Felipe Carvalho Perestrelo

Sistema Recomendador para Comércio Eletrônico

São Paulo - SP, Brasil

02 de dezembro de 2013

Felipe Carvalho Perestrelo

Sistema Recomendador para Comércio Eletrônico

Monografia desenvolvida para obtenção do grau de Bacharel em Ciência da Computação pelo Instituto de Matemática e Estatística da Universidade de São Paulo.

Supervisora: Kelly Rosa Braghetto

Instituto de Matemática e Estatística
Universidade de São Paulo

São Paulo - SP, Brasil

02 de dezembro de 2013

Sumário

Lista de Figuras

1 Introdução

Sistemas recomendadores "são ferramentas e técnicas de software que sugerem itens úteis a um usuário. As sugestões podem estar relacionadas a diversos processos de tomadas de decisão (nesse contexto, chamadas de *transações*), como quais itens comprar, que música ouvir ou que notícia online assistir"[14]. São desenvolvidos com algoritmos que avaliam dados históricos de transações para gerarem as recomendações estatisticamente mais prováveis de serem aceitas pelos usuários, com o objetivo de aumentar a quantidade de transações.

Normalmente utiliza-se sistemas recomendadores para aumentar, direta ou indiretamente, a receita em sítios de internet. No caso de um comércio eletrônico, um sistema recomendador tem relação direta com o faturamento pois o sucesso da recomendação faz com que o usuário compre um novo produto que, sem essa recomendação, não compraria. Para sítios de conteúdo, como de vídeos, músicas ou notícias, recomendações efetivas mantêm os usuários no sítio por mais tempo, aumentando as visualizações de páginas e, consequentemente, aumentando o número de exibições de peças publicitárias e, indiretamente, a receita gerada. Além disso, recomendações bem sucedidas podem também aumentar a fidelidade do usuário, estimulando-o a retornar ao sítio, também aumentando a quantidade de visualizações de páginas.

Um sistema recomendador utiliza grandes quantidades de dados e seus algoritmos são desenvolvidos com técnicas de mineração de dados. Berry e Liniff[6] definiram mineração de dados como "a exploração e a análise, por meio automático ou semiautomático, de grandes quantidades de dados, a fim de descobrir padrões e regras significativos". Para isso lança-se mão de técnicas de mineração de dados, que são: descrição, classificação, estimação (ou regressão), predição, agrupamento e associação. Esta última é a principal técnica utilizada em sistemas recomendadores, e será devidamente explicada neste documento.

A proposta deste trabalho é desenvolver um sistema recomendador eficiente e facil-

mente adaptável para aplicação em sítios de comércio eletrônico. A viabilidade prática deste trabalho foi validada por meio de um estudo de caso de um comércio eletrônico real focado no segmento de produtos infantis. A transação no contexto desse trabalho é, portanto, a venda de produtos.

Foram estudados conceitos de mineração de dados com aprofundamento na tarefa de associação, principal tarefa utilizada em sistemas recomendadores. Essa técnica identifica a relação entre diferentes atributos pela avaliação do histórico de transações. Por exemplo, a partir dos itens incluídos no carrinho de compras de um usuário ela identifica a probabilidade um determinado item ser comprado em conjunto com os outros. O sistema se utiliza dessa tarefa para recomendar os produtos com maiores probabilidades de serem comprados juntamente com os produtos que já foram escolhidos pelo usuário.

Existem diversos algoritmos que realizam tal tarefa, com diferentes entradas, estruturas de dados, objetivos, etc. Alguns exemplos são: *Apriori*[1] [5], *FP-Growth* [11], *Eclat* [16]. Para este trabalho foi escolhido o algoritmo Apriori como foco de estudo. Foi um dos primeiros algoritmos desenvolvidos para tal tarefa e até hoje é utilizado por sua eficiência e simplicidade de implementação.

Foram implementadas duas variações do algoritmo Apriori que tiveram seu desempenho avaliado e comparado com outras duas implementações já existentes. Uma das implementações propostas apresentou desempenho satisfatório e mostrou-se viável de ser usada em um comércio eletrônico real.

[1] Apesar do termo em Latim ter grafia *a priori*, o nome do algoritmo é escrito com os termos juntos (*Apriori*).

2 *Conceitos e Tecnologias Estudados*

Nesta seção serão apresentados os conceitos e tecnologias estudados. Falaremos mais profundamente sobre sistemas recomendadores, mineração de dados e suas tarefas, e o algoritmo de associação que utilizamos na implementação do sistema: Apriori.

2.1 Sistemas Recomendadores

Sistemas recomendadores são "ferramentas e técnicas de software que sugerem itens úteis a um usuário. As sugestões podem estar relacionadas a diversos processos de tomadas de decisão, como quais itens comprar, que música ouvir ou que notícia ler"[14].

Deste modo, a expectativa de sucesso de um sistema recomendador dependerá do modelo de mercado no qual ele está inserido. Em um comércio eletrônico, por exemplo, espera-se que ele gere sugestões relevantes de produtos para o usuário, aumentando a receita. Em um portal de conteúdo (vídeos, músicas, notícias, etc), espera-se que ele gere sugestões relevantes de conteúdo aumentando o tempo médio de interação por usuário.

2.2 Mineração de Dados

Mineração de dados é um processo computacional que visa descobrir padrões em grandes bases de dados. Foi definida por Berry e Liniff como "a exploração e a análise, por meio automático ou semiautomático, de grandes quantidades de dados, a fim de descobrir padrões e regras significativos"[6]. Envolve técnicas multidisciplinares, que utilizam conceitos de inteligência artificial, estatística, aprendizagem de máquinas, sistemas de banco de dados, etc, com o objetivo de realizar determinadas tarefas.

A mineração de dados é comumente classificada por sua capacidade de realizar determinadas tarefas [12]. As tarefas mais comuns, conforme descrito em [9], são:

- *Descrição:* É a tarefa utilizada para descrever os padrões e tendências revelados pelos dados. A descrição geralmente oferece uma possível interpretação para os resultados obtidos. A tarefa de descrição é muito utilizada em conjunto com as técnicas de análise exploratória de dados para comprovar a influência de certas variáveis no resultado obtido.

- *Classificação:* Uma das tarefas mais comuns, a Classificação, visa identificar a qual classe um determinado registro pertence. Nesta tarefa, o modelo analisa um conjunto de registros fornecidos como treinamento, com cada registro já contendo a indicação da classe a que ele pertence, a fim de aprender como classificar um novo registro. Por exemplo, categorizamos cada registro de um conjunto de dados contendo as informações sobre os colaboradores de uma empresa: Perfil Técnico, Perfil Negocial e Perfil Gerencial. O modelo analisa os registros e então é capaz de dizer em qual categoria um novo colaborador se encaixa.

  A tarefa de classificação pode ser usada, por exemplo, para:

  - Identificar, em uma escola, qual a turma mais indicada para um determinado aluno.
  - Determinar quando uma transação de cartão de crédito pode ser uma fraude.
  - Diagnosticar onde uma determinada doença pode estar presente.

- *Estimação ou Regressão:* A estimação é similar à classificação, porém é usada quando o registro é identificado por um valor numérico e não um categórico. Assim, pode-se estimar o valor de uma determinada variável analisando-se os valores das demais. Por exemplo, um conjunto de registros contendo os valores mensais gastos por diversos tipos de consumidores e de acordo com os hábitos de cada um. Após ter analisado os dados, o modelo é capaz de dizer qual será o valor gasto por um novo consumidor. A tarefa de estimação pode ser usada por exemplo para:

  - Estimar a quantia a ser gasta por uma família de quatro pessoas durante a volta às aulas.
  - Estimar a pressão ideal de um paciente baseando-se na idade, sexo e massa corporal.

- *Predição:* A tarefa de predição é similar às tarefas de classificação e estimação, porém ela visa descobrir o valor futuro de um determinado atributo. Exemplos:

  - Predizer o valor de uma ação três meses adiante.

  - Predizer o percentual que será aumentado de tráfego na rede se a velocidade aumentar.
  - Predizer o vencedor do campeonato baseando-se na comparação das estatísticas dos times.

  Alguns métodos de classificação e regressão podem ser usados para predição.

- *Agrupamento:* A tarefa de agrupamento visa identificar e aproximar os registros similares. Um agrupamento é uma coleção de registros similares entre si e diferentes dos outros registros nos demais agrupamentos. Essa tarefa difere da classificação pois não necessita que os registros sejam previamente categorizados (aprendizado não-supervisionado). Além disso, ela não tem a pretensão de classificar, estimar ou predizer o valor de uma variável, ela apenas identifica os grupos de dados similares, conforme mostra a figura 1, extraída de [10]. Exemplos:
  - Segmentação de mercado para um nicho de produtos.
  - Auditoria, separando comportamentos suspeitos.
  - Redução, para um conjunto de atributos similares, de registros com centenas de atributos.

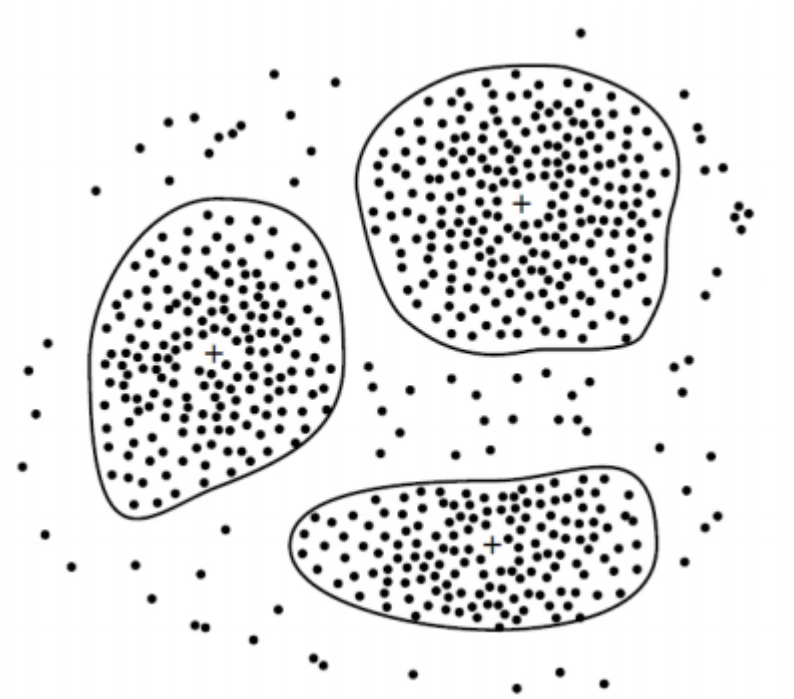

Figura 1: Exemplo de registros agrupados em 3 agrupamentos.

As aplicações das tarefas de agrupamento são as mais variadas possíveis: pesquisa de mercado, reconhecimento de padrões, processamento de imagens, análise de dados, taxonomia de plantas e animais, pesquisas geográficas, classificação de documentos da Web, entre outras [13]. Geralmente a tarefa de agrupamento é combinada com outras tarefas, além de serem usadas na fase de preparação dos dados.

- *Associação:* A tarefa de associação consiste em identificar quais atributos estão relacionados. A partir dela, extraem-se regras da forma: SE atributo X ENTÃO atributo Y. Ela é utilizada em análises de "Cestas de Compras", onde identificamos quais produtos são normalmente comprados juntos pelos consumidores. Alguns exemplos:
  - Determinar os casos onde um novo medicamento pode apresentar efeitos colaterais.
  - Identificar os usuários de planos que respondem bem a oferta de novos serviços.

A última tarefa descrita, de associação, é a principal tarefa utilizada em sistemas recomendadores de comércio eletrônico, conforme desenvolvido neste trabalho, portanto ela foi alvo do estudo para a implementação.

2.3 Tarefa de Associação com Mineração de Itens Frequentes

Para realizarmos a tarefa de associação, lançamos mão da técnica de *mineração de itens frequentes*, introduzida por Agrawal, Imielinski e Swami [4]. Tal técnica avalia o histórico de transações do objeto de estudo e tenta prever a ocorrência de um conjunto de itens baseado na ocorrência de um conjunto de outros itens em uma mesma transação. No caso de um varejo, por exemplo, ela avalia o histórico de vendas e tenta prever a probabilidade de um conjunto de itens ser adquirido juntamente com um conjunto de outros itens em uma mesma transação. Ela pode ser dividida em duas etapas: geração de conjuntos de itens frequentes e geração de regras de associação.

Para a primeira etapa da técnica de mineração de itens frequentes, definimos um *conjunto de itens de nível k* como um conjunto que contém k itens. Para cada conjunto de itens, definimos sua *frequência* como a quantidade de transações que o contém, e seu *suporte* como a razão entre as transações que o contém e o total de transações avaliadas.

Como medida de viabilidade determinamos um suporte mínimo (ou uma frequência mínima) aceitável. Um conjunto de itens frequente apresenta um suporte maior ou igual ao suporte mínimo definido. A saída da primeira etapa da técnica de mineração de itens frequentes é o conjunto de todos os conjuntos de itens frequentes.

Para a segunda etapa de mineração de itens frequentes, definimos uma *regra de associação* como uma expressão da forma X *implica* Y. Nesse caso, o suporte da regra é igual ao suporte do conjunto $X \cup Y$. Definimos, então, a *confiança* como a razão entre a frequência do conjunto $X \cup Y$ e o conjunto X, em outras palavras, a frequência com a qual Y aparece nas transações que contêm X. Assim, seja *conf(A => B)* a confiança da regra $A => B$, e *freq(A)* a frequência do conjunto de itens A, então *conf(X => Y) = freq(X ∪ Y) / freq(X)*. A saída da segunda etapa da técnica de mineração de itens frequentes é as regras com confiança mínima geradas a partir dos conjuntos de itens frequentes, obtidos na primeira etapa.

Existem diversos algoritmos que executam a tarefa de associação utilizando a técnica de mineração de itens frequentes, como: *Apriori* [5], *FP-Growth* [11], *Eclat* [16], etc. Por causa de seu uso mais comum - encontrar associações entre cestas de produtos de um varejo - eles comumente também são chamados de algoritmos de cesta de compras.

O algoritmo escolhido para a implementação do sistema proposto neste trabalho foi o Apriori. Foi um dos primeiros algoritmos desenvolvidos para tal tarefa e até hoje é utilizado por sua eficiência e simplicidade de implementação.

2.3.1 O Algoritmo Apriori

O algoritmo Apriori foi desenvolvido por Rakesh Agrawal e Ramakrishnan Srikant e divulgado em 1994 [5][1]. Conforme descrito anteriormente, realiza a tarefa de associação utilizando a técnica de mineração de itens frequentes, portanto é composto pelas duas etapas já descritas: geração dos conjuntos de itens frequentes e geração das regras de associação.

A primeira etapa do algoritmo recebe dados de transações como entrada e retorna uma estrutura com todos os conjuntos de itens frequentes das transações. A segunda recebe essa estrutura e retorna todas as regras que satisfazem a confiança mínima.

[1] O artigo completo pode ser encontrado em: `http://rakesh.agrawal-family.com/papers/vldb94apriori.pdf`

2.3.1.1 Etapa 1 - Geração dos Conjuntos de Itens Frequentes

A figura 2 apresenta a etapa 1 do algoritmo Apriori [5], a geração de conjuntos de itens frequentes. Chamaremos de L_k o conjunto dos conjuntos de itens frequentes de tamanho k, e de C_k o conjunto dos candidatos a L_k.

1) L_1 = {large 1-itemsets};
2) **for** ($k = 2$; $L_{k-1} \neq \emptyset$; k++) **do begin**
3) C_k = apriori-gen(L_{k-1}); // New candidates
4) **forall** transactions $t \in \mathcal{D}$ **do begin**
5) C_t = subset(C_k, t); // Candidates contained in t
6) **forall** candidates $c \in C_t$ **do**
7) c.count++;
8) **end**
9) $L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$
10) **end**
11) Answer = $\bigcup_k L_k$;

Figura 2: Algoritmo Apriori [5].

Na primeira passagem (linha 1), conta-se a ocorrência de cada um dos itens únicos para determinar sua frequência. Gera-se, então, o *large 1-itemsets*, ou L_1.

No passo seguinte inicia-se um laço para geração dos conjuntos de itens frequentes L_k de tamanhos maiores que 1 (linha 2). Dentro do laço cada execução é dividida em duas fases diferentes. Na primeira fase gera-se os candidatos a conjuntos de itens frequentes de nível k, ou C_K, com a função *apriori-gen()*. Note que é impossível gerar um conjunto frequente a partir da união de conjuntos onde pelo menos um desses conjuntos não seja frequente, portanto, na geração de C_k, são combinados, um a um, os conjuntos pertencentes a L_{k-1}.

Na segunda fase são eliminados os conjuntos de C_k que tenham subconjuntos de nível $k-1$ não frequentes. Exemplo: Seja L_3 $((1,2,3),(1,2,4),(1,3,4),(1,3,5)(2,3,4))$. Após a combinação, teremos C_4 igual a $((1,2,3,4),(1,3,4,5))$. O passo da eliminação corta o candidato $(1,3,4,5)$ pois seu subconjunto $(1,4,5)$ não é frequente. Note que esse passo não é necessário quando k = 2, já que o conjuntos dos subconjuntos de C_2 é exatamente L_1.

Após a geração de C_k, inicia-se um novo laço. Para cada transação a função *subset()* recebe o conjunto C_k e a transação em questão, e retorna o conjunto de candidatos de C_k

que são subconjuntos da transação, chamado de C_t. Todos os candidatos presentes em C_t têm sua frequência incrementada em 1. Ao final desse laço teremos a frequência de cada candidato pertencente a C_k e eliminamos os que não possuem suporte mínimo, obtendo L_k. Ao final do laço inicial, da linha 2 do algoritmo, teremos uma estrutura com todos os conjuntos de itens frequentes.

Esta etapa é crucial para o bom desempenho do algoritmo. Como temos que comparar todos os candidatos com todas as transações, para determinar a frequência de cada um, se não desenvolvermos uma implementação eficiente, o custo computacional deste passo pode ser extremamente elevado. Por esse motivo, a etapa de geração de itens frequentes é foco de estudo para melhorias no desempenho pensando tanto em tempo de execução quanto em consumo de memória.

Na implementação divulgada em [5], o conjunto de candidatos C_k é armazenado em uma *hash tree* diminuindo, assim, o tempo de acesso a cada candidato.

Abaixo ilustramos um exemplo de execução da geração de conjuntos de itens frequentes com suporte mínimo de 30%.

A figura 3 exibe um exemplo de tabela de banco de dados com o id de transação e os itens presentes em cada uma das transações.

ID Transações	Itens
ID001	{Pão,Leite,Ovos}
ID002	{Leite,Manteiga}
ID003	{Leite,Cerveja}
ID004	{Pão,Leite,Manteiga}
ID005	{Pão,Cerveja}
ID006	{Leite,Cerveja}
ID007	{Pão,Cerveja,Manteiga}
ID008	{Pão,Leite,Cerveja,Ovos}
ID009	{Pão,Leite,Cerveja}
ID0010	{Pão,Leite,Cerveja,Manteiga}

Figura 3: Exemplo de tabela com transações e itens armazenados.

A partir dos dados geramos a frequência de cada um dos n itens presentes nas transações e eliminamos aqueles que não cumprem com o suporte mínimo - no caso apenas $\{Ovos\}$ -, gerando o conjunto de itens frequentes de nível 1, ou L_1.

Item	N de transações
Pão	7
Leite	8
Cerveja	7
Manteiga	4
~~Ovos~~	~~2~~

Figura 4: L_1 - Conjuntos de itens frequentes de nível 1.

A partir de L_1, geramos C_2, ou o conjunto de candidatos a conjunto de itens frequentes de nível 2. Contamos a frequência de cada candidato e eliminamos o conjunto *{Cerveja,Manteiga}*, que não satisfez o suporte mínimo, gerando L_2.

Conjuntos	N de transações
{Pão,Leite}	5
{Pão,Cerveja}	5
{Pão,Manteiga}	3
{Leite,Cerveja}	5
{Leite,Manteiga}	3
~~{Cerveja,Manteiga}~~	~~2~~

Figura 5: L_2 - Conjuntos de itens frequentes de nível 2.

Com L_2, geramos C_3. Antes de contar as frequências de cada candidato, verificamos se existe algum conjunto que possui um subconjunto não frequente. É o caso do conjunto *{Leite,Cerveja,Manteiga}* que contém o subconjunto não frequente *{Cerveja,Manteiga}*. Assim, nós o eliminamos no passo anterior ao da contagem da frequência. Então contamos a frequência dos conjuntos restantes e eliminamos, também, *{Pão,Leite,Manteiga}*, que não satisfaz o suporte mínimo, e obtemos L_3.

Conjuntos	N de transações
{Pão,Leite,Cerveja}	3
~~{Pão,Leite,Manteiga}~~	~~2~~
~~{Leite,Cerveja,Manteiga}~~	

Figura 6: L_3 - Conjuntos de itens frequentes de nível 3.

Como L_3 possui apenas um elemento, C_4 será vazio e o laço será interrompido. Assim, a geração dos conjuntos de itens frequentes se encerra com o conjunto mostrado na figura 7:

Conjuntos	N de transações
{Pão}	7
{Leite}	8
{Cerveja}	7
{Manteiga}	4
{Pão,Leite}	5
{Pão,Cerveja}	5
{Pão,Manteiga}	3
{Leite,Cerveja}	5
{Leite,Manteiga}	3
{Pão,Leite,Cerveja}	3

Figura 7: L - Todos os conjuntos de itens frequentes.

2.3.1.2 Etapa 2 - Geração das Regras de Associação

A segunda etapa usa a estrutura gerada com todos os conjuntos de itens frequentes e calcula as regras de associação, retornando aquelas que satisfazem a confiança mínima.

Como exemplo, vamos gerar algumas regras a partir do conjunto de itens frequentes gerado na etapa anterior, e considerando confiança mínima de 50%. Lembrando que, seja *conf(A => B)* a confiança da regra $A => B$, e *freq(A)* a frequência do conjunto de itens A, então conf(X => Y) = freq(X U Y) / freq(X).

Exemplos de regras:

1. *{Pão} => {Manteiga} = 3/7 = 43%*
2. *{Pão,Leite} => {Cerveja} = 3/5 = 60%*
3. *{Leite} => {Pão,Cerveja} = 3/8 = 38%*
4. *{Pão} => {Cerveja} = 5/7 = 71%*
5. *{Manteiga} => {Leite} = 3/4 = 75%*
6. *{Leite} => {Manteiga} = 3/8 = 38%*

No exemplo acima apenas as regras *2, 4 e 5* satisfazem a confiança mínima e serão consideradas.

3 Desenvolvimento do Sistema

O objetivo do trabalho é implementar um sistema recomendador para comércio eletrônico, baseado no algoritmo Apriori, que possa ser facilmente adaptado para ser utilizado nesses tipos de sítios. Para tal foi escolhida a linguagem PHP, que é bastante utilizada em desenvolvimento *web*. A versão da PHP utilizada foi a 5.4.17.

Para o desenvolvimento do sistema, foi suposto que as transações, utilizadas como entrada para a geração das regras de associação, estão armazenadas em um banco de dados relacional, e que o sistema tem acesso a este banco para realizar as consultas e obter os dados.

Foram desenvolvidas duas implementações que se diferem, principalmente, na primeira etapa do Apriori, ou seja, na obtenção do conjunto de itens frequentes. Uma das versões, a qual chamaremos de *aprioriPHP*, se beneficia das estruturas de dados otimizadas da PHP e utiliza vetores de pares do tipo *(chave, valor)* de PHP para ganho de eficiência nessa tarefa. A outra, identificada como *aprioriSQL*, se beneficia da eficiência da execução de consultas em SQL para geração dos conjuntos de itens frequentes.

3.1 Entrada do Sistema

Ambas as versões recebem como parâmetros de entrada os valores de suporte mínimo e confiança mínima das regras a serem geradas, e os dados referentes às transações, que são obtidos a partir de consultas em SQL ao banco de dados.

A versão *aprioriPHP* consulta o banco de dados apenas no início da execução e recebe dados no formato *(id_transacao,id_item)*. A versão *aprioriSQL*, por sua vez, consulta o banco de dados diversas vezes para obter os conjuntos de itens frequentes, como será explicado na seção 3.1.2.1.

3.1.1 O Banco de Dados

Para o desenvolvimento do trabalho, foi criado um banco de dados relacional de acordo com os padrões mais comuns de um banco de dados de comércio eletrônico, considerando as tabelas necessárias para um sistema recomendador de produtos, que são a tabela de transações, a tabela que armazena os produtos, e a tabela de detalhamento dos pedidos, que é gerada a partir do relacionamento das duas anteriores. Deste modo, o sistema implementado se aplica a qualquer comércio eletrônico que tenha um banco de dados com tais características.

A figura 8 exibe um exemplo de relacionamento entre as tabelas de transações e itens que resulta na tabela *transacoes_itens*. Note que o que é necessário para um sistema recomendador de dados são apenas as colunas *id_transacao* e *id_item* contidas na tabela *transacoes_itens*. Os demais campos apresentados no exemplo são para efeito ilustrativo de um banco de dados padrão de um comércio eletrônico, pois não são utilizados para a tarefa de associação.

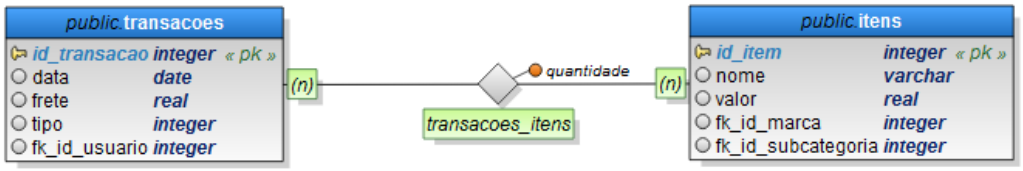

Figura 8: Exemplo de tabelas padrões de um comércio eletrônico necessárias para um sistema recomendador.

3.2 Implementação do Algoritmo Apriori

Nesta seção explicaremos como foi implementado o algoritmo Apriori no sistema, diferenciando a etapa da geração dos conjutos de itens frequentes em cada uma das versões.

3.2.1 Etapa 1 - Geração dos Conjuntos de Itens Frequentes

As implementações desenvolvidas - *aprioriPHP* e *aprioriSQL* - diferem entre si na etapa 1, da geração dos conjuntos de itens frequentes. Detalharemos aqui as duas versões dessa etapa.

3.2.1.1 Geração dos Conjuntos de Itens Frequentes na Versão *aprioriPHP*

A versão aprioriPHP inicia a execução com uma consulta ao banco de dados para a obtenção de um conjunto de pares na forma (id_transacao,id_item), com um par por linha da resposta. Ela transforma a entrada para um formato em que se tem uma transação por linha com cada item único separado por vírgula (figura 9).

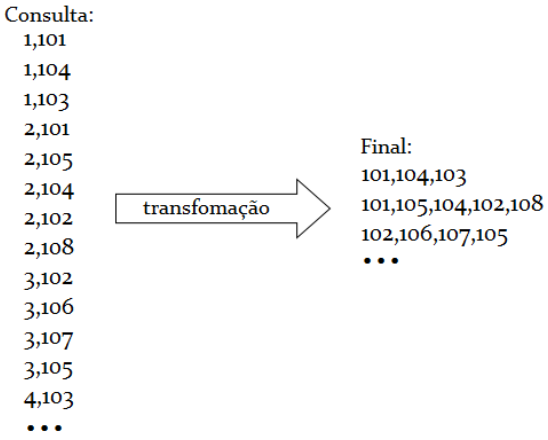

Figura 9: Exemplo do resultado e a sua transformação para o formato manipulado pelo Apriori.

A função *resolveApriori* recebe a entrada transformada e devolve a matriz com os conjuntos de itens frequentes.

A versão aprioriPHP mantém a estrutura do algoritmo original conforme apresentado em [5].

```php
function resolveApriori($transacoes){
    $L = array();
    $L[1] = large1itemsets($transacoes);
    for($k = 2; !empty($L[$k - 1]); $k++){
        $Ck = aprioriGen($L[$k - 1]);
        foreach($transacoes as $t){
            $Ct = subset($Ck, $t, $k);
            foreach($Ct as $candidato){
                $L[$k][$candidato]++;
            }
        }
        $L[$k] = filtraCandidatos($L[$k]);
    }
    return $L;
}
```

Figura 10: Versão reduzida da função geradora dos conjuntos de itens frequentes da aprioriPHP, contendo as principais funções.

Inicialmente, o algoritmo executa a função *large1Itemsets*, que gera L_1. Ela recebe o vetor de transações, percorre todos os itens de todas as transações e conta a ocorrência de cada um, devolvendo um array com os itens que satisfazem a frequência mínima em pares (chave => valor) no formato ([id_item] => frequência). A frequência mínima é passada a ela como variável global

```
[1] => 22
[2] => 23
[5] => 21
[8] => 36
[11] => 60
[12] => 206
[13] => 88
```

Figura 11: Exemplo de saída da função large1Itemsets.

Com o conjunto de itens frequentes de nível 1, o algoritmo entra no laço em que gera os conjuntos de itens frequentes de todos os níveis k para $k > 1$. O laço é executado até que o conjunto de itens frequentes de um determinado nível k seja vazio.

A função *aprioriGen* recebe o conjunto de itens frequentes de nível $k - 1$ e retorna os conjuntos de itens de nível k candidatos a satisfazer o suporte mínimo.

A próxima etapa é a comparação dos conjuntos de itens candidatos gerados por *aprioriGen* com cada uma das transações para determinar a frequência de cada conjunto de itens. Essa tarefa é realizada pela chamada da função *subset* para cada transação. Ela

recebe o vetor contendo todos os conjuntos de itens candidatos, a transação e o nível k, e devolve os conjuntos de itens candidatos contidos no conjunto de combinações de tamanho k da transação. Cada candidato tem seu valor correspondente a frequência incrementado em 1.

A última etapa do laço chama a função *filtraCandidatos*, que recebe o vetor de candidatos e retorna os conjuntos que satisfazem o suporte mínimo, ou seja, os conjuntos de itens frequentes. Ao final da execução do laço teremos uma matriz com todos os conjuntos de itens frequentes de cada nível k armazenada em L.

3.2.1.2 Geração dos Conjuntos de Itens Frequentes na Versão aprioriSQL

A versão *aprioriSQL* não segue a primeira parte do Apriori conforme [5]. A geração dos conjuntos de itens frequentes é realizada com consultas ao banco de dados, as quais retornam, para cada nível k, uma tabela com $k+1$ colunas, sendo uma coluna para cada um dos k itens do conjunto frequente, e uma coluna com a frequência do conjunto.

item	frequencia
1	31
2	30
5	28
6	21
8	51
11	71
12	220
13	100
14	270
15	158
16	47

item1	item2	frequencia
12	14	49
12	15	31
12	22	32
12	23	47
12	27	27
12	52	38
12	62	30
12	63	54
12	64	28
12	71	46
12	73	29

item1	item2	item3	frequencia
12	14	63	17
12	14	104	18
12	22	23	17
12	23	63	18
12	63	71	17
14	22	23	20
14	23	93	17
14	71	105	18
14	93	104	16
15	22	23	21
20	22	23	18

Figura 12: Exemplos do resultado das consultas para $k = 1$, $k = 2$ e $k = 3$.

O algoritmo de geração dos conjuntos de itens frequentes - função *resolveAprioriSQL* - fica bastante reduzido em comparação com o da versão aprioriPHP.

```
function resolveAprioriSQL(){
    $L = array();
    $L[1] = large1itemsets();
    for($k = 2; !empty($L[$k - 1]); $k++){
        $L[$k] = freqItemsetGen($L[$k - 1], $k);
    }
    return $L;
}
```

Figura 13: Versão reduzida da função geradora dos conjuntos de itens frequentes da aprioriSQL, contendo as principais funções.

A função *large1itemsets*, assim como na versão aprioriPHP, gera o conjunto de itens frequentes de nível 1, ou L_1. Para tal, ela executa uma consulta em SQL no banco de dados onde contabiliza a ocorrência de cada item e devolve apenas os itens que tenham ocorrência maior ou igual a da frequência mínima.

A figura 14 exibe um exemplo de uma consulta que gera L_1 com frequência mínima de 10 ocorrências. Para efeitos ilustrativos o nome da tabela com os dados é *pedido* e o nome da coluna de itens é *item*.

```
SELECT ped.item, count(*) as frequencia
  FROM pedido ped
  GROUP BY ped.item
  HAVING count(*) >= 10
  ORDER BY ped.item
```

Figura 14: Exemplo de consulta que gera L_1.

Após a obtenção de L_1 inicia-se um laço no qual são gerados os conjuntos de itens frequentes para todo nível $k > 2$. A função *freqItemsetGen* recebe L_{k-1} e o nível atual k, e devolve um vetor com os conjuntos de itens frequentes do nível k em pares *(chave,valor)* no formato *(conjunto => frequência)*. A função é composta por 3 etapas principais: a construção do texto da consulta, realizada pela função *geraQuery*; a consulta, realizada pela função *rodaQuey*; o tratamento dos dados e armazenamento no vetor no formato descrito acima.

```
function freqItemsetGen($freqItemsetAnterior, $k){
    $query = geraQuery($freqItemsetAnterior, $k);
    $resultado = executaQuery($query);
    $itemsets = array();
    foreach($resultado as $r){
        $prods = array();
        for($i = 1; $i <= $k; $i++)
            $prods[] = $r['id_produto_produto'. $i];
        $prods = implode(',', $prods);
        $itemsets[$prods] = $r['total'];
    }
    return $itemsets;
}
```

Figura 15: Versão reduzida da função *freqItemsetGen*, contendo as principais funções.

Para cada nível $k > 1$, a função *geraQuery* é chamada para construir o texto da consulta. A consulta gerada primeiro obtém uma listagem dos conjuntos de tamanho k que aparecem em cada transação armazenada no banco de dados e que contém ao menos um conjunto frequente de tamanho $k - 1$, identificado no nível anterior, e depois conta a

frequência de ocorrência de cada conjunto de itens de tamanho k obtido, e devolve como resposta somente os conjuntos que possuem uma frequência maior que a mínima definida.

A figura 16 mostra um exemplo de consulta gerada pela função *geraQuey* para o conjunto de itens frequentes {1, 2, 5, 7, 8, 10}, para $k = 3$ e frequência mínima = 10. Para efeitos ilustrativos o nome da tabela com os dados é *pedido* e o nome da coluna de itens é *item* e da coluna de ids de transação é *transacao*.

```
SELECT  ped1.item AS item1, ped2.item AS item2, ped3.item AS item3, COUNT(*) as frequencia
 FROM  pedido ped1 ,  pedido ped2 ,  pedido ped3
 WHERE  ped1.item IN (1, 2, 5, 7, 8, 10)
 AND  ped1.transacao = ped2.transacao
 AND  ped2.item IN (1, 2, 5, 7, 8, 10)
 AND  ped2.item > ped1.item
 AND  ped1.transacao = ped3.transacao
 AND  ped3.item IN (1, 2, 5, 7, 8, 10)
 AND  ped3.item > ped2.item
 GROUP BY  ped1.item, ped2.item, ped3.item HAVING count(*) >= 10
 ORDER BY  ped1.item, ped2.item, ped3.item
```

Figura 16: Exemplo de consulta que gera L_3.

Note que temos 4 condições que garantem o funcionamento do processo:

- $ped_{i-1}.transacao = ped_i.transacao\ para\ todo\ i > 1$: Condição que garante que estão sendo considerados itens de uma mesma transação.
- ped_i *IN ()*: Com essa condição, são selecionados apenas os itens únicos que fazem parte dos conjuntos de itens frequentes do nível anterior.
- $ped_{i-1}.item > ped_i.item\ para\ todo\ i > 1$: Essa condição garante a ordenação dos itens entre si, em cada linha, evitando a geração de linhas com mesmos itens.
- *HAVING count(*)* > 10: Condição de frequência mínima.

Após o término do laço, assim como na versão *aprioriPHP*, a função *resolveAprioriSQL* retorna a matriz L, que contém os conjuntos de itens frequentes de todos os níveis.

3.2.2 Etapa 2 - Geração das Regras

A função de geração das regras, *ruleGen*, é a mesma para ambas as implementações. Ela recebe a matriz L, com os conjuntos de itens frequentes de todos os níveis (que é a saída da primeira etapa das duas implementações) e devolve um vetor com as regras

que satisfazem a confiança mínima. O vetor devolvido utiliza os pares (chave, valor) no formato ([conjunto de itens => conjunto de itens] => confiança).

```
[41=>80] => 0.51219512195122
[79=>80] => 0.51546391752577
[14,22=>23] => 0.57142857142857
[22,122=>23] => 0.61904761904762
```

Figura 17: Exemplo de regras geradas pela função *ruleGen*.

4 *Avaliação de Desempenho dos Algoritmos Apriori Implementados*

Para avaliação do desempenho, as duas implementações feitas nesse trabalho foram submetidas a testes em conjunto com outras duas implementações já existentes. As implementações já existentes escolhidas para comparação de desempenho foram:

- *Weka:* Coletânea de algoritmos para mineração de dados desenvolvida em Java, e regida pela licença GNU[3]. Essa implementação foi escolhida por ser um arcabouço renomado e de fácil utilização.
- *Algoritmo de Ferenc Bodon:* Implementação acadêmica, em código aberto, desenvolvida em C++[7]. Essa implementação foi escolhida principalmente por sua eficiência em tempo de execução, sendo um bom parâmetro de comparação para as implementações propostas neste trabalho. Além disso tem a vantagem de ter sido desenvolvida em código aberto e ter extensa documentação, facilitando o estudo.

4.1 Metodologia dos Testes

Para os testes foi utilizado um banco de dados, com cerca de 80 mil transações e 16 mil itens, de um comércio eletrônico real do segmento de produtos infantis. Desse banco foram extraídos subconjuntos de transações de tamanhos: mil, 10 mil, 20 mil, 40 mil e 80 mil. Para cada tamanho, foram extraídas 5 versões aleatórias diferentes, exceto pela base de 80 mil, pois essa representa o banco completo, gerando um total de 21 versões de conjunto de transações diferentes.

O banco de dados foi mantido no sistema gerenciador de banco de dados PostgreSQL versão 9.2 instalado em um computador com processador Intel Core i5-3210M, 2,5GHz,

6,00GB de RAM, com sistema operacional Windows 8 Professional 64 bits. Durante os testes nenhum aplicativo além dos necessários ao sistema estava sendo executado.

Os testes realizados consistem na medição do tempo de execução de cada versão de implementação do Apriori sobre os conjuntos de transações gerados. Para cada versão dos conjuntos de transações foram realizados testes considerando como suporte mínimo os valores 1,0%, 0,5% e 0,1%, exceto pelas versões da base de mil transações, que não foi testada com o suporte mínimo de 0,1%, pois isso representaria frequência mínima de 1 e todos os itens seriam frequentes. Todos os testes foram executados 5 vezes para eliminar vieses estatísticos e foram realizados com confiança mínima de 50%. A confiança não foi alternada pois ela tem impacto apenas na etapa de geração das regras, que não tem tempo de execução considerável se comparado com a etapa de geração dos conjuntos de itens frequentes.

A execução dos testes foi feita em linha de comando, pelo prompt de comando do Windows. Foi criado um arquivo *batch* (.bat) para executar o seguinte processo: imprimir o timestamp do Windows, executar o teste, imprimir o timestamp do Windows. O tempo aferido é resultado da subtração entre o segundo e primeiro tempo medidos, e foram medido com precisão de centésimos de segundo, sendo apresentados no formato *mm:ss:cc*, onde *m* significa minuto, *s* significa segundo e *cc* significa centésimos de segundo. A entrada dos dois sistemas implementados foi alterada para se adaptar ao teste e receber os parâmetros em linha de comando.

4.2 Resultados

A figura 18 mostra o resultado dos testes para as bases com mil itens com cada um dos suportes mínimos definidos.

Base: 1k	1,0%	0,5%	0,1%
aprioriPHP	00:05.40	00:02.50	-
aprioriSQL	00:01.00	00:01.00	-
Weka	01:30.00	-	-
FerencBodon	00:00.11	00:00.12	-

Figura 18: Resultado dos testes para base de mil itens.

Após o primeiro teste ficou evidente a diferença de desempenho de Weka, que teve um tempo de execução extremamente alto comparado com as outras implementações, e

foi descartada como implementação eficiente, e não foi utilizada nos testes subsequentes.

A versão aprioriPHP teve problema com alocação de memória em uma das extrações de base no teste de suporte mínimo de 0,5%. Pelo fato do teste com essa base não ter entrado no cálculo da média, o tempo final do teste da versão aprioriPHP com suporte mínimo de 0,5% foi menor que o teste da mesma versão com suporte mínimo de 1,0%. Se excluirmos o tempo de execução do teste de 1,0% com a base em cujo teste de 0,5% excedeu a memória, teremos o resultado apresentado na figura 19, que exibe um tempo médio menor para o teste de 1,0%.

Base: 1k	1,0%	0,5%	0,1%
aprioriPHP	00:01.75	00:02.50	-
aprioriSQL	00:01.00	00:01.00	-
Weka	01:30.00	-	-
FerencBodon	00:00.11	00:00.12	-

Figura 19: Resultado dos testes para base de mil itens excluindo-se o teste de uma das bases.

Uma possível causa para esse resultado é a base em questão ter uma quantidade de itens com frequência satisfatória consideravelmente maior em relação às outras bases, fazendo com que sejam gerados mais conjuntos de itens frequentes e aumentando a necessidade de memória para armazenamento dos dados. Pelo fato de a versão aprioriPHP apresentar tais resultados na base de tamanho considerada pequena, essa implementação também foi descartada como implementação eficiente e não participou dos testes posteriores.

O resultado dos testes de bases de tamanho 10 mil, 20 mil, 40 mil e 80 mil, com cada um dos suportes mínimos definidos, pode ser visto nas figuras 20, 21, 22 e 23, respectivamente.

Base: 10k	1,0%	0,5%	0,1%
aprioriSQL	00:02.20	00:18.60	05:19.40
FerencBodon	00:00.13	00:00.13	00:00.13

Figura 20: Resultado dos testes para base de 10 mil itens.

Base: 20k	1,0%	0,5%	0,1%
aprioriSQL	00:13.40	00:38.00	04:47.00
FerencBodon	00:00.14	00:00.15	00:00.15

Figura 21: Resultado dos testes para base de 20 mil itens.

Base: 40k	1,0%	0,5%	0,1%
aprioriSQL	00:01.00	02:05.20	03:31.80
FerencBodon	00:00.11	00:00.17	00:00.14

Figura 22: Resultado dos testes para base de 40 mil itens.

Base: 80k	1,0%	0,5%	0,1%
aprioriSQL	00:23.00	03:28.00	10:14.00
FerencBodon	00:00.17	00:00.12	00:00.22

Figura 23: Resultado dos testes para base de 80 mil itens.

Pelos resultados dos outros testes, nota-se que o suporte mínimo é consideravelmente mais impactante para o desempenho do que o tamanho da base.

Os resultados também destacam a implementação de Ferenc Bodon pelo seu desempenho em relação ao tempo. Porém, a partir da comparação das regras geradas, para melhor análise dos resultados, notou-se que algumas regras não foram geradas pela versão de Ferenc Bodon, o que é um impacto extremamente relevante para um sistema recomendador. A versão aprioriSQL, entretanto, não acusou esse problema.

Além disso, um ponto que pode ser determinante, dependendo da configuração do servidor em que o sistema será utilizado, é o consumo de memória. Segundo [8], a implementação de Ferenc Bodon armazena o conjunto de candidatos a conjunto de itens frequente e as transações em uma *árvore de prefixos* - também conhecida como *trie* -, consumindo memória do sistema. A versão aprioriSQL gera os conjuntos a partir de consultas em SQL ao banco de dados, o que elimina a necessidade de armazenamento em memória principal de todos esses dados.

A versão aprioriSQL obteve maior tempo de execução em relação à versão de Ferenc Bodon, porém o tempo obtido é satisfatório para o objetivo do trabalho. Além disso tem a vantagem de não falhar na geração das regras de associação, de não consumir memória primária para armazenamento dos dados de entrada, e de ser facilmente adaptável e

utilizável por organizações comerciais ou mesmo por pesquisas acadêmicas. Assim, a implementação desta versão foi considerada bem sucedida em seu objetivo.

4.2.1 Nota sobre a Weka

A Weka é uma ferramenta renomada, que implementa uma coletânea de algoritmos para mineração de dados, entretanto se mostrou extremamente ineficiente para o objetivo deste trabalho. Alguns artigos na internet destacam a ineficiência da ferramenta para tarefas de associação utilizando o algoritmo Apriori.

Outro fato que pode virar um problema com essa ferramenta é a entrada de dados. Ela é feita via um arquivo de texto no formato *.arff* que é composto por 3 partes:

- @relation: Identificação da tarefa sendo estudada.
- @attribute: Atributo da tarefa estudada. Cada linha é composta pelo nome do atributo e seu tipo. Se o tipo for texto todas as alternativas que o atributo pode assumir devem ser discriminadas.
- @data: Dados da tarefa estudada. Cada linha contém uma instância com seus valores separados por vírgula. Uma linha deve ter valores para todos os atributos e na mesma ordem em que eles aparecem.

No contexto do nosso trabalho, cada item é um atributo, e cada transação é uma instância. A figura 24 mostra um exemplo de entrada extraído de [15].

```
% Compras em um mercado (completamente simulado)
@RELATION compras

@attribute leite {sim, não}
@attribute ovos {sim, não}
@attribute café {sim, não}
@attribute açúcar {sim, não}
@attribute fraldas {sim, não}
@attribute manteiga {sim, não}
@attribute farinha {sim, não}
@attribute cerveja {sim, não}

@data
sim,sim,sim,sim,sim,sim,não,não
sim,não,sim,não,não,não,sim,não
sim,sim,não,sim,não,não,não,não
não,não,sim,sim,não,não,não,não
não,não,não,não,sim,não,não,não
sim,sim,não,não,não,sim,não,não
não,sim,não,não,não,sim,sim,não
sim,sim,sim,sim,não,sim,não,não
não,não,sim,não,sim,não,não,sim
```

Figura 24: Exemplo de entrada da Weka.

O uso da Weka para tarefas de associação se torna inviável ao se considerar entradas de um comércio eletrônico real, como é o caso da proposta deste trabalho, devido às dimensões que a tabela de dados - @data - pode tomar. Apesar das desvantagens, Weka é uma ferramenta bastante adequada para o aprendizado de mineração de dados.

5 Conclusões

As técnicas e ferramentas de Mineração de dados são extremamente importantes para lidar com a grande quantidade de dados gerada nos dias de hoje, com o objetivo de identificar padrões para extrair informações relevantes. Os sistemas recomendadores se utilizam dessas técnicas para trazer valor ao usuário de sítios de internet com sugestões relevantes, e em um comércio eletrônio um sistema recomendador de produtos é uma ferramenta de grande importância para contribuir com o aumento do faturamento.

A proposta deste trabalho foi desenvolver um sistema recomendador para um comércio eletrônico eficiente e que pudesse ser facilmente adaptado e utilizado por empresas desse segmento. Duas implementações foram desenvolvidas com a linguagem PHP. Uma delas, a *aprioriPHP*, se beneficia das estruturas de dados otimizadas dessa linguagem e utiliza vetores de pares do tipo *(chave, valor)* para armazenamento dos dados e ganho de eficiência. A outra, *aprioriSQL*, se beneficia da eficiência de consultas em SQL.

Testes foram realizados para avaliar o desempenho em relação ao tempo de execução de cada uma das versões implementadas. Foram eleitas duas implementações já existentes que também passaram pelos testes para ser criada uma base de comparação. A versão aprioriPHP foi descartada como implementação eficiente por causa do seu alto consumo de memória. A versão *aprioriSQL*, entretanto, foi aceita como satisfatória, e considera-se que atingiu seu objetivo de ser um sistema eficiente e de fácil adaptação para qualquer sistema de comércio eletrônico de corporações de quaisquer tamanhos. Além disso, ela se mostrou confiável na geração das regras ao ser comparada com uma ferramenta desenvolvida em âmbito acadêmico que, apesar de ter desempenho de tempo de execução melhor, não obteve robustez na geração das regras, tarefa essencial para um sistema recomendador. Não obstante, a versão aprioriSQL pode ser adaptada inclusive em servidores com baixo suporte de memória primária, pois não há manipulação de grandes volumes de dados em memória principal durante sua execução.

6 Parte Subjetiva

Nesta seção contarei sobre os desafios e frustrações encontrados ao longo do desenvolvimento do trabalho, assim como quais foram as disciplinas do curso mais relevantes para o desenvolvimento do trabalho e que passos eu tomaria para aprimorar os conhecimentos da área de estudo caso eu desse continuidade ao projeto.

6.1 Desafios e Frustrações

A proposta inicial do trabalho era desenvolver um sistema recomendador que também levasse em consideração dados do usuário e de outras categorias dos produtos avaliados nas transações, tendo um escopo mais abrangente em relação ao estudo de mineração de dados. Porém, desde o início houve dificuldade para encontrar uma base de dados que fosse adequada como objeto de estudo para o desenvolvimento do trabalho. Busquei em diversos sítios do tema, como [1] e [2], e também pedi a colegas que trabalham no ramo.

Um dos colegas me doou duas bases de dados com transações de comércio eletrônico real, uma delas era do segmento de produtos de beleza para mulher, e a outra do segmento de produtos infantis. Para segurança dos dados, ambas as base foram descaracterizadas antes de ser entregue a mim, e todos os valores foram trocados por termos com identificadores que não eram inteligíveis como, por exemplo, *prod123*. Isto dificultou muito o entendimento dos dados e inviabilizou a base como objeto de estudo para a proposta inicial do trabalho.

No final de abril me envolvi em um projeto profissional de um comércio eletrônico do segmento de produtos de necessidades básicas masculinas, e tive permissão de uso da base descaracterizada para o desenvolvimento do trabalho proposto. O desenvolvimento do trabalho foi feito baseado nessa base de dados.

Com o passar do tempo, tive dificuldades para me dedicar o quanto queria no trabalho, principalmente por pendências acadêmicas, no primeiro semestre, e por causa da

necessidade de trabalhar, no segundo semestre. Em um certo momento, eu e minha supervisora notamos que o prazo estava curto para seguir com o escopo inicialmente proposto e tivemos que reduzi-lo, transformando-o no desenvolvimento de um sistema recomendador para comércio eletrônico que fosse facilmente adaptável a maioria dos sistemas de comércio eletrônico existentes, e que focasse na geração de regras de associação.

Com o escopo reduzido, e o foco em geração de regras de associação, tarefa que necessita apenas dos dados de transações, tornou-se viável a utilização das bases de dados inicialmente obtidas. A base de dados de produtos infantis foi utilizadas na realização dos testes de desempenho.

6.2 Disciplinas Relevantes para o Desenvolvimento do Trabalho

Pelo fato de o sistema desenvolvido manipular dados armazenados em um banco de dados, duas das disciplinas primordiais para o sucesso do trabalho foram MAC0439 - Laboratório de Banco de Dados e MAC0426 - Sistemas de Banco de Dados. A disciplina MAC0323 - Estrutura de Dados também foi importante por ter dado a base para o entendimento dos algoritmos estudados, cujo desempenho está fortemente relacionado à estrutura de dados escolhida em determinadas etapas do algoritmo.

6.3 Continuidade do Projeto

Para dar continuidade ao projeto, um passo importante seria aprofundar o estudo de mineração de dados, entendo melhor suas tarefas e como utilizá-las em conjunto para obter resultados mais relevantes.

Também seria importante o aprofundamento do estudo em estruturas de dados e banco de dados para que seja possível identificar pontos de melhoria nos algoritmos já existentes e desenvolver soluções mais eficientes.

Referências

[1] Frequent itemset mining dataset repository. `http://fimi.ua.ac.be/data/`. Acessado em: 02/12/2013.

[2] Sig kdd | bringing together the data mining, data science and analytics community. `http://www.kdd.org/kddcup/index.php`. Acessado em: 02/12/2013.

[3] Weka 3: Data mining software in Java. `http://www.cs.waikato.ac.nz/~ml/weka/`. Acessado em: 02/12/2013.

[4] AGRAWAL, Rakesh, IMIELINSKI, Tomasz, and SWAMI, Arun. Mining association rules between sets of items in large databases. Technical report, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, 1993.

[5] AGRAWAL, Rakesh and SRIKANT, Ramakrishnan. Fast algorithms for mining association rules. Technical report, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, 1994.

[6] BERRY, Michael and LINIFF, Gordon. *Data mining techniques for marketing, sales and customer support.* John Wiley and Sons, Nova Iorque, 1997.

[7] BODON, Ferenc. Apriori implementation of Ferenc Bodon. `http://www.cs.bme.hu/~bodon/en/apriori/`. Acessado em: 02/12/2013.

[8] BODON, Ferenc. A fast apriori implementation. In B. Goethals and M. J. Zaki, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, volume 90 of *CEUR Workshop Proceedings*, Melbourne, Florida, USA, 19. November 2003.

[9] CAMILO, Cássio Oliveira and DA SILVA, João Carlos. Mineração de dados: Conceitos, tarefas, métodos e ferramentas. Technical report, Instituto de Informática da Universidade Federal de Goiás - INF-UFG, 2009.

[10] HAN, Jiawei and KAMBER, Micheline. *Data Mining: Concepts and Techniques.* Elsevier, São Franciso, California, EUA, 2006.

[11] HAN, Jiawei, PEI, Jian, and YIN, Yiwei. New algorithms for fast discovery of association rules. Technical report, Computing Science Department - University of Rochester, Rochester, Nova Iorque, 14627, 2000.

[12] LAROSE, Daniel T. *Discovering Knowledge in Data: An Introduction to Data Mining.* John Wiley and Sons, Inc., Connecticut, 2005.

[13] OLIVEIRA, R. R. and CARVALHO, C. L. Algoritmos de agrupamento e suas aplicações. Technical report, Universidade Federal de Goiás - INF-UFG, 2008.

[14] RICCI, Francesco, ROKACH, Lior, SHAPIRA, Bracha, and KANTOR, Paul B. *Recommender Systems Handbook*. Springer, Nova Iorque, 2011.

[15] SANTOS, Rafael. Weka na munheca - um guia para uso do weka em *scripts* e integração com aplicações em Java. Technical report, Instituto Nacional de Pesquisas Espaciais, 2005.

[16] ZAKI, Mohammed J., PARTHASARATHY, Srinivasan, OGIHARA, Mitsunori, and LI, Wei. Mining frequent patterns without candidate generation. Technical report, School of Computing Science - Simon Fraser University, 8888 University Dr, Burnaby, BC V5A 1S6, Canada, 1997.