: informa o tamanho máximo do conjunto de dados sintéticos e o tamanho máximo do conjunto de dados reais sobre o qual o algoritmo de mineração de preferências foi avaliado;
- B/S (*Batch*/*Stream*): informa se a técnica proposta aborda a mineração *batch* apenas ou se pode ser utilizada no cenário de *data stream*.

Artigo	LR/ OR	QT/ QL	Modelo de Preferência	U/ A	Entrada	Saída	Medida de Avaliação	DS/ DR	Tam.DS/ Tam.DR	B/ S
[de Amo et al. 2012a]	OR	QL	Contextual	U	Banco de Dados de Preferências	Rede Bayesiana de Preferência (RBP)	Revocação, Precisão	DS, DR	500.000/ 30.000	B
[de Amo et al. 2012b]	OR	QL	Contextual	U/ A	Banco de Dados de Preferências	Perfil do Usuário	Revocação, Precisão	DR	30.000	B
[Beretta et al. 2011]	OR	QL	Contextual	A	Log de Consultas	Conjunto de $\alpha - preferences$	Revocação	DR	-	B
[Shivaswamy e Joachims 2011]	OR	QT	-	A	Resposta Implícita	Função de Utilidade	DCG* regret	DR	-	S
[Jiang et al. 2008]	OR	QL	Pareto	U	$\{Superiores\}$, $\{Inferiores\}$	*Satisfying Preference Set* (SPS)	Acurácia	DS, DR	200.000/-	B
[Jembere et al. 2007]	LR	QT	Contextual	A	Log de Dados	Sistema de Recomendação	Revocação, Precisão	DS	50.000	S
[Somefun e La Poutré 2007]	LR	QL	Pareto	A	Ofertas sobre Pacotes	Mecanismo de Aconselhamento	Percentagem Relativa, ...	DS	-	S
[Krause et al. 2006]	OR	QL	Contextual	A	Sensores e Interação com Celular	Configuração Preferida do Celular	Acurácia	DR	-	S
[Holland et al. 2003]	OR	QL	Composição Pareto e Priorizada	A	Log de Transações	Conjunto de preferências	Revocação, Precisão	DS, DR	50.000/-	B
[Delgado e Ishii 1999]	LR	QT	-	U	Matriz de *Score*	Sistema de Recomendação	Acurácia, Revocação, Precisão	DR	-	S
Algoritmos propostos nesta dissertação	OR	QL	Contextual	A	*Stream* de Preferências	Rede Bayesiana de Preferência (RBP)	Acurácia, Precisão, TC	DS, DR	100.000.000 /183.600	S

Tabela 3.1: Síntese dos Trabalhos Correlatos

3.1 Visão Geral dos Principais Trabalhos

A seguir são fornecidos mais detalhes sobre alguns trabalhos contemplados na Tabela 3.1, dando destaque nas principais diferenças destes para a proposta desta dissertação. Nenhum destes trabalhos trata especificamente o problema que o trabalho descrito nesta dissertação aborda.

Proposta de [de Amo et al. 2012a]

Esse artigo propõe o CPrefMiner, que é uma técnica para mineração de preferências contextuais do usuário no cenário *batch*. Esse algoritmo basicamente recebe como entrada um banco de dados de preferências e retorna como saída uma Rede Bayesiana de Preferência (RBP). Uma RBP é representada por um grafo de dependência dos atributos do banco de dados de preferências e um mapeamento que associa, para cada nó do grafo, uma tabela de probabilidade condicional de preferências. RBP's são usadas para comparar pares de tuplas. A tarefa de construir a RBP estática a partir dos dados de entrada foi dividida em duas fases:

1. *Construção do grafo*: nesta fase foi utilizado um algoritmo genético para evoluir uma população de grafos durante certo número de gerações e retornar o melhor deles de acordo com a função de *score* definida pelo artigo.
2. *Computação das tabelas de probabilidade condicional dos nós do grafo*: nesta fase foi utilizado o Princípio da Probabilidade Máxima, que dado a topologia do grafo, se baseia na ideia de frequência sobre o banco de dados de preferências para realizar a estimativa de probabilidade.

Proposta de [Beretta et al. 2011]

O artigo considera a situação em que estudantes e professores com dispositivos móveis estão andando no campus de uma universidade e efetuam consultas pelo dispositivo. Na arquitetura proposta, o dispositivo do usuário roda uma aplicação cliente que se conecta a um servidor. O usuário efetua consultas no servidor, e o servidor responde ao usuário de acordo com suas preferências e seu atual contexto. Para a mineração de preferências, o dispositivo grava um log das instâncias de dados acessadas pelo usuário. Esse artigo utiliza um algoritmo de mineração para inferir preferências contextuais em tuplas de bancos de dados relacionais que contêm ambos, dados estáticos e dados dinâmicos provenientes de sensores. Os dados de sensores são enviados através de um *stream*, mas são armazenados em um banco de dados relacional, de forma que todas as operações subsequentes são realizadas sobre o banco de dados e não sobre o *stream*. Este ponto, aliado ao fato de que o objetivo do artigo é responder a consultas do usuário de acordo com suas preferências, constituem as principais diferenças dessa proposta para a proposta desta dissertação.

Proposta de [Shivaswamy e Joachims 2011]

Esse artigo propõe um modelo de aprendizado *online* que aprende através de uma retroalimentação de preferências. O modelo é especialmente adequado para aplicações como busca web e sistemas de recomendação. É assumido que o usuário avalia *rankings* de acordo com uma função de utilidade $U(x, y)$ que é desconhecida ao algoritmo de aprendizado. O algoritmo proposto é chamado de *Preference Perceptron*. Ele mantém um vetor w_t e prediz o objeto com maior utilidade de acordo com w_t em cada iteração t. Ele então recebe o retorno do usuário e atualiza w_t em uma determinada direção de acordo com o retorno recebido. Como pôde ser visto, os problemas que esse artigo e o trabalho descrito nesta dissertação se propõe a resolver são distintos.

Proposta de [Jembere et al. 2007]

Esse artigo apresenta uma abordagem para modelar e minerar as preferências do usuário em um ambiente com múltiplos serviços cientes de contexto. Nesse artigo foi definido um meta-modelo de contexto para um ambiente com múltiplos serviços, com as seguintes descrições de contexto de alto nível: Horário, Localização, Influências e Atividade do Usuário. Além disso, foi desenvolvido um modelo para gerar contextos personalizados do usuário. O modelo recebe *streams* de dados de contexto e passa isso através de um algoritmo de *cluster* incremental para cada componente de alto nível do contexto. Se o componente do contexto atual é significativamente diferente dos *clusters* de contexto existentes no modelo, então ele irá formar uma semente para um novo *cluster*, o que irá resultar na criação de um novo contexto. Esse artigo realiza o aprendizado sobre *streams* apenas para o contexto, e não para as preferências do usuário, como é feito no trabalho descrito nesta dissertação.

Propostas de [Somefun e La Poutré 2007]

Esse artigo considera o problema de um agente de loja negociando bilateralmente com muitos clientes sobre um pacote de mercadorias com um preço. Para facilitar a busca do agente de loja por pacotes alternativos mutuamente benéficos, esse artigo propõe um método para o aprendizado *online* das preferências dos clientes respeitando sua privacidade. Para isso, o método usa dados anônimos de negociações passadas. Essa linha de pesquisa se difere da linha do trabalho descrito nesta dissertação por atuar sobre o problema de *label ranking*.

Proposta de [Krause et al. 2006]

Esse artigo propõe um método que, a partir de sensores utilizados junto ao corpo do usuário, consegue aprender preferências pessoais dependentes do contexto de forma *online*, identificando os estados do usuário e observando como este interage com o sistema nestes estados. Esse método foi avaliado em um estudo de caso de telefone celular. Porém, tal método diz apenas qual configuração é preferível que o celular esteja dado o contexto corrente do usuário. Mas este modelo não permite comparar duas configurações de celular

quaisquer dado um determinado contexto, e menos ainda provê uma ordem parcial estrita sobre o conjunto das configurações possíveis. Estas são as principais diferenças do modelo apresentado para o proposto nesta dissertação.

Proposta de [Delgado e Ishii 1999]

Como explicado nesse artigo, sistemas de recomendação, especificamente aqueles que utilizam a técnica de *Collaborative Filtering*, são sistemas de aprendizado que fazem uso de um banco de dados, representando preferências de múltiplos usuários sobre itens, para tentar predizer as preferências de um usuário particular sobre novos itens. Esse artigo propõe um novo algoritmo *online* para executar tal tarefa. Artigos nessa linha se diferem da proposta do trabalho descrito nesta dissertação por abordar o problema de *label ranking*, e não *object ranking*.

3.2 Considerações Finais

Este capítulo apresentou brevemente os principais trabalhos disponíveis na literatura cuja temática mais se aproxima do problema alvo abordado pelo trabalho descrito nesta dissertação. Também foi realizada uma síntese e análise comparativa sobre esses trabalhos correlatos, destacando os principais pontos que estes diferem da proposta desta dissertação.

Capítulo 4

Formalização do Problema

Este capítulo apresenta a formalização do problema abordado pelo trabalho descrito nesta dissertação. O problema em questão é intitulado como "Mineração de Preferências Contextuais no Cenário de *Data Stream*", e inicialmente foi proposto no cenário *batch* [de Amo et al. 2012a]. De forma a facilitar o entendimento sobre o problema no cenário de *data stream*, este capítulo fornece uma extensão do problema descrito no cenário *batch* (para maiores detalhes, veja a subseção 2.2.2).

É importante ressaltar que esta dissertação não pretende formalizar o problema genérico de mineração de preferências contextuais no cenário de *data stream*, mas apenas no âmbito endereçado pelos algoritmos propostos. Dessa forma, uma definição mais específica para o problema é dada por "Mineração de Redes Bayesianas de Preferências a partir de *Streams* de Preferências".

Neste capítulo, o problema em questão é estendido do cenário *batch* (subseção 2.2.2) para o cenário de *data stream*, e é proposta ainda uma forma de elicitação das preferências do usuário a partir de escolhas implícitas realizadas pelo mesmo.

4.1 Mineração de Preferências Contextuais no Cenário de Data Stream

As principais diferenças entre os cenários *batch* e *data stream* em relação ao problema de mineração de preferências contextuais, que é tratado no trabalho descrito nesta dissertação, podem ser resumidas da seguinte maneira:

- *Entrada*: nesta extensão, associa-se um *timestamp* t para cada bitupla (u, v) coletada automaticamente a partir de escolhas implícitas do usuário, onde t representa o instante de tempo em que esta escolha foi realizada. Seja T o conjunto infinito de todos os *timestamps*. Assim, os dados de entrada a partir dos quais o modelo de preferência será extraído são representados por um *stream de preferências*, definido

como um conjunto (possivelmente) infinito $P \subseteq Tup(R) \times Tup(R) \times T$, que é *temporalmente consistente*, ou seja, se $(u, v, t) \in P$, então $(v, u, t) \notin P$. A tripla (u, v, t), que será chamada de *bitupla temporal*, representa o fato de que o usuário prefere a tupla u à tupla v no instante de tempo t. Esta seção apresenta uma proposta de como transformar um *stream* de cliques e consultas do usuário (em um site) em um *stream* de preferências.

- *Saída*: o modelo de preferência a ser extraído do *stream* de preferências é uma *RBP temporal*, isto é, uma $PNet_t$ representando o estado do modelo em um instante de tempo t. A cada instante t, o algoritmo está pronto para retornar um modelo de preferência $PNet_t$ atualizado com os elementos do *stream* até o instante t, que será usado para predizer as preferências do usuário.
- *Ordem de preferência induzida pela RBP a cada instante de tempo t*: Em cada instante de tempo t, é possível comparar as tuplas u e v utilizando o modelo de preferência $PNet_t$, atualizado com os elementos do *stream* de preferências até o instante t. A ordem de preferência entre u e v é denotada por $>_t$, e é obtida como ilustrado no exemplo 2.4.
- *Acurácia, Taxa de Comparabilidade e Precisão no instante t*: em *streams*, a qualidade do modelo de preferência evolui ao longo do tempo. A dinâmica de avaliação do modelo é determinada pelo protocolo de amostragem utilizado. Nos experimentos apresentados no capítulo 9, utilizou-se dois protocolos de amostragem típicos do cenário de *data stream*: *Holdout* e *Prequential*. A seção 9.1 desse mesmo capítulo ilustra como o processo de avaliação ocorre com a utilização de cada um destes protocolos. Em ambos os casos, as fórmulas para o cálculo de *acc*, *TC* e *prec* são análogas às apresentadas na seção anterior.

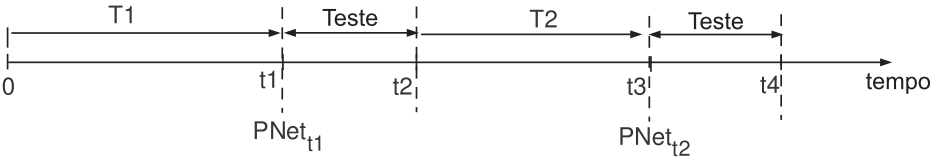


Figura 4.1: Processo de mineração e testes utilizando o protocolo *Holdout*

Como forma de ilustrar o processo dinâmico de minerar e testar o modelo de preferência sobre o tempo a partir de um *stream* de preferências, a Figura 4.1 exemplifica o mecanismo de avaliação do protocolo *Holdout* (explicado em mais detalhes na seção 9.1). Como será visto em capítulos posteriores, os algoritmos propostos são capazes de produzir o modelo de preferência utilizando somente um conjunto reduzido de *estatísticas suficientes* (detalhadas na seção 5.1) extraídas dos conjuntos de treinamento Ti.

Dado isso, é possível declarar o problema de "Mineração de Preferências Contextuais a partir de um *Stream* de Preferências", como:

Entrada: um esquema relacional $R(A_1, A_2, ..., A_n)$, e um *stream* de preferências sobre R.
Saída: sempre que requisitado, retorne a RBP mais atualizada até o momento, tendo uma boa acurácia, taxa de comparabilidade e precisão.

4.2 Elicitação de Preferências

Diferentes estratégias podem ser aplicadas em relação à maneira com que as informações de preferência do usuário são capturadas ao longo do tempo. Os algoritmos propostos nesta dissertação simplesmente assumem que bituplas temporais expressando as preferências do usuário são coletadas de alguma forma, e compõem um *stream de preferências*. Assim, esse tema não é foco do trabalho descrito nesta dissertação. Apesar disso, a seguir é apresentada uma ideia básica de como isso pode ser feito, abordando uma das possíveis maneiras de se recuperar as informações de preferências do usuário de forma automática e transformá-las em um *stream de preferências*.

O método proposto é baseado nos dados do *stream* de navegação *online* do usuário. Quando um usuário visualiza itens de determinada entidade de interesse (p. ex., Notícias) em um site, ele normalmente clica em alguns dos itens que visualizou. Os cliques do usuário demonstram que ele tem mais interesse em alguns itens do que em outros. Esse *stream* de navegação *online* contém informações valiosas que podem ser usadas para predizer as preferências do usuário.

Na sequência é explicado como, nesta abordagem, as bituplas temporais são geradas a partir do *stream* de navegação *online* do usuário. Suponha que em um site seja apresentado ao usuário um conjunto de itens de uma determinada entidade de interesse em uma única página. Suponha ainda que o usuário efetue cliques em alguns desses itens. Pode-se deduzir, de um modo geral, que os itens que o usuário clicou são preferidos a todos os itens não clicados, e a ordem dos cliques também infere uma ordem de preferência entre os itens clicados nessa página. Seja A o conjunto de tuplas representando itens que o usuário clicou na página P, e considere que este conjunto esteja ordenado pela ordem dos cliques do usuário. Seja B o conjunto de tuplas representando itens que o usuário não clicou na página P. A partir disso, é possível gerar um conjunto de bituplas temporais, da seguinte forma:

- Gera-se todas as bituplas temporais possíveis do tipo (a, b, t), onde $a \in A$, $b \in B$ e t é o *timestamp* do momento do clique do usuário no item representado pela tupla a;
- Gera-se todas as bituplas temporais possíveis do tipo (a_i, a_j, t), onde $a_i, a_j \in A$, $i < j$, representando, portanto, que o usuário clicou primeiro em a_i e posteriormente em a_j, e t é o *timestamp* do momento do clique do usuário no item representado pela tupla a_i.

A construção dessas bituplas temporais supõe que todos os itens estão em uma mesma página, e isso ocorre em uma mesma sessão do usuário. Assim, mesmo os cliques sendo em

instantes de tempo diferentes, consegue-se inferir que o usuário prefere mais a tupla que clicou primeiro em detrimento a todas as outras tuplas daquela página. Se por ventura, posteriormente, em outra sessão, o usuário voltar àquela página, a contabilização dos cliques para geração das bituplas temporais será refeita.

O usuário também pode não clicar em nenhuma das tuplas exibidas na página P, e optar por efetuar uma busca por valores de determinados atributos da entidade em questão. Neste caso, pode-se construir o conjunto de todas as bituplas temporais possíveis do tipo (s, p, t), onde $p \in P$, t é o *timestamp* do momento em que o usuário efetuou a busca, e s é semelhante a p, com a diferença de que nos atributos em que o usuário efetuou a busca, s receberá os valores informados na busca. Por exemplo, considere um esquema relacional $R(A, B, C, D)$, e suponha que um dos itens na página P seja $i = (a_1, b_1, c_1, d_1)$. Suponha que o usuário não tenha clicado em nenhum dos itens desta página, e tenha efetuado uma busca do tipo "$A = a_2 \wedge C = c_1$" no instante de tempo t. Neste caso, dentre as bituplas temporais geradas, tem-se a seguinte bitupla temporal: (j, i, t), onde $j = (a_2, b_1, c_1, d_1)$. Note que j é semelhante a i, diferindo apenas nos valores dos atributos em que a busca foi realizada.

Como mencionado anteriormente, a ideia apresentada é apenas uma tentativa de ilustrar uma das formas possíveis de se transformar o *stream* de cliques e consultas do usuário, isto é, um *stream* de tuplas, em um *stream de preferências*, que é um *stream* de bituplas temporais.

4.3 Considerações Finais

Este capítulo apresentou a formalização do problema abordado pelo trabalho descrito nesta dissertação. Em adicional, foi apresentada uma ideia básica de como extrair um *stream* de preferências automaticamente a partir de escolhas implícitas do usuário. Os próximos três capítulos (capítulos 5, 6 e 7) apresentam os algoritmos propostos nesta dissertação para solucionar o problema discutido neste capítulo.

Capítulo 5

Algoritmo FPSMining

O primeiro algoritmo proposto para resolver o problema de "Mineração de Preferências Contextuais do Usuário no cenário de *Data Stream*" é chamado FPSMining (iniciais de ***F**ast **P**reference **S**tream **Mining***). A motivação para o nome deste algoritmo remete ao fato do mesmo conseguir minerar de forma rápida *streams* de preferências contendo informações sobre a evolução das preferências do usuário ao longo do tempo. Este algoritmo, bem como os outros dois algoritmos propostos nesta dissertação, recebe como entrada um *stream* de preferências, e para realizar predições utiliza uma Rede Bayesiana de Preferência (RBP) construída sobre as estatísticas suficientes extraídas do *stream* de preferências. A principal característica deste algoritmo é construir o modelo de preferências (RBP) de uma forma não incremental.

Neste capítulo, primeiramente é apresentada a estrutura de estatísticas suficientes utilizada por todos os três algoritmos propostos nesta dissertação. Esse entendimento prévio se faz necessário para que o algoritmo FPSMining possa ser apresentado. Em seguida, são apresentados os detalhes do algoritmo FPSMining e a sua análise de complexidade.

5.1 Estatísticas Suficientes

Com o intuito de economizar tempo de processamento e memória, os algoritmos propostos nesta dissertação não armazenam elementos do *stream* de preferências, apenas coletam *estatísticas suficientes* deles de uma forma *online*. As estatísticas suficientes refletem o mínimo de informações necessárias para representar o *stream* de preferências.

Os algoritmos propostos mantêm estatísticas para cada atributo do *stream* de preferências. Visando atualizar as estatísticas, para cada bitupla temporal $l = (u_1, u_2, t)$ usada para treinar o modelo, primeiro determina-se o conjunto $\Delta(u_1, u_2)$, que é o conjunto de atributos onde u_1 e u_2 diferem. Apenas os atributos deste conjunto terão suas estatísticas atualizadas de acordo com l. A atualização basicamente consiste em determinar os possíveis contextos em l considerando a preferência particular de cada atributo em Δ. Para cada novo elemento que chega no *stream* de preferências, as estatísticas suficientes são

incrementalmente atualizadas, e o processo de treinamento ocorre através da extração de um modelo de preferência (uma RBP) destas estatísticas. O Exemplo 5.1 ilustra como as estatísticas suficientes são coletadas de um *stream* de preferências.

		$c_1 \succ c_2$	$c_2 \succ c_1$	$c_4 \succ c_5$
A	a_1	3	1	-
	a_2	-	-	2
B	b_3	1	-	1
	b_5	1	-	1

$c_1 \succ c_2$	4
$c_2 \succ c_1$	2
$c_4 \succ c_5$	3

(a)

		$c_1 \succ c_2$	$c_2 \succ c_1$	$c_4 \succ c_5$
A	a_1	3	2	-
	a_2	-	-	2
B	b_3	1	-	1
	b_5	1	-	1
	b_6	-	1	-

$c_1 \succ c_2$	4
$c_2 \succ c_1$	3
$c_4 \succ c_5$	3

(b)

	u_1			u_2		
T	A	B	C	A	B	C
t_1	a_1	b_3	c_1	a_1	b_3	c_2
t_2	a_1	b_3	c_2	a_1	b_5	c_1
t_3	a_2	b_5	c_2	a_1	b_3	c_1
t_4	a_2	b_3	c_4	a_2	b_6	c_5
t_5	a_1	b_5	c_1	a_1	b_5	c_2
t_6	a_2	b_3	c_4	a_2	b_3	c_5
t_7	a_1	b_3	c_1	a_1	b_5	c_2
t_8	a_2	b_5	c_1	a_1	b_6	c_2
t_9	a_1	b_5	c_4	a_2	b_5	c_5
t_{10}	a_1	b_6	c_2	a_1	b_6	c_1

(c)

Figura 5.1: (a) Estatísticas suficientes do atributo C no instante de tempo t_9. (b) Estatísticas suficientes do atributo C no instante de tempo t_{10}. (c) *Stream* de preferências S até o instante de tempo t_{10}.

Exemplo 5.1. *Extração das Estatísticas Suficientes.* Seja $R(A, B, C)$ um esquema relacional, com a_1, $a_2 \in \text{dom}(A)$, b_3, b_5, $b_6 \in \text{dom}(B)$ e c_1, c_2, c_4, $c_5 \in \text{dom}(C)$. Seja S o *stream* de preferências sobre R mostrado na Figura 5.1(c), onde a coluna T representa o tempo no qual a bitupla temporal foi gerada, e $u_1 \succ_{t_i} u_2$ (u_1 é preferível a u_2 em t_i) para cada bitupla temporal (u_1, u_2, t_i) no *stream* de preferências, com $1 \leq i \leq 10$. Considere as estatísticas suficientes do atributo C mostradas na Figura 5.1(a), coletadas do *stream* de preferências S até o instante de tempo t_9. A tabela de cima da Figura 5.1(a) mostra os **contadores de contexto** referentes às preferências sobre os valores do atributo C, e a tabela debaixo mostra os **contadores gerais** sobre C. Contadores de contexto contabilizam as causas possíveis de uma preferência particular sobre os valores de um atributo, e os contadores gerais armazenam o número de vezes que uma preferência particular sobre um atributo apareceu no *stream*. Com a chegada da bitupla temporal $l = (u_1, u_2, t_{10})$, onde $u_1 = (a_1, b_6, c_2)$ e $u_2 = (a_1, b_6, c_1)$, as estatísticas suficientes são atualizadas da seguinte maneira (veja a Figura 5.1(b)): (1) Calcule $\Delta(u_1, u_2)$, que é o conjunto de atributos onde u_1 e u_2 diferem em l. Neste exemplo, $\Delta(u_1, u_2) = \{C\}$, por isso somente o atributo C terá suas estatísticas atualizadas com a chegada de l; (2) Incremente os contadores de contexto a_1 e b_6 em relação à preferência $c_2 \succ c_1$ (tabela de cima da Figura 5.1(b)). Observe que na bitupla temporal l os valores a_1 e b_6 são possíveis contextos (causas) para a preferência $c_2 \succ c_1$, somente porque eles são iguais em ambas as tuplas (u_1 e u_2). Como o contexto b_6 ainda não existe, ele é inserido nas estatísticas; (3) Incremente o contador geral da preferência $c_2 \succ c_1$ (tabela debaixo da Figura 5.1(b)).

O capítulo 8 apresenta duas técnicas para tratamento de *concept drift*, que ao mesmo tempo limitam o crescimento das estatísticas suficientes.

5.2 Detalhes do Algoritmo

A principal ideia do algoritmo FPSMining é criar uma relação de preferências a partir das mais promissoras dependências entre os atributos de um *stream* de preferências.

Para medir a dependência entre um par de atributos, este algoritmo utiliza o conceito de grau de dependência. O *grau de dependência* entre um par de atributos (X, Y) em relação a um *snapshot* (foto) Q das estatísticas suficientes (descritas na seção 5.1), coletadas a partir do *stream* de preferências S no instante de tempo t, é um número real (entre 0 e 1) que estima como as preferências sobre os valores do atributo Y são influenciadas por valores do atributo X. No trabalho descrito nesta dissertação, adaptou-se o conceito de grau de dependência introduzido em [de Amo et al. 2012a] para lidar com as estatísticas suficientes ao invés de um completo conjunto de preferências. Seu cálculo é realizado de acordo com o Algoritmo 2.

Algoritmo 2: Grau de Dependência entre um par de atributos

Entrada: Q: um *snapshot* das estatísticas suficientes do *stream* de preferências S no instante de tempo *time*; (X, Y): um par de atributos; dois *thresholds* $\alpha_1 > 0$ e $\alpha_2 > 0$.

Saída: o grau de dependência de (X, Y) com relação a Q no instante de tempo *time*.

1 **para** *cada par* $(y, y') \in$ ***contadores gerais*** *sobre* Y *de* Q, $y \neq y'$ *e* (y, y') *comparável* **faça**

2 **para** *cada* $x \in$ ***dom***(X), *onde* x *é uma causa para* (y, y') *sendo comparável* **faça**

3 Seja $f_1(S^{time}_{x|(y,y')}) = \max\{N, 1 - N\}$, onde

$$N = \frac{|\{(u_1, u_2, t) \in S^{time}_{x|(y,y')} : u_1 >_t u_2 \wedge (u_1[Y] = y \wedge u_2[Y] = y')\}|}{|S^{time}_{x|(y,y')}|}$$

4 Seja $f_2(T^{time}_{yy'}) = \max\{f_1(S^{time}_{x|(y,y')}) : x \in \mathbf{dom}(X)\}$

5 Seja $f_3((X, Y), Q) = \max\{f_2(T^{time}_{yy'}) : (y, y') \in$ **contadores gerais** sobre Y de Q, $y \neq y'$, (y, y') comparável$\}$

6 **retorne** $f_3((X, Y), Q)$

Para facilitar a descrição do Algoritmo 2, foram introduzidas algumas notações como se segue: (1) Para cada par $(y, y') \in$ **contadores gerais** sobre Y, recuperados de Q (linha 1 do algoritmo), com $y \neq y'$, denota-se por $T^{time}_{yy'}$ o subconjunto finito de bituplas temporais $(u_1, u_2, t) \in S$, tais que $t \leq time$ e $[(u_1[Y] = y \wedge u_2[Y] = y')$ ou $(u_1[Y] = y' \wedge u_2[Y] = y)]$; (2) Seja suporte$((y, y'), Q) = \frac{|T^{time}_{yy'}|}{n}$, onde n é o número de elementos do *stream* de preferências processados até o instante de tempo *time* e $|T^{time}_{yy'}|$ é obtido pela soma dos valores de dois contadores gerais de Y, recuperados de Q: referentes a $y{>}y'$ e $y'{>}y$. É dito que o par $(y, y') \in$ contadores gerais sobre Y é *comparável* (linha 1), se suporte$((y, y'), Q) \geq \alpha_1$ para um dado *threshold* α_1, $0 \leq \alpha_1 \leq 1$; (3) Para cada $x \in \mathbf{dom}(X)$ (linha 2), denota-se por $S^{time}_{x|(y,y')}$ o subconjunto de $T^{time}_{yy'}$ contendo as bituplas temporais (u_1, u_2, t), tais que $u_1[X] = u_2[X] = x$; (4) Seja suporte$(S^{time}_{x|(y,y')}, Q) = \frac{|S^{time}_{x|(y,y')}|}{|\bigcup_{x' \in \mathbf{dom}(X)} S^{time}_{x'|(y,y')}|}$, onde $|S^{time}_{x|(y,y')}|$ é

obtido pela soma dos valores de dois **contadores de contexto** de Y, recuperados de Q: fixando a linha de $X = x$, e somando os valores das colunas $y{>}y'$ e $y'{>}y$; (5) É dito que x é uma *causa para* (y, y') *sendo comparável* (linha 2), se suporte($S^{time}_{x|(y,y')}$,Q) $\geq \alpha_2$ para um dado *threshold* α_2, $0 \leq \alpha_2 \leq 1$. O Exemplo 5.2 ilustra o cálculo do grau de dependência sobre as estatísticas suficientes, tornando mais acessível o seu entendimento.

Exemplo 5.2. *Grau de Dependência calculado sobre as Estatísticas Suficientes.* Considere o *stream* de preferências da Figura 5.1(c) até o instante de tempo t_{10} e o *snapshot* Q de suas estatísticas suficientes para o atributo C, mostradas na Figura 5.1(b). Para calcular o grau de dependência do par de atributos (A, C) em relação ao *snapshot* Q, primeiramente são identificados os **contadores de contexto** relativos ao atributo A na Figura 5.1(b). Os *thresholds* considerados neste exemplo são: $\alpha_1 = 0,1$ e $\alpha_2 = 0,2$. Os suportes de (c_1, c_2) e (c_4, c_5) são $(4 + 3)/10 = 0,7$ e $3/10 = 0,3$, respectivamente. Portanto, nenhuma destas preferências sobre o atributo C são descartadas. Entrando no laço interno para (c_1, c_2) (linha 2 do Algoritmo 2), existe apenas um conjunto, nomeado $S_{a_1|(c_1,c_2)}$. O suporte de $S_{a_1|(c_1,c_2)}$ é $5/5 = 1$ e $N = 3/5$. Com isso, $f_1(S_{a_1}) = 3/5$ e $f_2(T_{c_1c_2}) = 3/5$. Da mesma forma, para (c_4, c_5), tem-se $S_{a_2|(c_4,c_5)}$ com suporte $2/2 = 1$ e $N = 2/2 = 1$. Portanto, $f_1(S_{a_2|(c_4,c_5)}) = 1$ e $f_2(T_{c_4c_5}) = 1$. Assim, o grau de dependência do par de atributos (A, C) é $f_3((A, C), Q) = \max\{3/5, 1\} = 1$. Além disso, o grau de dependência do par de atributos (B, C) é $f_3((B, C), Q) = 1$.

Dado isso, este algoritmo é simples e constrói a RBP a partir das estatísticas suficientes, extraídas do *stream* de preferências, usando o Algoritmo 3.

A seguir são apresentados alguns comentários e explicações sobre o pseudo-código do Algoritmo 3:

- O *snapshot* das estatísticas suficientes (linha 1 do algoritmo) é necessário porque estas são constantemente atualizadas a cada novo elemento de S, e neste ponto do algoritmo é necessário uma versão das estatísticas atualizadas apenas até o instante de tempo t.
- O valor $0,5$ utilizado na linha 5 justifica-se pelo fato de que o grau de dependência entre um par de atributos ou é igual a 0 ou pertence ao intervalo $[0,5; 1]$. Isso pode ser facilmente identificado pela utilização da função *max* nas linhas 3, 4 e 5 do Algoritmo 2. Dessa forma, eliminar os pares de atributos (X, Y) cujo grau de dependência é menor do que $0,5$, significa eliminar os pares cujo grau de dependência é exatamente igual a 0 (não existe nenhuma causa possível em X para as preferências de Y que atenda ao suporte mínimo);
- Note que no grafo G são inseridas apenas arestas que não formam ciclos (linha 9). Isso se faz necessário porque se o grafo da RBP contiver ciclos, então não é possível garantir a ordem parcial estrita inferida pela RBP;

Algoritmo 3: FPSMining

Entrada: $R(A_1, A_2, ..., A_n)$: um esquema relacional; S: um *stream* de preferências sobre R.

Saída: sempre que requisitado, retorne a RBP mais atualizada até o momento.

1 Tire um *snapshot* Q das estatísticas suficientes de S no instante de tempo t.
2 **para** *cada par de atributos* (A_i, A_j), *com* $1 \leq i, j \leq n, i \neq j$ **faça**
3 Utilize o Algoritmo 2 para calcular o grau de dependência dd entre o par de atributos (A_i, A_j) de acordo com Q
4 Seja Ω o conjunto resultante destes cálculos, com elementos da forma (A_i, A_j, dd)
5 Remova de Ω todos os elementos onde $dd < 0.5$ (indica uma fraca dependência entre um par de atributos)
6 Ordene os elementos (A_i, A_j, dd) em Ω em ordem decrescente de acordo com seus respectivos dd
7 Inicialize o grafo G da RBP com um nó para cada atributo de R
8 **para** *cada elemento* $(A_i, A_j, dd) \in$ *conjunto ordenado* Ω **faça**
9 Insira a aresta (A_i, A_j) no grafo G somente se a inserção não formar ciclos
10 Após o grafo G da RBP ser criado, estime as tabelas de probabilidade condicional θ da RBP, utilizando o Princípio da Probabilidade Máxima [Jensen e Nielsen 2007] sobre Q
11 **retorne** *RBP*

- Na tentativa de construir um grafo sem ciclos (linha 9), o algoritmo pode vir a eliminar arestas que poderiam ser úteis para o modelo. Entretanto, como será visto no capítulo 9, a qualidade deste algoritmo é bastante satisfatória ainda com esta característica;
- A intuição subjacente ao *Princípio da Probabilidade Máxima* [Jensen e Nielsen 2007] (linha 10) é utilizar contagens de frequências como estimativas. Por exemplo, se é desejado estimar $P(C = c_1 >_t C = c_2 | A = a_1)$, então é necessário calcular $\frac{N(A=a_1, C=c_1)}{N(A=a_1, C=c_1)+N(A=a_1, C=c_2)}$, onde $N(A = a_1, C = c_1)$ é o número de casos em que $(A = a_1, C = c_1)$ é preferível a $(A = a_1, C = c_2)$ de acordo com Q, e $N(A = a_1, C = c_2)$ é o número de casos em que $(A = a_1, C = c_2)$ é preferível a $(A = a_1, C = c_1)$.

5.3 Análise de Complexidade

A análise de complexidade deste algoritmo foi dividida em três partes, conforme explicado abaixo.

$-$*A complexidade de processar cada elemento do stream e atualizar as estatísticas suficientes*: (i) O custo de varrer uma bitupla temporal com o intuito de salvar Δ (conjunto de atributos que tem valores diferentes nas duas tuplas) e B (conjunto de atributos que tem valores iguais nas duas tuplas) é $O(l)$, onde l é o número de atributos do *stream* de preferências; (ii) O custo de atualizar as estatísticas de um atributo é $O(|B|)$. Assim, o custo de atualizar as estatísticas para todos os atributos em Δ é $O(|\Delta|.|B|)$. Portanto, a

complexidade para processar um elemento do *stream* de preferências neste algoritmo, no pior caso, é $O(|\Delta|.|B|)$, onde $|\Delta| + |B| = l$, e no melhor caso é $O(l)$.

$-$*A complexidade de criar o modelo de preferências (RBP)*: (i) o cálculo do grau de dependência entre todos os possíveis pares de atributos é $O(l^2)$; (ii) a computação da topologia da rede de preferências é $O(l^2.e)$, onde e (satisfazendo $e \leq l.(l-1)/2$) é o número de arestas da RBP. As subetapas utilizadas neste cálculo foram: (a) a eliminação dos pares de atributos cujo grau de dependência $dd < 0,5$ é $O(l^2)$, (b) a ordenação dos elementos (A_i, A_j, dd) restantes é $O(l^2.\log l)$ no caso médio, e (c) para cada elemento (A_i, A_j, dd) restante, o teste de aciclicidade da inserção da aresta (A_i, A_j) no grafo é $O(e)$, o que implica que a complexidade total desta subetapa, no pior caso, é $O(l^2.e)$; (iii) a computação da tabela de probabilidade condicional referente a um atributo A_i é $O(r)$, onde $r = |$*contadores gerais de* $A_i|$ $\times$ $|$*contadores de contexto condicionados aos pais de* A_i *no grafo da RBP*$|$, ou seja, $r \leq \frac{|Q|}{l}$, onde $\frac{|Q|}{l}$ é o tamanho das estatísticas suficientes de cada atributo no caso médio. Dessa forma, a computação da tabela de probabilidade condicional de todos os atributos é $O(l.r)$. Portanto, a complexidade total para construir o modelo neste algoritmo é $O(l^2.e + l.r)$.

$-$*A complexidade de usar o modelo (RBP)* para ordenar uma bitupla temporal neste algoritmo é $O(l+e)$, que pode ser reduzido para $O(l)$ quando a RBP for um grafo esparso.

Note que o custo computacional para construir o modelo (RBP) é quadrático no número de atributos l e o custo computacional para usar o modelo é linear em l. Esse fato não caracteriza um problema porque o número de atributos em *streams* tende a ser pequeno. [Bifet et al. 2011] discute o típico cenário de aprendizado em *streams*, e de acordo com este trabalho é assumido que os dados possuem um pequeno e fixo número de atributos. Isso torna estes custos muito baixos, conforme pode ser constatado nos resultados apresentados no capítulo 9.

5.4 Considerações Finais

Este capítulo apresentou o algoritmo FPSMining, que é um dos algoritmos propostos para solucionar o problema abordado pelo trabalho descrito nesta dissertação. Este algoritmo é caracterizado principalmente por ser rápido e construir o modelo de preferências (RBP) de uma forma não incremental. Uma das vantagens deste algoritmo é que a qualquer instante de tempo t que se deseje, ele é capaz de fornecer o modelo mais atual de uma forma rápida e simples de acordo com as estatísticas suficientes extraídas do *stream* de preferências até o momento. Além disso, este algoritmo é eficiente em termos de tempo (veja a análise de complexidade na seção 5.3) e memória (utiliza um conjunto reduzido de estatísticas suficientes). Por fim, como pode ser visto no capítulo 9, este algoritmo produz resultados satisfatórios em *streams* com e sem a introdução explícita de *concept drift*.

Capítulo 6

Algoritmo IncFPSMining

O segundo algoritmo proposto para resolver o problema abordado pelo trabalho descrito nesta dissertação é chamado IncFPSMining (iniciais de ***Inc****remental* ***F****ast* ***P****reference* ***S****tream* ***Mining***). A motivação para o nome deste algoritmo remete ao fato do mesmo conseguir minerar *streams* de preferências de forma incremental e rápida. Este algoritmo, bem como os outros dois algoritmos propostos nesta dissertação, recebe como entrada um *stream* de preferências, e para realizar predições utiliza uma Rede Bayesiana de Preferência (RBP) construída sobre as estatísticas suficientes do *stream* de preferências. O algoritmo IncFPSMining é uma variante incremental do algoritmo FPSMining. A principal diferença entre os dois algoritmos está na maneira como estes constroem o modelo de preferências (RBP): o algoritmo FPSMining constrói o modelo completo a cada ponto de atualização, enquanto que o algoritmo IncFPSMining constrói o modelo incrementalmente ao longo do tempo. Como será visto no capítulo 9, foram projetados inúmeros testes com o intuito de comparar a performance desses dois algoritmos.

Neste capítulo, primeiramente são apresentados os detalhes deste algoritmo, seguidos da análise de complexidade do mesmo.

6.1 Detalhes do Algoritmo

O algoritmo IncFPSMining atualiza incrementalmente o modelo de preferências M (construído até o momento) a cada bloco de b bituplas temporais (parâmetro do algoritmo chamado "*grace period*") que chegam no *stream* de preferências. Para realizar essa atualização, o algoritmo IncFPSMining utiliza as estatísticas suficientes que foram coletadas com a chegada de cada elemento do *stream* de preferências. O modelo M consiste de um grafo com algumas arestas e_1,e_2,...,e_n, cada uma com um grau de dependência dd medido no momento em que M foi sendo atualizado. O *gap* de uma aresta e_i mede o quão próximo o seu valor de dd está do limite mínimo de $0,5$, isto é, $gap = dd - 0,5$. Somente as arestas que possuam *gap suficientemente altos* são admitidas em cada atualização. O *threshold* utilizado para isso é dado pelo *Hoeffding Bound* [Hoeffding 1963] ϵ associado à

variável aleatória *gap*.

Na área de mineração de *data streams*, tornou-se referência para a comunidade a utilização do *Hoeffding Bound* feita em [Domingos e Hulten 2000]. No próprio algoritmo IncFPSMining, utilizou-se o *Hoeffding Bound* de forma análoga a [Domingos e Hulten 2000]. A princípio, poderia ter sido feito uso de outros limites disponíveis na literatura, mas optou-se por este devido a sua ampla utilização na área de *data streams*, e pelo fato do mesmo ter apresentado resultados satisfatórios para os propósitos do trabalho descrito nesta dissertação. Um trabalho futuro consiste na utilização de outros limites no algoritmo IncFPSMining, a fim de avaliar a sua evolução de performance.

O *Hoeffding Bound* é calculado da seguinte maneira:

$$\epsilon = \sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}}, \text{ onde:}$$

- R é o tamanho do intervalo de valores da variável aleatória X associada ao problema considerado. Neste caso, $X = gap$. Portanto, o maior valor R para gap é $0,5$. Assim, $R = 0,5$.
- δ é a probabilidade de que $X_{corrente} - X_{futuro} > \epsilon$. Em [Domingos e Hulten 2000], considerou-se $\delta = 10^{-7}$.
- n é o número de elementos vistos até o momento.

O *Hoeffding Bound* garante (com uma probabilidade de erro δ) que, se o grau de dependência dd_t de uma aresta e no instante t satisfaz $dd_t - 0,5 \geq \epsilon$, quando o número de elementos vistos até o momento for n, então em qualquer instante futuro t_{futuro} o seu grau de dependência dd_{futuro} deve satisfazer $(dd_t - 0,5) - (dd_{futuro} - 0,5) \leq \epsilon$. Isto é, $dd_t - dd_{futuro} \leq \epsilon$ e, então, dd_{futuro} não estará muito distante do grau de dependência aceitável no instante t corrente. Assim, as arestas que foram introduzidas em um momento anterior não terão os seus valores de dd muito reduzidos no futuro, ou seja, eles não se aproximarão mais do limite $0,5$ do que anteriormente. O exemplo 6.1 ilustra este processo.

Exemplo 6.1. *Garantia do Hoeffding Bound.* Considere $\epsilon = 0,02$ e suponha que uma aresta e seja selecionada no instante t, tendo $dd_t = 0,55$. Então, o gap $(0,55 - 0,5)$ no instante t é $0,05 > 0,02$, considerado razoável. Assim, em qualquer instante futuro t_{futuro}, o gap não será muito modificado, ou seja, $GapFuturo = (dd_{futuro} - 0,5)$ não irá estar muito longe de $GapCorrente = (0,55 - 0,5)$. O *Hoeffding Bound* garante que este gap (que é a variável aleatória envolvida no *Hoeffding Bound*) irá satisfazer: $GapCorrente - GapFuturo \leq \epsilon$. Portanto, $GapFuturo \geq GapCorrente - 0,02 = 0,05 - 0,02 = 0,03$, o que é bastante aceitável.

Este algoritmo considera apenas as estatísticas relacionadas às arestas que não foram inseridas no grafo até o momento. Assim, primeiro seleciona-se arestas que não pertençam ao grafo atual, cujo dd satisfaça a condição de *Hoeffding Bound* ($gap = dd - 0,5 \geq \epsilon$).

Para cada uma destas arestas é verificado se a sua inclusão produz ciclos no grafo. Se sim, os valores de *dd* de todas as arestas no ciclo são avaliados, e a aresta com o pior *dd* é eliminada. Este é um mecanismo efetivo para renovar arestas antigas no grafo. No caso de empate na escolha da aresta a ser eliminada no ciclo, a aresta mais antiga é sempre escolhida para ser eliminada.

A *primeira versão* do algoritmo IncFPSMining está disposta no Algoritmo 4.

Algoritmo 4: IncFPSMining – Versão 1

Entrada: $R(A_1, A_2, ..., A_n)$: um esquema relacional; S: um *stream* de preferências sobre R.

```
1  Seja G um grafo com um nó para cada atributo de R
2  para cada bitupla temporal l de S faça
3      Incremente n, que é o número de elementos vistos até o momento
4      se n mod grace period = 0 então
5          Calcule o Hoeffding Bound ε
6          Tire um snapshot Q das estatísticas de S
7          para cada possível aresta e_i de fora do grafo G faça
8              Utilize o Algoritmo 2 para calcular o grau de dependência dd da aresta
               e_i de acordo com Q
9          Seja Ω o conjunto resultante destes cálculos, com elementos da forma (e_i, dd)
10         Ordene os elementos (e_i, dd) em Ω em ordem decrescente de acordo com
           seus respectivos dd
11         para cada par (e_i, dd) ∈ conjunto ordenado Ω faça
12             se dd − 0,5 ≥ ε então
13                 Insira a aresta e_i em G
14                 se e_i gerou ciclo em G então
15                     Remova de G a aresta com o menor dd no ciclo
16         Após o grafo G da RBP ser criado, estime as tabelas de probabilidade
           condicional θ da RBP, utilizando o Princípio da Probabilidade
           Máxima [Jensen e Nielsen 2007] sobre Q
```

Versões do Algoritmo

Nesta dissertação foram propostas duas versões do algoritmo IncFPSMining, que se diferem apenas na construção das tabelas de probabilidade condicional da RBP. Inicialmente, o algoritmo IncFPSMining foi projetado para efetuar a construção incremental do grafo da RBP, mas a tabela de probabilidade condicional referente a cada nó do grafo foi projetada para ser construída por completo a cada ponto de atualização (*grace period*). Sendo assim, as estatísticas suficientes atualizadas no intervalo entre dois *grace periods* não refletiam instantaneamente nas regras de probabilidade condicional utilizadas pelo modelo para realizar predições. Todos os experimentos do capítulo 9 que foram executados utilizando o protocolo *Holdout* fizeram uso dessa primeira versão do algoritmo IncFPSMining. Porém, com a utilização do protocolo *Prequential*, em que cada elemento do *stream* deve ser utilizado para testar e treinar o modelo, foi possível enxergar com

mais clareza os benefícios que se tem em atualizar as tabelas de probabilidade condicional com a chegada de cada elemento do *stream* de preferências. Dado isso, projetou-se uma segunda versão do algoritmo IncFPSMining, com a atualização das tabelas de probabilidade condicional com a chegada de cada elemento do *stream*, e não apenas a cada *grace period* (versão 1). Todos os experimentos do capítulo 9 que foram executados utilizando o protocolo *Prequential* fizeram uso dessa segunda versão do algoritmo IncFPSMining. Os benefícios principais da segunda versão do algoritmo são:

- *Diminuição da complexidade do algoritmo*: na primeira versão do algoritmo, todas as tabelas de probabilidade condicional são completamente refeitas a cada ponto de atualização do modelo, mesmo que o grafo da RBP não sofra alterações. Isso é necessário devido às atualizações das estatísticas suficientes ocasionadas pelos novos elementos do *stream*. Enquanto isso, com a segunda versão do algoritmo, as tabelas de probabilidade condicional são atualizadas com a chegada de cada elemento do *stream*. Note que isso reduz bastante o custo do algoritmo em atualizar o modelo.
- *Maior poder de predição*: na primeira versão deste algoritmo, as tabelas de probabilidade condicional não refletiam completamente as estatísticas suficientes no período entre cada ponto de atualização (*grace period*). Dessa forma, a segunda versão do algoritmo tende a melhorar bastante a qualidade de predição do modelo, visto que uma informação obtida no instante de tempo t já pode ser utilizada para predições no instante de tempo $t+1$, ao invés de ter que esperar para ser utilizada apenas no próximo ponto de atualização. A segunda versão do algoritmo também é bastante benéfica para a qualidade do modelo antes de seu primeiro ponto de atualização.

Considere A e B como sendo dois atributos de um *stream* de preferências. Com a segunda versão do algoritmo IncFPSMining, existem apenas os seguintes casos em que uma tabela de probabilidade condicional de fato necessite ser refeita por completo:

1. Considere que, em um determinado ponto de atualização, tinha-se a seguinte dependência: $A \rightarrow B$ (lê-se: as preferências sobre os valores do atributo B são influenciadas pelas preferências sobre os valores do atributo A); e no próximo ponto de atualização essa dependência seja removida. Neste caso, apenas a tabela do atributo B precisará ser refeita.
2. Considere que, em um determinado ponto de atualização, não existia dependência entre os atributos A e B; e no próximo ponto de atualização seja inserida a seguinte dependência: $A \rightarrow B$. Neste caso, apenas a tabela do atributo B precisará ser refeita.
3. Considere que, em um determinado ponto de atualização, tinha-se a seguinte dependência: $A \rightarrow B$; e no próximo ponto de atualização essa dependência seja invertida,

passando a ser: $B \rightarrow A$. Neste caso, as tabelas dos atributos A e B precisarão ser refeitas.

Ainda na segunda versão do algoritmo, note que para todos os outros tipos de mudanças na RBP, como o surgimento e desaparecimento de *preferências*, o algoritmo não terá custo algum com a construção das tabelas de probabilidade condicional em cada ponto de atualização, visto que as mesmas já vêm sendo atualizadas com cada elemento do *stream*. O custo da construção das tabelas será referente apenas às mudanças nas *dependências* entre os atributos, o que não é comum de ocorrer.

6.2 Análise de Complexidade

A complexidade da primeira versão do algoritmo IncFPSMining, no pior caso, é a mesma do algoritmo FPSMining. Dessa forma, a complexidade de atualizar o modelo de preferências na primeira versão do algoritmo IncFPSMining, no pior caso, também é $O(l^2.e + l.r)$. Entretanto, ao longo do tempo, o IncFPSMining (nas duas versões apresentadas) irá considerar apenas poucas arestas em seu processamento, diferentemente do FPSMining, que sempre considera todas as arestas possíveis. Assim, na prática, quanto mais atributos o *stream* de preferências tiver, maior será a vantagem do algoritmo IncFPSMining sobre o algoritmo FPSMining.

Por outro lado, a complexidade da segunda versão do algoritmo IncFPSMining se difere da complexidade do algoritmo FPSMining quanto à atualização do modelo. A análise de complexidade da segunda versão deste algoritmo foi dividida em três partes, conforme explicado abaixo.

$-$ *A complexidade de processar cada elemento do stream e atualizar as estatísticas suficientes*: este algoritmo realiza esta etapa da mesma maneira que o algoritmo FPSMining (seção 5.3). Observe que a adição da atualização das tabelas de probabilidade condicional com a chegada de cada elemento do *stream* não altera esta complexidade, visto que o processamento já existente para a atualização das estatísticas é aproveitado para essa atualização.

$-$ *A complexidade de atualizar o modelo de preferências (RBP)*: as etapas "(i) cálculo do grau de dependência ($O(l^2)$, onde l é o número de atributos do *stream* de preferências)" e "(ii) computação da topologia da rede de preferências ($O(l^2.e)$, onde e é o número de arestas da RBP)" possuem, no pior caso (conforme explicado no início desta seção), a mesma complexidade apresentada pelo algoritmo FPSMining (seção 5.3); a etapa "(iii) computação das tabelas de probabilidade condicional", no melhor caso, não possui custo algum, visto que as tabelas de probabilidade condicional são constantemente atualizadas com a chegada de cada elemento do *stream* de preferências. No caso médio, considere

que uma *dependência* seja alterada no grafo do modelo. Neste caso, a complexidade da etapa (iii) é $O(r)$, onde $r \leq \frac{|Q|}{l}$ (tamanho das estatísticas suficientes de cada atributo no caso médio). Portanto, a complexidade total para atualizar o modelo na segunda versão do algoritmo IncFPSMining é $O(l^2.e + r)$. Note que esta complexidade é inferior à apresentada pelo algoritmo FPSMining ($O(l^2.e + l.r)$).

$-$*A complexidade de usar o modelo (RBP)*: este algoritmo realiza esta etapa da mesma maneira que o algoritmo FPSMining (seção 5.3).

6.3 Considerações Finais

Este capítulo apresentou o algoritmo IncFPSMining, que é um dos algoritmos propostos para solucionar o problema abordado pelo trabalho descrito nesta dissertação. Este algoritmo é caracterizado principalmente por construir o modelo de preferências (RBP) incrementalmente. Duas versões deste algoritmo foram propostas nesta dissertação, uma onde as tabelas de probabilidade condicional são refeitas a cada *grace period* (versão 1) e outra em que estas são atualizadas com a chegada de cada elemento do *stream* de preferências (versão 2). Uma das vantagens da segunda versão deste algoritmo é o aprendizado rápido de informações que acabaram de chegar no *stream* de preferências, além de sua baixa complexidade de tempo de execução (veja a análise de complexidade na seção 6.2). Ademais, este algoritmo é eficiente em termos de memória, pois utiliza apenas um conjunto reduzido de estatísticas suficientes. Por fim, como poderá ser visto no capítulo 9, este algoritmo produz resultados satisfatórios em *streams* com e sem a introdução explícita de *concept drift*.

Capítulo 7

Algoritmo IGAPSMining

O terceiro algoritmo proposto para resolver o problema abordado pelo trabalho descrito nesta dissertação é chamado IGAPSMining (iniciais de ***I**ncremental **G**enetic **A**lgorithm for **P**reference **S**tream **Mining***). A motivação para o nome deste algoritmo remete ao fato do mesmo utilizar um algoritmo genético incremental para conseguir minerar *streams* de preferências. Este algoritmo, bem como os outros dois algoritmos propostos nesta dissertação, recebe como entrada um *stream* de preferências, e para realizar predições utiliza uma Rede Bayesiana de Preferência (RBP) construída sobre as estatísticas suficientes do *stream* de preferências. A principal característica deste algoritmo é construir o modelo de preferências (RBP) incrementalmente através de uma abordagem heurística. Nesta dissertação, este algoritmo será apenas apresentado e discutido, visto que o mesmo será alvo de implementação e testes em trabalhos futuros.

Neste capítulo, primeiramente são apresentados os detalhes deste algoritmo, seguidos da análise de complexidade do mesmo.

7.1 Detalhes do Algoritmo

O algoritmo IGAPSMining utiliza um método heurístico baseado em um algoritmo genético incremental (*Incremental Genetic Algorithm* - IGA) sobre um *stream* de preferências para realizar a fase de aprendizado da estrutura do grafo da Rede Bayesiana de Preferência.

Em um algoritmo genético tradicional, normalmente o mesmo conjunto de amostras fixas é utilizado como um todo para o cálculo de *fitness* após cada geração. Isso é possível porque se conhece a priori todo o conjunto de dados que será utilizado para o treinamento do algoritmo. Mas esta abordagem de cálculo de *fitness* não é possível em *streams*, visto que estes não podem ser completamente armazenados devido ao seu tamanho potencialmente infinito. Assim, a ideia principal deste algoritmo consiste em avaliar a população de indivíduos não sobre um banco de dados estático, como normalmente é feito por algoritmos genéticos tradicionais, mas sim sobre os dados vindos de um *stream*, que são

dinâmicos e evoluem com o tempo. Entretanto, ao contrário do IGA proposto por [Vivekanandan e Nedunchezhian 2011], que avalia a população sobre uma janela do *stream*, este algoritmo avalia a população sobre as estatísticas suficientes (com a mesma estrutura proposta no capítulo 5) coletadas do *stream* de preferências do usuário. Caso o tamanho destas estatísticas seja gerenciado, então essa abordagem implica em um baixo uso de memória, além de uma maior velocidade de processamento.

Dessa forma, neste algoritmo o processo de aprendizado do modelo consiste em duas fases, executadas em paralelo: (1) Atualização *online* das estatísticas suficientes para cada nova bitupla temporal que chegar no *stream* de preferências, assim como é feito pela segunda versão do algoritmo IncFPSMining (capítulo 6); (2) Execução do IGA, onde sua população é sempre avaliada levando em conta novos dados do *stream*. Em cada geração do IGA, o melhor indivíduo G da população é mantido em uma variável chamada BEST, juntamente com suas tabelas de probabilidade condicional θ, construídas incrementalmente da mesma maneira que na segunda versão do algoritmo IncFPSMining. Assim, sempre que o modelo for testado sobre o tempo, a RBP utilizada será aquela armazenada em BEST.

O Algoritmo 5 descreve como o IGAPSMining constrói a RBP sobre as estatísticas suficientes do *stream* de preferências.

Algoritmo 5: IGAPSMining

Entrada: $R(A_1, A_2, ..., A_n)$: um esquema relacional; S: um *stream* de preferências sobre R.

1 Gere randomicamente uma população inicial P_0 de grafos sem ciclos sobre R. Essa geração deve ser feita de forma que cada atributo em R seja um nó do grafo, e as arestas sejam colocadas randomicamente no grafo
2 Tire um *snapshot* Q_0 das estatísticas de S
3 Avalie os indivíduos da população P_0 sobre Q_0 de acordo com a função *score*
4 **para** *cada geração i, até que existam dados no stream* **faça**
5 Selecione os pares de indivíduos para o *crossover* em P_i
6 Aplique o operador de *crossover* para cada par selecionado, gerando uma população de filhos F_i
7 Aplique o operador de mutação sobre alguns indivíduos de F_i
8 Tire um *snapshot* Q_{i+1} das estatísticas de S
9 Avalie os indivíduos da população de filhos F_i sobre Q_{i+1}
10 A partir de $P_i \cup F_i$, selecione os $|P_i|$ melhores indivíduos de acordo com seu *score* para fazerem parte da próxima população P_{i+1} (operação de *reinserção ordenada*)
11 Armazene na variável BEST o melhor indivíduo da população $P_i \cup F_i$, juntamente com seu conjunto de tabelas de probabilidade condicional

Uma das justificativas para a implementação futura deste algoritmo consiste no fato do mesmo ter capacidade potencial para atingir qualquer ponto do espaço de busca, o que espera-se que reflita em modelos de alta qualidade.

Este algoritmo efetua o cálculo da função de *fitness* conforme explicado a seguir.

Função de Fitness

A função de *fitness* utilizada neste algoritmo é a mesma proposta por [de Amo et al. 2012a] – função *score*, baseada no conceito de *grau de dependência* entre um par de atributos (Algoritmo 2). Entretanto, neste algoritmo, a aplicação desta função será sobre um *snapshot* Q das estatísticas suficientes do *stream* de preferências. O *snapshot* é apenas para garantir que o processo de avaliação seja justo, de modo que indivíduos que façam parte de uma mesma população não sejam avaliados sobre dados diferentes. A função *score* atribui um número real entre [-1, 1] para uma estrutura candidata G, visando estimar o quão bem ela captura as dependências entre os atributos do *stream* de preferências. Neste caso, cada aresta (X, Y) do grafo G é *punida* ou *recompensada* de acordo com o grau de dependência correspondente do par (X, Y) em relação a Q. Assim, dado uma estrutura G, um *stream* de preferências S com n atributos e um *snapshot* Q das estatísticas suficientes de S, define-se $score(G, Q)$ como:

$$score(G, Q) = \frac{\sum\limits_{X,Y} g((X, Y), G)}{n(n-1)},$$

onde X e Y são atributos de S. A função g é calculada por um conjunto de regras, como se segue:

1. Se $f_3((X, Y), Q) \geq 0,5$ e a aresta $(X, Y) \in G$, então $g((X, Y), G) = f_3((X, Y), Q)$.
2. Se $f_3((X, Y), Q) \geq 0,5$ e a aresta $(X, Y) \notin G$, então $g((X, Y), G) = -f_3((X, Y), Q)$.

   Primeiramente, é preciso recordar que $f_3((X, Y), Q)$ representa o grau de dependência entre o par (X, Y) com relação a Q (veja o Algoritmo 2 para mais detalhes). Assim, a primeira regra modela situações onde os dados apresentam evidências da dependência (X, Y) e existe uma aresta em G para representar tal dependência. Esse fato implica que G deve ser *recompensado* com o valor de f_3. Enquanto isso, na segunda regra, apesar da dependência (X, Y) se evidenciar nos dados, não existe a aresta (X, Y) em G. Com isso, conclui-se que o grafo é incompatível com essa dependência, e portanto deve ser *punido* com o valor de $-f_3$.

3. Se $f_3((X, Y), Q) < 0,5$ e a aresta $(X, Y) \notin G$, então $g((X, Y), G) = 1$.
4. Se $f_3((X, Y), Q) < 0,5$ e a aresta $(X, Y) \in G$, então $g((X, Y), G) = 0$.

   A terceira regra declara que quando os dados não apresentam evidências da dependência (X, Y) e não existe uma aresta para representar essa dependência em G, então G deve ser recompensado com o valor 1. Já de acordo com a quarta regra, se os dados não apresentarem evidências da dependência (X, Y) e existir uma aresta em G para representar tal dependência, então deve-se assumir zero para o *score* dessa aresta.

7.2 Análise de Complexidade

A análise de complexidade deste algoritmo foi dividida em três partes, conforme explicado abaixo.

−A complexidade de processar cada elemento do stream e atualizar as estatísticas suficientes: este algoritmo realiza esta etapa da mesma maneira que a segunda versão do algoritmo IncFPSMining (seção 6.2).

−A complexidade de criar o modelo de preferências (RBP): (i) o cálculo do grau de dependência entre todos os possíveis pares de atributos é $O(l^2)$; (ii) a execução de uma geração do IGA, que produzirá uma topologia BEST a ser utilizada para predições, é $\mathbf{O(l^2.q)}$, onde q é o tamanho da população. As subetapas utilizadas neste cálculo foram: (a) a seleção dos pais para o *crossover* é $O(q)$, (b) a operação do *crossover* de dois pontos nos pais selecionados para a geração dos filhos é $O(l^2.q)$, (c) a operação de mutação por troca de bit em parte dos filhos gerados é $O(q)$, (d) o cálculo do *fitness*, considerando os graus de dependência já calculados, é $O(l^2.q)$, e (e) a seleção dos melhores indivíduos através de uma política de elitismo é $O(q.\log q)$ no caso médio, pois envolve a ordenação da população de acordo com o *fitness* de cada indivíduo; (iii) a computação das tabelas de probabilidade condicional é realizada da mesma maneira que na segunda versão do algoritmo IncFPSMining (seção 6.2) e, portanto, no caso médio, é $O(r)$, onde $r \leq \frac{|Q|}{l}$ (tamanho das estatísticas suficientes de cada atributo no caso médio). Portanto, a complexidade total para construir o modelo neste algoritmo é $O(l^2.q + r)$.

Note que a diferença de complexidade entre este algoritmo e a segunda versão do algoritmo IncFPSMining está na computação da topologia (em negrito no texto), onde o algoritmo IncFPSMining é dependente do número de arestas e (satisfazendo $e \leq l.(l-1)/2$) e este algoritmo é dependente do tamanho da população q do IGA. Assim, para um número pequeno de atributos ($e < q$), o algoritmo IncFPSMining tende a ter uma melhor performance do que este algoritmo. Por outro lado, para números maiores de atributos e tamanhos pequenos de população ($e > q$), este algoritmo tende a ser mais eficiente. Esse é outro ponto que viabiliza a implementação futura deste algoritmo, visto que ele tende a ser mais rápido do que o IncFPSMining com um número maior de atributos.

−A complexidade de usar o modelo (RBP): este algoritmo realiza esta etapa da mesma maneira que os algoritmos FPSMining e IncFPSMining (seções 5.3 e 6.2).

7.3 Considerações Finais

Este capítulo apresentou o algoritmo IGAPSMining, que é um dos algoritmos propostos para solucionar o problema abordado pelo trabalho descrito nesta dissertação. Esse

algoritmo é caracterizado principalmente por utilizar uma abordagem heurística para construir o modelo de preferências (RBP) incrementalmente. O IGAPSMining foi apenas apresentado e discutido neste capítulo, e será alvo de implementação e testes em trabalhos futuros. Existem duas justificativas principais que viabilizam a implementação deste algoritmo: (1) ele tem capacidade potencial para atingir qualquer ponto do espaço de busca, o que espera-se que reflita em modelos de alta qualidade; (2) ele é eficiente em termos de tempo (veja a análise de complexidade na seção 7.2) e memória (utiliza um conjunto reduzido de estatísticas suficientes), e tende a ser mais rápido do que o algoritmo IncFPSMining com um número maior de atributos.

Capítulo 8

Concept Drift em Preferências Contextuais

Neste capítulo o problema de *concept drift* é explorado no âmbito de preferências contextuais. *Streams* de preferências que possuem *concept drifts* são bastante relevantes dentro da problemática do trabalho descrito nesta dissertação, dado que normalmente as preferências do usuário não são estáticas. É natural que as pessoas evoluam seu modo de pensar com o tempo, principalmente no que diz respeito às suas preferências. Tanto que existem inúmeros trabalhos em outras áreas de conhecimento que estudam a evolução das preferências dos consumidores, por exemplo. Sendo assim, este capítulo destaca o tema de *concept drift*, que é alvo de grande atenção em pesquisas relacionadas a *data streams*.

Este capítulo está organizado como se segue. Na seção 8.1 são propostos e formalizados os tipos possíveis de *concept drift* no cenário de preferências contextuais. Na seção 8.2 é apresentado o gerador de *stream* de dados sintéticos, com possibilidade de introdução de *concept drift*, que foi proposto pelo trabalho descrito nesta dissertação. Por fim, na seção 8.3 são apresentadas as técnicas que foram incorporadas aos algoritmos propostos para o tratamento de *concept drift*.

8.1 Tipos de Concept Drift em Preferências Contextuais

Nesta seção é apresentada uma proposta referente à noção de *concept drift* para o cenário de preferências contextuais. A problemática de *concept drift*, ainda não explorada na literatura dentro do cenário de preferências contextuais, merece muita atenção e destaque, haja vista que normalmente as preferências do usuário são dinâmicas e evoluem com o tempo. Dessa forma, antes de se implementar técnicas para o tratamento da evolução das preferências contextuais do usuário ao longo do tempo, é preciso primeiramente que esse processo seja estudado e melhor compreendido.

De acordo com uma análise aprofundada neste assunto, as subseções a seguir apresentam um formalismo sobre os possíveis tipos de *concept drifts* em preferências contextuais.

Considerações gerais para a formalização proposta

A título de formalização dos diferentes tipos de *concept drifts* em preferências contextuais, considere, sem perda de generalidade, o esquema relacional $R(A_1, A_2, ..., A_n)$ e um usuário genérico U. Considere também que as reais preferências do usuário U estejam modeladas por uma RBP, e que para esse usuário "$A_i, A_r, ..., A_j \rightarrow A_k$" (lê-se: as preferências sobre os valores dos atributos $A_i, A_r, ..., A_j$ influenciam nas preferências sobre os valores do atributo A_k). Com isso, $A_i, A_r, ..., A_j$ são atributos pais do atributo A_k na estrutura do grafo da *RBP geradora* dos dados para o usuário U, ou seja, no modelo teórico que representa as reais preferências do usuário U. Denota-se por θ^t_{Ak} a tabela de probabilidade condicional θ referente ao atributo A_k no instante de tempo t, que faz parte da RBP geradora dos dados para o usuário U. Considere ainda que a variável *contexto* seja dada por "$A_i = a_{il}, A_r = a_{rm}, ..., A_j = a_{jo}$", com $a_{il} \in dom(A_i)$, $a_{rm} \in dom(A_r)$, ..., $a_{jo} \in dom(A_j)$.

8.1.1 Tipo 1: Surgimento de Preferência

Com o crescente aumento das opções sobre as quais os usuários têm acesso diariamente, é natural que novas preferências surjam com o passar do tempo. Aliás, não é difícil enxergar que provavelmente esse é o tipo mais comum de mudança nas preferências dos usuários.

O surgimento de uma nova preferência para um determinado atributo impacta diretamente na inserção de pelo menos uma nova regra na sua tabela de probabilidade condicional. Formalmente, pode-se definir o surgimento de uma nova preferência do usuário U para o atributo A_k, no instante de tempo t, como: "$a_{kp} > a_{kq}|contexto$" $\in \theta^t_{Ak}$, com $a_{kp}, a_{kq} \in dom(A_k)$, considerando que "$a_{kp} > a_{kq}|contexto$" $\notin \theta^{t-1}_{Ak}$ e "$a_{kq} > a_{kp}|contexto$" $\notin \theta^{t-1}_{Ak}$. Exemplo: O usuário U nunca havia utilizado a tecnologia *touchscreen* em substituição às teclas de seu aparelho celular. A partir do momento que utilizou, passou a ter a seguinte preferência quanto ao seu *Tipo de Interação* com o celular: "*touchscreen* > *teclas*".

8.1.2 Tipo 2: Desaparecimento de Preferência

A cada dia, mais e mais tendências antigas desaparecem para dar espaço ao surgimento de novas. Dessa forma, algumas preferências do usuário podem simplesmente desaparecer com o tempo, ou não ser mais relevantes dado o atual cenário tecnológico, por exemplo.

O desaparecimento de uma preferência para um determinado atributo impacta diretamente na remoção de pelo menos uma regra na sua tabela de probabilidade condicional.

Formalmente, pode-se definir o desaparecimento de uma preferência do usuário U para o atributo A_k, no instante de tempo t, como: "$a_{kp} > a_{kq}|contexto$" $\notin \theta^t_{Ak}$ e "$a_{kq} > a_{kp}|contexto$" $\notin \theta^t_{Ak}$, com $a_{kp}, a_{kq} \in dom(A_k)$, considerando que "$a_{kp} > a_{kq}|contexto$" $\in \theta^{t-1}_{Ak}$. Exemplo: O usuário U tinha a preferência "$Tec_1 > Tec_2$" para duas tecnologias disponíveis no mercado (Tec_1 e Tec_2), porém a tecnologia Tec_1 não fez sucesso e foi descontinuada, o que implicou no desaparecimento dessa preferência.

8.1.3 Tipo 3: Inversão de Preferência

Um tipo um pouco mais drástico de mudança nas preferências do usuário é a inversão de preferência. Esta é reflexo de uma mudança na opinião do usuário a respeito de algo. Esse tipo de mudança pode estar vinculado a uma alteração no estilo de vida do usuário.

A inversão de uma preferência para um determinado atributo impacta diretamente na modificação de pelo menos uma regra na sua tabela de probabilidade condicional. Formalmente, pode-se definir a inversão de uma preferência do usuário U para o atributo A_k, no instante de tempo t, como: "$P_t[a_{kp} > a_{kq}|contexto] = x$" (lê-se: a probabilidade, no instante de tempo t, de que "$a_{kp} > a_{kq}|contexto$" é igual a x), com $a_{kp}, a_{kq} \in dom(A_k)$ e $x > 0,5$, considerando que "$P_{t-1}[a_{kp} > a_{kq}|contexto] = y$" e $y < 0,5$. Exemplo: Durante o período eleitoral, o usuário U tinha a seguinte preferência em relação às categorias de notícias: "*política* > *esporte*". Depois, com a ocorrência das Olimpíadas, essa preferência particular do usuário mudou, e passou a ser: "*esporte* > *política*".

8.1.4 Tipo 4: Surgimento de Dependência

O crescente refinamento do usuário em relação às suas preferências normalmente torna as mesmas cada vez mais contextuais. Por exemplo, suponha que anteriormente o usuário tinha uma determinada preferência sem nenhum contexto. A adição de um *contexto* (condição) a essa preferência torna a mesma mais restritiva, e evidencia um refinamento maior do usuário em relação àquela preferência.

O surgimento de uma nova dependência para um determinado atributo impacta diretamente em toda a sua tabela de probabilidade condicional. Formalmente, pode-se definir o surgimento de uma nova dependência para o atributo A_k como: "$A_s \rightarrow A_k$", considerando que $A_s \notin \{A_i, A_r, ..., A_j\}$, ou seja, o atributo A_s é um novo pai para o atributo A_k no grafo da RBP geradora de dados do usuário U. Exemplo: Anteriormente, as preferências do usuário U em relação ao atributo *Local* da viagem não sofriam nenhuma influência. Assim, uma das preferências desse usuário em relação as suas viagens era "*praias* > *cidades históricas*". Com o passar do tempo, o usuário U refinou suas preferências em relação ao atributo *Local* da viagem, e as mesmas passaram a sofrer influência do atributo *Acompanhante* da viagem. Dessa forma, a preferência anteriormente mencionada do usuário U passou a ser dividida em duas novas preferências: "*praias* > *cidades históricas* |

Acompanhante = amigos" e "*cidades históricas > praias | Acompanhante = alunos*". Note que alterações (surgimento, desaparecimento ou inversão) nas dependências dos atributos implicam no surgimento e no desaparecimento de preferências.

8.1.5 Tipo 5: Desaparecimento de Dependência

O desaparecimento de dependência geralmente está vinculado a opiniões mais categóricas. Se antes um determinado atributo sofria uma dada influência, e com o tempo passou a não sofrer mais, então as preferências sobre aquele atributo se tornaram um pouco mais categóricas do que antes.

O desaparecimento de uma dependência para um determinado atributo impacta diretamente em toda a sua tabela de probabilidade condicional. Formalmente, pode-se definir o desaparecimento de uma dependência do atributo A_k como: "$A_i \nrightarrow A_k$" (lê-se: as preferências sobre os valores do atributo A_i não influenciam nas preferências sobre os valores do atributo A_k), ou seja, o atributo A_i deixou de ser pai do atribuo A_k no grafo da RBP geradora de dados do usuário U. Exemplo: Seguindo o mesmo exemplo da subseção anterior, considere que inicialmente as preferências do usuário U em relação ao atributo *Local* da viagem sofriam influência do atributo *Acompanhante*. Dessa forma, o usuário U tinha as seguintes preferências: "*praias > cidades históricas | Acompanhante = amigos*" e "*cidades históricas > praias | Acompanhante = alunos*". Com o passar do tempo, o usuário U passou a ter uma opinião um pouco mais categórica em relação ao atributo *Local* da viagem, e suas preferências sobre este atributo deixaram de sofrer influência do atributo *Acompanhante*. Assim, uma de suas preferências em relação a esse atributo passou a ser: "*praias > cidades históricas*".

8.1.6 Tipo 6: Inversão de Dependência

A inversão de dependência caracteriza uma mudança conceitual nas preferências do usuário, e normalmente provoca alterações com um maior grau de severidade nas suas preferências.

A inversão de uma dependência para um determinado par de atributos impacta diretamente nas tabelas de probabilidade condicional de ambos os atributos. Formalmente, pode-se definir a inversão de dependência de uma relação "$A_i \rightarrow A_k$" como: "$A_k \rightarrow A_i$", ou seja, o atributo A_i deixou de ser pai do atribuo A_k, e o atributo A_k passou a ser pai do atributo A_i no grafo da RBP geradora de dados do usuário U. Note que esse tipo de mudança pode ser obtido com uma remoção de dependência seguida de uma adição de dependência. Neste caso, deve-se obrigatoriamente seguir esta ordem (remoção seguida de inserção) de forma a não introduzir ciclos na RBP. Exemplo: Seguindo o mesmo exemplo da subseção anterior, considere que inicialmente as preferências do usuário U em relação ao atributo *Local* da viagem sofriam influência do atributo *Acompanhante*. Dessa forma,

o usuário U tinha a seguinte preferência: "*praias* > *cidades históricas* | *Acompanhante* = *amigos*". Com o passar do tempo, ocorreu uma inversão nessa dependência no modelo conceitual do usuário U, e suas preferências em relação ao atributo *Acompanhante* da viagem é que passaram a sofrer influência do atributo *Local* da viagem. Assim, uma de suas preferências passou a ser: "*amigos* > *alunos* | *Local* = *praias*".

8.2 Gerador de Dados Sintéticos

Com o intuito de avaliar a habilidade dos algoritmos propostos em lidar com grandes volumes de dados, foi implementado um gerador de *stream* sintético de preferências, de forma a possibilitar testes experimentais sobre dados sintéticos. Para isso, foi utilizado como base o gerador de dados sintéticos proposto em [de Amo et al. 2012a]. No trabalho descrito nesta dissertação, esse gerador foi adaptado para o ambiente de *data stream*. Com isso, inúmeros novos parâmetros foram introduzidos nesse gerador, de forma a possibilitar os mais variados testes em *streams* com e sem a introdução de *concept drift*.

O gerador proposto utiliza um algoritmo baseado em *Probabilistic Logic Sampling* [Jensen e Nielsen 2007], capaz de amostrar tuplas a partir de uma Rede Bayesiana. Este algoritmo foi adaptado para amostrar bituplas temporais para um *stream* de preferências S a partir de uma Rede Bayesiana de Preferências (RBP) com estrutura de grafo G e tabelas de probabilidade condicional θ. De forma a produzir o *stream* sintético de preferências, a ser utilizado como entrada pelos algoritmos propostos, o gerador produz randomicamente RBP's que obedecem às restrições impostas pelo conjunto de parâmetros (apresentados posteriormente nesta seção) configurados pelo usuário.

Em relação ao funcionamento dos parâmetros disponíveis, esta seção dá maior ênfase na explicação da abordagem utilizada para introdução de *concept drift* nos dados produzidos pelo gerador. De acordo com [Gama 2010], quando se tem a ocorrência de *concept drift* em *streams*, este não tem impacto em todo o espaço de instâncias, mas sim em regiões particulares. Seguindo essa ideia, as propostas desta dissertação de inserção de *concept drift* no *stream* de preferências estão relacionadas a mutações na RBP geradora dos dados, com a finalidade de impactar regiões particulares do espaço de instâncias.

Segundo [Gama 2010], *concept drift* significa que o conceito sobre o qual os dados estão sendo gerados pode mudar de tempos em tempos, após um período de permanência mínima. Assim, [Gama 2010] propõe o entendimento de um *stream* com *concept drift* da forma como se segue. Considere D uma distribuição não estacionária. Considere ainda que a distribuição que está gerando os elementos muda de tempos em tempos. Neste caso, o *data stream* pode ser visto como uma série de sequências $< S_1, S_2, ..., S_k, ... >$, onde cada sequência S_i é composta por um conjunto de elementos gerados por alguma distribuição estacionária D_i. Cada uma dessas sequências S_i é referenciada como uma fase[1] do stream.

[1] Cada período, referenciado neste trabalho como *fase*, é nomeado por [Gama 2010] como *contexto*.

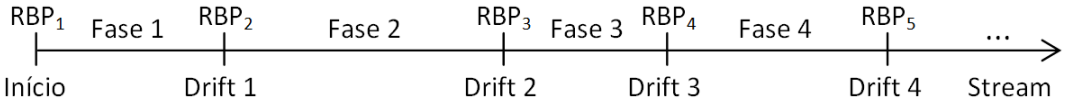


Figura 8.1: Simulação da ocorrência de *concept drifts* em um *stream* sintético de preferências

As sequências S_i podem possuir tamanhos variados, e o algoritmo de aprendizado precisa conseguir se adequar ao novo conceito após a ocorrência de cada *concept drift* no *stream*.

Baseando-se nos conceitos apresentados, em cada *fase i*, o gerador proposto nesta dissertação terá uma RBP_i como sendo a fonte geradora dos dados. Basicamente uma RBP_1, randomicamente gerada de forma a obedecer aos parâmetros configurados pelo usuário, começa sendo a fonte geradora dos dados para o *stream* de preferências (Fase 1). Após um determinado período estacionário, a RBP_1 sofre uma mutação (*Drift* 1), dando origem a RBP_2, e passa a gerar dados segundo a mutação sofrida (Fase 2). Essa mutação simula a ocorrência de um *concept drift*. Depois da ocorrência de cada mudança, a geração dos dados passa por um período mínimo estacionário. Esse processo pode ocorrer várias vezes ao longo do *stream* de preferências. A Figura 8.1 ilustra a dinâmica desse processo.

Dada essa explicação, o conjunto de parâmetros disponibilizados no gerador proposto é apresentado abaixo:

1. ATTRIBUTE_QTY: indica a quantidade de atributos do *stream* de preferências;
2. DOMAIN_QTY: indica a quantidade de valores possíveis no domínio de cada atributo;
3. LEVEL_QTY: indica a quantidade de níveis do grafo da RBP geradora dos dados;
4. PARENT_MIN_QTY: indica a quantidade mínima de nós pais para cada nó do grafo da RBP geradora dos dados, exceto para os nós do primeiro nível do grafo (nós sem pais);
5. BITUPLE_QTY: indica a quantidade de bituplas temporais que devem ser geradas, ou seja, indica o tamanho do *stream* de preferências;
6. SEED: indica a semente utilizada para a geração de números aleatórios. Utilizar uma mesma semente em dois testes executados sobre o mesmo conjunto de parâmetros possibilita a reprodução de experimentos;
7. PERC_MUT_STREAM: indica o percentual de *concept drifts* que devem ser inseridos no *stream* de preferências;
8. MIN_PHASE_QTY: indica o tamanho mínimo da fase entre a ocorrência de dois *concept drifts*;
9. CD: indica o tipo de mutação que será efetuada na RBP geradora dos dados com o intuito de introduzir *concept drift* no *stream* de preferências. Os tipos possíveis são:

a) CD_NONE: indica que não haverá introdução explícita de *concept drift* no *stream* de preferências;

b) CD_MUT_CREATE_PREFERENCE: indica a introdução de *concept drifts* do Tipo 1 - Surgimento de Preferência;

c) CD_MUT_REMOVE_PREFERENCE: indica a introdução de *concept drifts* do Tipo 2 - Desaparecimento de Preferência;

d) CD_MUT_INVERT_PREFERENCE: indica a introdução de *concept drifts* do Tipo 3 - Inversão de Preferência;

e) CD_MUT_CREATE_DEPENDENCY: indica a introdução de *concept drifts* do Tipo 4 - Surgimento de Dependência;

f) CD_MUT_REMOVE_DEPENDENCY: indica a introdução de *concept drifts* do Tipo 5 - Desaparecimento de Dependência;

g) CD_MUT_INVERT_DEPENDENCY: indica a introdução de *concept drifts* do Tipo 6 - Inversão de Dependência;

h) CD_MUT_ALL: indica a introdução randômica dos diferentes tipos de *concept drifts* citados anteriormente;

10. ATTRIBUTE_DRIFT_QTY: indica, para cada ocorrência de *concept drift*, o número de atributos que serão impactados;
11. PERC_CREATE_PREFERENCE: indica, para cada ocorrência de *concept drift*, o percentual de regras a serem criadas na tabela de probabilidade condicional de um atributo da RBP geradora dos dados;
12. PERC_REMOVE_PREFERENCE: indica, para cada ocorrência de *concept drift*, o percentual de regras a serem removidas da tabela de probabilidade condicional de um atributo da RBP geradora dos dados;
13. PERC_INVERT_PREFERENCE: indica, para cada ocorrência de *concept drift*, o percentual de regras cuja probabilidade X deve ser alterada para o seu complemento $(1 - X)$ na tabela de probabilidade condicional de um atributo da RBP geradora dos dados;
14. EDGE_CREATE_QTY: indica, para cada ocorrência de *concept drift*, o número de arestas a serem inseridas no grafo da RBP geradora dos dados;
15. EDGE_REMOVE_QTY: indica, para cada ocorrência de *concept drift*, o número de arestas a serem removidas do grafo da RBP geradora dos dados;
16. EDGE_INVERT_QTY: indica, para cada ocorrência de *concept drift*, o número de arestas a serem invertidas no grafo da RBP geradora dos dados;

Na seção 8.1 foram propostos seis tipos distintos de ocorrência de *concept drift* em preferências contextuais. Note que todos estes tipos estão contemplados no gerador proposto.

No capítulo 9 é apresentado um conjunto extenso de testes sobre dados sintéticos que faz uso do gerador apresentado nesta seção.

8.3 Técnicas para Tratamento de Concept Drift

Nesta seção são discutidas as técnicas que foram incorporadas aos algoritmos propostos para o tratamento do fenômeno de *concept drift* em *streams* de preferências. As duas técnicas utilizadas no trabalho descrito nesta dissertação envolvem o esquecimento de informações muito antigas, privilegiando as informações mais recentes. Essas técnicas conseguem lidar com *concept drift* e ao mesmo tempo gerenciar a memória ocupada pelas estatísticas suficientes.

8.3.1 Naive Blind

A primeira técnica utilizada, referenciada nesta dissertação por *Naive Blind*, consiste basicamente em, de tempos em tempos, avaliar quais informações das estatísticas suficientes estão mais desatualizadas e então descartá-las.

A seguir será explicado o funcionamento desta técnica em mais detalhes. As estatísticas suficientes utilizadas no trabalho descrito nesta dissertação podem ser abstraídas como uma árvore, com alguns nós intermediários representando a estrutura principal das preferências contextuais, e com as folhas sendo representadas por contadores gerais e contadores de contexto (explicados na seção 5.1). Esta técnica armazena, em cada folha, o tempo de sua última atualização. Periodicamente, de forma a reduzir o *overhead* de execução, essa técnica é executada com o intuito de descartar informações que não são atualizadas há bastante tempo. Basicamente, essa técnica consiste em eliminar folhas que não têm sido visitadas recentemente, isto é, o tempo de sua última atualização difere do tempo atual por uma quantidade maior do que um *threshold* (parâmetro informado pelo usuário). Quando uma folha é removida, é verificado, recursivamente, se os nós atravessados para se atingir essa folha também necessitam ser removidos. Neste caso, a remoção de um nó intermediário ocorrerá quando este não possuir mais nós filhos. Dessa maneira, é feita uma *varredura* completa na árvore das estatísticas suficientes, e não apenas em seus contadores. Isso torna possível o tratamento dos diferentes tipos de *concept drift* propostos na seção 8.1.

O único parâmetro desta técnica é o MAX_DOWNGRADE, que indica o período máximo que uma folha pode permanecer nas estatísticas suficientes sem ter sofrido nenhuma atualização. A definição de quanto em quanto tempo esta técnica será invocada é

realizada dentro dos algoritmos propostos que fazem uso da mesma.

Apesar de este ser um mecanismo simples, ele consegue reduzir consideravelmente o uso de memória utilizada pelas estatísticas suficientes dos algoritmos propostos, o que impacta diretamente em uma maior velocidade de execução dos mesmos.

8.3.2 Exponential Forgetting

Esta técnica é fortemente baseada no princípio de que as informações mais recentes são mais relevantes do que as mais antigas. Nesta técnica, o peso de cada informação que chega no *stream* de preferências diminui exponencialmente ao longo do tempo, dando assim mais importância às observações mais recentes, enquanto ainda não descarta as informações mais antigas por completo.

Basicamente, esta técnica consiste em multiplicar as informações das estatísticas suficientes por um fator de esquecimento $0 < \alpha\_\text{STATISTICS} < 1$. Quanto mais próximo de 0 for $\alpha\_\text{STATISTICS}$, mais rápido será o esquecimento. Normalmente, utiliza-se valores próximos de 1 para $\alpha\_\text{STATISTICS}$, o que proporciona um esquecimento lento e gradual.

A título de formalização, seja $S = \{e_1, e_2, e_3, ...\}$ um *stream* de preferências. Conforme visto no capítulo 5, as estatísticas suficientes Q são atualizadas com a chegada de cada novo elemento do *stream* de preferências. Segundo esta técnica, para atualizar as estatísticas com cada elemento e_i do *stream*, é efetuado o seguinte procedimento:

$$Q_i = Q_{i-1} * \alpha\_\text{STATISTICS} + A_{ei}, \qquad \text{(I)}$$

onde Q_i representa as estatísticas suficientes após a chegada do i-ésimo elemento do *stream* de preferências e A_{ei} são as atualizações referentes ao i-ésimo elemento (e_i).

Na equação (I), primeiro o fator $\alpha\_\text{STATISTICS}$ é aplicado sobre as estatísticas Q_{i-1}, para apenas após isso serem realizadas as atualizações referentes ao elemento e_i. Em relação ao termo $Q_{i-1} * \alpha\_\text{STATISTICS}$ de (I), aplica-se o fator de decaimento $\alpha_$STATISTICS nos contadores de contexto e contadores gerais das estatísticas suficientes. Ainda nesta técnica, após a aplicação do fator de decaimento $\alpha\_\text{STATISTICS}$, é verificado se algum dos contadores q que foram atualizados atende à restrição $q < \text{MIN\_STATISTICS}$, onde $0 < \text{MIN\_STATISTICS} < 1$ é um *threshold* passado como parâmetro pelo usuário. Se essa condição for satisfeita, então elimina-se q das estatísticas suficientes. Nesta técnica as estatísticas suficientes também podem ser abstraídas como uma árvore, e caso alguma folha q seja removida da árvore, também é verificado recursivamente a necessidade de se remover os nós atravessados para se atingir esta folha caso eles não possuam mais filhos. Assim, as estatísticas suficientes não irão crescer indefinidamente, pois quando um valor deixar de aparecer no *stream*, esta técnica acabará por eliminar ele das estatísticas. Essa é uma forma de manter o uso de memória reduzido.

Remetendo novamente à equação (I), o esquecimento exponencial efetuado por esta técnica ocorre devido à seguinte situação: em Q_i ter-se-á executado i vezes a aplicação do

fator de decaimento α_STATISTICS; em Q_{i+1} ter-se-á executado $i+1$ vezes a aplicação de α_STATISTICS, e assim por diante.

Note que esta técnica possui dois parâmetros, o α_STATISTICS e o MIN_STATISTICS, que representam, respectivamente, o fator de decaimento que será aplicado nas estatísticas com a chegada de cada elemento do *stream* e o valor mínimo que um contador pode assumir de forma a continuar fazendo parte das estatísticas.

De acordo com os testes realizados em dados sintéticos apresentados no capítulo 9, este mecanismo provou ser efetivo e ainda apresentou qualidade superior em relação ao *Naive Blind* e à abordagem de não aplicação de nenhuma técnica para tratamento de *concept drift*.

8.4 Considerações Finais

Este capítulo explorou o problema de *concept drift* no cenário de preferências contextuais. Primeiramente, foram propostos e formalizados seis tipos possíveis de *concept drift* em preferências contextuais. Em seguida, foi apresentado o gerador de *stream* sintético de preferências proposto nesta dissertação, que implementa todos os tipos de *concept drift* propostos. Esse gerador é bastante flexível em termos de parâmetros, e permite a construção de *streams* com a ocorrência de vários tipos de *concept drifts*. Por fim, foram apresentadas as técnicas *Naive Blind* e *Exponential Forgetting* que, incorporadas aos algoritmos propostos, representam uma tentativa de tratamento de *concept drift* e gerenciamento de memória das estatísticas suficientes.

Capítulo 9

Resultados Experimentais

Neste capítulo, são apresentados os resultados referentes à performance dos algoritmos FPSMining e IncFPSMining em experimentos executados sobre dados sintéticos e dados reais. Os dois algoritmos mencionados foram implementados em linguagem Java. Todos os experimentos apresentados neste capítulo foram executados em uma máquina com o sistema operacional Windows 7, processador Intel(R) Core(TM) i7 com frequência de 3.40 GHz e memória RAM de 12 GB.

Este capítulo está organizado como se segue. Primeiramente, são apresentados os protocolos de amostragem que foram utilizados nos experimentos realizados: *Holdout* e *Prequential*. Na sequência, são discutidos os testes executados sobre dados reais, destacando a fonte e o tipo de dados utilizados, os parâmetros configurados nos algoritmos e o *baseline* proposto. Por fim, são apresentados os extensos testes realizados sobre dados sintéticos, variando um conjunto significativo de parâmetros no gerador de *stream* sintético de preferências proposto, com e sem a introdução de *concept drift*.

9.1 Protocolos de Amostragem

Nesta seção, são apresentados os dois protocolos de amostragem utilizados nos experimentos realizados no trabalho descrito nesta dissertação. O protocolo *Holdout* foi utilizado tanto em experimentos com dados reais quanto em experimentos com dados sintéticos, enquanto que o protocolo *Prequential* foi utilizado apenas em experimentos com dados sintéticos.

9.1.1 Holdout

Com o intuito de avaliar os algoritmos propostos nesta dissertação, adaptou-se a técnica de amostragem apresentada no Algoritmo 1 (baseada no *Holdout* para *data stream*) para o cenário de mineração de preferências. Essa técnica de amostragem recebe como entrada três parâmetros: n_{train}, n_{test} e n_{eval}. Os parâmetros n_{train} e n_{test} representam,

respectivamente, o número de elementos do *stream* que são utilizados para treinar e testar o modelo em cada avaliação. O parâmetro n_{eval} representa o número de avaliações desejadas ao longo do *stream*.

A seguir é dado um exemplo ilustrativo de utilização desta técnica. Considere os valores[1] $n_{train} = 10\text{k}$, $n_{test} = 1\text{k}$ e $n_{eval} = 9090$. Considere $S = \{e_1, e_2, e_3, ...\}$ o *stream* de preferências utilizado nos testes. A dinâmica de avaliação para este exemplo é a seguinte: (1) os elementos e_1 a e_{10k} de S são usados para treinar o modelo; (2) os elementos e_{10001} a e_{11k} são usados para testar o modelo. As medidas de interesse *acc*, *TC* e *prec* do modelo são calculadas de acordo com este período de testes. Dessa maneira, é obtido um ponto no gráfico da qualidade do modelo ao longo do tempo; (3) os elementos e_{11001} a e_{21k} são usados para treinar o modelo, e assim por diante, por 9090 ciclos. A Figura 9.1 ilustra esse exemplo.

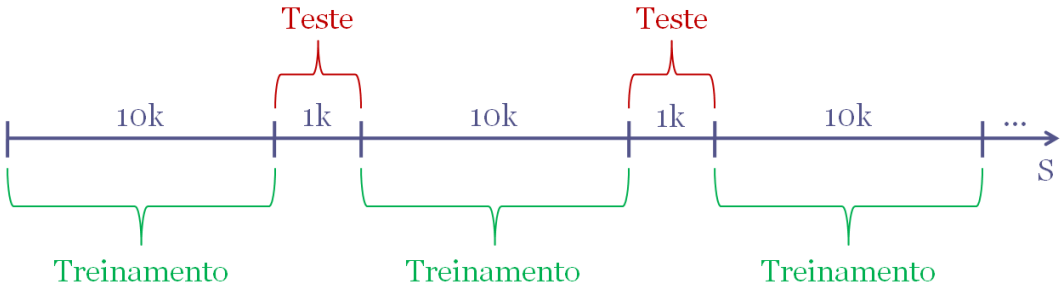


Figura 9.1: Exemplo ilustrativo do Holdout

Com este protocolo, no algoritmo FPSMining o modelo é produzido por completo a cada bloco de n_{train} bituplas temporais que chegam no *stream* de preferências. Note que com o algoritmo FPSMining não faz sentido produzir modelos intermediários (antes da finalização de um bloco de n_{train} elementos), haja vista que esse algoritmo refaz o modelo por completo, e o mesmo apenas será utilizado durante o período de testes. Por outro lado, o algoritmo IncFPSMining atualiza o modelo que foi incrementalmente gerado até o momento a cada *grace period*. Neste caso, o modelo a ser utilizado no período de testes é o modelo corrente, que vem sendo incrementalmente atualizado. É importante relembrar que com este protocolo foi utilizada a primeira versão do algoritmo IncFPSMining, onde as tabelas de probabilidade condicional são refeitas a cada *grace period*.

9.1.2 Prequential

O protocolo de amostragem *Prequential* foi utilizado em todos os experimentos realizados sobre *streams* gerados com a introdução explícita de *concept drift*. Este protocolo de amostragem recebe como entrada apenas um parâmetro: α_PREQUENTIAL, que representa o fator de decaimento aplicado sobre os erros e acertos passados do modelo (veja a seção 2.1.4 para maiores informações a respeito deste protocolo). Em todos os testes

[1] 1k = 1000.

realizados no trabalho descrito nesta dissertação, utilizou-se α_PREQUENTIAL = 1, com o intuito de se ter a dimensão exata de quantos elementos do *stream* de preferências o algoritmo acertou ou errou a predição, e quantos o mesmo não conseguiu comparar.

A seguir é dado um exemplo ilustrativo de utilização deste protocolo. Seja α_PREQUENTIAL = 1, e considere $S = \{e_1, e_2, e_3, ...\}$ o *stream* de preferências utilizado nos testes. A dinâmica de avaliação para este exemplo é a seguinte: (1) o elemento e_1 é usado para testar o modelo; (2) as medidas de interesse *acc*, *TC* e *prec* do modelo são atualizadas, levando em consideração o número de acertos e erros totais do modelo. Note que o divisor M das fórmulas de *acc* e *TC* (referente à definição 2.5) sempre será o número total de elementos do *stream* de preferências vistos até o momento. Assim, o valor de *acc* é dado por: $acc = \frac{N}{M}$, onde N é o número total de acertos até o momento e M é o número total de elementos vistos no *stream* até o momento. Com isso, é obtido um ponto no gráfico da qualidade do modelo ao longo do tempo; (3) o elemento e_1 é usado para treinar o modelo; (4) o elemento e_2 é usado para testar o modelo, e assim por diante, até que se tenham elementos no *stream*. A Figura 9.2 ilustra esse exemplo.

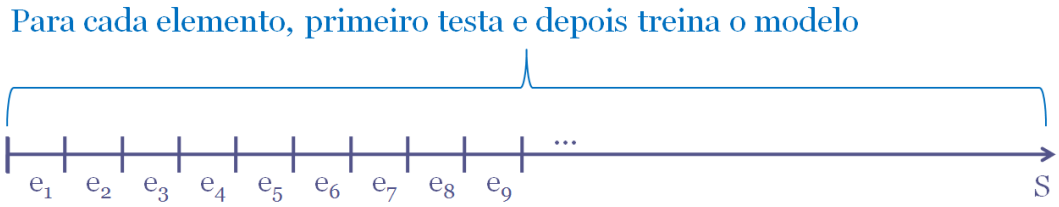


Figura 9.2: Exemplo ilustrativo do Prequential

Com este protocolo, os modelos de preferências são atualizados por ambos os algoritmos, FPSMining e IncFPSMining, a cada bloco de *grace period* bituplas temporais que chegam no *stream* de preferências. Este momento é nomeado como *ponto de atualização*. Em cada ponto de atualização, o algoritmo FPSMining produz seu modelo por completo, enquanto que o algoritmo IncFPSMining apenas atualiza o modelo que foi incrementalmente gerado (a cada *grace period*) até o momento. É importante relembrar que com este protocolo foi utilizada a segunda versão do algoritmo IncFPSMining, onde as tabelas de probabilidade condicional são atualizadas com a chegada de cada elemento do *stream* de preferências.

9.2 Experimentos com Dados Reais

Não é fácil encontrar grandes conjuntos de dados do mundo real que estejam publicamente disponíveis para *benchmarking*. Sendo assim, com o intuito de avaliar os algoritmos abordados neste capítulo sobre dados do mundo real, considerou-se no trabalho descrito nesta dissertação dados contendo preferências relacionadas a filmes, coletados pelo *Grou-*

pLens Research[2] a partir do site do *MovieLens*[3]. Nestes experimentos, um mecanismo de *baseline* foi realizado a fim de comparar a performance dos algoritmos propostos com dois classificadores clássicos de *data streams*.

9.2.1 Descrição dos Dados

Os dados relacionados a filmes que foram utilizados nestes testes são referentes a dez usuários distintos (nomeados U_i, para $i = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$) do *MovieLens*. *Streams* de preferências foram simulados a partir desses dados. Para tanto, primeiramente estipulou-se um intervalo de tempo λ, e cada tupla no conjunto de dados D_i, de filmes avaliados pelo usuário U_i, foi comparada a todos os outros filmes de D_i em um raio λ relativo ao seu *timestamp*, gerando assim as bituplas temporais para cada usuário U_i. A comparação de duas tuplas consiste em verificar para qual delas o usuário atribuiu o maior *score*, e então inserir no *stream* de preferências do usuário uma bitupla temporal do tipo: (tupla com o maior *score*, tupla com o menor *score*, *timestamp* gerado de acordo com λ).

A seguir será apresentado o *baseline* com dados reais realizado no trabalho descrito nesta dissertação. Neste *baseline*, o λ (em dias) foi calculado para cada usuário U_i de acordo com a distribuição das tuplas em relação ao seu *timestamp* dentro de cada conjunto de dados D_i, e os valores obtidos foram: $U_1 = 77$, $U_2 = 37$, $U_3 = 6$, $U_4 = 82$, $U_5 = 140$, $U_6 = 50$, $U_7 = 60$, $U_8 = 69$, $U_9 = 7$ e $U_{10} = 135$. O *stream* de preferências resultante S_i possui cinco atributos (diretor, gênero, linguagem, ator e ano), e os seus elementos correspondem a preferências sobre filmes referentes ao usuário U_i. Em relação ao período de avaliação dos filmes, o conjunto de dados D_1 é constituído pelos filmes avaliados pelo usuário U_1 de 5 de Agosto de 2001 a 3 de Janeiro de 2009. Já o período compreendendo as avaliações dadas pelos dez usuários é de Agosto de 2000 a Janeiro de 2009.

9.2.2 Sobre o Baseline

Até o momento, não foi encontrado na literatura nenhum outro algoritmo que trate exatamente o mesmo problema que o trabalho descrito nesta dissertação aborda. Isso dificulta a realização de comparações dos algoritmos propostos com outro algoritmo utilizado como *baseline*. Apesar disso, dentre as numerosas tarefas de mineração de dados existentes, a tarefa de classificação é a mais próxima da tarefa de mineração de preferências. Com base nisso, projetou-se um *baseline* utilizando algoritmos de classificação adaptados para os propósitos de mineração de preferências.

O objetivo do *baseline* é comparar a performance dos classificadores adaptados com a performance dos algoritmos implementados no trabalho descrito nesta dissertação. Nessa

[2]Disponível em `http://www.grouplens.org/taxonomy/term/14`
[3]Disponível em `http://movielens.umn.edu`

abordagem, as *classes* correspondem aos *scores* dados pelos usuários aos filmes, podendo assumir um dos seguintes valores: 1, 2, 3, 4 ou 5 (melhor *score* possível). A técnica de amostragem utilizada nos experimentos com o *baseline* foi o *Holdout*. Este *baseline* foi projetado de tal forma que, a cada ciclo do *Holdout*, as bituplas temporais utilizadas para treinamento e testes dos algoritmos propostos contêm os mesmos filmes envolvidos no conjunto de amostras de treinamento e no conjunto de amostras de testes dos classificadores. Isso garante que o processo de avaliação seja realizado sobre as mesmas informações.

Além disso, para um classificador conseguir um acerto, não é necessário que ele acerte exatamente o *score* dado pelo usuário para um determinado filme no conjunto de amostras de testes. Por exemplo, suponha que o classificador predisse a classe c_1 para o filme f_1 e a classe c_2 para o filme f_2, onde f_1 e f_2 estão no conjunto de amostras de testes. Neste caso, se f_1 foi melhor avaliado que f_2 pelo usuário, então é suficiente que $c_1 > c_2$ para o classificador conseguir um acerto. Por outro lado, se f_1 foi melhor avaliado que f_2 pelo usuário e $c_1 = c_2$, então o mecanismo de *baseline* infere que o par (f_1, f_2) é incomparável pelo classificador.

Considerou-se um teste t em par de *Student*, com uma distribuição ***unicaudal***[4], para medir a significância estatística das diferenças entre a acurácia (e também a taxa de comparabilidade (TC)) do algoritmo FPSMining (e IncFPSMining) e a respectiva medida para cada um dos algoritmos de *baseline*. A técnica utilizada para calcular a significância estatística, descrita em [Tan et al. 2005] (originalmente projetada para comparar classificadores no cenário *batch*), foi adaptada para a tarefa de mineração de preferências em *data stream*. Para cada uma das três medidas de qualidade d (onde $d \in \{acc, TC, prec\}$), seja $\overline{d}$ a diferença entre a medida d do algoritmo FPSMining (e IncFPSMining) e a respectiva medida d de um dado algoritmo de *baseline*. O principal objetivo do teste t é calcular o intervalo de confiança γ para d e testar se a Hipótese Nula (H0) é rejeitada. O intervalo de confiança γ é calculado da seguinte maneira: $\gamma = t_{(1-\alpha),n_{eval}-1} \times \hat{\sigma}_d$, onde $t_{(1-\alpha),n_{eval}-1}$ é um coeficiente obtido na tabela de distribuição t com $(1-\alpha)$ sendo o nível do teste t e $n_{eval} - 1$ sendo o grau de liberdade. Em todos os experimentos, considerou-se $\alpha = 0,95$ e n_{eval} variou de acordo com o tamanho do *stream*. A significância estatística de $\overline{d}$ foi avaliada, onde $d = acc$ e $d = TC$, uma vez que estas métricas são avaliadas sobre amostras de igual tamanho para ambos os algoritmos.

A Tabela 9.1 compara a performance do algoritmo FPSMining, através da metodologia de construção de *baseline* descrita anteriormente, com dois classificadores amplamente utilizados como *baselines* na literatura: *Hoeffding Tree* (HT) e *Naive Bayes* (NB). O algoritmo IncFPSMining (versão 1) foi submetido a esses mesmos dados. Os algoritmos FPSMining e IncFPSMining permaneceram estatisticamente empatados sobre esses dados, ou seja, as hipóteses de que um desses algoritmos é estritamente melhor do que o

[4] O capítulo 7 (mais especificamente, a subseção "*Hypothesis Testing and Type I Errors*") do livro [Urdan 2010] apresenta uma discussão a respeito da diferença dos testes unicaudal e bicaudal.

		FPSMining			Hoeffding Tree					Naive Bayes				
Usuário	Bituplas	$\overline{acc}$	$\overline{TC}$	$\overline{prec}$	$\overline{acc}$	γ	$\overline{TC}$	γ	$\overline{prec}$	$\overline{acc}$	γ	$\overline{TC}$	γ	$\overline{prec}$
U_1	183.600	0,65	0,93	0,70	0,36	0,03	0,64	0,02	0,54	0,35	0,02	0,49	0,02	0,71
U_2	178.500	0,58	0,82	0,70	0,41	0,03	0,66	0,05	0,63	0,39	0,03	0,56	0,05	0,69
U_3	127.500	0,59	0,94	0,63	0,32	0,04	0,59	0,04	0,52	0,26	0,03	0,40	0,05	0,66
U_4	112.200	0,60	0,93	0,64	0,37	0,05	0,66	0,04	0,54	0,28	0,05	0,47	0,05	0,59
U_5	112.200	0,60	0,90	0,66	0,33	0,03	0,63	0,03	0,52	0,24	0,04	0,40	0,06	0,61
U_6	102.000	0,59	0,94	0,62	0,31	0,05	0,55	0,08	0,48	0,24	0,05	0,36	0,05	0,69
U_7	91.800	0,68	0,91	0,74	0,39	0,03	0,66	0,03	0,55	0,39	0,02	0,54	0,04	0,72
U_8	86.700	0,61	0,90	0,69	0,36	0,06	0,60	0,08	0,52	0,35	0,02	0,54	0,03	0,65
U_9	76.500	0,58	0,90	0,64	0,32	0,07	0,64	0,08	0,47	0,30	0,05	0,51	0,08	0,59
U_{10}	76.500	0,61	0,91	0,66	0,39	0,07	0,62	0,07	0,58	0,30	0,05	0,48	0,04	0,61

Tabela 9.1: Baseline sobre Dados Reais

outro e vice-versa foram rejeitadas. Adicionalmente, também foi calculada a significância estatística da diferença de performance do algoritmo IncFPSMining e dos dois classificadores, e os resultados foram muito semelhantes aos apresentados pelo FPSMining. Por esta razão, foram apresentados somente os resultados referentes ao algoritmo FPSMining nestes testes.

Para estes experimentos, utilizou-se a implementação disponível no MOA [Bifet et al. 2010] para esses dois algoritmos de *baseline*, nas suas configurações padrão de parâmetros[5]. Os valores para os parâmetros do *Holdout* foram $n_{train} = 5k$ e $n_{test} = 100$. Para cada usuário U_i, calculou-se o valor de n_{eval} de forma a cobrir o *stream* S_i inteiro, e os seguintes valores foram obtidos: $U_1 = 36$, $U_2 = 35$, $U_3 = 25$, $U_4 = 22$, $U_5 = 22$, $U_6 = 20$, $U_7 = 18$, $U_8 = 17$, $U_9 = 15$ e $U_{10} = 15$. É importante destacar que para o algoritmo FPSMining foram utilizados $\alpha_1 = 0,2$ e $\alpha_2 = 0,1$ como valores padrão para os parâmetros do Algoritmo 2, referente ao cálculo do grau de dependência entre os atributos.

Nestes experimentos, a pergunta de interesse é dada por: Com $\alpha = 95\%$ de certeza, pode-se concluir que o algoritmo FPSMining supera os outros algoritmos utilizados como *baseline*? As hipóteses nula e alternativa para *acc* e *TC* são $H0 : FPSMining \leq HT$, NB e $HA : FPSMining > HT$, NB. Os resultados mostram que $H0$ é rejeitada (uma vez que todos os valores contidos no intervalo de confiança são positivos) e, portanto, HA é substanciada. Isso mostra que, ainda com *streams* limitados, o algoritmo FPSMining apresenta um bom desempenho, e supera ambos os algoritmos utilizados como *baselines*. Os resultados ruins produzidos por esses classificadores sugerem que classificadores não são muito adequados para tarefas de mineração de preferências. Assim, isso mostra que o algoritmo FPSMining, um algoritmo especificamente projetado para minerar preferências, obteve um desempenho melhor do que classificadores clássicos adaptados para a tarefa de mineração de preferências.

Apesar dos dados reais utilizados nestes experimentos se adequarem perfeitamente

[5] *Hoeffding Tree: gracePeriod* $g = 200$, *splitConfidence* $c = 10^{-7}$, *tieThreshold* $t = 0,05$, *numericEstimator* $n = GAUSS10$. *Naive Bayes*: não possui parâmetros.

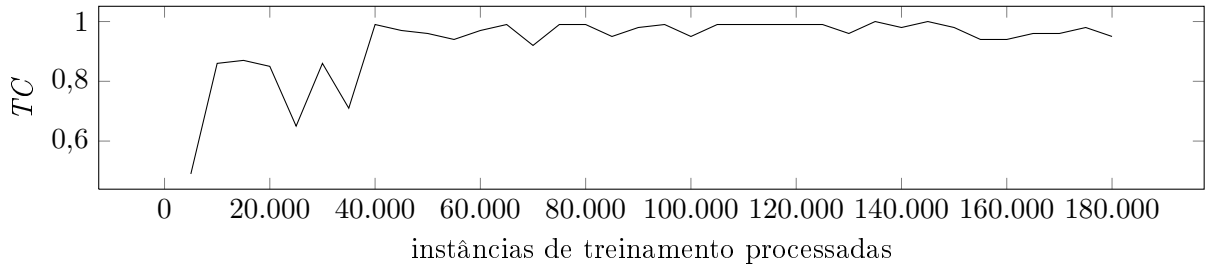


Figura 9.3: Evolução da Taxa de Comparabilidade do FPSMining sobre Dados Reais

para o cenário de preferências, o número de filmes avaliados pelos usuários é ainda muito pequeno para o cenário de *streams*. Em uma tentativa de amenizar isso, nestes experimentos teve-se o cuidado de considerar apenas os usuários que mais avaliaram filmes nesse conjunto de dados.

A seguir, é discutida em mais detalhes a performance do algoritmo FPSMining sobre um dos *streams* utilizados neste *baseline*.

Análise de Performance do algoritmo FPSMining. A Figura 9.3 mostra a evolução da taxa de comparabilidade (TC) do algoritmo FPSMining sobre o número de instâncias de treinamento processadas. Neste teste, utilizou-se o *stream* S_1 (referente ao usuário U_1 do *baseline* realizado) e plotou-se todos os valores de TC utilizados para calcular a média $\overline{TC}$ de 93%, mostrada na Tabela 9.1. Note que, com poucas instâncias de treinamento processadas, o algoritmo FPSMining não pôde comparar muitos pares de filmes, devido à informação limitada que havia recebido até o momento. Com o aumento do número de instâncias de treinamento processadas, o algoritmo FPSMining pôde aprender mais sobre as preferências dos usuários e, portanto, foi capaz de comparar mais pares de filmes. Note também que, a partir de 40.000 instâncias de treinamento processadas, o algoritmo FPSMining foi capaz de comparar quase todos os pares de filmes que foram apresentados a ele.

9.3 Experimentos com Dados Sintéticos

Com o intuito de avaliar a habilidade dos algoritmos implementados em lidar com grandes volumes de dados e com *concept drifts*, foram executados testes experimentais sobre dados sintéticos. Os dados sintéticos foram gerados pelo gerador[6] de *stream* sintético de preferências proposto na seção 8.2.

Esta seção foi dividida em duas partes: a primeira parte é referente a experimentos em *streams* com distribuição estacionária e a segunda parte é referente a experimentos com todos os tipos de *concept drifts* propostos na seção 8.1.

[6]Disponível em `http://lsi.facom.ufu.br/dataStreamMining/syntheticGenerator`

9.3.1 Streams sem Concept Drift

Estes testes têm o intuito de avaliar tanto a escalabilidade dos algoritmos implementados, quanto sua qualidade em *streams* com distribuição estacionária. Nestes testes, foram consideradas diferentes estruturas de grafo (variando o número de nós) e tabelas de probabilidade condicional, usando a configuração mostrada abaixo. Dessa forma, foram executados testes com *streams* de até 100 milhões[7] de elementos.

# atributos	|dom| por atributo	|stream|
6, 8, 10, 12, 14, 16	10 elementos	10m, 25m, 50m, 75m, 100m

Nestes experimentos, utilizou-se o mesmo teste estatístico considerado na seção anterior (seção 9.2) para o cálculo da significância estatística. Além disso, utilizou-se a técnica *Naive Blind* (com MAX_DOWNGRADE = 10k) apenas a título de gerenciamento de memória das estatísticas suficientes. Os valores padrão para os parâmetros do *Holdout* foram: n_{train} = 10k e n_{test} = 1k. Para os testes com 10m, 25m, 50m, 75m e 100m de elementos, foram utilizados valores de n_{eval} de forma a considerar o *stream* inteiro: $n_{eval_{10m}} = 909$ (obtido por: 10m / (10k + 1k)), $n_{eval_{25m}} = 2272$, $n_{eval_{50m}} = 4545$, $n_{eval_{75m}} = 6818$ e $n_{eval_{100m}} = 9090$. No cálculo do grau de dependência, utilizou-se $\alpha_1 = 0,2$ e $\alpha_2 = 0,1$. Além disso, os valores padrão utilizados no algoritmo IncFPSMining (versão 1) foram: *grace period* = 1k e $\delta = 10^{-7}$.

A Tabela 9.2 e a Figura 9.4 apresentam os resultados obtidos com esses testes. A seguir, esses resultados são analisados nos âmbitos de performance e tempo de execução.

1. Análise de Performance

As Tabelas 9.2(a) e (b) mostram os valores médios de *acc*, *TC* e *prec* obtidos com os algoritmos IncFPSMining e FPSMining para *streams* com diferentes números de atributos e tamanhos, respectivamente. Na Tabela 9.2(a), utilizou-se 50m de elementos como tamanho padrão do *stream*, e na Tabela 9.2(b), utilizou-se 10 atributos como número de atributos padrão. Note que a qualidade de ambos os algoritmos permaneceu estável sobre os diferentes *streams*. Além disso, os dois algoritmos analisados conseguiram comparar praticamente todas as bituplas de testes a que eles foram submetidos (coluna $\overline{TC}$). Nestes testes, também foi calculada a significância estatística considerando a leve melhora apresentada pelo algoritmo IncFPSMining comparado ao algoritmo FPSMining. A pergunta de interesse aqui é: Com $\alpha = 95\%$, pode-se concluir que o algoritmo IncFPSMining supera o algoritmo FPSMining? As hipóteses nula e alternativa para *acc* e *TC* são $H0 : IncFPSMining \leq FPSMining$ e $HA : IncFPSMining > FPSMining$. Os resultados mostram que $H0$ é rejeitada e, portanto, HA é substanciada. Assim, embora a diferença de performance entre os dois algoritmos seja extremamente pequena, ela é estatisticamente significante, visto que o número de elementos do *stream* é muito grande.

[7] No texto desta dissertação, geralmente abreviou-se *milhões* por *m*.

	IncFPSMining			**FPSMining**				
# atributos	$\overline{acc}$	$\overline{TC}$	$\overline{prec}$	$\overline{acc}$	γ	$\overline{TC}$	γ	$\overline{prec}$
6	0,95168	1	0,95168	0,94915	2×10^{-4}	0,99741	2×10^{-4}	0,95160
8	0,94936	1	0,94936	0,94322	4×10^{-4}	0,99347	4×10^{-4}	0,94942
10	0,95239	1	0,95239	0,95162	1×10^{-4}	0,99919	1×10^{-4}	0,95239
12	0,95059	1	0,95059	0,94831	3×10^{-4}	0,99762	3×10^{-4}	0,95057
14	0,95053	1	0,95053	0,94858	2×10^{-4}	0,99795	3×10^{-4}	0,95053
16	0,95115	1	0,95115	0,94932	2×10^{-4}	0,99808	2×10^{-4}	0,95115

(a)

	IncFPSMining			**FPSMining**				
|stream|	$\overline{acc}$	$\overline{TC}$	$\overline{prec}$	$\overline{acc}$	γ	$\overline{TC}$	γ	$\overline{prec}$
10m	0,95236	1	0,95236	0,95151	3×10^{-4}	0,99908	3×10^{-4}	0,95238
25m	0,95243	1	0,95243	0,95165	1×10^{-4}	0,99916	1×10^{-4}	0,95244
50m	0,95239	1	0,95239	0,95162	1×10^{-4}	0,99919	1×10^{-4}	0,95239
75m	0,95235	1	0,95235	0,95155	1×10^{-4}	0,99915	1×10^{-4}	0,95235
100m	0,95240	1	0,95240	0,95160	9×10^{-5}	0,99916	9×10^{-5}	0,95240

(b)

Tabela 9.2: Comparação dos algoritmos IncFPSMining e FPSMining sobre Dados Sintéticos sem *Concept Drift*

Com o intuito de fornecer mais detalhes sobre a análise de performance efetuada, as Figuras 9.4(a) e (b) ilustram como os valores de *acc* e *prec* do algoritmo FPSMining evoluem sobre o tempo. Para estes testes foi escolhido o algoritmo FPSMining, por ele ter apresentado menor qualidade dentre os dois algoritmos analisados. Neste experimento, considerou-se o *stream* com 100m de elementos e 10 atributos da Tabela 9.2(b). As curvas de aprendizado para *acc* e *prec* são muito similares, com exceção de alguns poucos picos inferiores apresentados pelo *acc*, evidenciando que os valores de *prec* são ligeiramente melhores. Essas curvas mostraram que os valores de *acc* e *prec* do algoritmo FPSMining foram razoavelmente estáveis sobre o tempo.

2. Análise do Tempo de Execução

A Figura 9.4(c) mostra o tempo, medido em segundos, gasto pelo algoritmo FPSMining para gerar o modelo em cada ponto de atualização. Os mesmos dados utilizados nos experimentos (a) e (b) foram considerados aqui. Este *stream* produz cerca de 20 GB de dados. Note que o tempo gasto para atualizar o modelo é muito pequeno, na ordem de milissegundos. Embora o algoritmo FPSMining construa a RBP inteira a cada ciclo do *Holdout*, as estatísticas suficientes são atualizadas incrementalmente, isto é, elas estão sempre prontas para uso, o que torna este processo rápido. Note também que o tempo permaneceu praticamente constante com o aumento do número de instâncias de treinamento processadas, principalmente devido à eficiência do gerenciamento de memória proporcionado pelo uso da técnica *Naive Blind*. Assim, esta figura mostra que o algoritmo FPSMining é rápido e satisfaz as restrições de tempo do cenário de *streams*.

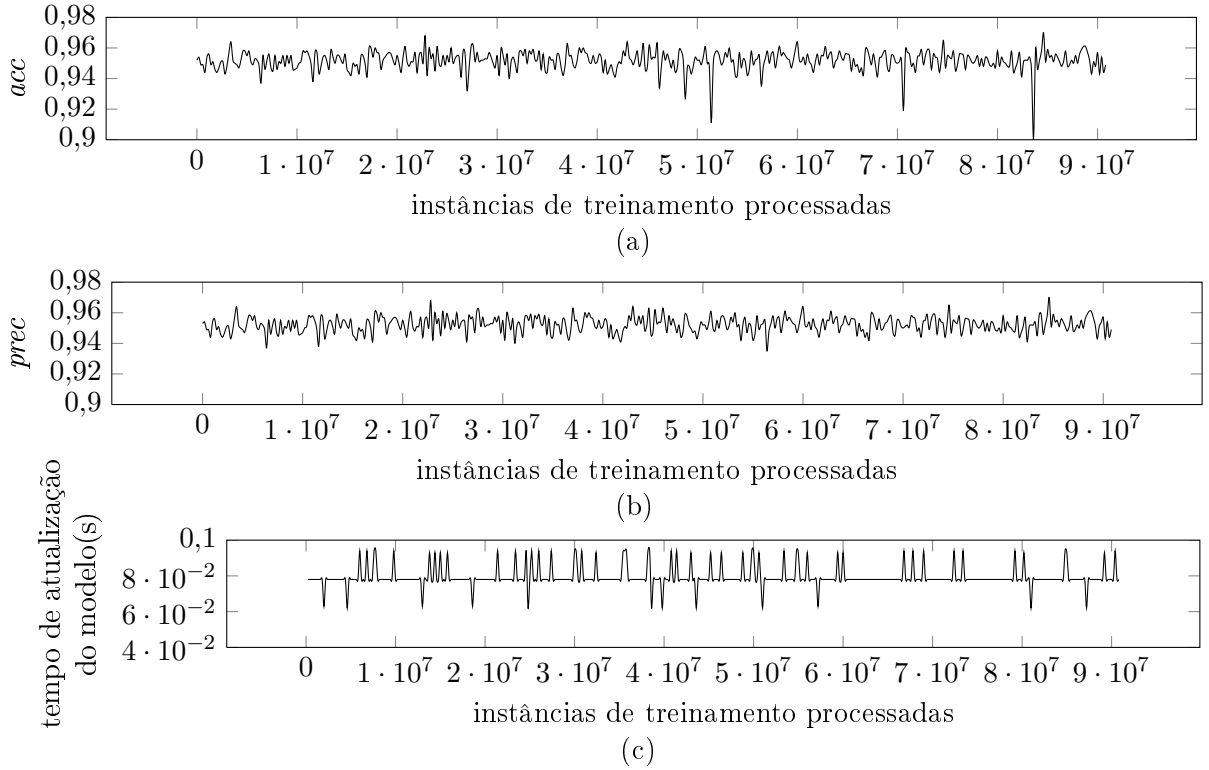


Figura 9.4: Detalhes da execução do FPSMining sobre Dados Sintéticos sem *Concept Drift*

9.3.2 Streams com Concept Drift

Com o intuito de avaliar os algoritmos FPSMining e IncFPSMining (versão 2) em *streams* com *concept drift*, utilizou-se o gerador de *stream* sintético de preferências proposto na seção 8.2. Esse gerador contempla todos os seis tipos distintos de *concept drift* em preferências contextuais propostos na seção 8.1. Na presente seção, é apresentado um amplo conjunto de testes que exploram o comportamento dos algoritmos propostos frente a diferentes tipos de *concept drift*. Ainda nesta seção, são apresentados e analisados os resultados referentes à performance dos algoritmos propostos (1) sem a introdução de técnica para tratamento de *concept drift* e (2) com a introdução das técnicas *Naive Blind* e *Exponential Forgetting*, descritas também no capítulo 8 (seção 8.3).

A seguir, são apresentados e discutidos os resultados dos experimentos realizados. Um total de nove conjuntos de testes foi executado. Os seis primeiros conjuntos de testes envolvem a análise aprofundada de cada um dos tipos de *concept drift* individualmente. O sétimo conjunto de testes envolve a análise detalhada de recuperação do algoritmo FPSMining frente à ocorrência de um *concept drift*. O oitavo conjunto de testes envolve a ocorrência de vários tipos distintos de *concept drifts* em um mesmo *stream*. Por fim, o nono conjunto de testes apresenta os resultados referentes ao tempo gasto pelos algoritmos para atualizar o modelo a cada *grace period*.

Os resultados da maioria dos conjuntos de testes foram dispostos em três tabelas (a, b, c), sendo uma para cada medida de interesse: acurácia (*acc*), taxa de comparabilidade

(TC), e precisão ($prec$). Note que cada um dos valores nessas tabelas foi calculado de acordo com a explicação realizada na subseção 9.1.2, ou seja, utilizando o protocolo *Prequential*. Cada uma das tabelas apresenta a performance tanto do algoritmo FPSMining quanto do algoritmo IncFPSMining, em três abordagens distintas: algoritmos *puros* (sem nenhuma técnica para tratamento de *concept drift*) e com a introdução das técnicas *Naive Blind* e *Exponential Forgetting*.

Em relação ao gerador de *stream* sintético de preferências, os parâmetros comuns utilizados nos nove conjuntos de testes foram: ATTRIBUTE_QTY = 10, DOMAIN_QTY = 10, LEVEL_QTY = 4, PARENT_MIN_QTY = 1, BITUPLE_QTY = 100k, SEED = 1L, PERC_MUT_STREAM = 0,01% e MIN_PHASE_QTY = 10k. Assim, nestes testes, o tamanho do *stream* foi configurado para 100k, de forma a tornar os testes mais gerenciáveis em relação à clareza do comportamento dos algoritmos propostos frente aos *concept drifts*. A combinação dos parâmetros BITUPLE_QTY = 100k e PERC_MUT_STREAM = 0,01% indica que devem ocorrer dez *concept drifts* ao longo do *stream*. Por outro lado, a adição do parâmetro MIN_PHASE_QTY = 10k permite avaliar o comportamento dos algoritmos propostos frente a exatos nove *concept drifts*, cada um ocorrendo a cada 10k elementos do *stream* de preferências (o décimo *concept drift* ocorre no último elemento do *stream* e portanto não deve ser contabilizado). Todos esses parâmetros do gerador de *stream* sintético de preferências são explicados na seção 8.2.

Para os algoritmos FPSMining e IncFPSMining, os parâmetros comuns utilizados nos nove conjuntos de testes foram: *grace period* = 200, α_1 e α_2 (utilizados no cálculo do grau de dependência) iguais a 0,01 para o FPSMining e 0,028 para o IncFPSMining (melhor configuração de parâmetros obtida em testes experimentais). Para o algoritmo IncFPSMining, utilizou-se $\delta = 10^{-7}$. Para todos os conjuntos de testes, utilizou-se como técnica de amostragem o *Prequential*, com α_PREQUENTIAL = 1. Na técnica *Naive Blind* foi utilizado MAX_DOWNGRADE = 10k, e na técnica *Exponential Forgetting* foram utilizados MIN_STATISTICS = 0,5 e α_STATISTICS = 0,9999.

O ***Conjunto 1*** de testes é referente ao *Tipo 1* (Surgimento de Preferência) de *concept drift* (CD = CD_MUT_CREATE_PREFERENCE), explicado em detalhes na seção 8.1. Neste conjunto de testes, fixou-se PERC_CREATE_PREFERENCE = 50% e variou-se ATTRIBUTE_DRIFT_QTY = 1, 2, 3, 4, 5 e 6 atributos, com o intuito de avaliar o impacto do aumento do número de atributos envolvidos com a ocorrência de cada *concept drift*. Por exemplo, se ATTRIBUTE_DRIFT_QTY = 6, então significa que a cada ocorrência de *concept drift* devem ser impactados seis atributos aleatórios do grafo da RBP geradora dos dados, e a mudança deve consistir no acréscimo de 50% de novas regras na tabela de probabilidade condicional referente a cada um dos seis atributos. A Tabela 9.3 exibe os resultados obtidos com o *Conjunto 1* de testes, referente às métricas de acurácia (acc), taxa de comparabilidade (TC) e precisão ($prec$).

De acordo com a Tabela 9.3(a), os valores de acc dos algoritmos FPSMining e IncFPS-

ATTRIBUTE DRIFT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 atributo	0,91207	0,87248	0,89230	0,92199	0,88687	0,90565
2 atributos	0,89831	0,85704	0,88008	0,91187	0,87195	0,89408
3 atributos	0,89973	0,87267	0,88532	0,91035	0,88173	0,89367
4 atributos	0,89307	0,88326	0,89150	0,90371	0,88994	0,89780
5 atributos	0,89699	0,87225	0,88440	0,90102	0,88109	0,89315
6 atributos	0,89651	0,86528	0,87676	0,88871	0,87324	0,88608

(a) *acc*

ATTRIBUTE DRIFT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 atributo	0,96399	0,92353	0,94439	0,97311	0,93566	0,95479
2 atributos	0,95276	0,91018	0,93496	0,96550	0,92327	0,94596
3 atributos	0,95853	0,92611	0,93977	0,96778	0,93819	0,95064
4 atributos	0,95700	0,93961	0,94825	0,96564	0,95192	0,96006
5 atributos	0,95825	0,93111	0,94408	0,95912	0,94045	0,95324
6 atributos	0,96150	0,92959	0,94166	0,95093	0,93750	0,95061

(b) *TC*

ATTRIBUTE DRIFT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 atributo	0,94614	0,94472	0,94484	0,94747	0,94785	0,94853
2 atributos	0,94285	0,94162	0,94130	0,94445	0,94441	0,94516
3 atributos	0,93866	0,94230	0,94206	0,94066	0,93982	0,94007
4 atributos	0,93320	0,94003	0,94015	0,93587	0,93489	0,93515
5 atributos	0,93607	0,93679	0,93679	0,93942	0,93688	0,93696
6 atributos	0,93241	0,93082	0,93108	0,93457	0,93146	0,93212

(c) *prec*

Tabela 9.3: Conjunto de Testes 1 – Surgimento de preferência com variação do número de atributos com *concept drift*

Mining permaneceram razoavelmente estáveis para os seis diferentes *streams*, com pequenas variações. No algoritmo IncFPSMining com a utilização da técnica *Exponential Forgetting*, é possível notar um comportamento levemente decrescente dos valores de *acc* frente ao aumento do número de atributos com *concept drift*, mas a variação apresentada ainda é muito sutil. Assim, esse tipo de *concept drift* parece não acarretar em grandes impactos na qualidade dos algoritmos analisados.

Além disso, a variação do número de atributos com *concept drift* passou praticamente despercebida pelos algoritmos de mineração. Provavelmente isso ocorreu porque, apesar de se aumentar em 50% o número de regras das tabelas de probabilidade condicional de cada um dos atributos com *concept drift* na RBP geradora dos dados, dois terços das regras totais das tabelas em questão, após a mutação, ainda consistem de regras já conhecidas pelos algoritmos. Além disso, a estrutura do grafo da RBP geradora dos

dados não é alterada por este tipo de *concept drift*. Dessa forma, a cada novo elemento do *stream*, gerado de acordo com uma das regras criadas, inicialmente os algoritmos propostos provavelmente não conseguirão distinguir a tupla preferida, mas rapidamente devem aprendê-la.

Note que em ambos os algoritmos a técnica *Exponential Forgetting* apresentou resultados levemente melhores do que as demais abordagens. Isso já era de se esperar, visto que na técnica *Exponential Forgetting* os elementos mais recentes do *stream* possuem um peso maior em relação aos mais antigos, o que evidencia mais facilmente o aparecimento de novas regras. Porém, neste conjunto de testes, a técnica *Naive Blind* apresentou resultados levemente piores do que os algoritmos puros, possivelmente porque sua abordagem consiste em eliminar lotes de regras antigas e com baixíssima frequência, mas note que este tipo de *concept drift* não elimina as regras já existentes nas tabelas de probabilidade condicional, apenas insere novas. Por outro lado, a vantagem da utilização da técnica *Naive Blind* em relação aos algoritmos puros consiste no gerenciamento de memória das estatísticas suficientes, que é consequência de sua execução.

Observe também que o algoritmo IncFPSMining apresentou, na grande maioria das vezes, uma performance levemente superior a do algoritmo FPSMining, possivelmente devido à construção incremental de suas tabelas de probabilidade condicional. Com isso, ele consegue aprender mais rapidamente novas regras, ao contrário do algoritmo FPSMining, onde isso ocorre apenas a cada *grace period*.

Em relação às Tabelas 9.3(b) e (c), os valores de TC e *prec* dos algoritmos FPSMining e IncFPSMining também permaneceram razoavelmente estáveis para os seis diferentes *streams*, com pequenas variações. Isso é decorrência dos fatos explicados anteriormente. Entretanto, é interessante ressaltar dois pontos em relação a essas tabelas: 1) Note que na Tabela 9.3(b) o algoritmo IncFPSMining apresenta valores de TC levemente superiores aos do algoritmo FPSMining (exceto para um único valor dos 18 valores apresentados), o que ratifica a afirmação anterior em relação ao benefício da construção incremental das tabelas de probabilidade condicional para este tipo de *concept drift*. Note ainda que a maior taxa de comparabilidade apresentada nesta tabela é referente ao algoritmo IncFPSMining com a técnica *Exponential Forgetting* e ATTRIBUTE_DRIFT_QTY = 1 atributo; 2) Na Tabela 9.3(c), os valores de *prec* foram muito semelhantes entre as diferentes abordagens e os dois algoritmos analisados, variando em cerca de 93% e 94%.

Ainda com o *Tipo 1* de *concept drift*, realizou-se também experimentos fixando ATTRIBUTE_DRIFT_QTY = 5 atributos e variando PERC_CREATE_PREFERENCE = 10%, 20%, 30%, 40%, 50% e 60%, com o intuito de avaliar o impacto do aumento do percentual de regras a serem criadas dentro da tabela de probabilidade condicional de cada atributo com *concept drift*. Os resultados obtidos foram muito semelhantes aos resultados apresentados na Tabela 9.3.

O ***Conjunto 2*** de testes é referente ao *Tipo 2* (Desaparecimento de Preferência) de

concept drift (CD = CD_MUT_REMOVE_PREFERENCE). Neste conjunto de testes, fixou-se PERC_REMOVE_PREFERENCE = 50% e variou-se ATTRIBUTE_DRIFT_QTY = 1, 2, 3, 4, 5 e 6 atributos, com o intuito de avaliar o impacto do aumento do número de atributos com *concept drift*. Por exemplo, se ATTRIBUTE_DRIFT_QTY = 6, então significa que a cada ocorrência de *concept drift* devem ser impactados seis atributos aleatórios do grafo da RBP geradora dos dados, e a mudança deve consistir na diminuição de 50% das regras da tabela de probabilidade condicional referente a cada um dos seis atributos. A Tabela 9.4 exibe os resultados obtidos com o *Conjunto 2* de testes.

ATTRIBUTE DRIFT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 atributo	0,93208	0,88979	0,90237	0,94754	0,90334	0,91548
2 atributos	0,92888	0,90222	0,90825	0,94512	0,91629	0,92169
3 atributos	0,93077	0,91121	0,91464	0,94681	0,92495	0,92786
4 atributos	0,93175	0,90883	0,91420	0,94723	0,92177	0,92708
5 atributos	0,92946	0,91256	0,91459	0,94480	0,92634	0,92783
6 atributos	0,93240	0,91098	0,91459	0,94865	0,92600	0,92911

(a) *acc*

ATTRIBUTE DRIFT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 atributo	0,98077	0,93704	0,95030	0,99563	0,94855	0,96094
2 atributos	0,97998	0,95265	0,95891	0,99563	0,96456	0,97006
3 atributos	0,98049	0,96048	0,96408	0,99621	0,97241	0,97540
4 atributos	0,98157	0,95810	0,96365	0,99693	0,96884	0,97434
5 atributos	0,98089	0,96349	0,96570	0,99567	0,97639	0,97793
6 atributos	0,98009	0,95809	0,96216	0,99636	0,97238	0,97567

(b) *TC*

ATTRIBUTE DRIFT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 atributo	0,95036	0,94958	0,94956	0,95170	0,95234	0,95269
2 atributos	0,94786	0,94706	0,94717	0,94927	0,94996	0,95014
3 atributos	0,94929	0,94870	0,94872	0,95041	0,95119	0,95126
4 atributos	0,94924	0,94858	0,94868	0,95015	0,95142	0,95150
5 atributos	0,94757	0,94714	0,94707	0,94891	0,94874	0,94877
6 atributos	0,95134	0,95083	0,95056	0,95212	0,95230	0,95228

(c) *prec*

Tabela 9.4: Conjunto de Testes 2 – Desaparecimento de preferência com variação do número de atributos com *concept drift*

De acordo com os resultados dispostos na Tabela 9.4(a), os valores de *acc* dos algoritmos analisados também permaneceram praticamente estáveis dentro de cada uma das três abordagens. Os valores de *acc* obtidos foram altos, e assim permaneceram com o aumento do número de atributos com *concept drift*. Observe que os valores de *acc* dos

algoritmos frente a esse tipo de *concept drift* foram superiores aos valores de *acc* frente a *concept drifts* do *Tipo 1* (Surgimento de Preferência). Isso porque este tipo de mutação apenas remove regras da tabela de probabilidade condicional dos atributos com *concept drifts*, mas não introduz nenhuma novidade que impacte diretamente no aprendizado dos algoritmos. Dessa forma, era de se esperar que este tipo de *concept drift* não resultasse em piora na qualidade dos algoritmos, muito pelo contrário, uma vez que um conjunto menor e frequente de informações, ocasionado pela redução de 50% das regras dos atributos com *concept drift* na RBP geradora dos dados, tende a facilitar o aprendizado do modelo. Tanto que, considerando apenas as duas primeiras casas decimais, pode-se observar, em grande parte dos casos, um comportamento levemente crescente na qualidade dos algoritmos frente ao aumento do número de atributos com *concept drifts*. Além disso, novamente nota-se uma leve melhora da técnica *Exponential Forgetting* em relação às outras abordagens, e um desempenho bastante similar entre a técnica *Naive Blind* e os algoritmos puros.

Em relação à Tabela 9.4(b), vale ressaltar que o algoritmo IncFPSMining com a técnica *Exponential Forgetting* conseguiu comparar praticamente todas as bituplas temporais apresentadas a ele, resultando em valores de TC acima de 99%. Como o conjunto de regras da RBP foi sucessivamente reduzido, era de se esperar que a taxa de comparabilidade dos algoritmos fosse alta. Além disso, o uso da técnica *Exponential Forgetting* possivelmente ajustou corretamente a estrutura de grafo do modelo (resultante dos cálculos de grau de dependência entre os atributos), o que fez com que sua taxa de comparabilidade fosse maior do que simplesmente utilizando o algoritmo puro.

Por fim, a Tabela 9.4(c) mostra que os valores de *prec* dos algoritmos analisados sob as diferentes abordagens variaram pouquíssimo entre 94% e 95%, o que mostra um alto índice de acerto dentre os elementos que o modelo conseguiu comparar.

Ainda com o *Tipo 2* de *concept drift*, realizou-se também experimentos fixando ATTRIBUTE_DRIFT_QTY = 5 atributos e variando PERC_REMOVE_PREFERENCE = 10%, 20%, 30%, 40%, 50% e 60%, com o intuito de avaliar o impacto do aumento do percentual de regras a serem removidas dentro da tabela de probabilidade condicional de cada atributo com *concept drift*. Os resultados obtidos foram muito semelhantes aos resultados apresentados na Tabela 9.4.

O ***Conjunto 3*** de testes é referente ao *Tipo 3* (Inversão de Preferência) de *concept drift* (CD = CD_MUT_INVERT_PREFERENCE). Neste conjunto de testes, fixou-se PERC_INVERT_PREFERENCE = 50% e variou-se ATTRIBUTE_DRIFT_QTY = 1, 2, 3, 4, 5 e 6 atributos, com o intuito de avaliar o impacto do aumento do número de atributos com *concept drift*. Por exemplo, se ATTRIBUTE_DRIFT_QTY = 6, então significa que a cada ocorrência de *concept drift* devem ser impactados seis atributos aleatórios do grafo da RBP geradora dos dados, e a mudança deve consistir no complemento da probabilidade X (após a mudança ficará "$1 - X$") de 50% das regras da tabela de

probabilidade condicional referente a cada um dos seis atributos. A Tabela 9.5 exibe os resultados obtidos com o *Conjunto 3* de testes.

ATTRIBUTE DRIFT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 atributo	0,87947	0,78138	0,80294	0,89768	0,79396	0,81309
2 atributos	0,84520	0,74109	0,76235	0,86428	0,75363	0,77010
3 atributos	0,82025	0,69287	0,71258	0,83925	0,70314	0,71836
4 atributos	0,79436	0,66120	0,67859	0,81325	0,67196	0,68552
5 atributos	0,74816	0,61400	0,63050	0,78628	0,62472	0,63769
6 atributos	0,72732	0,56526	0,58234	0,76174	0,57784	0,59171

(a) *acc*

ATTRIBUTE DRIFT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 atributo	0,97712	0,90173	0,93878	0,99498	0,91325	0,94368
2 atributos	0,97694	0,90146	0,93727	0,99495	0,91291	0,94025
3 atributos	0,97696	0,90152	0,93774	0,99496	0,91285	0,94098
4 atributos	0,97694	0,90098	0,93632	0,99496	0,91235	0,94087
5 atributos	0,97677	0,90083	0,93609	0,99103	0,91149	0,94007
6 atributos	0,97697	0,90016	0,93511	0,99097	0,91171	0,94020

(b) *TC*

ATTRIBUTE DRIFT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 atributo	0,90006	0,86653	0,85530	0,90221	0,86938	0,86162
2 atributos	0,86515	0,82210	0,81337	0,86867	0,82552	0,81904
3 atributos	0,83959	0,76856	0,75989	0,84350	0,77027	0,76342
4 atributos	0,81311	0,73387	0,72474	0,81737	0,73652	0,72860
5 atributos	0,76595	0,68159	0,67355	0,79340	0,68538	0,67834
6 atributos	0,74447	0,62796	0,62275	0,76868	0,63380	0,62934

(c) *prec*

Tabela 9.5: Conjunto de Testes 3 – Inversão de preferência com variação do número de atributos com *concept drift*

A Tabela 9.5 apresenta resultados bastante interessantes. Observe que na Tabela 9.5(a) todos os valores de *acc* diminuíram gradativamente com o aumento do número de atributos com *concept drift*. Isso ocorreu porque, como já havia sido mencionado na seção 8.1, esse tipo de mudança é um pouco mais drástico em relação às demais referentes a regras de preferências. A inversão de preferência muito provavelmente conduz o modelo ao erro, uma vez que, nessa situação, o modelo já havia aprendido previamente algo que agora está incorreto. Dessa forma, o aumento do número de atributos com *concept drift* impacta diretamente de forma negativa na qualidade dos algoritmos. Note também que a qualidade dos algoritmos com a utilização da técnica *Exponential Forgetting* foi consideravelmente superior à qualidade dos algoritmos puros. Esse conjunto de testes

torna nítido o ganho com qualidade que se tem ao se utilizar esta técnica. Observe também que, apesar da mudança drástica proporcionada por esse tipo de *concept drift*, a qualidade dos algoritmos com a utilização da técnica *Exponential Forgetting* ainda foi razoável. Por outro lado, novamente a utilização da técnica *Naive Blind* apresentou resultados levemente piores do que os algoritmos puros, confirmando não ser muito efetiva em termos de melhora de qualidade para tipos de *concept drift* que envolvem apenas mudanças nas tabelas de probabilidade condicional (e não no grafo da RBP). Apesar disso, como já mencionado anteriormente, a vantagem do uso da técnica *Naive Blind* está no seu baixo uso de memória. Ainda nesta tabela, é possível notar que o decaimento na qualidade dos algoritmos puros com o aumento do número de atributos com *concept drift* é muito mais acelerado do que com a utilização da técnica *Exponential Forgetting*. É possível notar também que novamente o algoritmo IncFPSMining apresentou valores levemente maiores de *acc* em relação ao algoritmo FPSMining, provavelmente devido a uma maior adaptação frente às mudanças, proporcionada, principalmente, pela construção incremental das tabelas de probabilidade condicional.

Os valores de TC da Tabela 9.5(b) apresentaram um comportamento bastante estável para os seis diferentes *streams*, tanto que as suas duas primeiras casas decimais foram iguais dentro de cada abordagem. Os resultados mostraram também que a utilização da técnica *Exponential Forgetting* permitiu comparar mais elementos do que os algoritmos puros (possivelmente devido a uma melhor modelagem do grafo do modelo), que por sua vez obtiveram uma maior taxa de comparabilidade do que a técnica *Naive Blind*. Por outro lado, os valores de *prec* da Tabela 9.5(c) apresentaram um comportamento decrescente e acelerado com o aumento do número de atributos com *concept drift*, seguindo uma distribuição parecida com a seguida pelos valores de *acc*. Isso indica que os baixos valores de *acc* não foram ocasionados por uma taxa de comparabilidade ruim, mas sim por erros cometidos pelos algoritmos durante as comparações realizadas, o que já era de se esperar, visto que diversas regras tiveram sua probabilidade alterada para o seu complemento.

Ainda com o *Tipo 3* de *concept drift*, realizou-se também experimentos fixando ATTRIBUTE_DRIFT_QTY = 5 atributos e variando PERC_INVERT_PREFERENCE = 10%, 20%, 30%, 40%, 50% e 60%, com o intuito de avaliar o impacto do aumento do percentual de regras cuja probabilidade X foi alterada para o seu complemento (após a mudança ficará "$1 - X$") dentro da tabela de probabilidade condicional de cada atributo com *concept drift*. Os resultados obtidos foram muito semelhantes aos resultados apresentados na Tabela 9.5, o que mostra que esse tipo de *concept drift* impacta diretamente na qualidade dos algoritmos.

O ***Conjunto 4*** de testes é referente ao *Tipo 4* (Surgimento de Dependência) de *concept drift* (CD = CD_MUT_CREATE_DEPENDENCY). Neste conjunto de testes, variou-se o parâmetro EDGE_CREATE_QTY = 1, 2, 3, 4, 5 e 6 arestas, com o intuito de avaliar o impacto do aumento do número de dependências criadas entre os atributos

do *stream* de preferências. Por exemplo, se EDGE_CREATE_QTY = 6, então significa que a cada ocorrência de *concept drift* devem ser inseridas seis arestas aleatórias no grafo da RBP geradora dos dados. A Tabela 9.6 exibe os resultados obtidos com o *Conjunto 4* de testes.

EDGE CREATE QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 aresta	0,92842	0,80526	0,82382	0,92461	0,85614	0,87133
2 arestas	0,92980	0,81009	0,81905	0,90035	0,84412	0,85763
3 arestas	0,92599	0,77038	0,81477	0,89310	0,84846	0,88858
4 arestas	0,92892	0,68864	0,71203	0,90128	0,86283	0,88269
5 arestas	0,92875	0,75881	0,75158	0,89916	0,84798	0,86953
6 arestas	0,92419	0,85707	0,85044	0,90408	0,90013	0,91297

(a) *acc*

EDGE CREATE QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 aresta	0,97515	0,84677	0,86701	0,96984	0,89727	0,91348
2 arestas	0,97618	0,85146	0,86118	0,94471	0,88479	0,89866
3 arestas	0,97364	0,81237	0,85963	0,93872	0,89059	0,93173
4 arestas	0,97517	0,72516	0,75039	0,94539	0,90358	0,92427
5 arestas	0,97473	0,79692	0,79003	0,94287	0,88769	0,90995
6 arestas	0,97289	0,90439	0,89810	0,95089	0,94372	0,95699

(b) *TC*

EDGE CREATE QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 aresta	0,95208	0,95098	0,95019	0,95336	0,95416	0,95386
2 arestas	0,95249	0,95141	0,95108	0,95304	0,95403	0,95434
3 arestas	0,95106	0,94831	0,94781	0,95140	0,95269	0,95369
4 arestas	0,95257	0,94964	0,94888	0,95334	0,95490	0,95501
5 arestas	0,95283	0,95218	0,95133	0,95364	0,95527	0,95558
6 arestas	0,94994	0,94768	0,94693	0,95077	0,95381	0,95400

(c) *prec*

Tabela 9.6: Conjunto de Testes 4 – Surgimento de dependência

Primeiramente, é importante comentar que o grafo acíclico da RBP geradora dos dados de um *stream* de preferências com 10 atributos (valor utilizado nestes testes) pode possuir no máximo 45 arestas. Sendo assim, em um cenário em que o grafo da RBP geradora dos dados não tenha nenhuma aresta, com EDGE_CREATE_QTY = 5 arestas, após a ocorrência do nono *concept drift*, a RBP poderá atingir o número máximo de arestas permitido (ou um número menor, visto que isso é dependente dos sorteios intermediários das arestas para compor o grafo sem formar ciclos). O parâmetro EDGE_CREATE_QTY = 6 arestas é interessante dado que simula o crescimento rápido do número de

arestas no grafo, mas em suas últimas aplicações (no mínimo nas duas últimas) pode ser que não seja possível inserir mais nenhuma aresta.

Sabendo disso, a Tabela 9.6(a) mostra que a utilização da técnica *Exponential Forgetting* nos dois algoritmos analisados apresentou valores de *acc* relativamente estáveis, mesmo com a variação do número de arestas inseridas no grafo. Por outro lado, note que a qualidade do algoritmo FPSMining puro decresceu nos quatro primeiros *streams* com o aumento do número de arestas inseridas a cada *concept drift*, porém esse mesmo comportamento não foi observado nos dois últimos *streams*. Isso se deve, possivelmente, ao fato de que inserir mais arestas na RBP geradora dos dados quando a mesma ainda está gerando o começo do *stream*, não oferece tanto impacto quanto inseri-las continuamente ao longo de todo o *stream*. Com a inserção de 5 e 6 arestas a cada *concept drift*, como dito anteriormente, os *concept drifts* finais não irão ocorrer, haja vista que não existirão mais arestas possíveis a serem inseridas. Dessa forma, é possível notar um comportamento crescente na qualidade do algoritmo FPSMining puro em relação aos *streams* com EDGE_CREATE_QTY = 4, 5 e 6 arestas. Note também que o algoritmo IncFPSMining puro, com EDGE_CREATE_QTY = 6 arestas, apresentou um valor de *acc* maior do que os resultados que vinham sendo obtidos com os demais valores de EDGE_CREATE_QTY.

Os valores de *TC* exibidos na Tabela 9.6(b) apresentaram o mesmo comportamento dos valores de *acc* (Tabela 9.6(a)). Isso indica que o número de acertos dos dois algoritmos analisados foi quase que proporcional ao número de bituplas que estes conseguiram comparar. Por fim, note na Tabela 9.6(c) que os valores de *prec* se mantiveram estáveis nas três abordagens dos dois algoritmos analisados, com valores de 94% e 95%, o que indica que quando a RBP foi capaz de comparar as tuplas de uma bitupla temporal, esta normalmente o fez corretamente.

O ***Conjunto 5*** de testes é referente ao *Tipo 5* (Desaparecimento de Dependência) de *concept drift* (CD = CD_MUT_REMOVE_DEPENDENCY). Neste conjunto de testes, variou-se o parâmetro EDGE_REMOVE_QTY = 1, 2, 3, 4, 5 e 6 arestas, com o intuito de avaliar o impacto do aumento do número de dependências removidas entre os atributos do *stream* de preferências. Por exemplo, se EDGE_REMOVE_QTY = 6, então significa que a cada ocorrência de *concept drift* devem ser removidas seis arestas aleatórias do grafo da RBP geradora dos dados. A Tabela 9.7 exibe os resultados obtidos com o *Conjunto 5* de testes.

A análise da Tabela 9.7(a) também é bastante interessante. Note que, no geral, a qualidade dos algoritmos puros diminuiu com o aumento do número de arestas removidas. Mas note também que, na abordagem dos algoritmos puros, a queda de qualidade no algoritmo FPSMining é bastante superior a do algoritmo IncFPSMining. Mais que isso, o valor de *acc* do algoritmo IncFPSMining puro com EDGE_REMOVE_QTY = 6 arestas é superior ao valor de *acc* do algoritmo FPSMining puro com EDGE_REMOVE_QTY =

EDGE REMOVE QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 aresta	0,92619	0,85528	0,74376	0,86958	0,78396	0,79001
2 arestas	0,92573	0,84618	0,72045	0,86115	0,77084	0,77198
3 arestas	0,92277	0,81737	0,59367	0,86150	0,77142	0,77243
4 arestas	0,92762	0,86740	0,63177	0,85994	0,76809	0,76837
5 arestas	0,92588	0,87481	0,61604	0,86042	0,76820	0,76850
6 arestas	0,92484	0,87001	0,60602	0,86107	0,76891	0,76925

(a) *acc*

EDGE REMOVE QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 aresta	0,97361	0,90003	0,78239	0,91369	0,82406	0,83028
2 arestas	0,97296	0,88969	0,75728	0,90518	0,80998	0,81144
3 arestas	0,96947	0,85978	0,62476	0,90531	0,81078	0,81204
4 arestas	0,97472	0,91174	0,66524	0,90542	0,80815	0,80864
5 arestas	0,97453	0,92084	0,64831	0,90573	0,80825	0,80877
6 arestas	0,97443	0,91736	0,63824	0,90562	0,80818	0,80876

(b) *TC*

EDGE REMOVE QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 aresta	0,95129	0,95028	0,95063	0,95172	0,95134	0,95150
2 arestas	0,95146	0,95110	0,95137	0,95136	0,95168	0,95137
3 arestas	0,95183	0,95067	0,95024	0,95161	0,95145	0,95122
4 arestas	0,95168	0,95137	0,94969	0,94977	0,95043	0,95020
5 arestas	0,95008	0,95001	0,95022	0,94997	0,95045	0,95021
6 arestas	0,94911	0,94838	0,94952	0,95081	0,95141	0,95115

(c) *prec*

Tabela 9.7: Conjunto de Testes 5 – Desaparecimento de dependência

1 aresta. Provavelmente, essa queda de qualidade mais rigorosa se deve ao fato de que, a cada *grace period*, o algoritmo FPSMining constrói o grafo completo do modelo, enquanto que o algoritmo IncFPSMining constrói o mesmo incrementalmente. Com isso, pode ser que as arestas removidas do grafo da RBP geradora dos dados nem tenham chegado a existir no modelo gerado pelo algoritmo IncFPSMining. Enquanto isso, as arestas removidas da RBP geradora dos dados podem demorar vários *grace periods* até desaparecerem do grafo completo gerado pelo algoritmo FPSMining, que é baseado no estado atual das estatísticas suficientes. Por outro lado, com a utilização da técnica *Exponential Forgetting*, o algoritmo FPSMining obteve resultados melhores do que o algoritmo IncFPSMining, possivelmente porque o esquecimento proveniente dessa técnica aproximou o grafo completo gerado pelo algoritmo FPSMining do grafo da RBP geradora dos dados. Outro fato interessante é que, com o uso da técnica *Exponential Forgetting*, os valores de *acc* dos algoritmos analisados permaneceram estáveis, ao contrário do comportamento decrescente

apresentado pelos algoritmos puros. É importante ressaltar também que no algoritmo IncFPSMining o uso da técnica *Naive Blind* obteve praticamente resultados idênticos aos do algoritmo puro. Entretanto, no algoritmo FPSMining, a técnica *Naive Blind* mostrou ser eficiente, e superou consideravelmente os resultados do algoritmo puro, possivelmente porque acelerou o processo de esquecimento das arestas removidas do grafo da RBP geradora dos dados. Ainda assim, a técnica *Exponential Forgetting* foi a que obteve melhores resultados dentre as abordagens apresentadas.

A Tabela 9.7(b) mostra que os valores de *TC* seguiram uma distribuição semelhante à seguida pelos valores de *acc*, o que indica que o número de acertos nestes testes está intimamente ligado à taxa de comparabilidade do modelo. Enquanto isso, a Tabela 9.7(c) mostra que os valores de *prec* permaneceram estáveis para os dois algoritmos analisados sob as três abordagens distintas, variando em torno de 94% e 95%.

Por fim, como mencionado anteriormente, para estes testes o grafo acíclico da RBP geradora dos dados pode possuir no máximo 45 arestas, visto que existem 10 atributos. Sendo assim, em um cenário em que o grafo da RBP geradora dos dados tenha o número máximo de arestas possíveis, com EDGE_REMOVE_QTY = 5 arestas, após a ocorrência do nono *concept drift*, o grafo dessa RBP não terá mais nenhuma aresta. O parâmetro EDGE_REMOVE_QTY = 6 arestas é interessante dado que simula o decaimento rápido do número de arestas no grafo, mas em suas últimas aplicações (no mínimo nas duas últimas) podem não haver mais arestas a serem removidas.

O ***Conjunto 6*** de testes é referente ao *Tipo 6* (Inversão de Dependência) de *concept drift* (CD = CD_MUT_INVERT_DEPENDENCY). Neste conjunto de testes, variou-se o parâmetro EDGE_INVERT_QTY = 1, 2, 3, 4, 5 e 6 arestas, com o intuito de avaliar o impacto do aumento do número de dependências invertidas entre os atributos do *stream* de preferências. Por exemplo, se EDGE_INVERT_QTY= 6, então significa que a cada ocorrência de *concept drift* devem ser invertidas seis arestas aleatórias no grafo da RBP geradora dos dados. A Tabela 9.8 exibe os resultados obtidos com o *Conjunto 6* de testes.

Conforme já havia sido mencionado na seção 8.1, esse tipo de mudança é um pouco mais drástico em relação às demais relativas a alterações nas dependências entre os atributos. Observe na Tabela 9.8(a) que os valores de *acc* dos algoritmos puros com EDGE_INVERT_QTY = 6 arestas (cerca de 73% no algoritmo IncFPSMining e 50% no algoritmo FPSMining) foram consideravelmente inferiores aos valores apresentados na Tabela 9.7(a), referentes a EDGE_REMOVE_QTY = 6 arestas (cerca de 76% para o algoritmo IncFPSMining e 60% para o algoritmo FPSMining). Outro ponto a se ressaltar é que os valores de *acc* dos algoritmos puros, salvo algumas exceções (como o algoritmo IncFPSMining nos *streams* com EDGE_INVERT_QTY = 4 e 5 arestas), apresentaram um comportamento decrescente com o aumento do número de arestas invertidas, ao contrário do comportamento relativamente estável apresentado por ambos os algoritmos com a utilização da técnica *Exponential Forgetting*. Note ainda que a utilização da técnica *Exponential*

EDGE INVERT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 aresta	0,92263	0,81926	0,75291	0,90945	0,84637	0,86680
2 arestas	0,91736	0,80359	0,65101	0,86834	0,73562	0,73912
3 arestas	0,92064	0,76866	0,66784	0,86876	0,71480	0,72806
4 arestas	0,91750	0,79243	0,57920	0,85897	0,77886	0,78278
5 arestas	0,92017	0,72354	0,56466	0,86143	0,77861	0,78187
6 arestas	0,91875	0,68782	0,50316	0,86021	0,73454	0,73630

(a) *acc*

EDGE INVERT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 aresta	0,97130	0,86435	0,79417	0,95642	0,88905	0,91011
2 arestas	0,96377	0,84571	0,68528	0,91193	0,77204	0,77602
3 arestas	0,96640	0,80769	0,70250	0,91149	0,74979	0,76394
4 arestas	0,96392	0,83309	0,60934	0,90218	0,81809	0,82299
5 arestas	0,96482	0,75910	0,59346	0,90258	0,81549	0,81979
6 arestas	0,96436	0,72250	0,52907	0,90244	0,77054	0,77319

(b) *TC*

EDGE INVERT QTY	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1 aresta	0,94989	0,94783	0,94805	0,95089	0,95199	0,95241
2 arestas	0,95185	0,95020	0,94999	0,95220	0,95283	0,95245
3 arestas	0,95265	0,95168	0,95066	0,95312	0,95333	0,95303
4 arestas	0,95184	0,95119	0,95054	0,95210	0,95205	0,95114
5 arestas	0,95372	0,95316	0,95147	0,95441	0,95478	0,95374
6 arestas	0,95270	0,95200	0,95103	0,95320	0,95328	0,95229

(c) *prec*

Tabela 9.8: Conjunto de Testes 6 – Inversão de dependência

Forgetting foi extremamente benéfica para os algoritmos analisados frente a este tipo de *concept drift*, haja vista que além da diferença de qualidade ser considerável com e sem a utilização desta técnica, a diferença ainda aumenta conforme o aumento do número de arestas invertidas (como dito anteriormente, com esta técnica os valores são estáveis, e sem a mesma eles tendem a ser decrescentes). Além disso, novamente a utilização da técnica *Naive Blind* com o algoritmo FPSMining superou consideravelmente os valores de *acc* do algoritmo puro, e a diferença só aumentou com o aumento do número de arestas invertidas a cada *concept drift*. Assim, os resultados das Tabelas 9.7 e 9.8 mostraram que a técnica *Naive Blind* em conjunto com o algoritmo FPSMining é eficiente para tipos de *concept drifts* que envolvem remoção ou inversão na dependência entre os atributos. Por outro lado, no algoritmo IncFPSMining os valores de *acc* com a utilização da técnica *Naive Blind* ficaram próximos dos valores de *acc* do algoritmo puro. Ademais, novamente os valores de *acc* dos algoritmos puros foram superiores com o algoritmo IncFPSMining,

sendo que com a utilização da técnica *Exponential Forgetting*, ao contrário, os valores de *acc* foram superiores com o algoritmo FPSMining.

A Tabela 9.8(b) mostra que os valores de TC também seguiram uma distribuição semelhante à seguida pelos valores de *acc*, o que indica que o número de acertos nestes testes está intimamente ligado à taxa de comparabilidade do modelo. Enquanto isso, a Tabela 9.8(c) mostra que os valores de *prec*, assim como nos resultados apresentados pela Tabela 9.7(c), também permaneceram estáveis para os dois algoritmos analisados sob as três abordagens distintas, variando em torno de 94% e 95%.

Por fim, é importante ressaltar que com um número alto de EDGE_INVERT_QTY, uma mesma aresta pode ser invertida em mais de uma ocorrência de *concept drift*, voltando assim ao seu estado inicial ao longo do *stream*. Isso tende a complicar o aprendizado do modelo, pois ele pode aprender uma determinada dependência, depois precisa perceber que essa dependência está errada (teve sua primeira inversão), e posteriormente precisa aprender que a dependência está correta novamente (foi invertida mais uma vez).

O ***Conjunto 7*** de testes é referente à evolução de qualidade do algoritmo FPSMining com a técnica *Exponential Forgetting* frente à ocorrência de um *concept drift* do *Tipo 6* (Inversão de Dependência). Neste conjunto de testes, os seguintes parâmetros foram fixados: CD = CD_MUT_INVERT_DEPENDENCY e EDGE_INVERT_QTY = 3 arestas. Assim, como será explicado na sequência, este conjunto de testes consiste em um *zoom* dado em uma parte específica do *Conjunto 6* de testes. O período do *stream* de preferências que foi avaliado nestes testes é desde a chegada do elemento e_{20k} até a chegada do elemento e_{30k}. Durante a geração do elemento e_{20k} do *stream* de preferências, a RBP geradora dos dados sofreu uma mutação, que consistiu na inversão de três arestas de seu grafo. Isso marcou a ocorrência de um *concept drift*. Até a geração do elemento e_{20k} do *stream*, já havia ocorrido um *concept drift* durante a geração do elemento e_{10k}. Dessa forma, o objetivo deste teste é entender o comportamento e a recuperação do algoritmo FPSMining com a técnica *Exponential Forgetting* após a ocorrência do segundo *concept drift* desse *stream*. Os gráficos (a), (b) e (c) da Figura 9.5 exibem os resultados obtidos com o *Conjunto 7* de testes.

A Figura 9.5(a) mostra que os valores de *acc* apresentados pelo algoritmo com a chegada do elemento e_{20k} estavam em cerca de 92%. A partir desse ponto, é possível visualizar claramente uma queda considerável nos valores de *acc* apresentados pelo algoritmo, até quando este atinge valores próximos de 83%. Essa queda se deve, sobretudo, a uma queda na taxa de comparabilidade (Figura 9.5(b)) do algoritmo, observada nesse mesmo período. Após essa queda acentuada, os valores de *acc* passaram por um período razoavelmente estável, e por volta da chegada do elemento e_{23k} o algoritmo começou a se recuperar. A partir desse ponto, os valores de *acc* sobem até aproximadamente 95%.

Outro ponto interessante nesta análise é que o comportamento das curvas de evolução das medidas *acc* e TC são bastante semelhantes, o que indica que neste teste o com-

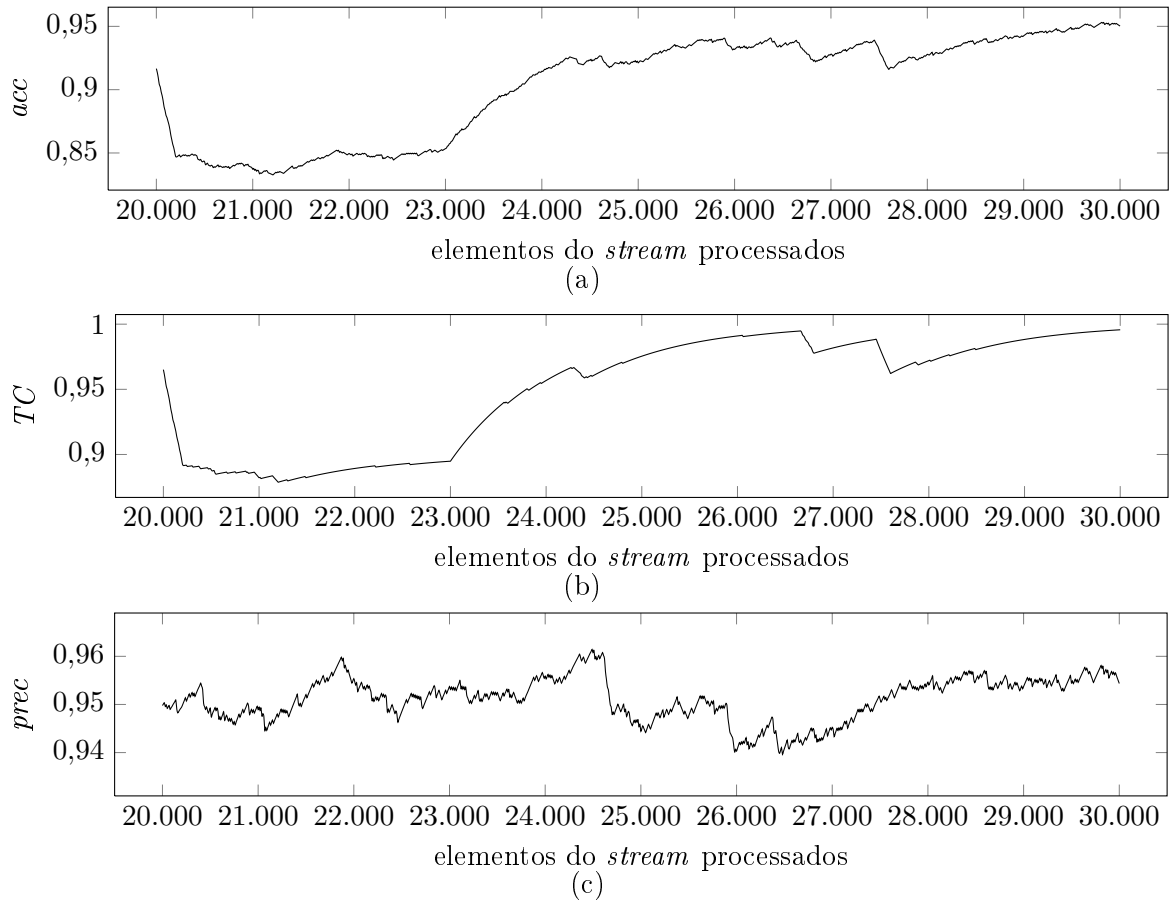


Figura 9.5: Conjunto de Testes 7 – Evolução da qualidade do algoritmo FPSMining com a técnica *Exponential Forgetting* após a ocorrência do segundo *concept drift* do Tipo 6

portamento dos valores de *acc* é muito influenciado pelos valores de TC. Apesar disso, note que o intervalo de variação dos valores de *acc* e TC são distintos: cerca de 83% a 95% para *acc* e 88% a 99% para TC. Isso indica que o algoritmo errou uma pequena parte dos elementos que conseguiu comparar. Note ainda, que os valores de *prec* (Figura 9.5(c)) sofreram uma pequena variação neste trecho do *stream*, permanecendo razoavelmente estáveis: entre 94% e 96%. Por fim, é importante destacar que os valores de *prec* são responsáveis pela diferença de variação entre as medidas *acc* e TC, visto que estas três medidas estão interligadas.

O ***Conjunto 8*** de testes pode envolver potencialmente todos os seis tipos de *concept drifts* propostos nesta dissertação (CD = CD_MUT_ALL). Neste conjunto de testes, os seguintes parâmetros foram fixados: ATTRIBUTE_DRIFT_QTY = 5, PERC_CREATE_PREFERENCE = 30%, PERC_REMOVE_PREFERENCE = 30%, PERC_INVERT_PREFERENCE = 30%, EDGE_CREATE_QTY = 3, EDGE_REMOVE_QTY = 3 e EDGE_INVERT_QTY = 3. O parâmetro que sofreu variação nestes testes foi SEED = 1L, 2L, 3L, 4L e 5L. Por exemplo, se SEED = 1L, então significa que a cada ponto de ocorrência de *concept drift* deve ser sorteado um dos seis tipos propostos de *concept drift* de acordo com a semente 1L para a geração de números aleatórios. Após definida a ordem dos tipos de *concept drifts* que devem ocorrer ao longo do *stream*, os valores

dos parâmetros utilizados para cada tipo são aqueles que foram fixados e mencionados anteriormente. A Tabela 9.9 exibe os resultados obtidos com o *Conjunto 8* de testes. A coluna SEED dessas tabelas apresenta a semente utilizada para a definição dos tipos de *concept drifts* que fazem parte do *stream*. Veja abaixo quais foram os tipos de *concept drifts* sorteados (Tipos de 1 a 6) e a sua respectiva ordem de ocorrência (totalizando nove ocorrências no *stream*, assim como nos demais conjuntos de testes) com a utilização de cada uma das sementes escolhidas para estes testes.

- SEED = 1L: Tipos 4 (Surgimento de Dependência), 5 (Desaparecimento de Dependência), 2 (Desaparecimento de Preferência), 4, 3 (Inversão de Preferência), 5, 3, 5 e 5, respectivamente;
- SEED = 2L: Tipos 5, 1 (Surgimento de Preferência), 3, 2, 4, 1, 1, 4 e 2, respectivamente;
- SEED = 3L: Tipos 3, 3, 1, 2, 1, 1, 4, 5 e 2, respectivamente;
- SEED = 4L: Tipos 3, 5, 4, 5, 4, 4, 2, 5 e 1, respectivamente;
- SEED = 5L: Tipos 6 (Inversão de Dependência), 5, 3, 3, 1, 6, 5, 6 e 3, respectivamente.

Os resultados da Tabela 9.9 são muito interessantes pois envolvem *streams* com a ocorrência de *concept drifts* de diversos tipos. Assim como explicado anteriormente, o *stream* gerado pela semente igual a 1L envolve a ocorrência de um *concept drift* do Tipo 4 (após 10k elementos do *stream*), seguido de um *concept drift* do Tipo 5 (após 20k elementos do *stream*), e assim por diante. Observe que o número total de combinações que envolvam um ou mais tipos de *concept drifts* (considerando os seis tipos propostos), totalizando a ocorrência de nove *concept drifts*, é muito grande: 6^9, o que resulta em mais de 10 milhões de possibilidades. Justamente por isso, esta seção dá ênfase no entendimento individual de cada tipo de *concept drift* (Conjuntos de Testes 1 até 6), visto que isto permite a análise posterior de qualquer uma destas combinações. Sendo assim, a Tabela 9.9 exibe uma pequena amostra de cinco conjuntos de testes dentre as possibilidades existentes.

De acordo com as Tabelas 9.9(a) e (b), os resultados que mais chamam a atenção são os referentes às sementes 3L (melhores valores de *acc* e TC) e 5L (piores valores de *acc* e TC), devido à discrepância de qualidade obtida pelos algoritmos puros. Um ponto de destaque nesta análise é a ocorrência de vários *concept drifts* dos Tipos 3, 5 e 6 com a semente 5L (oito das nove ocorrências de *concept drifts* são de um destes tipos), que conforme visto anteriormente (Conjuntos de Testes 3, 5 e 6), afetam de forma mais drástica a qualidade dos algoritmos analisados. Note que estes tipos de *concept drifts* ocorreram em uma proporção bem menor com a semente 3L (apenas três das nove ocorrências de *concept drifts*). Isso explica em parte essa diferença de qualidade apresentada pelos algoritmos

SEED	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1L	0,88054	0,74863	0,68987	0,82906	0,75303	0,76270
2L	0,92925	0,89409	0,69929	0,77165	0,75900	0,75344
3L	0,89669	0,81515	0,80423	0,89230	0,87340	0,87352
4L	0,90703	0,78950	0,59225	0,77368	0,78626	0,73980
5L	0,88707	0,78830	0,52506	0,65829	0,64228	0,64003

(a) *acc*

SEED	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1L	0,96864	0,84785	0,78612	0,90966	0,85349	0,86880
2L	0,98012	0,95812	0,75405	0,81459	0,81580	0,81649
3L	0,97560	0,90809	0,89784	0,96922	0,97343	0,97361
4L	0,97311	0,84591	0,64050	0,82946	0,84859	0,80090
5L	0,97452	0,89434	0,60370	0,71936	0,73287	0,73766

(b) *TC*

SEED	FPSMining			IncFPSMining		
	Exponential Forgetting	Naive Blind	Nenhum	Exponential Forgetting	Naive Blind	Nenhum
1L	0,90905	0,88297	0,87756	0,91140	0,88230	0,87788
2L	0,94810	0,93317	0,92738	0,94729	0,93038	0,92278
3L	0,91912	0,89765	0,89574	0,92064	0,89724	0,89720
4L	0,93209	0,93331	0,92467	0,93275	0,92655	0,92371
5L	0,91026	0,88143	0,86974	0,91511	0,87639	0,86765

(c) *prec*

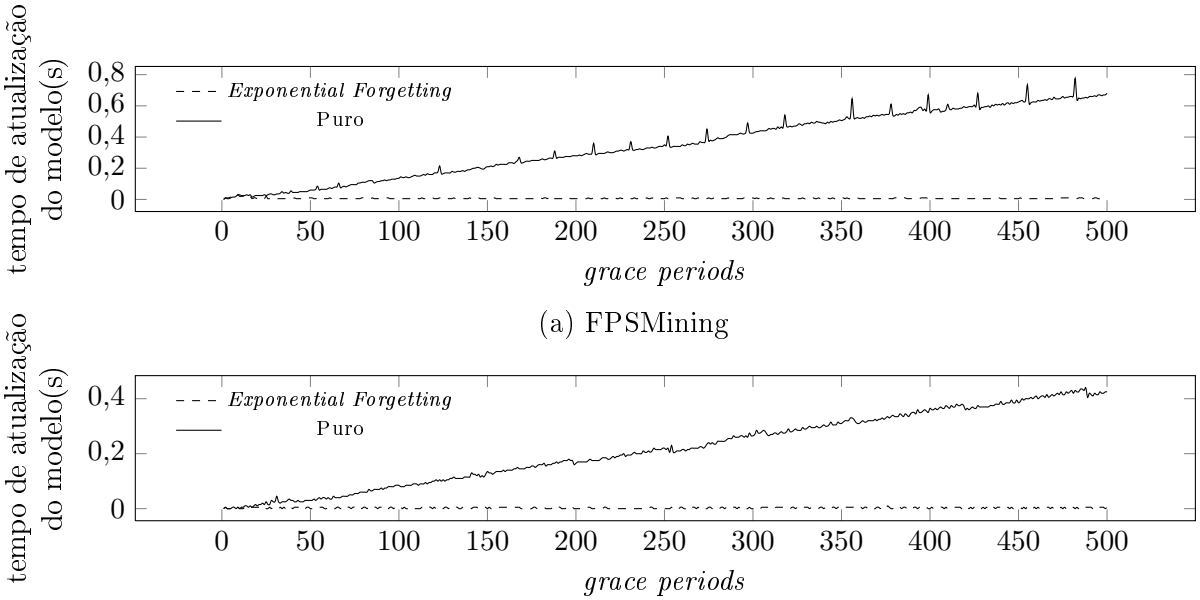
Tabela 9.9: Conjunto de Testes 8 – Vários tipos de *concept drifts*

nestes dois *streams*. Note ainda que com a semente 3L ocorreram vários tipos de mutações que, conforme visto anteriormente, não ocasionam grandes impactos na qualidade dos algoritmos (entre o *surgimento* e *desaparecimento* de preferências, teve-se um total de cinco ocorrências). Outro item a se considerar é a sequência de ocorrência dos diferentes tipos de *concept drifts*. Note que com a semente 5L, como oito das nove ocorrências de *concept drifts* são dos Tipos 3, 5 ou 6, os algoritmos propostos mal tiveram tempo de se recuperar de um tipo drástico de *concept drift*, e já ocorreu outro tão impactante quanto. Por outro lado, note que com a semente 3L existem duas ocorrências do Tipo 3 no começo do *stream*, e uma ocorrência do Tipo 5 no final do *stream*, dando um intervalo razoável para que os algoritmos pudessem se recuperar.

Note também que os resultados de *acc* dos algoritmos puros entre as sementes 1L e 2L (cerca de 68% e 69% para o FPSMining e 76% e 75% para o algoritmo IncFPSMining) foram muito similares, embora essa diferença tenha aumentado consideravelmente com a utilização das técnicas *Exponential Forgetting* (para ambos os algoritmos) e *Naive Blind* (apenas para o algoritmo FPSMining). Ainda com essas duas sementes, note que o al-

goritmo FPSMining apresentou melhores resultados (*acc* e *prec* apenas) com o *stream* de semente 2L em comparação com o *stream* de semente 1L, enquanto que o algoritmo IncFPSMining apresentou melhores resultados (*acc* e *TC* apenas) com o *stream* de semente 1L. Assim, o algoritmo IncFPSMining obteve melhores resultados de *acc* para a semente 1L porque neste *stream* ele conseguiu comparar consideravelmente mais tuplas do que no *stream* oriundo da semente 2L, apesar de também ter errado mais nas comparações realizadas do que com a semente 2L. Já o algoritmo FPSMining, apesar dele ter conseguido efetuar menos comparações com a semente 2L, ele acertou grande parte delas, o que fez subir seus valores de *acc*.

O ***Conjunto 9*** de testes é referente à evolução do tempo de atualização do modelo dos algoritmos FPSMining e IncFPSMining puros e com a técnica *Exponential Forgetting* frente à ocorrência de *concept drifts* de vários tipos. O *stream* utilizado nestes testes é o mesmo utilizado no *Conjunto 8* de testes com o parâmetro SEED = 5L. Os gráficos (a) e (b) da Figura 9.6 exibem os resultados obtidos com o *Conjunto 9* de testes.



(a) FPSMining

(b) IncFPSMining

Figura 9.6: Conjunto de Testes 9 – Tempo de atualização do modelo

A Figura 9.6(a) mostra a evolução do tempo gasto para a geração do modelo no algoritmo FPSMining ao longo do tempo. Como pode ser visto, o tempo de geração do modelo, com o algoritmo FPSMining puro, foi crescente com o aumento da quantidade de elementos processados no *stream* (aumento do número de *grace periods*, onde 500 *grace periods* = 100k elementos processados no *stream*). Isso já era esperado, pois o aumento do volume das estatísticas suficientes impacta na velocidade de geração do modelo. Porém, com a utilização da técnica *Exponential Forgetting*, o tempo de geração do modelo se manteve constante durante todo o *stream*. Este comportamento também já era esperado, pois o objetivo da técnica *Exponential Forgetting* é justamente reduzir a importância dos

elementos mais antigos com o passar do tempo, de forma a dar mais importância aos novos elementos, o que resulta em eliminar informações muito antigas. Essa redução das informações antigas faz o volume das estatísticas suficientes se manter praticamente constante e, com isso, o tempo de geração do modelo praticamente não sofre alterações.

Enquanto isso, a Figura 9.6(b) mostra a evolução do tempo gasto para a atualização do modelo no algoritmo IncFPSMining ao longo do tempo. Como pode ser visto, o comportamento apresentado nesta figura foi bastante similar ao observado na Figura 9.6(a). Dessa maneira, a análise desse comportamento é semelhante ao da Figura 9.6(a). Apesar desta grande semelhança no comportamento dos dois algoritmos, é importante destacar que o tempo de atualização do modelo apresentado pelo IncFPSMining foi cerca da metade do tempo apresentado pelo FPSMining. Isso apenas confirma a eficiência da atualização das tabelas de probabilidade condicional com a chegada de cada elemento do *stream* (versão 2 do algoritmo IncFPSMining, utilizada nos conjuntos de testes de 1 a 9), o que resulta na redução da complexidade de tempo do algoritmo.

Por fim, a ***Tabela 9.10*** apresenta uma síntese acerca dos resultados de acurácia (*acc*) referentes aos seis tipos de *concept drifts* propostos (conjuntos de testes de 1 a 6). As colunas "Exponential Forgetting", "Naive Blind" e "Nenhum" exibem o algoritmo que apresentou melhor performance na maior parte das vezes com a técnica em questão, enquanto que a coluna "Melhor Algoritmo - Técnica" apresenta o par de algoritmo e técnica que superou os demais dentro do conjunto de testes em questão.

Tipo de Concept Drift	**Exponential Forgetting**	**Naive Blind**	**Nenhum**	**Melhor Algoritmo - Técnica**
1 - Surgimento de Preferência	IncFPS-Mining	IncFPS-Mining	IncFPS-Mining	IncFPSMining - Exponential Forgetting
2 - Desaparecimento de Preferência	IncFPS-Mining	IncFPS-Mining	IncFPS-Mining	IncFPSMining - Exponential Forgetting
3 - Inversão de Preferência	IncFPS-Mining	IncFPS-Mining	IncFPS-Mining	IncFPSMining - Exponential Forgetting
4 - Surgimento de Dependência	FPSMining	IncFPS-Mining	IncFPS-Mining	FPSMining - Exponential Forgetting
5 - Desaparecimento de Dependência	FPSMining	FPSMining	IncFPS-Mining	FPSMining - Exponential Forgetting
6 - Inversão de Dependência	FPSMining	empate	IncFPS-Mining	FPSMining - Exponential Forgetting

Tabela 9.10: Síntese dos resultados referentes aos seis tipos de *concept drift* propostos

Note que com a técnica *Exponential Forgetting*, o algoritmo IncFPSMining obteve melhores resultados para os tipos de *concept drift* referentes a *preferências*, enquanto que o algoritmo FPSMining obteve melhores resultados para os tipos de *concept drift* referentes a *dependências*. Note também que a técnica *Exponential Forgetting* foi a que

apresentou melhores resultados para todos os tipos de *concept drift*. Além disso, na avaliação dos algoritmos puros (coluna "Nenhum"), o algoritmo IncFPSMining obteve melhor desempenho para todos os tipos de *concept drift*, conforme já era esperado, haja vista que este atualiza as tabelas de probabilidade condicional com a chegada de cada elemento do *stream*. Por fim, note ainda que com a técnica *Naive Blind*, o algoritmo IncFPSMining apresentou melhores resultados para a maior parte dos tipos de *concept drift*, sendo superado pelo algoritmo FPSMining apenas no *Tipo 5* de *concept drift*.

9.4 Considerações Finais

Os algoritmos FPSMining e IncFPSMining foram implementados e amplamente analisados através de uma série de experimentos com dados reais e dados sintéticos.

Um experimento que merece bastante destaque é a execução do mecanismo de *baseline* proposto com dois classificadores amplamente utilizados na literatura: *Hoeffding Tree* e *Naive Bayes*. Os resultados apresentados mostraram que o algoritmo FPSMining supera, com significância estatística, as medidas de acurácia e taxa de comparabilidade dos dois algoritmos utilizados como *baselines*.

Além disso, dentre os experimentos realizados, comparou-se os algoritmos FPSMining e IncFPSMining sobre dados sintéticos com distribuição estacionária e, de acordo com a realização de uma análise de significância estatística, o algoritmo IncFPSMining obteve melhores resultados.

Em relação aos testes em dados sintéticos com *concept drifts*, conclui-se que a técnica *Exponential Forgetting* apresenta melhores resultados, na maioria das vezes consideráveis, em relação à utilização dos algoritmos puros. Enquanto isso, a técnica *Naive Blind* apresenta melhores resultados apenas frente a tipos de *concept drifts* que impactem diretamente na *remoção ou inversão das dependências* entre os atributos do grafo da RBP. Para os tipos de *concept drift* referentes apenas às regras de preferência ou a criação de dependência, a vantagem apresentada pela técnica *Naive Blind* em relação aos algoritmos puros é que ela consegue atingir um resultado similar com menos informações, isto é, com um conjunto bastante reduzido das estatísticas suficientes.

Além disso, nos tipos de *concept drifts* relativos às modificações de *dependências*, os valores de TC seguem uma distribuição semelhante à seguida pelos valores de acc, o que indica que o número de acertos em *streams* com esses tipos de *concept drifts* está intimamente ligado à taxa de comparabilidade do modelo. Enquanto isso, para esses mesmos tipos de *concept drifts*, os valores de $prec$ permanecem estáveis para os dois algoritmos analisados, sob as três abordagens distintas.

Por fim, conclui-se que os tipos de *concept drifts* que mais impactam a qualidade dos algoritmos analisados são os Tipos 3 (Inversão de Preferência), 5 (Desaparecimento de Dependência) e 6 (Inversão de Dependência).

Capítulo 10

Conclusão

Evidências de preferências dos usuários estão em toda parte, mas pouco uso se tem feito dessas informações valiosas. O fluxo dessas informações de preferências normalmente não é armazenado, e está disponível no formato de *streams*. Muitas vezes, essas preferências fazem parte de padrões conscientes ou subconscientes que o usuário obedece quando procura por determinadas notícias, quando busca pela compra de um livro, ou quando investiga filmes de seu interesse. O objetivo principal desta pesquisa consistiu em propor algoritmos para descobrir esses padrões e transcrevê-los em regras de preferências com alto poder expressivo - regras de preferências contextuais.

Nesta dissertação foi introduzido o problema de "Mineração de Preferências Contextuais do Usuário em *Data Streams*". Este problema foi estendido de um problema prévio existente no cenário *batch*. Em suma, este problema consiste na extração de uma Rede Bayesiana de Preferência a partir de um *stream* de preferências. Na tentativa de solucionar este problema, foram propostos três algoritmos: FPSMining, IncFPSMining e IGAPSMining. Estes algoritmos mantêm um conjunto reduzido de estatísticas suficientes extraídas ao longo do tempo de um *stream* de preferências, e com base nisso conseguem evoluir uma Rede Bayesiana de Preferência, capaz de predizer as preferências do usuário.

O primeiro algoritmo proposto - FPSMining - caracteriza-se por construir o modelo de preferências de uma forma não incremental. De tempos em tempos, esse algoritmo reconstrói seu modelo a partir das estatísticas suficientes que foram extraídas ao longo do tempo. Apesar disso, esse algoritmo se mostra eficiente em termos de qualidade (de acordo com os resultados apresentados no capítulo 9) e tempo de execução (de acordo com sua análise de complexidade exibida na seção 5.3).

Enquanto isso, o segundo algoritmo proposto - IncFPSMining - constrói o modelo de preferências incrementalmente, com a chegada de cada elemento do *stream* de preferências. Foram propostas duas versões referentes a este algoritmo, sendo que a segunda versão, que atualiza as tabelas de probabilidade condicional com a chegada de cada novo elemento do *stream*, mostrou-se mais eficiente do que a primeira em termos de qualidade e tempo de execução.

Os algoritmos FPSMining e IncFPSMining foram implementados e amplamente analisados em uma série de experimentos com dados sintéticos e dados reais. Dentre os experimentos realizados, comparou-se estes dois algoritmos sobre dados sintéticos e, de acordo com a realização de uma análise de significância estatística, o algoritmo IncFPSMining obteve melhores resultados. Outro experimento que merece bastante destaque é a execução do mecanismo de *baseline* proposto com dois classificadores largamente utilizados na literatura: *Hoeffding Tree* e *Naive Bayes*. Conforme mencionado anteriormente, os algoritmos FPSMining e IncFPSMining superaram, com significância estatística, as medidas de acurácia e taxa de comparabilidade dos dois algoritmos utilizados como *baselines*.

Dentro desta dissertação, foram propostos ainda seis tipos de *concept drifts* em preferências contextuais. Implementou-se um gerador de *stream* sintético de preferências, capaz de simular *streams* com cada um dos tipos de *concept drift* propostos. Como uma tentativa de lidar com estes *concept drifts*, os algoritmos propostos foram utilizados em conjunto com as técnicas *Naive Blind* e *Exponential Forgetting* (detalhadas no capítulo 8). A utilização da técnica *Exponential Forgetting* sempre apresentou melhora em relação aos algoritmos puros, enquanto que a utilização da técnica *Naive Blind* apresentou melhora apenas em casos específicos, envolvendo mudanças nas dependências entre os atributos. Apesar disso, ambas as técnicas conseguem reduzir significativamente o uso de memória das estatísticas, proporcionando, assim, uma maior velocidade de execução para os algoritmos. As análises sobre os resultados obtidos evidenciaram, por exemplo, qual das medidas, *prec* ou TC, foi mais impactada com a ocorrência de cada um dos tipos de *concept drift* e, portanto, mais impactaram nos resultados de *acc*. Com isso, é possível entender se a qualidade foi reduzida devido a uma baixa taxa de comparabilidade ou devido à introdução de mais erros por parte dos algoritmos.

Enquanto isso, o terceiro algoritmo proposto – IGAPSMining – é caracterizado por utilizar uma técnica heurística para construir o modelo de preferências incrementalmente. Esse algoritmo foi apenas apresentado e discutido nesta dissertação, reservando a sua implementação e análise para trabalhos futuros. A análise de complexidade deste algoritmo viabiliza a sua implementação.

Por fim, é importante destacar que os códigos fontes dos dois algoritmos[1] implementados, bem como do gerador[2] de *stream* de dados sintéticos proposto, estão publicamente disponíveis para facilitar o uso de futuros *baselines*.

10.1 Artigos aceitos

A pesquisa envolvida nesta dissertação foi alvo de um total de cinco artigos aceitos até o momento. A seguir, são fornecidos os principais dados sobre cada um desses artigos,

[1] Disponível em `http://lsi.facom.ufu.br/dataStreamMining/algorithms`
[2] Disponível em `http://lsi.facom.ufu.br/dataStreamMining/syntheticGenerator`

em ordem cronológica de submissão.

1. Artigo entitulado *"Strategies for Mining User Preferences in a Data Stream Setting"*, apresentado no *Symposium on Knowledge Discovery, Mining and Learning* (KDMiLe 2013). Nesse artigo, foram propostas duas estratégias na tentativa de solucionar o problema abordado pelo trabalho descrito nesta dissertação. Essas duas estratégias representam as ideias iniciais que posteriormente deram origem aos algoritmos FPSMining (capítulo 5) e IGAPSMining (capítulo 7).
2. Artigo entitulado *"FPSMining: A Fast Algorithm for Mining User Preferences in Data Streams"*, apresentado no *Simpósio Brasileiro de Banco de Dados* (SBBD 2013). Uma das estratégias propostas no artigo mencionado no item 1 foi melhor trabalhada e validada, dando origem ao algoritmo FPSMining (capítulo 5). Ainda dentro do SBBD 2013, este trabalho de mestrado foi apresentado e discutido no *Workshop de Teses e Dissertação* (WTDBD 2013).
3. Artigo estendido do KDMiLe 2013 (mencionado no item 1), convidado e aceito para uma edição especial do *Journal of Information and Data Management* (JIDM). Esse artigo apresenta, dentre outros, o mecanismo de *baseline* realizado com os classificadores *Hoeffding Tree* e *Naive Bayes* (seção 9.2 desta dissertação), bem como a análise de complexidade das estratégias propostas (seções 5.3 e 7.2).
4. Artigo estendido do SBBD 2013 (mencionado no item 2), convidado e aceito para uma edição especial do JIDM. Nesse artigo, dentre outros, foi introduzido o gerador (sem *concept drift*) de *stream* sintético de preferências (seção 8.2) para a avaliação de escalabilidade dos algoritmos frente a grandes volumes de dados.
5. Artigo entitulado *"Mining Contextual Preferences in Data Streams"*, aceito na conferência *Florida Artificial Intelligence Research Society* (FLAIRS 2014). O foco desse artigo consiste em uma comparação experimental entre os algoritmos FPSMining (capítulo 5) e IncFPSMining (capítulo 6) sobre dados reais e dados sintéticos.

Artigos premiados

O artigo entitulado *"FPSMining: A Fast Algorithm for Mining User Preferences in Data Streams"*, referenciado na listagem anterior (item 2), ganhou o prêmio de *Best Short Paper* no *Simpósio Brasileiro de Banco de Dados* (SBBD 2013).

10.2 Artigos a serem submetidos

A curto prazo, pretende-se submeter dois artigos, um para cada uma das conferências listadas abaixo.

1. *25th International Conference on Database and Expert Systems Applications* (DEXA 2014), na metade de março de 2014. O foco desse artigo será os principais pontos abordados no capítulo 8, que são: a) os tipos propostos de *concept drift* em preferências contextuais, b) o gerador de *stream* de dados sintéticos com introdução de *concept drift* e c) as técnicas que foram incorporadas aos algoritmos propostos para o tratamento de *concept drift*.
2. *16th International Conference on Data Warehousing and Knowledge Discovery* (DaWaK 2014), no final de março de 2014. O foco desse artigo será a implementação e validação do algoritmo IGAPSMining (capítulo 7).

A longo prazo, pretende-se submeter todo o trabalho descrito nesta dissertação para um *Journal* (a ser definido).

10.3 Trabalhos Futuros

Apesar de se ter avançado bastante neste problema com o trabalho descrito nesta dissertação, existem várias propostas de trabalhos futuros para darem continuidade a essa pesquisa. A seguir, são listadas as principais propostas existentes até o momento.

1. Implementação e avaliação do algoritmo IGAPSMining, descrito no capítulo 7
    - As principais justificativas para a implementação deste algoritmo são: (1) ele tem capacidade potencial para atingir qualquer ponto do espaço de busca, o que espera-se que reflita em modelos de alta qualidade; (2) a análise de complexidade efetuada mostra que ele é eficiente em termos de tempo e tende a ser mais rápido do que o algoritmo IncFPSMining com um número maior de atributos.
2. Agregação de outros experimentos interessantes
    - No capítulo 9 foi apresentada uma bateria extensa de testes em *streams* com *concept drifts*. Apesar disso, pretende-se ainda estudar o comportamento dos algoritmos propostos frente a determinadas sequências específicas de *concept drifts*. No Conjunto 8 de Testes (CD = CD_MUT_ALL) da seção 9.3.2, tentou-se efetuar esta análise sobre sequências aleatórias, mas não sobre sequências determinadas de acordo com um estudo prévio;
    - Planeja-se também efetuar uma análise dos algoritmos propostos com as técnicas *Naive Blind* e *Exponential Forgetting* sobre dados reais, com o intuito de avaliar se a qualidade dos mesmos aumenta devido a possíveis *concept drifts* nas avaliações dos usuários ao longo do tempo;

- Pretende-se ainda analisar os algoritmos propostos em conjunto com uma técnica que efetua a detecção explícita de *concept drift* como, por exemplo, o *Drift Detection Method* (DDM) [Gama et al. 2004]. Com isso, pode-se inclusive detectar as possíveis ocorrências de *concept drift* sobre os dados reais.

3. Utilização de um parâmetro para limitar explicitamente o uso de memória

   - Apesar das técnicas *Naive Blind* e *Exponential Forgetting* reduzirem consideravelmente o uso de memória dos algoritmos propostos, pretende-se ainda implementar um parâmetro indicando explicitamente o limite máximo de memória que estes podem utilizar.

4. Análise do algoritmo IncFPSMining com a utilização de outros limites disponíveis na literatura

   - Apesar do *Hoeffding Bound* ser amplamente utilizado na literatura e o seu uso no algoritmo IncFPSMining ter apresentado bons resultados, planeja-se também avaliar o desempenho deste algoritmo com a utilização de outros limites promissores disponíveis na literatura.

5. Implementação de um algoritmo de *ensemble*

   - Pretende-se ainda implementar um algoritmo de *ensemble* que utiliza os algoritmos propostos nesta dissertação como *base learners* (aprendedores base), com o intuito de avaliar o aumento no poder de predição em relação aos algoritmos individuais.

6. Avaliação dos algoritmos propostos integrados a sistemas de recomendação baseados em conteúdo

   - Por fim, pretende-se avaliar o desempenho dos algoritmos propostos dentro de sistemas de recomendação baseados em conteúdo. É importante ressaltar, entretanto, que o problema tratado no trabalho descrito nesta dissertação é um problema de personalização (*object ranking*, veja mais detalhes no capítulo 2), e não de recomendação (*label ranking*). Contudo, os algoritmos propostos nesta dissertação podem ser adaptados para trabalharem dentro de sistemas de recomendação baseados em conteúdo, que é o foco deste trabalho futuro.

Referências Bibliográficas

[Bach e Maloof 2010] Bach, S. H. e Maloof, M. A. (2010). A Bayesian Approach to Concept Drift. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., e Culotta, A. (editores), *NIPS*, pp. 127–135. Curran Associates, Inc.

[Beretta et al. 2011] Beretta, D., Quintarelli, E., e Rabosio, E. (2011). Mining Context-Aware Preferences on Relational and Sensor Data. In Morvan, F., Tjoa, A. M., e Wagner, R. (editores), *DEXA Workshops*, pp. 116–120. IEEE Computer Society.

[Bifet e Gavaldà 2007] Bifet, A. e Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *In SIAM International Conference on Data Mining*.

[Bifet et al. 2010] Bifet, A., Holmes, G., Kirkby, R., e Pfahringer, B. (2010). MOA: Massive Online Analysis. *J. Mach. Learn. Res.*, 11:1601–1604.

[Bifet et al. 2011] Bifet, A., Holmes, G., Kirkby, R., e Pfahringer, B. (2011). Data stream mining: A practical approach. Technical report, The University of Waikato.

[Bifet et al. 2009] Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., e Gavaldà, R. (2009). New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pp. 139–148, Paris, France, New York, NY, USA. ACM.

[Brzezinski 2010] Brzezinski, D. (2010). Mining Data Streams with Concept Drift. Master's thesis, Poznan University of Technology.

[Crammer e Singer 2001] Crammer, K. e Singer, Y. (2001). Pranking with Ranking. In Dietterich, T. G., Becker, S., e Ghahramani, Z. (editores), *NIPS*, pp. 641–647, Vancouver, British Columbia, Canada. MIT Press.

[Dawid 1984] Dawid, A. P. (1984). Statistical theory: The prequential approach. *Journal of the Royal Statistical Society-A*, 147:278–292.

[de Amo et al. 2012a] de Amo, S., Bueno, M. L. P., Alves, G., e Silva, N. F. (2012a). CPrefMiner: An Algorithm for Mining User Contextual Preferences Based on Bayesian Networks. In *Proceedings of the 2012 IEEE 24th International Conference on Tools with Artificial Intelligence - Volume 01*, ICTAI '12, pp. 114–121, Washington, DC, USA. IEEE Computer Society.

[de Amo et al. 2012b] de Amo, S., Diallo, M. S., Diop, C. T., Giacometti, A., Li, H. D., e Soulet, A. (2012b). Mining contextual preference rules for building user profiles. In *Proceedings of the 14th international conference on Data Warehousing and Knowledge Discovery*, DaWaK'12, pp. 229–242, Vienna, Austria, Berlin, Heidelberg. Springer-Verlag.

[De Sá et al. 2011] De Sá, C. R., Soares, C., Jorge, A. M., Azevedo, P., e Costa, J. (2011). Mining Association Rules for Label Ranking. In *Proceedings of the 15th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, PAKDD'11, pp. 432–443, Shenzhen, China, Berlin, Heidelberg. Springer-Verlag.

[Delgado e Ishii 1999] Delgado, J. e Ishii, N. (1999). Online learning of user preferences in recommender systems. In *International Joint Conference on Artificial Intelligence (IJCAI-99), Workshop on Machine Learning for Information Filtering*, Stockholm, Sweden.

[Domingos e Hulten 2000] Domingos, P. e Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pp. 71–80, Boston, Massachusetts, United States, New York, NY, USA. ACM.

[Freund et al. 2003] Freund, Y., Iyer, R., Schapire, R. E., e Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969.

[Fürnkranz e Hüllermeier 2010] Fürnkranz, J. e Hüllermeier, E. (2010). Preference Learning: An Introduction. In Fürnkranz, J. e Hüllermeier, E. (editores), *Preference Learning*, pp. 1–17. Springer-Verlag, Barcelona, Spain.

[Gaber et al. 2005] Gaber, M. M., Zaslavsky, A., e Krishnaswamy, S. (2005). Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26.

[Gama 2010] Gama, J. (2010). *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, Minneapolis, Minnesota, USA, 1st edition.

[Gama et al. 2004] Gama, J., Medas, P., Castillo, G., e Rodrigues, P. P. (2004). Learning with Drift Detection. In Bazzan, A. L. C. e Labidi, S. (editores), *SBIA*, volume 3171 de *Lecture Notes in Computer Science*, pp. 286–295. Springer.

[Hoeffding 1963] Hoeffding, W. (1963). Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30.

[Holland et al. 2003] Holland, S., Ester, M., e Kießling, W. (2003). Preference Mining: A Novel Approach on Mining User Preferences for Personalized Applications. Technical report, Universitätsbibliothek der Universität Augsburg, Universitätsstr. 22, 86159 Augsburg.

[Hüllermeier et al. 2008] Hüllermeier, E., Fürnkranz, J., Cheng, W., e Brinker, K. (2008). Label Ranking by Learning Pairwise Preferences. *Artif. Intell.*, 172(16-17):1897–1916.

[Jembere et al. 2007] Jembere, E., Adigun, M. O., e Xulu, S. S. (2007). Mining Context-based User Preferences for m-Services Applications. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, WI '07, pp. 757–763, Washington, DC, USA. IEEE Computer Society.

[Jensen e Nielsen 2007] Jensen, F. V. e Nielsen, T. D. (2007). *Bayesian Networks and Decision Graphs*. Springer Publishing Company, Incorporated, 2nd edition.

[Jiang et al. 2008] Jiang, B., Pei, J., Lin, X., Cheung, D. W., e Han, J. (2008). Mining preferences from superior and inferior examples. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pp. 390–398, Las Vegas, Nevada, USA, New York, NY, USA. ACM.

[Krause et al. 2006] Krause, A., Smailagic, A., e Siewiorek, D. P. (2006). Context-Aware Mobile Computing: Learning Context-Dependent Personal Preferences from a Wearable Sensor Array. *IEEE Transactions on Mobile Computing*, 5(2):113–127.

[Oza e Russell 2001] Oza, N. C. e Russell, S. (2001). Online Bagging and Boosting. In *In Artificial Intelligence and Statistics 2001*, pp. 105–112. Morgan Kaufmann.

[Rajaraman e Ullman 2011] Rajaraman, A. e Ullman, J. D. (2011). *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.

[Shivaswamy e Joachims 2011] Shivaswamy, P. K. e Joachims, T. (2011). Online Learning with Preference Feedback. *CoRR*, abs/1111.0712.

[Somefun e La Poutré 2007] Somefun, D. J. A. e La Poutré, J. A. (2007). A fast method for learning non-linear preferences online using anonymous negotiation data. In *Proceedings of the 2006 AAMAS workshop and TADA/AMEC 2006 conference on Agent-mediated electronic commerce: automated negotiation and strategy design for electronic markets*, TADA/AMEC'06, pp. 118–131, Hakodate, Japan, Berlin, Heidelberg. Springer-Verlag.

[Tan et al. 2005] Tan, P.-N., Steinbach, M., e Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Tsymbal 2004] Tsymbal, A. (2004). The problem of concept drift: definitions and related work. Technical Report TCD-CS-2004-15, The University of Dublin, Trinity College, Department of Computer Science, Dublin, Ireland.

[Urdan 2010] Urdan, T. (2010). *Statistics in Plain English, Third Edition*. Taylor & Francis.

[Vivekanandan e Nedunchezhian 2011] Vivekanandan, P. e Nedunchezhian, R. (2011). Mining data streams with concept drifts using genetic algorithm. *Artif. Intell. Rev.*, 36(3):163–178.

[Wilson 2004] Wilson, N. (2004). Extending CP-nets with Stronger Conditional Preference Statements. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pp. 735–741, San Jose, California. AAAI Press.

[Yong et al. 2012] Yong, Z., Jun, H., e He, G. (2012). *Bio-Inspired Computational Algorithms and Their Applications*. InTech.