Pontifícia Universidade Católica
do Rio de Janeiro

Franklin Anderson de Amorim

Mineração de Itens Frequentes em Sequências de Dados: Uma Implementação Eficiente Usando Vetores de Bits

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Departamento de Informática da PUC-Rio.

Orientador: Prof. Marco Antonio Casanova

Rio de Janeiro
Setembro de 2015

Franklin Anderson de Amorim

Mineração de Itens Frequentes em Sequências de Dados: Uma Implementação Eficiente Usando Vetores de Bits

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Departamento de Informática do Centro Técnico e Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Marco Antonio Casanova
Orientador
Departamento de Informática – PUC-Rio

Prof. Bernardo Pereira Nunes
Departamento de Informática – PUC-Rio

Prof[a]. Giseli Rabello Lopes
UFRJ

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 04 de setembro de 2015

Franklin Anderson de Amorim

Graduou-se em Ciência da Computação pela Universidade Iguaçu em dezembro de 1999. Tem experiência na área de Ciência da Computação, com ênfase em Banco de Dados. Atualmente trabalha como Especialista de Banco de Dados na empresa Globo.com.

Ficha Catalográfica

de Amorim, Franklin Anderson

Mineração de Itens Frequentes em Sequências de Dados: Uma Implementação Eficiente Usando Vetores de Bits/ Franklin Anderson de Amorim; orientador: Marco Antonio Casanova. - 2015.

48 f. : il. ; 29,7 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2015.

Inclui referências bibliográficas.

1. Informática - Teses. 2. Sequências de Dados. 3. Conjuntos de Itens Frequentes. 4. Mineração de Dados. I. Casanova, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

À minha esposa, Áurea,
que amo muito.

Agradecimentos

Ao meu orientador, Professor Marco Antonio Casanova, pela oportunidade de ter me acolhido como orientando, pelos ensinamentos ao longo do caminho e sobretudo pela compreensão e apoio transmitidos ao longo de todo o período.

À minha esposa, Áurea, pelo amor, carinho, incentivo e apoio incondicional em todos os momentos.

À minha família, aos meus amigos e aos meus professores que me ensinaram muitas coisas e tiveram paciência em lidar com minhas dificuldades.

À Globo.com, não só por custear o mestrado, como por me dar tempo e espaço para me dedicar aos estudos e pesquisas necessários.

Ao amigo Rodrigo Senra por algumas contribuições fundamentais que ajudaram a melhorar muito esta dissertação.

À Giseli Rabello Lopes e Bernardo Pereira Nunes pelo apoio e orientação.

Aos meus colegas da PUC-Rio, que foram muito receptivos e tornaram agradáveis os momentos na universidade.

Resumo

de Amorim, Franklin Anderson; Casanova, Marco Antonio. **Mineração de Itens Frequentes em Sequências de Dados: Uma Implementação Eficiente Usando Vetores de Bits.** Rio de Janeiro, 2015. 48p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A mineração de conjuntos de itens frequentes em sequências de dados possui diversas aplicações práticas como, por exemplo, análise de comportamento de usuários, teste de software e pesquisa de mercado. Contudo, a grande quantidade de dados gerada pode representar um obstáculo para o processamento dos mesmos em tempo real e, consequentemente, na sua análise e tomada de decisão. Sendo assim, melhorias na eficiência dos algoritmos usados para estes fins podem trazer grandes benefícios para os sistemas que deles dependem. Esta dissertação apresenta o algoritmo MFI-TransSW+, uma versão otimizada do algoritmo MFI-TransSW, que utiliza vetores de bits para processar sequências de dados em tempo real. Além disso, a dissertação descreve a implementação de um sistema de recomendação de matérias jornalísticas, chamado ClickRec, baseado no MFI-TransSW+, para demonstrar o uso da nova versão do algoritmo. Por último, a dissertação descreve experimentos com dados reais e apresenta resultados da comparação de performance dos dois algoritmos e dos acertos do sistema de recomendações ClickRec.

Palavras-chave

Sequências de Dados; Conjuntos de Itens Frequentes; Mineração de Dados.

Abstract

de Amorim, Franklin Anderson. Casanova, Marco Antonio. (Advisor). **Mining Frequent Itemsets in Data Streams: An Efficient Implementation using Bit Vectors.** Rio de Janeiro, 2015. 48p. MSc. Dissertation - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The mining of frequent itemsets in data streams has several practical applications, such as user behavior analysis, software testing and market research. Nevertheless, the massive amount of data generated may pose an obstacle to processing then in real time and, consequently, in their analysis and decision making. Thus, improvements in the efficiency of the algorithms used for these purposes may bring great benefits for systems that depend on them. This thesis presents the MFI-TransSW+ algorithm, an optimized version of MFI-TransSW algorithm, which uses bit vectors to process data streams in real time. In addition, this thesis describes the implementation of a news articles recommendation system, called ClickRec, based on the MFI-TransSW+, to demonstrate the use of the new version of the algorithm. Finally, the thesis describes experiments with real data and presents results of performance and a comparison between the two algorithms in terms of performance and the hit rate of the ClickRec recommendation system.

Keywords

Datastream; Frequent Itemsets; Data Mining.

Sumário

1 Introdução **11**
1.1 Motivação 11
1.2 Objetivos 12
1.3 Organização 13

2 Conceitos Básicos **14**
2.1 Mineração de Dados 14
2.2 Sequências de Dados 14
2.3 Conjuntos de Itens Frequentes 15
2.4 Filtragem Colaborativa 16
2.5 TF-IDF 17

3 Trabalhos Relacionados **19**

4 MFI-TransSW e sua Otimização **21**
4.1 MFI-TransSW 21
4.2 Otimização de MFI-TransSW para Sequências de Cliques 24

5 ClickRec **34**
5.1 Geração de Recomendações 34
5.2 Arquitetura 36
5.3 Processamento da Sequência de Dados 37
5.4 Mineração dos Conjuntos de Itens Frequentes 37
5.5 Geração de Recomendações 38
5.6 Implementação 38

6 Experimentos e Resultados **40**
6.1 Horizon: Sistema para Captura e Distribuição de Eventos da Globo.com 40
6.2 MFI-TransSW+: Experimento e Análise dos Resultados 41
6.3 ClickRec: Experimento e Análise dos Resultados 42

7 Conclusão e Trabalhos Futuros **46**

Referências Bibliográficas **47**

Lista de figuras

4.1 Lista Ordenada de Usuários (LOU): sequência de cliques. 25
4.2 Vetores de bits: sequência de cliques. 27
4.3 Vetores de bits: após deslocamento. 27
4.4 Vetores de bits: com ponteiro. 27
4.5 Lista dos vetores por posição. 27
4.6 Vetores de bits: após limpeza. 28
4.7 Lista de Vetores de Bits por Transação (LVBPT): exemplo. 28
4.8 Lista Ordenada de Usuários (LOU): sequência de cliques. 29
4.9 Lista de Vetores de Bits por Transação (LVBPT): sequência de cliques. 29
4.10 Algoritmo MFI-TransSW (original). 30
4.11 Algoritmo MFI-TransSW+ (otimizado). 32

5.1 Arquitetura. 36
5.2 Arquitetura Física. 39

6.1 Performance MFI-TransSW+. 42
6.2 Número de cliques por usuário. 43
6.3 ClickRec: Acertos nas recomendações do Globoesporte. 44
6.4 ClickRec: Acertos nas recomendações do GShow. 45

Lista de tabelas

4.1 Primeiras transações. 22
4.2 Vetores de bits: posição inicial. 22
4.3 Nova transação. 22
4.4 Vetores de bits: após nova transação. 23
4.5 Vetores de bits: após deslocamento. 23
4.6 Vetores de bits: novo exemplo. 24
4.7 Vetores de bits: executando operação AND. 24
4.8 Exemplo de sequência de cliques. 25
4.9 Outro exemplo de sequência de cliques. 26
4.10 Vetores de bits: representando uma sequência de cliques. 26
4.11 Exemplo de sequência de cliques (cont.). 29
4.12 Vetores de bits. 29

5.1 Anotações semânticas. 37
5.2 Vetores de bits: por anotação semântica. 37

6.1 Estrutura do clickstream da Globo.com. 41
6.2 Tempos de execução. 42

1
Introdução

1.1
Motivação

Diversas revoluções tecnológicas, sociais e comportamentais têm levado a um aumento sem precedentes da quantidade de informações geradas a cada dia, trazendo novas oportunidades e desafios. Para conseguir processar estes grandes volumes de dados de forma eficiente, indivíduos e organizações dependem de algoritmos de mineração de dados (*data mining*). Estes algoritmos normalmente trabalham sobre grandes bancos de dados estáticos. Entretanto, existe um caso onde os algoritmos tradicionais de mineração normalmente não são adequados: sequências de dados (*datastreams*). Estes fluxos contínuos de informação, que chegam em alta velocidade e em grande volume, possuem um valor significativo, que está diretamente associado à capacidade de processar os dados à medida que eles chegam, de preferência em tempo real. Por exemplo, sistemas de corretoras de ações da bolsa de valores precisam detectar padrões e emitir ordens de compra e venda em frações de segundos, roteadores de tráfego da Internet precisam analisar milhares de pacotes por segundo para detectar e bloquear possíveis ataques maliciosos. Em qualquer um desses casos a demora em processar os dados e tomar uma ação pode gerar grandes danos às organizações. Para estes casos novos algoritmos foram desenvolvidos, focando especificamente o problema de minerar sequências de dados [Gaber et al. 2005].

Dentre as diversas técnicas de mineração de dados uma das mais famosas e usadas é conhecida como *descoberta de conjuntos de itens frequentes* (*finding frequent itemsets*) [Agrawal et al. 1993]. Dada uma massa de dados de transações, seu objetivo é identificar conjuntos de itens que aparecem repetidas vezes em um grande número de transações. Já existem diversos algoritmos eficientes para descobrir conjuntos de itens frequentes em grandes bancos de dados tradicionais [Goethals et al. 2004]. Entretanto, ainda há muita pesquisa para desenvolver e otimizar algoritmos para descobrir conjuntos de itens frequentes em sequências de dados [Cheng et al. 2008]. Diferentemente da mineração de grandes bancos de dados tradicionais, a mineração de sequências de dados apre-

senta novos desafios, como a impossibilidade de ler os dados mais de uma vez, o grande volume e velocidade com que os dados chegam, a imprevisibilidade quanto a sua evolução e sua natureza virtualmente ilimitada.

1.2 Objetivos

O objetivo desta dissertação é aperfeiçoar a técnica de minerar conjuntos de itens frequentes em sequências de dados em tempo real ou, mais especificamente, o processamento de sequências de cliques em janelas deslizantes com baixo consumo de memória e latência. MFI-TransSW [Li et al. 2009] é um algoritmo promissor pois usa vetores de bits para descobrir conjuntos de itens frequentes em uma sequência de dados, o que o torna eficiente do ponto de vista de processamento e no consumo de memória. Entretanto a falta de uma etapa mais eficiente de deslizamento de janela e o não processamento de sequências de cliques abre espaço para melhorias. Assim, nesta dissertação apresentaremos o MFI-TransSW+, uma versão otimizada de MFI-TransSW, para processar sequências de cliques (*clickstreams*) em tempo real.

Além disso, descreveremos a implementação do ClickRec, um sistema de recomendações de itens que utiliza o MFI-TransSW+. Este sistema processa sequências de transações consumidas por usuários para gerar recomendações. O sistema foi desenvolvido com foco em portais de notícias e recomendação de matérias jornalísticas. Porém, sua abordagem permite que ele seja usado com qualquer aplicação que receba sequências de transações em alta velocidade, como Web sites de comércio eletrônico ou de transmissão de vídeos.

Sabemos que um dos principais desafios dos portais de notícias na Web encontra-se na apresentação de conteúdos relevantes aos usuários com o intuito de aumentar o tempo de permanência e navegação dos mesmos em seus Web sites. Por outro lado, dada a grande quantidade de notícias geradas diariamente, é cada vez mais difícil apresentar conteúdos relevantes aos usuários. Assim, um sistema automático de recomendação de notícias pode ser de grande auxílio para os portais de notícias. A ideia básica é que, ao terminar de ler um artigo, o usuário seja apresentado a um pequeno conjunto de outros artigos que possam lhe interessar. Diversas técnicas podem ser usadas para gerar estas recomendações, como agrupamento (*clustering*), árvores de decisão (*decision trees*) e similaridade de cosseno (*cosine similarity*). Nessa proposta, focaremos em uma técnica conhecida como filtragem colaborativa. Esta técnica, que baseia-se na recomendação de itens para usuários com preferências semelhantes, funciona muito bem com mineração de conjuntos frequentes de itens e será descrita em mais detalhes no Capítulo 2.

Resumindo, esta dissertação tem como principais objetivos:

- Apresentar o algoritmo MFI-TransSW+, uma versão otimizada do algoritmo de descoberta de conjuntos de itens frequentes em sequências de dados MFI-TransSW.
- Apresentar um sistema de recomendação baseado no MFI-TransSW+ chamado ClickRec.

1.3 Organização

Além do capítulo introdutório que apresenta a motivação, objetivos e contribuições esperadas como resultado deste trabalho, o restante desta dissertação está organizado da seguinte forma:

- Capítulo 2: Define conceitos básicos de mineração de dados, sequência de dados e conjuntos de itens frequentes utilizados ao longo desta dissertação.
- Capítulo 3: Revisa trabalhos relacionados bem como descreve o estado da arte para a mineração de conjuntos de itens frequentes em sequências de dados.
- Capítulo 4: Apresenta o algoritmo de mineração de conjuntos de itens frequentes em sequências de dados MFI-TransSW+.
- Capítulo 5: Introduz o sistema ClickRec, que utiliza técnicas de mineração de sequências de dados associadas a técnicas de filtragem colaborativa e similaridade entre documentos para gerar recomendações de itens a partir de sequências de cliques.
- Capítulo 6: Descreve experimentos realizados com dados reais (sequências de cliques extraídos de um portal de notícias) e os resultados obtidos.
- Capítulo 7: Apresenta as conclusões seguidas de uma discussão dos resultados e trabalhos futuros.

2
Conceitos Básicos

2.1
Mineração de Dados

Mineração de dados pode ser definida como o processo de descoberta de modelos em conjuntos de dados [Leskovec et al. 2014]. Este termo é normalmente usado para se referir ao processamento de grandes volumes de dados em busca de padrões, tais como regras de associação e séries temporais, que podem ser úteis em outros processos.

Nos últimos anos técnicas de mineração de dados têm sido amplamente adotadas e aplicadas a problemas na indústria, na academia e no governo gerando um impacto significativo em nossa sociedade. A percepção crescente de que a mineração de dados pode agregar valor real levou à demanda cada vez mais intensa por técnicas mais eficientes e eficazes.

Exemplos de uso de mineração de dados incluem detecção de fraude, processamento de anúncios em páginas da Web, processamento de imagens médicas, análise de DNA para detecção de doenças genéticas, análise de padrões de compra de consumidores em redes de varejo, análise de crédito de novos clientes de bancos, etc.

2.2
Sequências de Dados

Uma variada gama de aplicações geram grandes volumes de dados em tempo real. Isto cria um novo desafio para os sistemas que buscam obter informações a partir destas fontes de dados. *Sequências de dados* (*datastreams*) são fluxos contínuos de dados, sem fim e normalmente em alta velocidade, que podem mudar sua distribuição com o passar do tempo [Babcock et al. 2002].

Alguns exemplos de sequências de dados são dados provenientes de sensores, sequências de transações online em lojas de varejo, logs de registros de servidores, sequências de cliques de Web sites, tráfego de redes, cotação de bolsa de valores, entre outros.

Devido ao volume e velocidade com que os dados chegam, o processamento de cada item acaba sendo realizado de forma limitada, ou seja, o pro-

cessamento é realizado uma única vez já que armazenar toda a sequência em memória ou disco pode ser bastante custoso e ineficiente. A maioria dos algoritmos disponíveis para tratamento de sequências de dados gera resumos dos dados, descartando elementos desnecessários e calculando dados aproximados a partir de amostras da sequência de dados. Por exemplo, pode-se estimar o número de elementos distintos que apareceram na sequência de dados usando amostras dos dados, o que consome muito menos memória do que seria necessário se armazenássemos uma lista com todos os itens distintos existentes.

Entre os vários formatos de sequência de dados existentes, nesta dissertação focamos na *sequência de transações*, que consiste de uma sequência contínua e ilimitada de transações. Uma amostra que engloba um conjunto contínuo de transações da sequência de dados é chamado de janela (*window*). Podemos categorizar estas janelas em *janelas baseadas em tempo* (*time-based window*) e *janelas baseadas em contagem* (*count-based window*). Paralelo a isso, podemos classificar ainda as janelas em *janelas fixas* (*landmark window*) e *deslizantes* (*sliding window*).

Janelas baseadas em tempo: W é considerada uma janela baseada em tempo se W consiste de uma sequência de unidades de tempo de comprimento fixo, onde um número variável de transações pode chegar dentro de cada unidade de tempo.

Janelas baseadas em contagem: W é uma janela baseada em contagem se W é composto por uma sequência de lotes, em que cada lote é constituído por um número igual de transações.

Janelas fixas e deslizantes: W é uma janela fixa, se $W = \{T_1, T_2, ..., T_n\}$. Por outro lado, W é uma janela deslizante, se $W = \{T_{n-w+1}, ..., T_n\}$, onde cada T_i é uma unidade de tempo ou de um lote. T_1 e T_n são as unidades de tempo (ou lote) mais antiga e atual respectivamente, e w é o número de unidades de tempo (ou lotes) da janela deslizante.

2.3 Conjuntos de Itens Frequentes

Conforme visto em [Leskovec et al. 2014], para entender o conceito de conjuntos de itens frequentes devemos primeiro examinar o modelo de "cesta de produtos" (*market-basket*). Uma cesta de produtos é uma relação muitos-para-muitos entre dois tipos de objetos: cestas e itens. Cada cesta contém um conjunto, normalmente pequeno, de itens. Já o número de cestas, que também podem ser chamadas de transações, pode ser muito grande a ponto de não poder ser processado em memória principal. Note que este é exatamente o modelo empregado em um mercado, onde clientes selecionam produtos e os colocam

em uma cesta para depois levar a cesta ao caixa e pagar pela compra.

Espera-se que pessoas com preferências similares comprem os mesmos conjuntos de produtos. Assim, se um conjunto de itens (*itemset*) aparece em muitas cestas ele é considerado um conjunto de itens frequente (*frequent itemset*).

De acordo com Agrawal et al. [Agrawal et al. 1993], a mineração de conjuntos de itens frequentes diz respeito à identificação de conjuntos de itens que aparecem juntos em um grande número de transações. Grandes redes de supermercados foram as primeiras a utilizarem esta técnica para identificar produtos que costumam ser comprados juntos com o objetivo de desenvolver promoções e ações de marketing.

O problema da mineração de conjuntos de itens frequentes em uma sequência de dados pode ser definido da seguinte forma. Seja $I = \{x_1, x_2, ..., x_n\}$ um conjunto de itens e $X \subset I$ um subconjunto de itens. Um fluxo de dados transacional é uma sequência de transações recebidas, $D = (T_1, T_2, ..., T_N)$, onde uma transação T_i é um conjunto de itens e N é o número total de transações em D. O número de transações que contêm X em D é chamado o suporte de X, denotado como $sup(X)$. Um conjunto de itens X é frequente se e somente se $sup(X) \geq N \cdot s$, em que s é um limite, definido pelo usuário, chamado suporte mínimo tal que $s \in [0, 1]$.

O desafio da mineração de conjuntos de itens frequentes em sequências de dados está ligado à natureza da explosão combinatória do número de conjuntos de itens e ao espaço de memória máximo necessário para mineração dos conjuntos. Dado um domínio de I itens distintos, o número de conjuntos de itens possíveis é igual a $2^I - 1$. Conforme o comprimento da sequências de dados se aproxima de um número muito grande a possibilidade de um conjunto de itens ser frequente torna-se maior e mais difícil de controlar com memória limitada.

2.4 Filtragem Colaborativa

Sistemas de recomendação automatizam, total ou parcialmente, a tarefa de apresentar conteúdo relevante ou interessante para os usuários. São amplamente utilizados em várias plataformas online, incluindo Web sites de comércio eletrônico, redes sociais e portais de notícias, trazendo comprovada vantagem competitiva às empresas.

Podemos creditar o desenvolvimento dos sistemas de recomendação ao crescimento vertiginoso da Web e ao surgimento dos Web sites de comércio eletrônico (*e-commerce*). Esta tecnologia começou a ser estudada de forma

mais aprofundada nos anos 90 [Adomavicius et al. 2005].

De modo geral, o problema de recomendação pode ser descrito como o problema de prever as notas ou graus que um usuário atribuiria aos itens que ainda não qualificou.

Formalmente, seja C o conjunto de todos os usuários e S o conjunto de todos os itens que podem ser recomendados. u é uma função utilitária que mede a utilidade de um item s para um usuário c, ou seja, $u : C \times S \rightarrow R$, onde R é um conjunto ordenado de itens. Então, para cada $c \in C$, queremos escolher o item $s' \in S$ que maximize a utilidade para o usuário. Formalmente temos, para cada $c \in C$:

$$s'_c = \max_{s \in S} u(c, s)$$

Uma das mais conhecidas e bem sucedidas técnicas de recomendação é a filtragem colaborativa (*collaborative filtering*)[Resnick et al. 1994]. Esta técnica é baseada na suposição de que pessoas que concordaram no passado provavelmente irão concordar no futuro. Por exemplo, se o usuário A e o usuário B qualificaram de forma semelhante um conjunto de itens, é possível sugerir itens ao usuário A baseado nos itens que ele não qualificou mas que receberam boas qualificações do usuário B.

2.5 TF-IDF

De forma simplificada, o processo de consulta e recuperação de documentos a partir de palavras-chave se resume em analisar todos os documentos disponíveis e calcular o número de vezes que as palavras-chave da consulta aparecem nestes documentos. Os documentos que possuem maior pontuação devem ser a resposta à consulta do usuário. Digamos, por exemplo, que temos uma coleção de documentos sobre filmes e queremos achar os documentos que tratam do filme "Birdman". Neste caso, contamos o número de vezes que o termo "Birdman" aparece em cada um dos documentos da coleção. Aqueles que tem o maior número de ocorrências devem ser a resposta procurada. Entretanto, se buscarmos pelo filme "O Fabuloso Destino de Amélie Poulain", podemos encontrar documentos que têm várias ocorrências dos termos "O", "Fabuloso", "Destino" e "de" e dar uma pontuação alta para estes documentos, mesmo que nenhum deles possua os termos "Amélie" e "Poulain". Uma das formas de resolver este problema é usar a técnica TF-IDF, que consiste em dar maior peso às palavras mais raras, ou seja, que aparecem em menos documentos [Ramos et al. 2004]. TF-IDF determina a frequência relativa dos termos em um documento, comparada com a proporção inversa destes termos em todos os

documentos da coleção. Assim, termos que aparecem em todos os documentos recebem menor peso, e os que aparecem em poucos documentos recebem maior peso.

Formalmente, seja D uma coleção de documentos (*corpus*), w um termo e $d \in D$ um documento. Podemos calcular TF-IDF da seguinte forma:

$$w_d = f_{w,d} \cdot \log\left(\frac{|D|}{f_{w,D}}\right)$$

O $f_{w,d}$ é o número de vezes que o termo w aparece em d, $|D|$ é o tamanho do *corpus*, e $f_{w,D}$ é o número de documentos em que w aparece. Assim, se o termo aparece muitas vezes nesse documento, mas poucas vezes nos outros documentos do *corpus*, teremos um w_d com valor alto, enquanto que se o termo aparece muitas vezes neste documento mas também aparece em muitos outros documentos do *corpus*, teremos um w_d com valor baixo.

3
Trabalhos Relacionados

Os algoritmos de mineração de conjuntos de itens frequentes em sequência de dados podem ser classificados em duas categorias [Cheng et al. 2008]: os que usam janelas fixas e os que usam janelas deslizantes. Além disso, eles são classificados de acordo com os resultados que produzem: alguns são exatos e outros aproximados. Entre os aproximados podemos ainda classificar os algoritmos em falsos-positivos e falsos-negativos.

Agrawal et al. [Agrawal et al. 1994] propuseram o algoritmo A-Priori, uma das mais importantes contribuições para o problema de encontrar conjuntos de itens frequentes. O algoritmo é baseado na observação de que se um conjunto de itens i é frequente, então todos os subconjuntos de i também são frequentes.

Manku e Motwani [Manku & Motwani 2002] desenvolveram dois algoritmos de uma passada, Sticky-Sampling e Lossy Counting, para minerar conjuntos de itens frequentes em janelas fixas. Além disso, desenvolveram um método composto de três módulos chamado BTS (Buffer-Trie-SetGen) para a mineração de conjuntos de itens frequentes sobre sequências de dados. Ambos os algoritmos usam pouca memória, mas geram resultados aproximados, com falsos-positivos.

Li et al. [Li et al. 2004] propuseram o algoritmo de uma passada DSM-FI e Li et al. [Li et al. 2005] definiram o algoritmo DSM-MFI, para extrair o conjunto de todos os conjuntos de itens frequentes (FI) e os conjuntos de itens frequentes máximo (MFI) ao longo de toda a história de uma sequência de dados. Estes algoritmos utilizam uma estrutura de dados baseada em árvores de prefixo (*prefix-trees*) para armazenar os itens, seus respectivos valores de apoio, ids de janelas e links apontando para outros nós.

Yu et al. [Yu et al. 2006] propuseram algoritmos falso-positivo e falso-negativo baseados em Chernoff Bound [Chernoff 1952] para a mineração de conjuntos de itens frequentes em sequência de dados de alta velocidade. Os algoritmos usam um parâmetro de erro em tempo de execução para eliminar conjuntos de itens e usam um parâmetro de confiabilidade para controlar a memória.

Lee et al. [Lee et al. 2005] propuseram um algoritmo para filtragem de janelas deslizantes (SWF) para mineração incremental de conjuntos de itens frequentes dentro de uma janela deslizante. Chang et al. [Chang et al. 2004] propuseram um algoritmo baseado em BTS, chamado *SWFI-stream*, para encontrar conjuntos de itens frequentes dentro de uma janela deslizante sensível à transação. Chi et al. [Chi et al. 2006] propuseram o primeiro algoritmo para sequências de dados, chamado *Moment*, para extrair os conjuntos de itens frequentes fechados (*closed frequent itemsets*) dentro de uma janela deslizante sensível à transação. Li et al. [Li et al. 2009] desenvolveram o algoritmo MFI-TransSW, para minerar conjuntos de itens frequentes em janelas deslizante sobre sequências de dados usando vetores de bits e operações binárias.

O algoritmo MFI-TransSW+ proposto nesta dissertação processa sequências de cliques usando janelas deslizantes. Ao armazenar as transações em vetores de bits ele ocupa muito menos memória que outros algoritmos similares. Além disso, ele usa uma versão do algoritmo A-Priori sobre os vetores de bits para encontrar todos os conjuntos de itens frequentes sem apresentar resultados falsos-positivos ou falsos-negativos. Por último, seu grande diferencial é usar uma atualização circular dos vetores de bits ao fazer o deslizamento da janela, o que resulta em um ganho expressivo de desempenho, conforme veremos no capítulo 6.

4
MFI-TransSW e sua Otimização

Neste capítulo descreveremos o algoritmo MFI-TransSW e as otimizações implementadas para seu uso no sistema ClickRec.

4.1
MFI-TransSW

MFI-TransSW é um algoritmo de mineração de conjuntos de itens frequentes em sequências de dados. O algoritmo lê os dados apenas uma vez (*single pass mining*) e usa uma estrutura de vetores de bits para processar uma janela deslizante e extrair todos os conjuntos de itens frequentes, sem falsos-positivos ou falsos-negativos.

Cada item X presente na janela é representado por uma sequência de bits, ou vetor binário, chamado Bit(X), onde cada posição da sequência representa uma transação na janela e um bit 1 indica que o item está presente naquela transação. Por exemplo, se uma janela de tamanho $W = 3$ for composta das transações $T_1 = (ab)$, $T_2 = (b)$ e $T_3 = (abc)$, os vetores binários correspondentes aos itens da janela W seriam: Bit(a)=101, Bit(b)=111 e Bit(c)=001.

O algoritmo é composto de três etapas, descritas a seguir: (1)*inicialização da janela*; (2)*deslizamento da janela*; e (3)*geração dos conjuntos de itens frequentes*.

4.1.1
Inicialização da Janela

Ao iniciar, cada transação da sequência de dados é lida, uma a uma, na ordem em que chegam. A primeira etapa, *inicialização da janela*, conforme processa as transações, verifica se já existe um vetor de bits para cada novo item processado. Caso não exista, um novo vetor de bit é criado. O vetor tem um número de bits igual ao número de transações da janela, todos com valor 0, com exceção do bit correspondente à transação que contém o item, que tem valor 1. Caso o vetor já exista, o bit na posição correspondente à transação sendo processada é atualizado para 1. Podemos ver um exemplo nas tabelas 4.1 e 4.2. Quando o número de transações processadas nesta etapa é igual ao

tamanho da janela dizemos que a janela está preenchida e podemos começar a segunda etapa.

Janela	Tempo	Transação
W_1	T_1	abc
	T_2	c
	T_3	ac
	T_4	acd

Tabela 4.1: Primeiras transações.

Vetor	Bits
bit(a)	1011
bit(b)	1000
bit(c)	1111
bit(d)	0001

Tabela 4.2: Vetores de bits: posição inicial.

4.1.2 Deslizamento da Janela

Assim que a janela é preenchida, iniciamos a segunda etapa, *deslizamento da janela*. A cada nova transação lida é adicionado um bit no final de cada vetor de bits. O valor do bit é 1, se o item daquele vetor estiver contido na transação, e 0, em caso contrário. Caso não exista um vetor de bits para algum item da transação, um novo vetor é criado seguindo as mesmas regras da etapa 1.

No exemplo da tabela 4.3, a nova transação T_5 contém os itens a e b, de modo que na tabela 4.4 os vetores bit(a) e bit(b) recebem um novo bit 1 enquanto os outros vetores recebem o bit 0.

Janela	Tempo	Transação
W_2	T_2	c
	T_3	ac
	T_4	acd
	T_5	ab

Tabela 4.3: Nova transação.

Em seguida é executada uma operação de deslocamento de bits para a esquerda (*bitwise left shift*) eliminando o primeiro bit de cada vetor, como podemos ver na tabela 4.5. Neste momento, é feita uma operação de limpeza: contar o número de bits 1 em cada vetor de bits e eliminar os vetores que contiverem 0 bits 1.

Vetor	Bits
bit(a)	1011**1**
bit(b)	1000**1**
bit(c)	1111**0**
bit(d)	0001**0**

Tabela 4.4: Vetores de bits: após nova transação.

Vetor	Bits
bit(a)	0111
bit(b)	0001
bit(c)	1110
bit(d)	0010

Tabela 4.5: Vetores de bits: após deslocamento.

Este processo de *adicionar e deslocar* continua indefinidamente até que não haja mais transações na sequência de dados ou que o processamento seja interrompido.

4.1.3 Geração dos Conjuntos de Itens Frequentes

A terceira etapa, *geração dos conjuntos de itens frequentes*, pode ser executada a qualquer momento, por solicitação do usuário ou sempre que for necessário gerar os conjuntos de itens frequentes. Esta etapa é uma adaptação do algoritmo A-Priori [Agrawal et al. 1994] usando vetores de bits.

Ao ser acionada, o primeiro passo é calcular o suporte dos itens a partir da contagem dos bits 1 de todos os vetores de bits. Os itens que têm suporte maior ou igual ao suporte mínimo são selecionados como conjuntos de itens frequentes de tamanho 1 (FI_1). Em seguida é gerada uma lista de candidatos de tamanho 2 (CI_2) a partir da combinação dos itens de FI_1. Para verificar se os candidatos de CI_2 são frequentes, combinamos os vetores de bits dos itens de cada candidato usando uma operação *bitwise* AND, gerando um novo vetor binário, e contamos os bits 1 destes vetores. Aqueles que têm suporte maior ou igual ao suporte mínimo são selecionados como conjuntos de itens frequentes de tamanho 2 (FI_2). O processo continua de forma semelhante para gerar o CI_3 e FI_3, e assim por diante, até que não existam mais conjuntos de itens frequentes.

Na tabela 4.6 temos um exemplo. Se considerarmos, por exemplo, um suporte mínimo $s = 0,5$ e uma janela de tamanho $w = 4$ todo item que aparecer em pelo menos $(4 \times 0,5 = 2)$ transações é frequente.

Assim, no nosso exemplo, apenas os itens a e c são frequentes. Para identificar se o conjunto ac também é frequente, precisamos apenas gerar um

Vetor	Bits	Qtd bits 1
bit(a)	0111	3
bit(b)	0001	1
bit(c)	1110	3
bit(d)	0010	1

Tabela 4.6: Vetores de bits: novo exemplo.

novo vetor de bits bit(ac) usando a operação AND, conforme tabela 4.7.

Vetor	Bits	Qtd bits 1
bit(a)	0111	3
bit(c)	1110	3
bit(ac)	0110	2

Tabela 4.7: Vetores de bits: executando operação AND.

Umas das vantagens dessa abordagem é o baixo consumo de memória. Como cada item consome um número de bits igual ao tamanho da janela, o consumo total de memória em bytes é $((i \times w)/8)$, onde i=número de itens na janela e w=número de transações na janela. Assim, para uma janela de tamanho 100K transações e com 1K itens distintos, MFI-TransSW utiliza pouco menos de 12MB de memória.

4.2 Otimização de MFI-TransSW para Sequências de Cliques

Neste capítulo descreveremos as alterações feitas ao algoritmo MFI-TransSW original gerando uma versão otimizada, chamada MFI-TransSW+. Esta versão opera sobre sequências de cliques.

4.2.1 Processamento de Sequências de Cliques

Um sequência de cliques, ou *clickstream*, é um registro dos cliques de usuários recebidos nas várias páginas de um Web site. Os dados das sequências de cliques fornecem informações sobre a sequência de páginas ou o caminho percorrido pelo usuário ao navegar no site [Montgomery et al. 2004].

Uma sequência de cliques é um tipo de sequência de dados, onde os registros são normalmente no formato *Timestamp*, *Usuário*, *Página* e *ação*, conforme tabela 4.8.

Timestamp indica a hora que o clique foi executado, *Usuário* é o identificador do usuário que executou o clique, *Página* é a identificação da página acessada pelo clique, e *Ação* é a ação executada pelo usuário (exemplo: acessar, curtir, votar, etc). Como trataremos apenas de sequências de cliques

Timestamp	*Usuário*	*Página*	*Ação*
ts_1	1701	91001	1
ts_2	2000	91001	1
ts_3	0514	91003	1
ts_4	1701	91002	1
ts_5	1864	91003	1

Tabela 4.8: Exemplo de sequência de cliques.

com apenas uma ação (acessar a página) omitiremos este campo nos próximos exemplos.

O algoritmo MFI-TransSW original foi projetado para processar um sequência de transações, onde cada transação é um conjunto finito e bem conhecido de itens. Entretanto, em um sequência de cliques, cada item é um clique em um *link* (ou um *pageview*). Nunca sabemos quando uma transação está completa, uma vez que cada registro representa apenas um item acessado por um usuário naquele momento, e não sabemos se ele já terminou sua sessão ou se está apenas começando. Assim, no exemplo da tabela 4.8, após receber o quinto registro, sabemos apenas que o usuário 1701 acessou as páginas 91001 e 91002.

Para processar um sequência de cliques precisaremos fazer uma alteração no MFI-TransSW para manter um registro dos identificadores dos usuários (uid) que pertencem à janela atual. Este registro será na forma de uma Lista Ordenada de Uid's (LOU). Assim, cada usuário é considerado uma transação, e sua posição na lista LOU indica sua posição no vetor de bits. Um usuário que não existe na lista LOU é acrescentado no fim da lista. O tratamento em seguida é quase o mesmo que executamos ao recebermos uma nova transação, adicionando bits aos vetores de bits ou criando o vetor caso ele não exista e, caso a janela esteja preenchida, executando um *bitwise left shift*. A diferença é que ao fazer o *left shift* nos vetores de bits precisamos também executar um *left shift* na lista LOU, eliminando o primeiro uid. Porém, se o usuário já existe na lista LOU usamos sua posição na lista para atualizar o vetor de bits da página acessada para bit 1.

Na tabela 4.9 temos uma amostra de uma sequência de cliques e na figura 4.1 e tabela 4.10 a lista LOU e os vetores de bits resultantes, respectivamente.

0	1	2	3
001	002	003	004

Figura 4.1: Lista Ordenada de Usuários (LOU): sequência de cliques.

Timestamp	Usuário	Página
ts_1	001	A
ts_2	002	B
ts_3	003	A
ts_4	004	C
ts_5	002	A

Tabela 4.9: Outro exemplo de sequência de cliques.

Vetor	Bits
bit(A)	1110
bit(B)	0100
bit(C)	0001

Tabela 4.10: Vetores de bits: representando uma sequência de cliques.

Apesar de termos 5 itens na janela (ver tabela 4.9), existem apenas 4 usuários distintos. Por isso os vetores de bits só tem 4 bits cada. O quinto item da janela não criou mais um bit nos vetores de bits já existentes, apenas atualizou para 1 o segundo bit do vetor de bits bit(B) por se tratar do segundo usuário registrado na janela.

4.2.2 Vetor Circular de Bits

Um dos pontos principais do algoritmo MFI-TransSW é o uso do processo *adicionar e deslocar* nos vetores de bits para fazer o deslocamento da janela deslizante. Esta operação usa função *bitwise shift*, que é muito eficiente. Entretanto, ela precisa ser executada em todos os vetores de bits da janela para cada nova transação, mesmo os bits que representam itens que não façam parte da nova transação.

A solução para este problema foi trocar o processo de *adicionar e deslocar* por um preenchimento circular dos vetores de bits, que chamamos de *limpar e atualizar*, reduzindo substancialmente o número de vetores atualizados durante o processo de deslizamento de janela.

Por exemplo, digamos que recebemos uma nova transação e precisamos colocar-la na nossa janela, representada pelos vetores de bits da figura 4.2. Como a janela está completa, com todas as 8 posições dos vetores de bits preenchidas, precisamos efetuar o deslizamento de 1 bit para a esquerda de todos os vetores de bits para salvar a nova transação. Para isso teremos que executar a operação de *bitwise shift* em cada um dos cinco vetores de bits, terminando com a configuração mostrada na figura 4.3, liberando a posição 7 para ser usada para registrar a nova transação.

Em nossa abordagem trocamos o deslocamento por uma limpeza e

	0	1	2	3	4	5	6	7
A	0	1	0	0	0	0	1	0
B	1	1	0	0	0	1	0	1
C	0	0	1	0	0	0	1	0
D	0	0	0	1	0	0	0	0
E	0	0	0	0	1	0	0	1

Figura 4.2: Vetores de bits: sequência de cliques.

	0	1	2	3	4	5	6	7
A	1	0	0	0	0	1	0	0
B	1	0	0	0	1	0	1	0
C	0	1	0	0	0	1	0	0
D	0	0	1	0	0	0	0	0
E	0	0	0	1	0	0	1	0

Figura 4.3: Vetores de bits: após deslocamento.

atualização circular do grupo de vetores. Isso é feito mantendo um ponteiro da última posição preenchida da janela e uma lista com os vetores de bits preenchidos com bit 1 em cada posição, conforme figuras 4.4 e 4.5.

							↓	
	0	1	2	3	4	5	6	7
A	0	1	0	0	0	0	1	0
B	1	1	0	0	0	1	0	1
C	0	0	1	0	0	0	1	0
D	0	0	0	1	0	0	0	0
E	0	0	0	0	1	0	0	1

Figura 4.4: Vetores de bits: com ponteiro.

0	1	2	3	4	5	6	7
B	AB	C	D	E	B	AC	BE

Figura 4.5: Lista dos vetores por posição.

Para fazer o preenchimento circular, ao receber uma nova transação o processo calcula a próxima posição a ser preenchida a partir do módulo do

tamanho da janela, com a fórmula abaixo:

$$(p + 1) \bmod w$$

Onde p é o ponteiro da última posição preenchida e w é o tamanho da janela. Assim, usando o exemplo da figura 4.4, ao invés de fazer um deslocamento, calculamos a próxima posicão $((7 + 1) \bmod 8 = 0)$. Em seguida verificamos a lista da figura 4.5 para identificar quais vetores de bits precisam ser "apagados" (atualizados para bit 0) na posição 0 (nesse caso, apenas o vetor B). Agora esta posição está pronta para receber a nova transação, conforme figura 4.6.

	↓							
	0	1	2	3	4	5	6	7
A	0	1	0	0	0	0	1	0
B	0	1	0	0	0	1	0	1
C	0	0	1	0	0	0	1	0
D	0	0	0	1	0	0	0	0
E	0	0	0	0	1	0	0	1

Figura 4.6: Vetores de bits: após limpeza.

Voltando ao algoritmo MFI-TransSW+, durante a primeira etapa deste, a janela é preenchida da forma regular, mas, para cada transação, guardamos o identificador dos vetores que tem o bit 1 para aquela transação em uma Lista de Vetores de Bit por Transação (LVBPT). Vale ressaltar que, em sequências de cliques, o conjunto de páginas acessadas por um usuário costuma ser uma pequena fração do total de páginas acessadas por todos os usuários na janela (conforme veremos nos experimentos com dados reais no capítulo 6). Consequentemente, é muito mais eficiente manter a lista LVBPT do que simplesmente "zerar" todos os vetores de bits da posição sendo liberada.

Usando o exemplo da tabela 4.9 e figura 4.1 criamos a lista LVBPT vista na figura 4.7.

0	1	2	3
A	BA	A	C

Figura 4.7: Lista de Vetores de Bits por Transação (LVBPT): exemplo.

Na segunda etapa do MFI-TransSW+, substituímos o *adicionar e deslocar* pelo *limpar e atualizar*: sobrescreveremos os vetores de bits a partir da

primeira posição conforme executamos o deslizamento da janela. Assim, para uma janela de tamanho W, ao processar a transação de número T_{W+1}, voltamos a apontar para a posição 0 dos vetores de bits e "apagamos" (atualizamos para bit 0) os vetores de bits listados na posição 0 da lista LVBPT. Em seguida, ainda tratando da posição 0, atualizamos com bit 1 o vetor de bits da página que está sendo acessada. Para terminar, atualizamos as listas LOU e LVBPT com os dados da transação T_{W+1}. Para a transação seguinte, T_{W+2}, executamos o mesmo procedimento, mas com a posição 1, e assim por diante.

Continuando o exemplo da tabela 4.9, se considerarmos que a janela tem tamanho $W = 4$, ao recebermos uma nova transação T_6 de um novo usuário que não está na janela atual, executamos um deslizamento de janela, como mostrado nas tabelas 4.11 e 4.12 e nas figuras 4.8 e 4.9.

Timestamp	Usuário	Página
ts_1	001	A
ts_2	002	B
ts_3	003	A
ts_4	004	C
ts_5	002	A
ts_6	005	C

Tabela 4.11: Exemplo de sequência de cliques (cont.).

0	1	2	3
005	002	003	004

Figura 4.8: Lista Ordenada de Usuários (LOU): sequência de cliques.

Vetor	Bits
bit(A)	0110
bit(B)	0100
bit(C)	1001

Tabela 4.12: Vetores de bits.

0	1	2	3
C	BA	A	C

Figura 4.9: Lista de Vetores de Bits por Transação (LVBPT): sequência de cliques.

Assim, o usuário 005 assume a posição 0 da janela, que antes era ocupada pelo usuário 001. Usando a lista LVBPT (figura 4.7) atualizamos para 0 o

primeiro bit do vetor bit(A). Com isso, a posição 0 de todos os vetores de bits está setado para 0. Em seguida atualizamos o vetor de bits da página sendo acessada pela transação T_6 para 1, conforme vemos na tabela 4.12. Por último, atualizamos a lista de usuários com o ponteiro para o vetor de bits que foi atualizado pelo usuário 005, conforme figura 4.9.

Como os usuários acessam apenas uma fração das páginas disponíveis, o processo de *limpar e atualizar* é bem mais rápido do que o de *adicionar e deslocar*. Nos nossos testes identificamos um redução no tempo de processamento da ordem de 100 vezes usando o algoritmo otimizado. No capítulo 6 apresentaremos os resultados e análises destes experimentos.

```
input  : TDS (uma sequência de dados de transações), s (o suporte
         mínimo, definido pelo usuário, no intervalo de [0, 1]), e w (o
         tamanho da janela, especificado pelo usuário).
output: um conjunto de frequent itemsets, FI-Output.
1  TransSW ← NULL;
2  for T_i in TDS do
3  |  for i in T_i do
4  |  |  Execute bit-sequence transform(i);
5  |  end
6  |  if TransSW = FULL then
7  |  |  Execute bitwise-shift em todos os vetores de bits da janela
   |  |  TransSW, eliminando os que tiverem suporte = 0;
8  |  end
9  end
   /* A seguir a geração dos frequent itemsets. Esta parte é
      executada por demanda do usuário.                        */
10 FI_1 ← {frequent 1-itemsets};
11 for (k←2; FI_{k-1} ≠ NULL; k++) do
12 |  CI_k ← CIGA(FI_{k-1}) ;
13 |  Executa bitwise AND para achar os suportes de CI_k;
14 |  for c_k ∈ CI_k do
15 |  |  if sup(c_k)^{TransSW} ≥ w · s then
16 |  |  |  FI_k ← ∪c_k;
17 |  |  end
18 |  end
19 |  FI-Output ← ∪_k FI_k;
20 end
```

Figura 4.10: Algoritmo MFI-TransSW (original).

Na figura 4.10 temos o algoritmo MFI-TransSW (original). Foi mantido o pseudocódigo utilizando a descrição originalmente proposta em [Li et al. 2009] com o objetivo de realçar as diferenças entre o algoritmo original e o otimizado (proposto nesta dissertação).

Primeiramente, na linha 1 do algoritmo original, é inicializada uma janela vazia TransSW. As linhas 2 a 9 processam cada transação recebida de TDS. A linha 2 lê uma transação da sequência de dados. As linhas 3 a 5 lêem cada item da transação e executam a função *bit-sequence* $transform$. Esta função cria o vetor de bits para o item i. Caso o vetor já exista, ela adiciona um bit 1 no fim do vetor de i. Em seguida a função adiciona um bit 0 a todos os outros vetores. As linhas 6 a 8 executam o deslizamento da janela. A linha 6 verifica se a janela já está preenchida. A linha 7, que é executada se a janela estiver preenchida, faz o deslocamento dos bits de todos os vetores da janela uma posição para a esquerda, eliminando o primeiro bit. Esta mesma função, ao deslocar o vetor de bits, verifica se o mesmo ainda possui algum bit 1. Caso não tenha, o vetor é eliminado.

As linhas 10 à 20 geram os conjuntos de itens frequentes da janela TransSW atual. Esta parte do algoritmo é executada sob demanda. A linha 10 seleciona todos os conjuntos de itens frequentes de tamanho $k = 1$. Esta operação é feita simplesmente contando os bits 1 dos vetores. Aqueles que tiverem mais bits 1 do que o suporte mínimo da janela TransSW ($s \cdot w$) são armazenados em FI_1. As linhas 11 a 20 são o *loop* que é executado para cada tamanho k de conjunto de itens frequente. A linha 11 inicializa o loop com $k = 2$, e a cada execução incrementa k de 1, até que o último *loop* executado $(k - 1)$ não tenha retornado nenhum conjunto de itens frequentes.

A linha 12 chama a função $CIGA$ para gerar os candidatos CI_k a partir da lista de conjuntos de itens frequentes de tamanho $k - 1$. Para calcular o suporte dos novos candidatos é executado a função *bitwise AND* nos conjuntos de itens de CI_k. As linhas 14 a 18 identificam quais candidatos são frequentes. Na linha 14 selecionamos os candidatos da lista de candidatos CI_k. Na linha 15 verificamos se o suporte do candidato c_k é maior que o suporte mínimo. Caso seja, na linha 16 o candidato é adicionado a lista FI_k dos conjuntos de itens frequentes de tamanho k. Por último, a lista de conjuntos de itens frequentes FI_k é unida à lista total dos conjuntos de itens frequentes da janela *FI-Output*.

Na figura 4.11 temos o algoritmo MFI-TransSW+ (otimizado). Sua principal diferença para o algoritmo original está nas linhas 2 a 14. Diferentemente do algoritmo original, o MFI-TransSW+ processa sequências de cliques (linha 2), mantém uma lista de usuários da janela (linhas 3 e 4), além de fazer o preenchimento circular dos vetores de bits durante o deslizamento da janela (linhas 8 a 12).

Na linha 1 do algoritmo otimizado uma janela vazia TransSW é inicializada. As linhas 2 a 14 são o *loop* que processa os cliques recebidos da sequência de cliques CDS. A linha 2 lê o usuário u e o item i. A linha 3

```
   input  : CDS (uma sequência de cliques), s (o suporte mínimo,
            definido pelo usuário, no intervalo de [0, 1]), e w (o
            tamanho da janela, especificado pelo usuário).
   output: um conjunto de frequent itemsets, FI-Output.
 1 TransSW ← NULL;
 2 for (u, i) in CDS do
 3 |   if u ∈ U^TransSW then
 4 |   |   Execute user-visit-document(u, i);
 5 |   else
 6 |   |   if TransSW ≠ FULL then
 7 |   |   |   Execute bit-sequence transform(i);
 8 |   |   else
 9 |   |   |   p ← (p + 1) mod w ;
10 |   |   |   Execute clean-position(p) (eliminando os vetores de bits
   |   |   |   que tiverem suporte = 0);
11 |   |   |   Execute user-visit-document(u, i);
12 |   |   end
13 |   end
14 end
   /* A seguir a geração dos frequent itemsets. Esta parte é
      executada por demanda do usuário.                      */
15 FI_1 ← {frequent 1-itemsets};
16 for (k←2; FI_{k-1} ≠ NULL; k++) do
17 |   CI_k ← CIGA(FI_{k-1}) ;
18 |   Executa bitwise AND para achar os suportes de CI_k;
19 |   for c_k ∈ CI_k do
20 |   |   if sup(c_k)^TransSW ≥ w · s then
21 |   |   |   FI_k ← ∪c_k;
22 |   |   end
23 |   end
24 |   FI-Output ← ∪_k FI_k;
25 end
```

Figura 4.11: Algoritmo MFI-TransSW+ (otimizado).

verifica se o usuário já existe na janela TransSW. Se existir, na linha 4 é executada a operação *user-visit-document*, que atualiza o vetor de bits do item i, na posição $pos(u)^{TransSW}$ com bit 1. As linhas 5 a 13 são executadas se o usuário u for um novo usuário na janela TransSW. Na linha 6 é verificada se a janela está preenchida. Se não estiver, na linha 7 é executada a função *bit-sequence* $transform$ (a mesma do algoritmo original). As linhas 8 a 12 são executadas se a janela já estiver preenchida. A linha 9 move o ponteiro p para a próxima posição da janela a ser sobrescrita. A linha 10 executa a operação que limpa apenas os vetores ocupados pelo usuário atual da posição p, atualizado nestes vetores a posição p de bit 1 para bit 0. Ao atualizar o vetor, é

verificado se o mesmo ficou apenas com bits 0. Se este for o caso, o vetor é eliminado. A linha 10 executa a operação *user-visit-document*. As linhas 15 a 25 são exatamente iguais às linhas 10 a 20 do algoritmo original.

5
ClickRec

Neste capítulo descreveremos o sistema ClickRec, seus componentes, sua arquitetura e implementação.

O principal objetivo do sistema ClickRec é demonstrar o uso do algoritmo MFI-TransSW+ em uma aplicação de grande volume de dados do mundo real.

5.1
Geração de Recomendações

O ClickRec é um sistema de recomendação para Web sites de notícias que processa em tempo real as sequências de cliques de um portal de notícias Web e, usando o algoritmo MFI-TransSW+, minera conjuntos de itens frequentes para extrair as seguintes informações da sequência de dados:

- Quais são as matérias comumente acessadas juntas?
- Quais usuários acessam os mesmos conjuntos de matérias?

Inicialmente descrevemos o sistema ClickRec e, em seguida, apresentamos três abordagens testadas para geração de recomendações, além de apresentar a abordagem escolhida e sua justificativa.

5.1.1
Complemento de Matérias

Nesta primeira abordagem vamos identificar conjuntos de matérias que são acessadas juntas e identificar usuários que acessaram um subconjunto destes conjuntos e lhes recomendar o complemento.

Para ilustrar, vamos supor que, em um portal de notícias, as matérias A, B e C foram acessadas conjuntamente por um grande número de usuários durante a última hora. A partir desta informação podemos deduzir que estas três matérias possuem alguma relação, direta ou indireta. Como um usuário costuma consumir as matérias em uma sequência, podemos concluir que ao acessar uma das matérias algo o fez acessar as outras duas. Podemos utilizar esta lógica para gerar as recomendações: se, na hora seguinte, um usuário u acessar a matéria A, podemos concluir que ele também terá interesse nas matérias B e C, e assim lhe recomendar estas matérias.

5.1.2 Complemento de Metadados

Outra abordagem envolve usar os metadados das matérias, como tags ou anotações semânticas, e minerar conjuntos de itens frequentes a partir destes metadados. Se, por exemplo, a matéria A possui as anotações semânticas a_1 e a_2, e a matéria B possui a anotação semântica b_1, e em uma janela W um usuário acessar as duas matérias, sua transação em W será $t = \{a_1, a_2, b_1\}$. Ao usar os metadados das matérias como itens das transações dos usuários podemos executar o processo de mineração de conjuntos de itens frequentes sobre os metadados.

Podemos agora gerar recomendações a partir dos complementos de metadados. Inicialmente mineramos os conjuntos de itens frequentes da janela W. Em seguida, para cada usuário u que acessa uma matéria m, extraímos os metadados d_m da matéria. Por fim buscamos nos conjuntos de itens frequentes da janela W um conjunto d_w que seja um superconjunto de d_m. Se encontramos, usamos o complemento, ou seja, a diferença entre d_w e d_m, para encontrar matérias na janela W que vamos recomendar ao usuário u.

Digamos, por exemplo, que em uma janela W de um portal de notícias de esporte identificamos que o conjunto de anotações semânticas <barcelona>, <neymar> e <flamengo> é frequente. Caso um usuário u, em seguida, acesse uma matéria com as anotações <barcelona> e <neymar>, podemos sugerir matérias com a anotação <flamengo> para este usuário.

5.1.3 Similaridade de Metadados

Uma terceira abordagem, ainda usando metadados, como tags, anotações semânticas ou mesmo editorias, seria recomendar matérias por similaridade de metadados. Para isso, primeiro mineramos os conjuntos de itens mais frequentes da janela W. Em seguida, quando um usuário u acessa uma matéria m comparamos os metadados d_m da matéria acessada com os conjuntos de itens frequentes da janela W, para encontrar o conjunto mais similar s. Uma vez identificado o conjunto mais similar s, procuramos dentre as matérias acessadas na janela as N matérias com metadados mais similares ao conjunto s. Estas N matérias serão as recomendadas aos usuários.

Por exemplo, se o usuário acessa uma matéria que tem o conjunto de anotações <barcelona>, <neymar> e <flamengo>, e identificamos que na última janela o conjunto de itens frequente mais similar ao conjunto do usuário é <barcelona>, <neymar>, <flamengo> e <copa-america>, usamos

este conjunto de anotações para encontrar outras matérias acessadas na janela que tenham um conjunto similar de anotações semânticas.

Em nosso sistema utilizamos TF-IDF para fazer as comparações de similaridades.

5.1.4 Abordagem Implementada

As três abordagens são válidas e poderiam gerar bons resultados. Porém, durante a análise das sequências de cliques, identificamos que a grande maioria dos usuário (55% $\sim$ 70%) acessa apenas uma página (estes usuários são comumente chamados de *bounce users*). Isso dificulta a geração de conjuntos de itens frequentes baseados apenas em matérias, invalidando a primeira abordagem. Depois de executar vários experimentos com dados reais, concluímos que a terceira abordagem seria a mais eficiente para o nosso sistema. No capítulo 6 apresentaremos os resultados destes experimentos.

5.2 Arquitetura

Como podemos ver na figura 5.1, o sistema ClickRec é composto de 3 módulos.

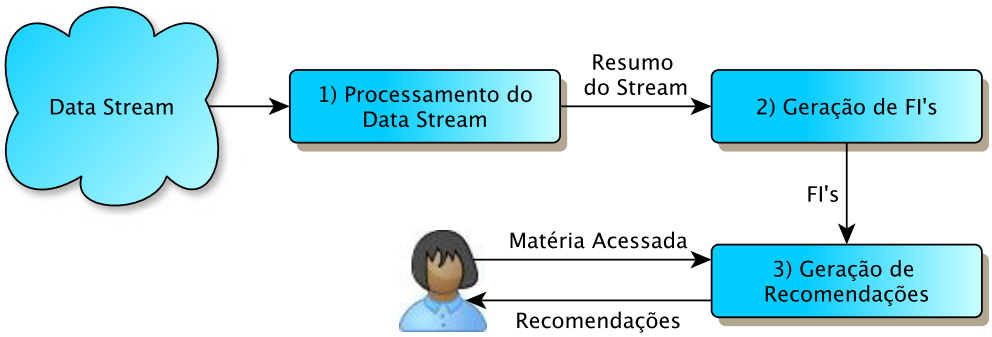

Figura 5.1: Arquitetura do sistema ClickRec.

O módulo 1 do sistema lê continuamente a sequência de dados e gera um resumo. Este resumo é usado pelo módulo 2 para gerar os conjuntos de itens frequentes. Por último, o módulo 3 gera as recomendações para o usuário a partir da matéria acessada e dos conjuntos de itens frequentes recebidos do módulo 2.

A seguir detalharemos estes três módulos: *processamento da sequência de dados*, *mineração de conjuntos de itens frequentes* e *geração de recomendações*.

5.3 Processamento da Sequência de Dados

Este módulo é responsável por processar continuamente a sequência de cliques, mantendo uma janela de transações com o resumo. Usando as etapas 1 e 2 do algoritmo MFI-TransSW+ descrito no capítulo anterior, ele cria ou atualiza os vetores de bits, e suas estruturas relacionadas, dos itens que fazem parte da janela.

O módulo aceita como parâmetro o tamanho da janela, em número de transações, que deve ser definido pelo usuário ao iniciar o módulo.

Para poder atender à abordagem definida na seção anterior precisamos manter vários vetores de bits para cada matéria acessada, de acordo com as várias anotações semânticas associadas a cada matéria.

Usando o exemplo da tabela 4.9, digamos que as matérias desta janela estão anotadas conforme a tabela 5.1.

Matéria	*A*	*B*	*C*
Anotações	<flamengo>, <neymar>, <barcelona>	<fluminense>, <fred>	<fluminense>

Tabela 5.1: Anotações semânticas.

Considerando uma janela de tamanho $w = 4$ teríamos os vetores de bits conforma a tabela 5.2.

Vetor	Bits
bit(<flamengo>)	1110
bit(<neymar>)	1110
bit(<barcelona>)	1110
bit(<fluminense>)	0101
bit(<fred>)	0100

Tabela 5.2: Vetores de bits: por anotação semântica.

5.4 Mineração dos Conjuntos de Itens Frequentes

Este módulo é executado a qualquer momento depois que a janela de transações for preenchida. Pode ser acionado pontualmente conforme necessário ou pode ser programado para executar em intervalos regulares de tempo.

Ao ser acionado, o módulo processa a janela atual em busca dos conjuntos de itens frequentes. Para tanto é executada a etapa 3 do algoritmo MFI-TransSW+ sobre os vetores de bits gerados na etapa anterior.

O suporte mínimo é um parâmetro que pode ser definido a cada execução do módulo, mas que normalmente fica salvo em um arquivo de configuração. Outro parâmetro é o tamanho máximo dos conjuntos de itens frequentes, que também pode ser definido a cada execução. Em nossos experimentos identificamos que conjuntos de itens frequentes maiores que 4 demoram a ser gerados e não trazem melhores resultados na recomendação.

A saída desse módulo é uma lista de conjuntos de itens frequentes de anotações semânticas.

5.5 Geração de Recomendações

Este módulo utiliza os conjuntos de itens frequentes minerados na etapa anterior para gerar recomendações para os usuários do Web site. O módulo é acionado toda vez que é necessário gerar recomendações para um usuário que está acessando uma página. Por ser bastante rápido podemos servir a página solicitada pelo usuário já com as recomendações geradas em tempo real.

O processo acontece da seguinte forma: o usuário entra com uma URL no seu *Web browser* solicitando uma página ao servidor de páginas Web. O servidor Web, antes de retornar a página ao usuário, faz uma consulta via API ao módulo *geração de recomendações* do sistema ClickRec. O módulo, usando a informação da página acessada e com base nos conjuntos de itens frequentes pré-gerados pelo módulo *mineração de conjuntos de itens frequentes*, gera um conjunto de recomendações, que é retornado ao servidor Web, que então monta e serve a página solicitada ao usuário já com as recomendações.

Para gerar as recomendações, o módulo, ao receber a identificação da matéria que o usuário está solicitando, extrai o conjunto de anotações semânticas da página. Usando a técnica TF-IDF, identifica o conjunto de itens frequentes mais similar a este conjunto de anotações. Então, usando TF-IDF mais uma vez, identifica as páginas que têm as anotações mais semelhantes. Este conjunto é a recomendação que será devolvida ao usuário.

5.6 Implementação

O sistema foi implementado usando Python[1], linguagem de programação de código aberto criada por Guido van Rossum em 1991 e que tem se tornado muito popular no desenvolvimento de aplicações Web. Entre suas vantagens estão a legibilidade do código fonte, boa performance, constante atualização e popularidade, tanto na academia quanto na indústria.

[1] `https://www.python.org/`

Para armazenar os vetores de bits e outras estruturas de dados foi escolhido o Redis[2], um banco de dados em memória de código aberto, criado em 2009 por Salvatore Sanfilippo. Usando uma estrutura de chave-valor, sua principal vantagem, além da simplicidade, é sua ótima performance e um baixo custo computacional. Além disso, o Redis possui uma funcionalidade que o torna especialmente útil para nosso trabalho: um conjunto de funções que permitem efetuar operações binárias com cadeias de caracteres, que são tratadas como vetores de bits.

Na figura 5.2 podemos ver uma representação da arquitetura do sistema.

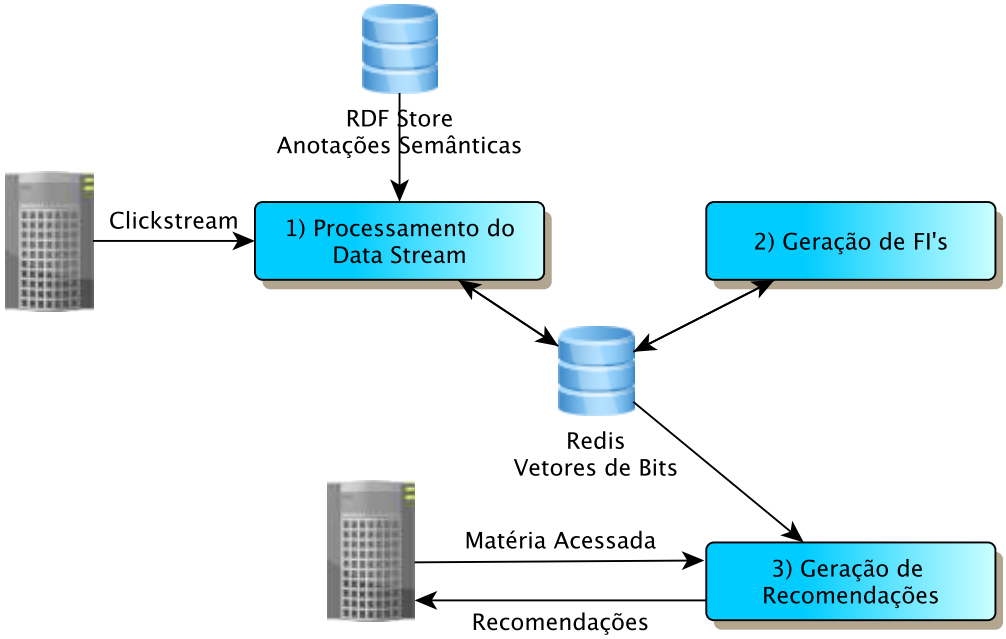

Figura 5.2: Arquitetura física do sistema ClickRec.

[2]http://redis.io/

6
Experimentos e Resultados

Neste capítulo apresentamos dois experimentos realizados com o algoritmo MFI-TransSW+. O primeiro experimento compara diretamente os desempenhos dos algoritmos MFI-TransSW (original) e MFI-TransSW+ (otimizado). O segundo experimento valida os resultados do sistema de recomendações ClickRec. Ambos os experimentos usam a sequência de cliques capturada pelo sistema Horizon, da Globo.com, descrito a seguir.

6.1
Horizon: Sistema para Captura e Distribuição de Eventos da Globo.com

Horizon é um sistema da Globo.com para captura de ações do usuário, como (*pageviews*, cliques, avaliações, respostas de questionários, etc). Além disso, o sistema possui uma API para consulta às ações capturadas.

Os dados capturados têm dois destinos, um cluster de servidores Apache Kafka[1] e um cluster de bancos de dados Apache HBase[2].

Atualmente apenas as ações de *pageview* são capturadas e para um número limitado de editorias (G1[3], Globo Esporte[4], Ego[5], GShow[6], Techtudo[7], Música[8] e Educação[9]). No total, estas sete editorias geram em média 25 milhões de *pageviews* por dia.

Para os testes realizados usamos os arquivos extraídos do HBase, em formato CSV. Cada arquivo equivale a um período de uma hora de sequência de cliques das sete editorias. Por exemplo, o arquivo `rt-actions-read-2015_01_14_23.log` possui os cliques do dia 14 de janeiro de 2015, das 23:00:00 até as 23:59:59, das sete editorias.

Na tabela 6.1 temos a estrutura de dados desses arquivos.

[1] `http://kafka.apache.org/`
[2] `http://hbase.apache.org`
[3] `http://g1.globo.com/index.html`
[4] `http://globoesporte.globo.com/`
[5] `http://ego.globo.com/`
[6] `http://gshow.globo.com/`
[7] `http://www.techtudo.com.br/`
[8] `http://musica.com.br/`
[9] `http://g1.globo.com/educacao/`

Atributo	Descrição
Document Id	Identificador da página acessada
Product Id	Identificador da editoria
Provider Id	Globo, Facebook ou anônimo
User Id	Identificador do usuário
Timestamp	Data e hora do acesso

Tabela 6.1: Estrutura do clickstream da Globo.com.

Usamos o campo *Product Id* para filtrar a editoria que estamos processando. Os campos *User Id*, *Document Id* e *Timestamp* são usados pelo sistema ClickRec para identificar o usuário, a página acessada e a data e hora do acesso, respectivamente. O Campo *Provider Id* é ignorado.

6.2 MFI-TransSW+: Experimento e Análise dos Resultados

Para comparar a performance dos algoritmos MFI-TransSW (original) e MFI-TransSW+ (otimizado) usamos um arquivo com uma amostra de uma hora de sequência de cliques extraída do HBase, conforme descrito na seção anterior. O teste possui duas etapas: primeiro, o programa lê do arquivo CSV o número necessário de ações até preencher a janela de tamanho w de vetores de bits. Em seguida, na segunda etapa, o programa lê o número de ações do arquivo CSV necessárias para executar 10k deslizamentos de janela. Fazemos a medição do tempo gasto apenas na segunda etapa.

Os dois algoritmos foram implementados em Python e os vetores de bits foram criados com o módulo Bitarray[10]. Todos os testes foram executados em um computador com processador Intel Core I5 de 2,5GHz e 16GB de memória RAM.

Efetuamos os testes com janelas de tamanhos $w = 1.000$ até $w = 10.000$, com intervalos de 1.000. Na tabela 6.2 temos os resultados para os vários tamanhos de janelas e na figura 6.1 quantas vezes mais rápido é o algoritmo otimizado em comparação ao algoritmo original.

A diferença substancial no desempenho dos dois algoritmos se deve ao fato de o MFI-TransSW+ executar muito menos operações a cada deslizamento. Em uma janela de tamanho $w = 10.000$ chegamos a ter mais de 20k vetores de bits. Neste momento, com a janela cheia, a cada nova transação o algoritmo original precisa efetuar o *bitwise shift* em cada um dos vetores de bits. Já a nossa versão otimizada só precisa executar a limpeza dos vetores de bits ocupados pelo usuário que está sendo sobrescrito.

[10] `https://github.com/ilanschnell/bitarray`

Tamanho da Janela	Tempo de execução (segundos)	
	MFI-TransSW	MFI-TransSW+
1.000	41,45	0,40
2.000	136,73	0,63
3.000	272,23	0,95
4.000	395,54	1,17
5.000	533,10	1,28
6.000	761,30	1,59
7.000	996,09	1,91
8.000	1.295,16	2,07
9.000	1.484,10	2,22
10.000	1.928,76	2,36

Tabela 6.2: Tempos de execução.

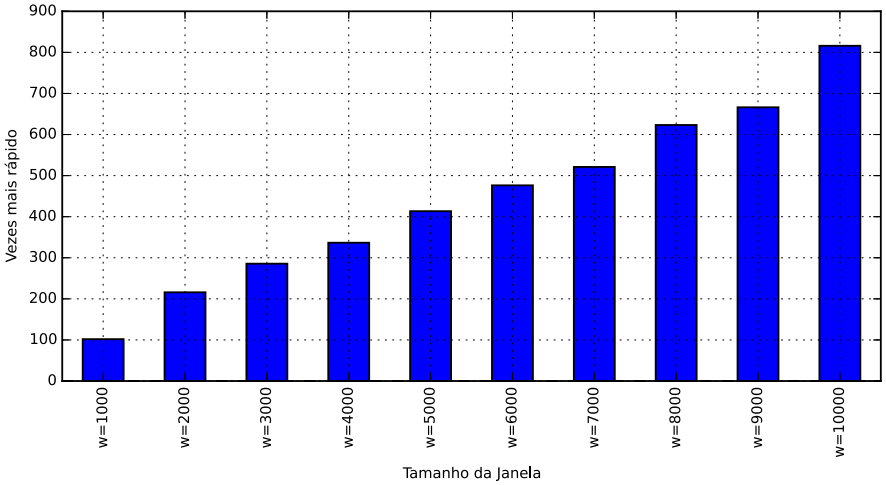

Figura 6.1: Performance do algoritmo MFI-TransSW+ comparado com o original.

6.3 ClickRec: Experimento e Análise dos Resultados

Uma análise dos dados apontou que a grande maioria dos usuários é composta de *bounce users*, ou seja, usuários que acessam apenas uma página do site durante sua navegação. Em uma amostra de todos os cliques durante o período de 15h00 às 15h59 do dia 14 de janeiro de 2015, 61% dos usuários do GShow e 63% dos usuários do Globoesporte acessaram apenas 1 página, conforme figura 6.2.

Isso é um problema para algoritmos de mineração de conjuntos de itens frequentes, já que desta forma temos poucos conjuntos para minerar. Para resolver esse problema nossa abordagem consiste em minerar os metadados das matérias para gerar os conjuntos de itens frequentes. Conforme discutido no capítulo 5, praticamente todas as matérias das sete editorias citadas possuem

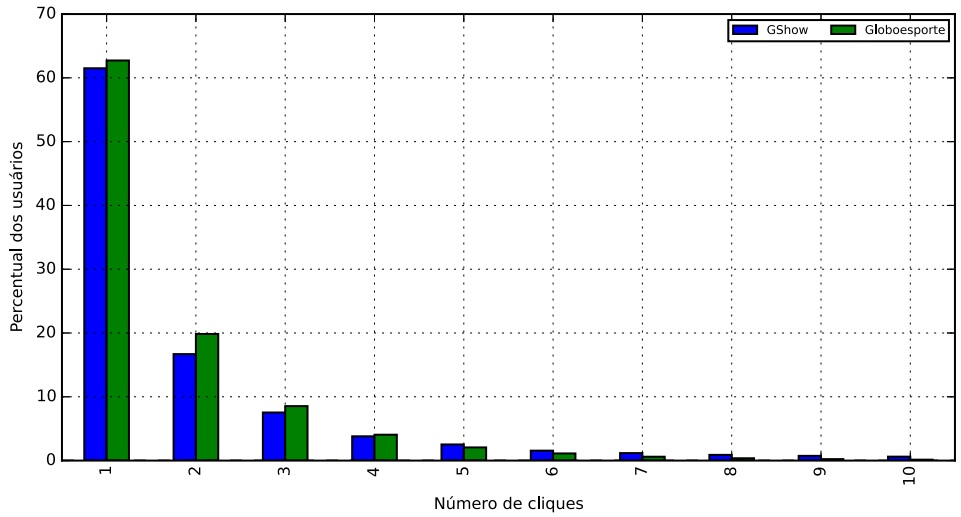

Figura 6.2: Número de cliques por usuário.

pelo menos uma anotação semântica. Assim, estas anotações serão usadas no experimento.

O sistema ClickRec foi desenvolvido como um serviço REST que recebe requisições via uma API HTTP. Entretanto, para nossos testes, rodamos o sistema no modo *batch*, ou seja, lendo arquivos com os cliques capturados anteriormente nos servidores Web. Utilizamos a mesma massa de dados adotada para os testes de performance do algoritmo MFI-TransSW+, ou seja, arquivos CSV extraídos do HBase, cada um com uma hora de sequências de cliques. A massa completa utilizada nos testes é composta de 24 horas de cliques do dia 14 de janeiro de 2015 para o teste da editoria Globoesporte e outras 24 horas de cliques do dia 08 de agosto de 2015 para a editoria GShow.

Para este experimento processamos as sequências de cliques em pares de duas horas seguidas. As sequências capturadas durante a primeira hora são usadas para criar a janela e gerar as recomendações. As sequências capturadas durante a segunda hora são usadas para extrair uma amostra de usuários e medir a eficiência das recomendações. A amostra é composta apenas de usuários que acessaram mais de uma página durante esta segunda hora.

Neste experimento, precisamos dos usuários que acessaram mais de uma página pela seguinte razão: para cada usuário da amostra extraída da segunda hora iremos selecionar a primeira página acessada e passar estes dados para o módulo de recomendação do sistema ClickRec. Este módulo retorna 10 recomendações. Comparamos estas recomendações com as páginas acessadas logo em seguida pelo usuário. Se acertamos uma ou mais recomendações marcamos este usuário como sucesso.

Por exemplo, usamos os cliques executados na hora 00h (que vai de 00h00 até 00h59) para carregar a janela e gerar os conjuntos de itens frequentes. Em seguida, usamos os cliques da hora 01h (que vai de 01h00 até 01h59) para selecionar uma amostra de usuários que clicaram em mais de uma página. Depois de gerar as recomendações e verificar os resultados fazemos o processo de novo, desta vez carregando a janela com a hora 01h e selecionando a amostra de usuários com a hora 02h, e assim sucessivamente até completar as 24 horas.

Em resumo, executamos os seguintes passos para cada par de horas:

- **Carregar janela:** Carregar a janela com primeira hora de sequência de cliques e gerar os conjuntos de itens frequentes. Este passo equivale à execução das etapas 1 e 2 do sistema ClickRec, conforme visto na figura 5.2.
- **Extrair amostra de usuários:** Extrair uma amostra de 10k usuários da segunda hora de cliques.
- **Solicitar recomendações:** Para cada usuário gerar recomendações, com base na primeira página que foi acessada. Este passo equivale à execução da etapa 3 do sistema ClickRec (figura 5.2).
- **Comparar recomendações:** Para cada usuário, verificar se alguma das páginas acessadas pelo usuário depois do primeiro acesso (primeira página acessada) é uma das páginas recomendadas. Caso positivo, marcamos um acerto.

Nas figuras 6.3 e 6.4 vemos os resultados para as editorias Globoesporte e GShow, respectivamente.

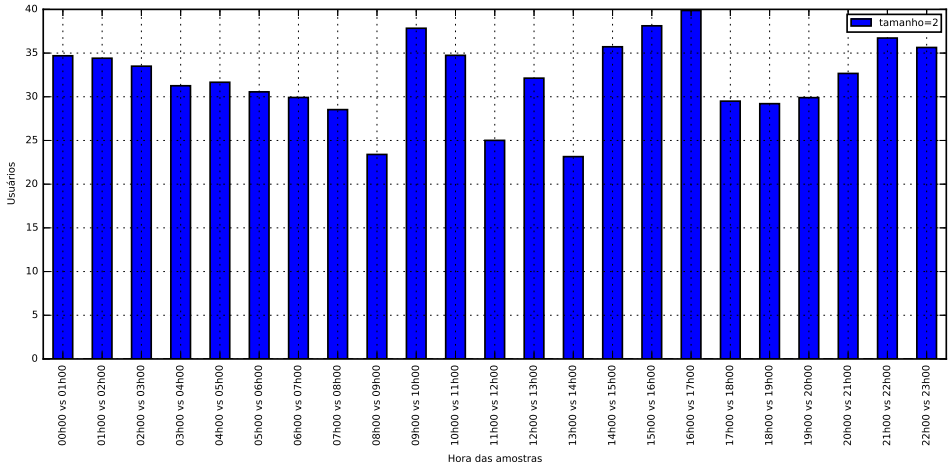

Figura 6.3: ClickRec: Acertos nas recomendações do Globoesporte.

A grande maioria dos Web sites da Globo.com implementa recomendação automática apenas na página principal (*home*) das editorias. Apenas dois sites

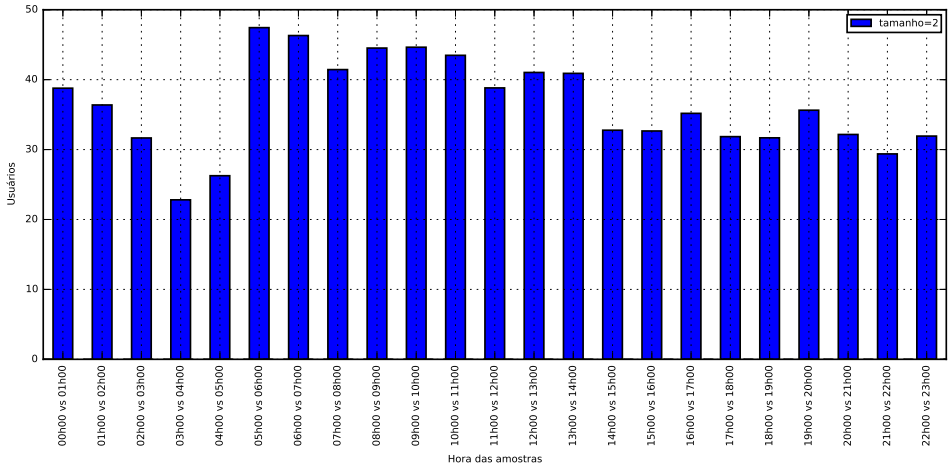

Figura 6.4: ClickRec: Acertos nas recomendações do GShow.

da Globo.com possuem recomendação automática na matéria. Estes tem uma taxa de conversão (quando um usuário clica em uma matéria recomendada) de menos de 10%. O pior resultado do ClickRec está acima de 20%, demonstrando a validade da nossa abordagem.

7
Conclusão e Trabalhos Futuros

Nesta dissertação apresentamos o algoritmo MFI-TransSW+, um algoritmo de mineração de conjuntos de itens frequentes em sequências de dados. Seu desenvolvimento foi baseado no algoritmo MFI-TransSW e suas principais diferenças são processar sequências de cliques (onde não é possível dizer quando uma transação começou ou terminou) e usar uma atualização circular dos vetores de bits.

Também apresentamos os resultados de experimentos com estes algoritmos, realizados com uma massa de dados real extraída do portal Globo.com. Estes experimentos comprovam o ganho significativo de performance da versão proposta em comparação com o algoritmo original: MFI-TransSW+ é de 100 à 900 vezes mais rápido que o algoritmo original para janelas de 1k à 10k transações, e essa diferença de performance continua crescendo de forma linear conforme aumentamos o tamanho da janela de transações.

Além disso, apresentamos o sistema ClickRec, um sistema de recomendação que processa em tempo real as sequências de cliques de um portal de notícias Web e, usando o algoritmo MFI-TransSW+, minera conjuntos de anotações semânticas frequentes anotadas nas matérias para gerar recomendações. Realizamos experimentos com uma massa de dados real extraída do portal Globo.com e, ao comparar nossas recomendações com a sequência de cliques executados pelo usuário, comprovamos que o sistema apresentou uma taxa de acertos pelo menos duas vezes maior que os sistemas usados atualmente pelo portal Globo.com.

Como trabalho futuro esperamos rodar o ClickRec em um ambiente de produção. Assim teremos a oportunidade de apresentar as recomendações para os usuários do portal em um ambiente real, na hora que eles acessam as matérias. Este experimento nos daria um melhor entendimento da eficiência do sistema ClickRec e sua aplicabilidade em portais de notícias.

Referências Bibliográficas

Agrawal, R. e Srikant, R. **Fast Algorithms for Mining Association Rules**. *Proc. 20th int. conf. very large data bases, VLDB*, pages 1–32, 1994.

Babcock, B., Babu, S., Datar, M., Motwani, R. e Widom, J. **Models and issues in data stream systems**. *Proceedings of the twentyfirst ACM SIGMODSIGACTSIGART symposium on Principles of database systems PODS 02*, pages(2002-19):1, 2002.

Chang, J. H. e Lee, W. S. **A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams**. *Presented at Journal of Information Science and Engineering*, 20(4):753–762, 2004.

Ramos, J., Eden, J. e Edu, R. **Using TF-IDF to Determine Word Relevance in Document Queries**. *Processing*, 2003.

Resnick, P., Iacovou, N. e Suchak, M. **GroupLens: an open architecture for collaborative filtering of netnews**. *Proceedings of the 1994 ACM conference on Computer supported cooperative work.*, pp(1):175–186, 1994.

Adomavicius, G. e Tuzhilin, A. **Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions**. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

Agrawal, R., Imieliński, T. e Swami, A. **Mining association rules between sets of items in large databases**. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

Chernoff, H. **A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations**. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

Cheng, J., Ke, Y. e Ng, W. **A survey on algorithms for mining frequent itemsets over data streams**. *Knowledge and Information Systems*, 16(1):1–27, 2008.

Chi, Y., Wang, H., Philip, S. Y. e Muntz, R. R. **Catch the moment: maintaining closed frequent itemsets over a data stream sliding window**. *Knowledge and Information Systems*, 10(3):265–294, 2006.

Gaber, M. M., Zaslavsky, A. e Krishnaswamy, S. **Mining data streams: a review**. *ACM Sigmod Record*, 34(2):18–26, 2005.

Goethals, B. e Zaki, M. J. **Advances in frequent itemset mining implementations: report on fimi'03**. *ACM SIGKDD Explorations Newsletter*, 6(1):109–117, 2004.

Lee, C. H., Lin, C. R. e Chen, M. S. **Sliding window filtering: an efficient method for incremental mining on a time-variant database**. *Information Systems*, 30(3):227–244, 2005.

Leskovec, J., Rajaraman, A. e Ullman, J. D. **Mining of massive datasets**. Cambridge University Press, 2014.

Li, H. F., Lee, S. Y. e Shan, M. K. **An efficient algorithm for mining frequent itemsets over the entire history of data streams**. In *Proc. of First International Workshop on Knowledge Discovery in Data Streams*, 2004.

Li, H. F., Lee, S. Y. e Shan, M. K. **Online mining (recently) maximal frequent itemsets over data streams**. In *Research Issues in Data Engineering: Stream Data Mining and Applications, 2005. RIDE-SDMA 2005. 15th International Workshop on*, pages 11–18. IEEE, 2005.

Li, H. F. e Lee, S. Y. **Mining frequent itemsets over data streams using efficient window sliding techniques**. *Expert Systems with Applications*, 36(2):1466–1477, 2009.

Manku, G. S. e Motwani, R. **Approximate frequency counts over data streams**. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

Montgomery, A. L., Li, S., Srinivasan, K. e Liechty, J. C. **Modeling online browsing and path analysis using clickstream data**. *Marketing Science*, 23(4):579–595, 2004.

Yu, J. X., Chong, Z., Lu, H., Zhang, Z. e Zhou, A. **A false negative approach to mining frequent itemsets from high speed transactional data streams**. *Information Sciences*, 176(14):1986–2015, 2006.