UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Elaine Venson

Um Modelo de Sistema de Recomendação Baseado em Filtragem Colaborativa e Correlação de Itens para Personalização no Comércio Eletrônico

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Prof[a]. Elizabeth Sueli Specialski

Florianópolis, abril de 2002.

Um Modelo de Sistema de Recomendação baseado em Filtragem Colaborativa e Correlação de Itens para Personalização no Comércio Eletrônico

Elaine Venson

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Fernando A. Ostuni Gauthier, Dr.
Coordenador CPGCC, UFSC

Banca Examinadora

_____

Prof[a]. Elizabeth Sueli Specialski, Dr[a].
Orientadora INE, UFSC

_____

Prof. José Marcos Silva Nogueira, Dr.
DCC, UFMG

_____

Prof. João Bosco Mangueira Sobral, Dr.
INE, UFSC

_____

Prof[a]. Maria Marta Leite, M. Sc.
INE, UFSC

Para os meus queridos pais, Volmir e Maria Irene,

sem seu apoio e seus sábios ensinamentos esta jornada não seria possível.

Para o meu doce Du,

um verdadeiro anjo da guarda.

Agradecimentos

O meu especial agradecimento à professora Marta, que encorajou e acompanhou de perto todo o trabalho.

À professora Beth, que me acolheu e confiou em meu potencial.

À Universidade Federal de Santa Catarina e aos professores membros da banca José Marcos Silva Nogueira e João Bosco Mangueira Sobral.

Aos colegas e amigos que, de várias formas, contribuíram para o desenvolvimento deste trabalho.

Ao Supermercado Hippo Ltda., na pessoa de Joarez Wernke, por ter contribuído com os dados necessários para a avaliação do modelo.

Sumário

Lista de Abreviaturas ..... **VII**
Lista de Figuras ..... **VIII**
Lista de Tabelas ..... **IX**
Resumo ..... **X**
Abstract ..... **XI**
1 Introdução **1**
1.1 Motivação 2
1.2 Objetivo 3
1.3 Organização Do Trabalho 3
2 A Internet e o Gerenciamento do Relacionamento com os Clientes **5**
2.1 Introdução 5
2.2 Comércio Eletrônico 6
2.2.1 O Comércio Eletrônico no Brasil 7
2.3 CRM Eletrônico 7
2.4 Personalização 10
2.4.1 Personalização e Customização 13
2.4.2 Técnicas de Personalização 14
3 Sistemas de Recomendação **16**
3.1 Introdução 16
3.2 Evolução dos Sistemas de Recomendação 17
3.3 Taxonomia 21
3.3.1 Entradas e Saídas Funcionais 22
3.3.2 Métodos de Recomendação 26
3.3.3 Outros Aspectos de Projeto 28
3.4 Correlação Pessoa-a-Pessoa 30
3.4.1 O Processo de Obtenção de Recomendações 31
3.4.2 Problemas 37

3.5 Correlação Item-a-Item 39
3.5.1 O Processo de Obtenção de Recomendações 40
3.6 Métricas de Avaliação 43
3.6.1 Cobertura 43
3.6.2 Precisão Estatística 44
3.6.3 Precisão de Suporte a Decisão 45
4 O Modelo Proposto 48
4.1 Introdução 48
4.2 O Modelo da Aplicação de Recomendação 48
4.3 O Método de Recomendação Híbrido 50
4.3.1 Representação 51
4.3.2 Construção do Modelo 54
4.3.3 Geração das Recomendações 57
4.4 Arquitetura do Modelo 58
5 Avaliação Experimental 61
5.1 Introdução 61
5.2 Implementação do Modelo 61
5.3 Os Conjuntos de Dados 62
5.4 Metodologia e Métrica de Avaliação 63
5.5 Resultados Experimentais 64
5.5.1 Efeito dos Algoritmos de Similaridade 65
5.5.2 Sensibilidade ao Peso do Cliente-Alvo 67
5.5.3 Sensibilidade ao Tamanho do Modelo 68
5.5.4 Comparação com Outros Métodos de Recomendação 69
6 Conclusão 72
7 Referências Bibliográficas 74
Anexo 1 – Código De Implementação Das Aplicações 78

Lista de Abreviaturas

B2B	*Business-to-business*
B2C	*Business-to-consumer*
CE	Comércio Eletrônico
CO	Cosseno
CP	Correlação de Pearson
CRM	*Customer Relationship Management*
ECRM	*Electronic Customer Relationship Management*
EDI	*Electronic Data Interchange*
FC	Filtragem Colaborativa
LSI	*Latent Semantic Indexing*
MAE	*Mean Absolute Error*
MCI	Modelo baseado em Correlação de Itens
MFC	Modelo baseado em Filtragem Colaborativa
MMV	Modelo de produtos mais vendidos
MP	Modelo proposto
RMSE	*Root Mean Squared Error*
ROC	*Receiver Operating Curve*
TI	*Tecnologia da Informação*

Lista de Figuras

Figura 2.1 – Crescimento das categorias de produtos no CE 7

Figura 3.1 – Esquema de funcionamento de um site utilizando um sistema de recomendação 21

Figura 3.2 – As três partes principais de um Sistema de Recomendação 31

Figura 3.3 – Resultado obtido pela correlação baseada em itens relativo ao resultado obtido pela filtragem colaborativa 40

Figura 3.4 – Exemplo de gráfico ROC 46

Figura 4.1 – Modelo de personalização proposto 50

Figura 4.2 – Correspondência entre os processos dos modelos tradicionais e os processos do modelo proposto 51

Figura 4.3 – Formas de representação 53

Figura 4.4 – Os subprocessos da construção do modelo 54

Figura 4.5 – Algoritmos de filtragem colaborativa e correlação item-a-item 55

Figura 4.6 – As partes componentes do modelo 59

Figura 5.1 – A aplicação do modelo e a aplicação para simulação 62

Figura 5.2 – Impacto das combinações de medidas de similaridade sobre o resultado .. 66

Figura 5.3 – Sensibilidade do modelo proposto ao valor de p 67

Figura 5.4 – Sensibilidade do modelo proposto ao valor de m 68

Figura 5.5 – Comparação do modelo proposto com os demais modelos 70

Lista de Tabelas

Tabela 3.1 – Matriz cliente-produto 32

Tabela 5.1 – As características dos conjuntos de dados utilizados na avaliação 63

Tabela 5.2 – Testes das medidas de similaridade 66

Resumo

A evolução dos sites de comércio eletrônico, que tem aumentado a interatividade com os usuários, e os novos modelos de gestão centrados no cliente requerem ferramentas computacionais capazes de processar a imensa quantidade de dados obtida, a fim de gerar conhecimento e personalizar a experiência de compra.

Os sistemas de recomendação têm obtido destaque na Internet, e em especial no comércio eletrônico, como uma solução para se alcançar a personalização e agregar valor ao relacionamento com os clientes. Novas abordagens que observem as características da comercialização de produtos são necessárias para que se disponha de um modelo abrangente para criar aplicações nesta área.

Este trabalho propõe a criação de um modelo híbrido de sistema de recomendação baseado nos dois métodos de recomendação mais conhecidos: a filtragem colaborativa e a correlação item-a-item. O modelo proposto aborda questões como freqüência e quantidade de compra dos produtos.

O método híbrido foi implementado e testado com dados reais de compra. Os resultados mostraram que o modelo proposto supera em média a qualidade das recomendações geradas pelas outras abordagens e também comprovaram a validade da utilização do método em casos em que os produtos são comprados repetidamente.

Palavras-chave: filtragem colaborativa, correlação item-a-item, personalização, comércio eletrônico.

Abstract

The evolution of e-commerce sites, increasing user's interactivity, and the new customer centric management models apply for computer tools capable of processing the huge amount of gathered data, in order to generate knowledge and personalize the buying experience.

Recommender systems have gained prominence on the Internet, specially in electronic commerce, as a solution for reaching personalization and delivering value to customers. New approaches that examine the products commercialization characteristics are needed to provide a full coverage model to develop applications in this area.

This work proposes the creation of a hybrid recommender system model based on the two best-known recommendation methods: collaborative filtering and item-to-item correlation. The proposed model attempts to issues as frequency and quantity of purchased products.

The hybrid method was implemented and tested with real purchase data. Results showed that the proposed model overcomes, in average, the quality of the recommendations generated by other approaches and also proved the usefulness of the method in cases where products are bought repeatedly.

Keywords: collaborative filtering, item-to-item correlation, personalization, electronic commerce.

1 Introdução

Quando houve o *boom* do comércio eletrônico, muitas empresas correram para a Internet, abandonando todas as regras do comércio tradicional, principalmente as do varejo. Depois de muitos insucessos, quebras e prejuízos, as organizações remanescentes perceberam que não podiam simplesmente cortar o cordão umbilical que as unia com as antigas práticas. Ou seja, as regras que servem para o mercado tradicional também se aplicam ao mercado digital.

A duras penas, o mercado eletrônico vem evoluindo. Ano a ano, apesar das crises econômicas globalizadas, aumenta o número de internautas e crescem as estatísticas do mercado digital e as cifras que o movimentam. Investir neste setor é ainda um bom negócio.

Ao encontro deste cenário, onde ainda estão amadurecendo os modelos de gestão organizacionais, surge uma nova filosofia que promete revolucionar a maneira de conduzir os negócios: o CRM (*Customer Relationship Management*). O CRM está transformando o modo com que as empresas vêem seus clientes. O advento de tecnologias como a Internet, o *datawarehouse* e o *data mining*, aliado à Tecnologia da Informação, vem permitindo às empresas uma nova forma de interagir com seus clientes, muito mais ágil, barata e eficaz.

Também denominado por alguns de “Marketing de Relacionamento”, o CRM é uma nova estratégia de negócios utilizada para ampliar e principalmente manter a base de clientes. O objetivo é atuar de forma mais incisiva sobre os clientes mais valiosos. Conforme advoga o Princípio de Pareto (Hafner, 2001), apenas 20% dos clientes geram 80% da receita de uma empresa. Este princípio ilustra a importância de se conquistar a fidelidade do cliente e uma das maneiras de fazê-lo é tornar seu relacionamento com a empresa tão conveniente que ele tenha que gastar muita energia para passar para uma empresa concorrente, ou seja, criar uma barreira de troca. Pode-se fazer isto acontecer estabelecendo um diálogo empresa-cliente que seja interessante para ambas as partes.

De um lado, a empresa retém o seu cliente, enquanto do outro, o cliente recebe serviços agregados ao produto.

A proposição de valor ao cliente através de relacionamentos individuais não é uma tarefa simples e exige pessoas capacitadas, tecnologia da informação e algoritmos sofisticados. Hoje em dia a quantidade de clientes, principalmente das corporações presentes na Internet, conta-se aos milhares e oferecer atendimento pessoal a todos eles torna-se impraticável. Sendo assim, é imprescindível contar com ferramentas que sejam capazes de automatizar algumas tarefas de modo a oferecer tratamento diferenciado e obter vantagem competitiva à frente das organizações concorrentes.

1.1 Motivação

Uma enorme quantidade de dados em volume e diversidade pode ser obtida através da interação com o cliente via Internet. Como trabalhar essa massa de dados, gerando informações úteis e conhecimento é ainda um grande desafio para as organizações que desejam manter um diálogo com seus clientes.

Uma nova tecnologia que se propõe a fazer uso dos dados obtidos dos clientes para prover um serviço personalizado e agregar valor aos relacionamentos é filtragem colaborativa ou *collaborative filtering* (FC). Esta tecnologia faz parte de uma classe maior de métodos de filtragem de informação, que são os chamados sistemas de recomendação.

Na literatura estudada constam várias pesquisas nas quais são feitas análises comparativas de algoritmos de recomendação (Karypis, 2000; Sarwar et al, 2000; Kitts, 2000; Schafer et al, 2001; Billsus, Pazzani, 2001). Em geral estas análises incluem testes que utilizam conjuntos de dados de comércio eletrônico ou de sites experimentais de pesquisa. Entretanto, nos conjuntos de dados de *e-commerce*, questões inerentes às características de comercialização dos diferentes produtos, tais como a freqüência e quantidade de compra, não são abordadas. Sendo assim, faz-se necessário um modelo genérico para o desenvolvimento de aplicações que atendam a estas diferentes características, de modo a ampliar o número de organizações e clientes beneficiados pelos sistemas de recomendação.

Por outro lado, apesar da crescente demanda por este tipo de sistema nos sites de comércio eletrônico estrangeiros, ainda não se tem notícia da utilização de tais técnicas na Internet brasileira. Assim, a pesquisa científica nesta área também visa a contribuir com a sociedade através da geração de conhecimento, para que as empresas nacionais tenham recursos que lhes permitam criar ferramentas de personalização. Desta forma as organizações "ponto-com-ponto-br" poderão agregar valor ao relacionamento com seus clientes através de sites mais atrativos, aumentar as vendas e transformar simples visitantes em compradores; ao mesmo tempo em que os clientes poderão ver os dados que fornecem às empresas convertidos também em próprio benefício.

1.2 Objetivo

Nas técnicas de personalização, as organizações utilizam o conhecimento que possuem sobre o cliente para proporcionar-lhe uma melhor experiência de compra, oferecendo, por exemplo, produtos e serviços condizentes com o seu perfil.

Este trabalho visa fornecer subsídios para o desenvolvimento de uma aplicação que, através do emprego de algoritmos de recomendação, seja capaz de gerar sugestões de compra personalizadas para os clientes de uma loja virtual. Pretende-se criar um modelo genérico e que possa adaptar-se a vários segmentos de varejo. Por exemplo: a aplicação deve poder ser configurada tanto para um supermercado, onde compram-se produtos repetidamente, quanto para uma loja de discos e livros, onde é mais provável que o cliente compre apenas um exemplar de cada produto.

Após a implementação de um algoritmo específico de sistema de recomendação, tenciona-se verificar a validade e eficácia dos resultados, utilizando para isso dados reais de compra de um site de comércio eletrônico.

1.3 Organização do Trabalho

O presente trabalho está organizado da seguinte maneira: o capítulo dois aborda o conceito de gerenciamento do relacionamento com os clientes e sua aplicação na Internet, de modo a contextualizar a personalização no comércio eletrônico, servindo

também como uma base para o capítulo seguinte. Neste são tratados em detalhes os sistemas de recomendação, incluindo um esboço dos algoritmos mais importantes pesquisados na bibliografia. Na seqüência, o capítulo quatro apresenta a proposta de desenvolvimento de um modelo genérico com um método híbrido de sistema de recomendação. A avaliação empírica do modelo concebido utilizando dados de compra é mostrada no capítulo cinco. E, por fim, os capítulos seis e sete incluem respectivamente a conclusão do trabalho e as referências bibliográficas.

2 A Internet e o Gerenciamento do Relacionamento com os Clientes

2.1 Introdução

A Internet, como um meio de comunicação globalmente acessível e independente de plataforma, vem revolucionando o relacionamento entre pessoas e organizações. A constante evolução das tecnologias, o barateamento dos equipamentos de informática e a ampliação das redes de telecomunicações fizeram com que a Internet começasse a alcançar inclusive as camadas menos favorecidas da população.

O comércio eletrônico (CE), apesar da assustadora crise atravessada pelas chamadas empresas "ponto-com", está conseguindo crescer. Até o momento, o impacto causado por esta nova forma de fazer negócios não foi muito grande, entretanto algumas pesquisas, como a do *Boston Consulting Group* (Wenrich & Becerra, 2001), mostram que o mercado eletrônico está amadurecendo no Brasil.

Este crescimento do comércio eletrônico e a entrada de mais empresas no mercado acirraram a disputa pelos clientes. A localização geográfica já não pode mais ser considerada como uma vantagem competitiva, pois a Internet é capaz de derrubar barreiras e ampliar mercados. No comércio eletrônico, a distância entre uma empresa e seu concorrente mede-se à base de "cliques de *mouse*" e, além disso, o meio eletrônico favorece a competitividade entre pequenas e grandes empresas.

Num cenário como este, muitas organizações estão descobrindo que investir em relacionamento pode ser o grande diferencial para manterem-se competitivas. Conhecer os clientes, reter os que dão lucros para a empresa e gerenciar este relacionamento é uma estratégia mais do que nunca defendida pelo pessoal da área de marketing. Esta é a nova filosofia das empresas que estão focadas não mais nos produtos e sim nos clientes, amplamente difundida pela sigla CRM (*Customer Relationship Management*) e também conhecida como **Gerenciamento do Relacionamento com os Clientes**.

Um dos fatores impulsionadores deste novo modo de gerir organizações foi justamente a Internet, por ser um meio de fácil obtenção de informações sobre os clientes. Hoje o CRM almeja a integração de todos os canais de comunicação da empresa com o cliente, de forma a **fornecer uma visão única da empresa** e, ao mesmo tempo, **ter uma visão única do cliente**. Portanto, o web site é apenas mais um canal de venda para as empresas que possuem lojas físicas, mas pode ser muito bem utilizado para obter informações valiosas e compartilhá-las com os outros pontos de contato com o cliente. Pela importância deste segmento, o componente do CRM na Internet é denominado de CRM eletrônico ou eCRM.

Neste capítulo são trazidas as questões relacionadas a três temas: o comércio eletrônico, o eCRM e, por fim, a personalização das ações de marketing direcionadas aos clientes.

2.2 Comércio Eletrônico

Segundo Zwass, o comércio eletrônico ou *e-commerce* pode ser definido como "compartilhamento de informações de negócio, manutenção de relacionamento de negócio e condução de transações de negócio por meio de redes de telecomunicações" (apud Romano, 2001), sendo que, neste caso, as redes de telecomunicações referem-se principalmente à Internet.

Dentro do comércio eletrônico pode-se classificar as relações comerciais em duas categorias:

- *Business-to-business* (B2B): relações entre empresas;
- *Business-to-consumer* (B2C): relações entre empresas e consumidores finais.

O B2C foi o segmento que causou o *boom* do comércio eletrônico em todo o mundo nos últimos anos. Antes disso, o CE era basicamente realizado entre empresas que utilizavam a troca eletrônica de dados ou EDI (*electronic data interchange*) através de redes privadas para efetuar transações eletrônicas.

2.2.1 O Comércio Eletrônico no Brasil

Em um recente estudo (Wenrich, Becerra, 2001), o Boston Consulting Group estimou, para a América Latina, um crescimento de 137% do varejo on-line no ano 2001 em relação ao ano anterior, quando seria atingida uma cifra de U$ 1.28 milhões em transações. Segundo este mesmo estudo, o Brasil representa dois terços do mercado eletrônico latino e é indicado como líder mundial em vendas diretas de automóveis através da internet. As taxas de crescimento das várias categorias de produtos comercializados on-line na América Latina podem ser observadas na figura 2.1.

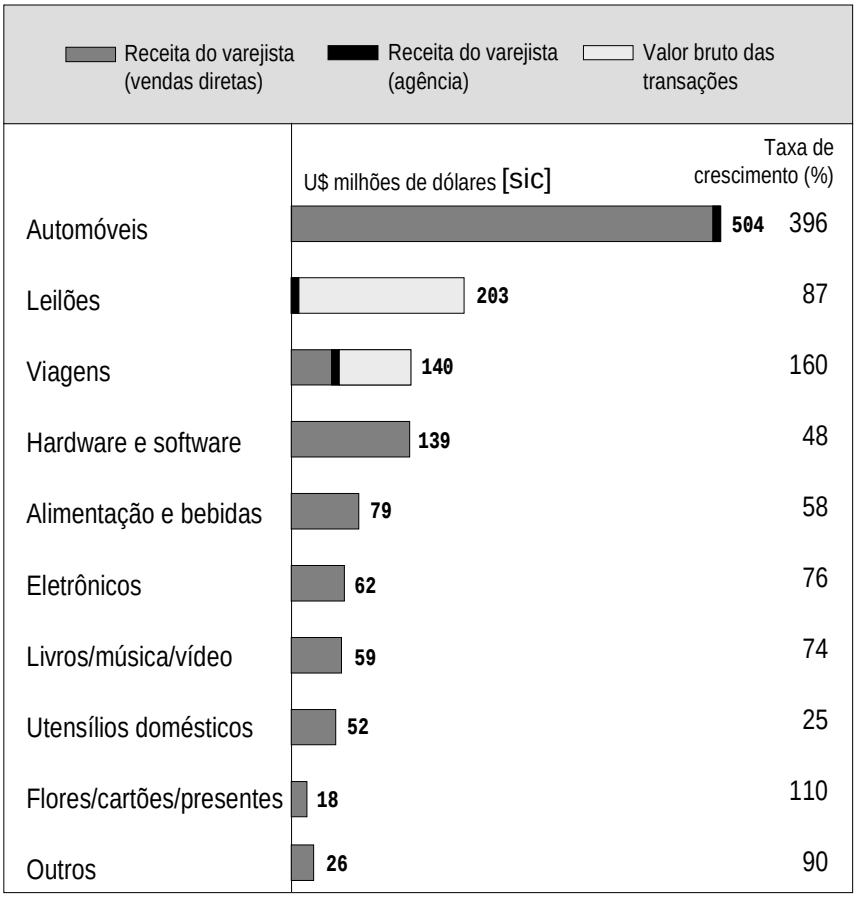


Figura 2.1 – Crescimento das categorias de produtos no CE

Fonte: Wenrich, Becerra, 2001, pg. 3 (tradução minha)

2.3 CRM Eletrônico

A tecnologia trouxe aos consumidores muita informação e mais possibilidades de escolha. Hoje em dia os clientes são informados, exigentes e impacientes, por este motivo suprir suas expectativas é crítico para o sucesso dos negócios. Além disso, as

organizações que atuam em mercados muito competitivos têm buscado um diferencial, que vai além do tradicional *mix* de marketing (produto, preço, ponto de venda e promoção), a fim de garantir a sua sobrevivência e destacar-se entre os concorrentes.

O "novo" fator competitivo a ser explorado é o relacionamento com os clientes. Já se sabe que é muito mais caro adquirir novos clientes que manter e reter os antigos. É por isso que as organizações estão mudando seu foco que antes era nos produtos para o foco nos clientes e tentando conhecê-los melhor. Visando tornarem-se cada vez mais competitivas, as organizações cujas estratégias estão voltadas para o cliente e que realizam negócios eletrônicos estão adotando uma nova abordagem: o eCRM.

O eCRM pode ser definido como a utilização da internet para desenvolver, educar e gerenciar múltiplos relacionamentos, incluindo clientes, parceiros e fornecedores (Sindell, 2000). Para Paul Greenberg (2001), o eCRM é o CRM on-line, é ser capaz de servir os clientes via internet ou permitir que eles próprios se sirvam. Além disso, esta nova abordagem implica um meio de comunicação adicional e um maior nível de interatividade com o cliente. Embora a filosofia, a metodologia e os processos sejam os mesmos do CRM tradicional, a tecnologia e a arquitetura do eCRM são diferentes, ou seja, muitas das questões que diferenciam o eCRM do CRM são aquelas relativas à Internet (Greenberg, 2001).

Sendo o eCRM um meio adicional de comunicação, é importante ressaltar que quando houver outros canais de contato com o cliente, como por exemplo lojas físicas ou *call center*, deve haver uma integração completa com o CRM on-line, sob o risco de deixar o cliente frustrado. Da mesma forma, se os dados utilizados na Internet não vierem dos outros canais, o cliente poderá estar sendo atendido de forma errada ou incompleta. O eCRM deve ser instalado em conjunto com o CRM tradicional ou o resultado pode até ser negativo (Greenberg, 2001).

Uma questão chave para o CRM tradicional como também para o eCRM é a retenção dos clientes. Segundo Steve Dorio, diretor da *IMT Strategies*, o caminho para reter os clientes é construir barreiras de saída ou barreiras de troca. Dorio (apud Sindell, 2000) destaca nove maneiras de fazê-lo:

- **Melhorar a curva de aprendizado do cliente**: o web site deve possuir interfaces amigáveis e fáceis de usar. Além disso, devem estar disponíveis

informações e instruções claras sobre os processos de compra, pagamento, entrega etc, bem como uma lista com as dúvidas mais freqüentes.

- **Integrar processos**: o cliente deve ter uma visão única da empresa. Se a organização trabalha com parceiros e intermediários, os processos precisam ser completamente integrados. Uma boa maneira de conseguir isto é através do gerenciamento da cadeia de suprimentos, que além de agilizar os processos contribui para a diminuição dos preços.
- **Oferecer customização de massa e seleção de produtos**: o site pode tornar-se mais atrativo com o oferecimento de uma grande variedade de produtos e serviços através de parcerias, customização de massa e incentivo de preços. Por exemplo, os clientes da General Motors no Brasil podem customizar e comprar o Celta no site www.celta.com.br, selecionando atributos como cor e opcionais do automóvel a um preço menor que nas concessionárias. Para que isto fosse possível a GM instalou sua fábrica em um local onde tem como vizinhos os fornecedores de componentes para o seu produto.
- **Reduzir riscos e aumentar a confiança**: a organização deve manter o cliente informado sobre sua política de segurança e privacidade. O cliente precisa estar confiante de que seus dados pessoais e número do cartão de crédito não sejam utilizados para outros propósitos que não sejam a compra naquela loja. Explicar como os dados são coletados e armazenados ajuda a desenvolver a confiança do cliente.
- **Instituir programas de lealdade on-line**: oferecer um desconto para a segunda compra ou trocar pontos acumulados nas diversas compras por prêmios é uma estratégia eficiente para reter clientes. Não exclusivo do *e-commerce*, este artifício pode e, se houver outros canais, deve estar presente em todos os pontos de venda.
- **Estabelecer uma identidade de marca on-line**: também como nos mercados tradicionais, construir uma marca pode ser muito importante para se obter sucesso. Neste caso, o fato de a organização contar com lojas físicas e já ter uma identidade de marca pode favorecê-la junto aos clientes.

- **Colaborar com organizações relacionadas**: as lojas podem expandir sua base de clientes e área de atuação mantendo relações com parceiros comerciais. Estas parcerias permitem que, por exemplo, uma floricultura estabelecida em Florianópolis ofereça entrega de flores e cestas de café da manhã em Fortaleza.
- **Criar padrões on-line**: estabelecer padrões que melhorem a experiência on-line dos clientes pode ajudar a torná-los fiéis. Isto pode ser feito através de tecnologias e práticas de gerenciamento revolucionárias, pelas quais o cliente nem esperava.
- **Personalizar a experiência on-line**: deve-se utilizar todas as informações obtidas tais como o fluxo de cliques, o comportamento de compra e as preferências dos clientes para segmentá-los, fazer venda cruzada e oferecer recomendações de produtos personalizadas. Um bom exemplo de uma organização que proporciona experiência de compra personalizada é a Amazon.com, que emprega aplicações baseadas em filtragem colaborativa para fazer recomendações individuais. Estas recomendações permitem realizar *up-sell* e *cross-sell* dos seus produtos. A personalização e os sistemas de recomendação são o foco deste trabalho e serão tratados em mais detalhes na seqüência.

2.4 Personalização

Uma organização que deseja integrar todos os pontos de contato com o cliente e fornecer uma visão única de si deve ser capaz de utilizar os dados provenientes de todos os canais em todos os canais para todas as suas atividades, inclusive a personalização. Sendo assim, juntamente com o CRM surgiu o novo conceito de empresas **centradas em personalização** (*personalization-centric*) (Larsen, Tutterow, 2001), nas quais não só o web site usa tecnologia de personalização, como também toda a organização, permitindo desta forma tomar decisões em tempo real a cada transação. O que antes era uma tecnologia de web site foi transformada pelo CRM em uma filosofia corporativa. Pode-se observar que a Internet impulsionou a personalização e foi o fenômeno

inspirador das novas tecnologias que estão contribuindo para agregar valor ao relacionamento com o cliente em todos os pontos de contato.

As primeiras técnicas de personalização eram baseadas em perfis, segmentação e regras de comportamento médio. Estas regras, que consistiam em atributos como por exemplo, classe social, nível de escolaridade, dados demográficos, sexo, idade, e assim por diante, eram utilizadas para descrever o perfil dos clientes adequados para determinados produtos. A base de dados dos clientes era filtrada por estes perfis e assim eram obtidos os segmentos de mercado. Às pessoas que compunham cada grupo eram oferecidos os produtos que se encaixassem no perfil do seu segmento.

A mudança de foco do produto para o cliente, levantada pelo CRM, sugere que as formas de classificar o cliente não sejam mais baseadas em segmentos de mercado, mas sim no comportamento real de compra. Ao invés de olhar para os produtos e definir quais os clientes podem comprá-los, a nova filosofia do marketing de relacionamento rege que se deve olhar para cada cliente, se possível no segmento de um (marketing *one-to-one*), e verificar quais produtos ele compraria.

Já é consenso que esta mudança de abordagem provou que os antigos segmentos de mercado não definiam tão bem grupos de pessoas (Gordon, 1998) (Sharma, Lambert, 1998). Conhecer realmente o cliente é saber suas preferências sobre música, livros, viagens, restaurantes etc. e identificar o seu comportamento através do histórico de compras. As aplicações que fazem uso deste tipo de informação são capazes de oferecer produtos de uma forma muito mais precisa do que através dos segmentos de mercado. Parte-se do pressuposto de que não existe cliente médio, existem clientes individuais e eles possuem preferências em comum com outros clientes. Dizer que clientes que gostam dos mesmos discos tendem a comprar itens em comum é muito diferente de dizer que clientes que moram na mesma região gostam das mesmas coisas.

Um conceito muito importante do CRM é que o relacionamento deve ser uma via de mão dupla, ou seja, o relacionamento deve ser **mutuamente** benéfico para empresa e cliente. A seguir são citadas as vantagens da personalização para ambas as partes **(**Larsen e Tutterow, 2001**)**.

Benefícios da personalização do ponto de vista do cliente:

- **Casar as necessidades e desejos dos clientes com os produtos disponíveis** é a principal funcionalidade de muitas aplicações de personalização mais modernas, como a correlação pessoa-a-pessoa.
- **Economia de tempo** ao auxiliar o cliente a encontrar os melhores produtos condizentes com o seu perfil e apresentá-los durante a navegação. A recomendação em tempo real pode ocorrer em vários canais de comunicação como web site, *call center*, quiosques, balcão de serviços, etc.
- **Propiciar o conhecimento de novos produtos** que superam a expectativa do cliente através de sugestões condizentes com seu perfil. Antes, este era um benefício exclusivo da experiência de compra dentro de uma loja tradicional (de tijolo e cimento).
- **Demonstrar reconhecimento pessoal durante as transações** dá ao cliente uma sensação de confiança e conforto: “estas pessoas me conhecem”. Um cliente que retorna gosta de saber que a organização lembra de suas compras anteriores, pois isto acelera o processo de decisão e assegura que a escolha certa foi feita para aquele cliente.
- **Demonstrar reconhecimento pessoal em atividades promocionais** melhora a opinião do cliente sobre a empresa direcionando somente o material pertinente a ele e não enchendo a caixa de correio (seja eletrônico ou não) com informações inúteis e indesejadas. As recomendações individualizadas podem ser vistas não como propagandas, mas sim como um serviço ao cliente. Por exemplo, “Caetano Veloso e Gilberto Gil lançam disco acústico com coletânea de suas melhores músicas” pode ser uma boa notícia para um cliente que é fã de música popular brasileira e gosta tanto do Caetano quanto do Gil.

Benefícios da personalização do ponto de vista da organização:

- **Lealdade**. Quanto melhores forem satisfeitas as necessidades do cliente, mais provável será que ele se torne leal à empresa. A personalização aumenta as chances de o cliente receber sempre a melhor oferta de acordo com o seu perfil e que atenda ou supere as suas expectativas. Se ele perceber que é conhecido

em todos os pontos de contato é mais provável que ele visite vários destes pontos, e fazendo isso ele se torna mais leal e lucrativo.

- **Redução de custos por cliente**. A automação é o único caminho para atender os clientes individualmente quando o atendimento cara-a-cara não é possível. E mesmo quando for possível, se for permitido ao vendedor consultar as informações sobre o cliente, este poderá ser atendido de uma forma muito mais eficaz. Sendo assim, as ferramentas de personalização auxiliam o oferecimento de um alto nível de serviço a um custo relativamente baixo.
- **Otimização das oportunidades de *cross-sell* e *up-sell***. Esta é uma característica muito atrativa para lojas que possuem um catálogo de produtos extremamente extenso, como as livrarias on-line. Os sistemas de personalização permitem que o cliente navegue por milhares de produtos e ao mesmo tempo receba recomendações precisas, que tendem a aumentar as vendas.
- **Promoções bem-sucedidas**. As tecnologias de personalização aplicadas a propagandas, promoções dirigidas por carta, serviço ao cliente, entre outras possibilidades, melhoram a aplicabilidade de qualquer item ao receptor da promoção.

2.4.1 Personalização e Customização

É comum ver os termos personalização e customização sendo utilizados no mesmo contexto, às vezes até com o mesmo significado. Para se ter clareza neste estudo é bom que ambos sejam distinguidos, pois têm significados muito diferentes.

A customização na Internet é tida como uma atividade que é controlada diretamente pelo usuário ou cliente. Em contrapartida, o conteúdo da personalização on-line é geralmente gerenciado pela organização responsável pelo site (Sindell, 2000).

Na customização o cliente seleciona aquilo que ele quer ver e tem total controle sobre suas atitudes. Sites de notícias como o do *The New York Times* (www.nyt.com) permitem que o usuário selecione os tipos de notícias que deseja receber por e-mail. Em

outros, as próprias páginas web podem ser customizadas para mostrar somente o que o cliente deseja.

As interfaces para customização são geralmente apresentadas como menus de opções, onde o cliente seleciona atributos e itens de interesse após ter feito o seu cadastro no site. É uma forma de oferecer valor, pois atende às necessidades individuais das pessoas de forma eficiente. No entanto, deve-se ressaltar que esta abordagem funciona bem somente quando o cliente sabe o quer, sabe o que vai ver quando selecionar uma opção e o que vai deixar de ver se não selecionar alguma.

Embora o objetivo seja o mesmo: agregar valor ao serviço prestado para o cliente, na personalização a atividade de gerar páginas individualizadas é de responsabilidade dos web sites. O software de personalização tem a função de inferir o que cada cliente deseja tendo como base o seu perfil. A personalização é uma tarefa delicada, já que o cliente pode ficar frustrado se o software falhar. Por outro lado, nos casos em que as necessidades dos clientes possam ser facilmente descritas em termos computacionais, a personalização pode trazer imensos benefícios.

2.4.2 Técnicas de Personalização

Para efetivar a personalização é preciso conhecer o cliente utilizando as informações disponíveis e, na medida do possível, evitar exigir o preenchimento de formulários para constituição do perfil sem que antes sejam obtidos benefícios. Sindell (2000) destaca duas formas de personalizar um site:

- **Filtragem de regras de negócios**: é um processo de levantamento de dados a respeito do cliente na qual ele preenche um conjunto de critérios que serão utilizados para gerar conteúdo personalizado. Os critérios aplicados são os mais variados, tais como endereço, sexo e tipo de computador que possui.
- **Filtragem colaborativa**: é um processo em que as preferências de um cliente são combinadas com as preferências de outros clientes que tenham interesses similares, a fim de gerar conteúdo personalizado e relevante. Por exemplo, se José freqüentemente compra livros de ficção científica e os seus pares, ou seja, aquelas pessoas com as quais ele compartilha preferências,

> estão comprando o livro “3001: A Odisséia Final”, o software de filtragem colaborativa o recomendará a José, já que a probabilidade de ele gostar do livro é alta. Neste caso, os outros clientes estão colaborando com José como se estivessem fazendo propaganda boca-a-boca, mas sem estarem realmente interagindo.

De forma mais abrangente consideram-se as aplicações de filtragem colaborativa como parte de uma classe maior de técnicas e algoritmos: os chamados **sistemas de recomendação**. Na verdade, a filtragem colaborativa foi a primeira tecnologia de recomendação e a que até hoje obteve mais sucesso na Internet. Os sistemas de recomendação e em particular a filtragem colaborativa ou correlação pessoa-a-pessoa são o tema do próximo capítulo.

3 Sistemas de Recomendação

3.1 Introdução

Muitos estudiosos do Marketing de Relacionamento devem concordar com o CEO da Amazon.com, Jeff Bezoz, quando ele declarou “Se eu tenho 2 milhões de clientes na Web, devo ter 2 milhões de lojas virtuais”. Sem dúvida, uma das formas de materializar algumas das teorias do Marketing de Relacionamento ou Marketing Um-a-Um é personalizar a experiência de compra do cliente na web. Entretanto, sem a utilização de sistemas e algoritmos sofisticados, a personalização torna-se cara, pois o volume de informações de clientes e produtos que precisa ser processado é imenso.

Uma área que vem sendo amplamente pesquisada e também aplicada comercialmente com o intuito de fornecer subsídios para a personalização de massa é área de sistemas de recomendação. Um sistema de recomendação é um mecanismo capaz de aprender com as experiências dos usuários para poder recomendar, dentre os produtos disponíveis, o que seja mais interessante para um usuário particular. Técnicas de estatística e *knowledge discovery* são aplicadas para obter-se as recomendações.

Os sistemas de recomendação estão sendo utilizados pelos sites de *e-commerce* para oferecer sugestões de compra para os seus clientes, que desta forma têm uma loja adaptada às suas características e necessidades. As recomendações podem ser baseadas em vários tipos de informação – como por exemplo os produtos mais vendidos de um site, as informações demográficas do cliente, ou análises do histórico de compras para predizer comportamento de compra futuro.

Segundo Schafer (1999), os sistemas de recomendação melhoram as vendas no comércio eletrônico de três maneiras:

- **Convertendo visitantes em compradores**: os visitantes freqüentemente navegam pelos sites sem efetivamente efetuar uma compra. Um sistema de recomendação pode ajudar o visitante a encontrar o que ele deseja.

- **Aumentando a venda cruzada**: os sistemas de recomendação podem sugerir ao cliente a compra de produtos adicionais, como a oferta de produtos baseada no que o cliente já tem no carrinho na hora de fechar o pedido.
- **Construindo lealdade**: os sistemas de recomendação melhoram a lealdade agregando valor ao relacionamento entre o site e o cliente. Eles aprendem as preferências dos clientes e apresentam interfaces personalizadas, auxiliando-o no processo de compra. Assim, o cliente retorna ao site que oferece os produtos que condizem com a sua necessidade. Além disso, quanto mais o cliente usa o sistema de recomendação – ensinando o que ele quer – mais leal ele será, pois mesmo que o site concorrente venha a implementar um sistema semelhante, o cliente terá de gastar mais energia e tempo para ensinar novamente suas preferências.

Com o objetivo de tornar mais clara esta abordagem de personalização e justificar a aplicabilidade dos sistemas de recomendação, este capítulo apresenta o tema da seguinte maneira: primeiramente é descrito o panorama histórico contendo os principais fatos que proporcionaram a evolução deste campo; na seqüência analisa-se uma taxonomia destes sistemas, proposta por estudiosos da área; duas tecnologias de sistema de recomendação mais específicas e de interesse, a filtragem colaborativa ou correlação pessoa-a-pessoa e a correlação item-a-item, são discutidas em seguida, quando são abordadas questões teóricas de seu funcionamento, juntamente com desafios e problemas encontrados. Por fim, são apresentados vários métodos de avaliação da qualidade deste tipo de aplicação.

3.2 Evolução dos Sistemas de Recomendação

Os primeiros tipos de sistemas de recomendação, baseados em filtragem colaborativa, nasceram de uma necessidade de gerenciar a grande carga de informação disponível na Internet. Diariamente os usuários de correio eletrônico têm suas caixas de e-mail inundadas por mensagens que nem sempre são de interesse. Grandes portais de

comércio eletrônico comercializam uma infinidade de produtos. As notícias são atualizadas a cada hora. A oferta de informações on-line aumenta cada vez mais.

Uma abordagem utilizada para solucionar este problema de sobrecarga de informações é a filtragem baseada em conteúdo. É possível, por exemplo, configurar um cliente de e-mail para não receber mensagens de determinado destinatário, ou procurar artigos filtrando alguns termos. Entretanto, esta técnica de filtragem, embora seja muito útil, apresenta algumas limitações (Shardanand, Maes, 1995):

- Os itens precisam ser automaticamente analisados – como textos – ou alguns atributos devem ser manualmente inseridos – é o caso de sons, fotografias, arte, vídeo ou itens físicos que não podem ser automaticamente analisados para a informação de atributos relevantes.
- Não possui a capacidade de gerar recomendações valiosas e que superem as expectativas do usuário.
- Não possui a capacidade de filtrar itens com base em qualidade, estilo e ponto de vista. Por exemplo, não é possível distinguir entre um artigo bem escrito e um mal escrito se ambos usam os mesmos termos.

As técnicas de filtragem colaborativa conseguiram suprir estas necessidades e acabaram se tornando as mais bem sucedidas das tecnologias de sistemas de recomendação. A filtragem colaborativa funciona construindo uma base de dados de preferências de itens (ou produtos) por usuários (ou clientes). Para se obter recomendações para uma pessoa, são comparadas suas características com as de outras pessoas na base de dados a fim de descobrirem-se **vizinhos**, ou seja, pessoas que tenham historicamente os mesmos gostos. Os produtos que os vizinhos gostam são recomendados para esta pessoa, porque provavelmente ela irá gostar também.

O sistema **Tapestry** (Goldberg et al, 1992) foi uma das primeiras implementações de um sistema de filtragem colaborativa. Foi seu autor, Goldberg, quem cunhou o termo filtragem colaborativa, definido por ele como uma atividade na qual as pessoas auxiliam umas às outras a filtrar informações através da anotação das suas reações a cada documento lido (basicamente se é interessante ou não). O princípio do Tapestry era possibilitar o acesso destas reações ou anotações a outros filtros. Ele foi

utilizado em *newsgroups* moderados: através desta prática, cada grupo poderia ter vários moderadores. Se, por exemplo, alguém quisesse ver o *newsgroup* como se ele fosse moderado por Maria, deveria simplesmente filtrar os artigos aprovados por ela. Além desta funcionalidade, o Tapestry oferecia muitos outros tipos de filtros e servia também como repositório de e-mails enviados no passado.

O Tapestry não era automático e exigia que os usuários construíssem *queries* complexas em uma linguagem especificamente projetada. Além disso, o usuário precisava conhecer as outras pessoas para saber quem tinha gostos semelhantes aos seus. No entanto, os sistemas de recomendação não poderiam ficar limitados a pequenas comunidades e depender de que as pessoas se conhecessem. A necessidade fez com que estes sistemas evoluíssem e assim uma série de sistemas de recomendação baseados em correlação de avaliações ou pontuações (*ratings*) foi desenvolvida.

O primeiro sistema de filtragem colaborativa automática foi introduzido pelo **GroupLens** (Konstan et al, 1997; Resnick et al, 1994), com o objetivo de predizer individualmente, ou seja, de forma personalizada, o interesse dos usuários sobre os artigos postados no Usenet *netnews*. Este sistema estendia o Tapestry, trazendo duas novas e importantes características: (1) sua arquitetura possibilitava que as avaliações fossem compartilhadas por vários sites e (2) implementava avaliações agregadas de vários usuários, com base na correlação de suas avaliações passadas, livrando desta forma o usuário de conhecer de antemão quais avaliações utilizar (Resnick et al, 1994).

Logo surgiram os sistemas de recomendações de música **Ringo (**Shardanand, Maes, 1995) e de vídeo **Bellcore** (Hill et al, 1995), semelhantes ao algoritmo utilizado pelo GroupLens. A abordagem utilizada nestes dois sistemas era designada por seus autores de *social information filtering*, ou filtragem de informação social, em vez de filtragem colaborativa. Entretanto, apesar da diferente denominação, ambas as técnicas seguiam os mesmos princípios.

Com a popularização da Internet, os usuários/clientes de cada site ou serviço passaram a ser contados em milhões e, juntamente com crescimento do catálogo de produtos, a matriz cliente-produto – fonte de dados valiosa para os algoritmos mais aprimorados – tornou-se cada vez mais esparsa, aumentando consideravelmente a complexidade dos cálculos. Conseqüentemente, surgiram novos desafios: aumentar a

escalabilidade dos sistemas e ao mesmo tempo a qualidade das recomendações. Estes desafios são conflitantes entre si, pois quanto menos tempo um algoritmo gasta para encontrar os vizinhos, mais escalável ele é, porém pior é a sua qualidade.

Resnick et al (1997) também apontou o problema de exigir do usuário avaliações – no caso, notas de 1 a 5 – para cada artigo lido no Usenet *netnews*. O modelo ideal sugere a utilização de avaliações implícitas, convertendo dados como o tempo que o usuário leva para ler um artigo, os livros emprestados de uma biblioteca, os produtos comprados em uma loja, as viagens realizadas através de uma agência, etc. em avaliações. Além de trazer benefícios para o cliente, a avaliação implícita também contribui para amenizar o problema da esparsidade da matriz cliente-produto, já que as avaliações são mais facilmente obtidas.

Pesquisadores do GroupLens propuseram a utilização de agentes do tipo *filter-bots* capazes de filtrar e avaliar artigos, para tentar melhorar a precisão das recomendações e diminuir o problema da esparsidade. Estes agentes eram integrados ao sistema como se fossem usuários comuns, mas com o diferencial de terem capacidade para avaliar todos os produtos – o que pode significar a leitura de milhares de mensagens por dia (Resnick et al, 1997) (Sarwar et al, 1998).

Outra abordagem adotada para tentar contornar o problema da esparsidade e que tem sido aceita por sua ampla aplicabilidade é a redução de dimensionalidade do conjunto de itens. Esta técnica, aplicada à representação da matriz cliente-produto, pode diminuir o tempo de cálculo das recomendações drasticamente e, da mesma forma, aumentar a escalabilidade dos sistemas. Entretanto, pesquisas precisam ser desenvolvidas para entender por que a redução de dimensionalidade funciona bem para algumas aplicações e não tão bem para outras (Sarwar et al, 2000).

Melhorias efetivas na precisão das predições dos sistemas de recomendação, baseados em filtragem colaborativa, foram apresentadas por Herlocker et al (1999) de forma empírica através de um *framework* de algoritmos.

Trabalhos sobre sistemas de recomendação no comércio eletrônico surgiram recentemente. Schafer, Konstan e Riedl (1999, 2001) apresentam uma explicação de como os sistemas de recomendação ajudam os sites de comércio eletrônico a aumentar as vendas e analisam seis sites que utilizam um ou mais sistemas de recomendação.

Com base nos exemplos, é definida uma taxonomia de sistemas de recomendação, incluindo as interfaces que eles apresentam para os clientes, as tecnologias utilizadas para criar as sugestões e as entradas que eles requerem dos usuários.

Sarwar et al (2000) mostra diferentes técnicas para os diferentes subprocessos dos sistemas de recomendação e compara qualidade e desempenho. Os problemas decorrentes do ambiente do *e-commerce* também são abordados.

3.3 Taxonomia

Esta seção tem como objetivo classificar os vários sistemas de recomendação existentes e auxiliar o entendimento deste vasto universo de aplicações. A taxonomia apresentada é baseada no trabalho de Schafer, Konstan e Riedl (2001).

O esquema na figura 3.1 apresenta, de forma simplificada, um sistema de recomendação, podendo ser observadas três categorias de atributos: entrada/saída funcional, método de recomendação e outros aspectos de projeto (apresentação e grau de personalização).

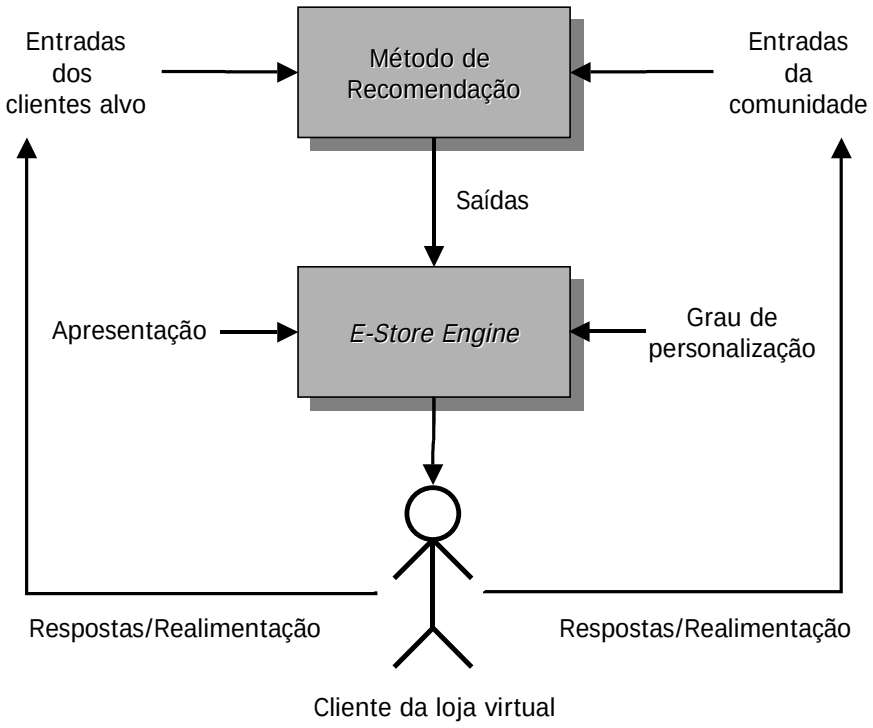


Figura 3.1 – Esquema de funcionamento de um site utilizando um sistema de recomendação

As aplicações de recomendação combinam as entradas do cliente em questão com outras informações sobre produtos e comunidades de usuários para gerar recomendações. Os sites definem o nível de personalização e o modo de apresentação para produzir pacotes de recomendações específicos. A resposta a estas recomendações pode gerar entradas adicionais para futuras recomendações.

3.3.1 Entradas e Saídas Funcionais

Um sistema de recomendação utiliza vários tipos de dados de entrada, tais como perfil do consumidor, dados sobre produtos e informações provenientes da comunidade de usuários, para gerar a saída, que são as recomendações. A seguir são apresentadas as várias dimensões destas entradas e saídas.

Entradas do cliente-alvo

As entradas do cliente-alvo no sistema de recomendação são os dados que permitem produzir recomendações personalizadas. Uma aplicação que não faz uso deste tipo de dados gera somente recomendações não-personalizadas.

Uma das formas de categorizar os dados de entrada do cliente-alvo é em relação à maneira com que os dados são fornecidos. Pode ser através de:

- **Navegação implícita**. Muitas aplicações utilizam os **dados de navegação** da sessão atual para produzir ou refinar recomendações. Os dados são inferidos do comportamento do cliente e alimentam o sistema sem que ele perceba. Estes dados podem ser de um produto específico que o cliente esteja vendo em uma determinada página ou de produtos que estejam na cesta de compras.
- **Navegação explícita**. Neste caso, o site fornece uma lista de itens na forma de links, sobre a qual os clientes navegam de forma intencional de acordo com seus interesses, a fim de informar ao sistema de recomendação suas preferências.
- **Palavras-chave e atributos de itens**. Em alguns casos, as entradas não podem ficar limitadas às categorias ou características dos produtos. Neste

tipo de entrada, palavras-chave ou atributos de itens que foram obtidos explícita ou implicitamente, derivados dos itens sendo observados, são interpretados como entrada do sistema de recomendação.

- **Avaliações**. Este é o caso em que o cliente avalia itens que ele já conhece ou possui, de forma explícita, dentro de uma escala numérica (“avalie este item numa escala de 1 a 5”) ou binária (“você gostou deste item?”). Neste tipo de situação o cliente sente que não está navegando, nem procurando, mas sim configurando suas preferências para obter um serviço mais personalizado.
- **Histórico de compra.** É uma outra forma de obter dados de forma implícita. Os produtos comprados expressam preferências concretas dos clientes.

Entradas da Comunidade

As entradas da comunidade denotam como os múltiplos indivíduos ou a comunidade por completo percebem os itens. Entradas que refletem a opinião da comunidade:

- **Atributos de item**. É a associação de rótulos e categorias a itens, como por exemplo, categorias de livros que refletem um consenso em toda a sociedade.
- **Popularidade de item externo**. Similar aos atributos de item, porém reflete a opinião da sociedade de forma mais ampla, como listas nacionais de itens mais vendidos. Podem ser listas de recomendação manualmente selecionadas.
- **Histórico de compra**. Como no cliente-alvo, o histórico de compras pode ser empregado como um conjunto de avaliações implícitas sobre produtos. Estes dados são utilizados para gerar listas de produtos mais vendidos em um site particular ou submetidos a processos de *data mining* para descobrir similaridades e tendências de compra.

Entradas associadas a membros individuais da comunidade:

- **Avaliações**. Também como os clientes alvo, a comunidade é incentivada a avaliar os produtos através de uma escala de pontos. Estes dados são então aplicados na geração de recomendações.
- **Comentários textuais**. Há sites que encorajam seus clientes a darem sua opinião sobre os produtos. Mais tarde estes comentários são disponibilizados para outros clientes, como um meio de facilitar o processo de tomada de decisão.

A origem de todos estes dados de entrada é muito diversa. A maioria dos sites pesquisados por Schafer, Konstan e Riedl (2001) utiliza informações sobre seus clientes obtidas diretamente das páginas do próprio site, combinadas com dados de pesquisa de seus produtos. Os atributos de itens são geralmente obtidos através de serviços que publicam catálogos digitais compreendendo categorização e descrição dos produtos. Estes dados de terceiros são comumente complementados com uma pequena quantidade de dados específicos do site. Os itens externos são quase sempre largamente pesquisados para obter uma medida ampla do interesse do consumidor. As informações de compra da comunidade são sempre próprias do site, da mesma forma que são primariamente os comentários textuais e as avaliações.

Todas estas informações poderiam ser unificadas e compartilhadas entre os sites, aumentando assim a eficiência dos sistemas de recomendação. Entretanto, ainda não são conhecidos exemplos de organizações que façam isso. Existem também dificuldades que impedem esta unificação, como o desafio da integração de informações e questões de ética e privacidade.

Saídas

As saídas de recomendações de itens específicos variam em tipo, quantidade e forma como são apresentadas para o cliente. A mais comum é a **sugestão**, que geralmente aparece na forma de “experimente isto” ou simplesmente colocando o item na página web que está sendo visitada.

A forma mais simples de apresentação é expor individualmente o item, que tem a vantagem de chamar mais a atenção do cliente e toma pouco tempo para ser

processada. Por outro lado, há casos em que é arriscado utilizar somente um item, pois o cliente pode já possuí-lo.

Mais comumente, os sistemas de recomendação oferecem um conjunto de sugestões para o cliente em um determinado contexto. Algumas aplicações apresentam uma lista não ordenada ou em ordem alfabética, para evitar a rejeição de toda a lista caso o primeiro produto seja desinteressante e também para evitar dar a impressão de que a primeira opção é a melhor de todas (embora toda a lista gerada por um sistema de recomendação possua uma ordem). Outras aplicações preferem mostrar a lista ordenada, já que esta forma oferece uma informação extra e pode auxiliar o cliente na escolha.

Muitos sistemas que utilizam avaliações explícitas apresentam **predições** das avaliações que o cliente faria sobre determinados itens. Estas estimativas podem ser personalizadas para consumidores individuais ou não-personalizadas para membros da comunidade. De forma geral, as predições podem auxiliar o cliente a entender o valor da recomendação, já que representam a avaliação que o cliente faria sobre itens específicos.

Em comunidades pequenas, nas quais os membros são conhecidos entre si, pode ser útil mostrar as avaliações individuais dos membros da comunidade, permitindo que o próprio cliente tire conclusões sobre o peso da recomendação. Esta técnica também tem seu valor quando as avaliações são acompanhadas de críticas (*reviews*), que podem explicar porque determinado item foi favorecido ou desfavorecido. As críticas podem ser de grande ajuda no processo de decisão de compra, porém, como os sistemas não distinguem comentários positivos de negativos, os sistemas que utilizam esta técnica requerem que juntamente com a crítica seja feita uma avaliação numérica do item. Atualmente, os sites de comércio eletrônico não têm apresentado estas críticas de forma personalizada, embora isto possa ser feito de forma relativamente fácil, verificando a concordância histórica do cliente com a pessoa que escreveu a crítica. Outra forma seria permitir que os clientes avaliassem a crítica. Ainda que interessantes, não existem por enquanto tais mecanismos.

3.3.2 Métodos de Recomendação

A seguir é apresentada uma visão geral dos vários processos utilizados pelos atuais sistemas de recomendação de comércio eletrônico. É interessante observar que muitos sistemas combinam mais de um método, assim como podem combinar entradas e saídas. Cada método compreende uma família de algoritmos e abordagens, alguns deles são mostrados mais adiante neste trabalho.

Consulta (*Raw Retrieval*)

Consiste em uma interface de procura, através da qual os clientes podem consultar uma base de dados de itens. Neste caso a recomendação é um processo binário e sintático no qual o sistema apresenta o que foi requisitado. Embora tecnicamente não seja uma aplicação de recomendação, este método pode parecer como tal para o cliente. Ao pedir livros de um determinado autor, por exemplo, o sistema pode retornar livros que o cliente ainda não conhecia. Esta forma de recomendação é altamente difundida nas aplicações de comércio eletrônico.

Seleção Manual

São aplicações que valorizam personalidades. Os itens são manualmente selecionados por editores, artistas, críticos e outros especialistas. Estes “recomendadores humanos” identificam itens baseados em seus próprios gostos, interesses e objetivos e criam uma lista de itens recomendados, disponível para membros da comunidade. Comentários geralmente acompanham a avaliação e ajudam os clientes a entenderem a recomendação. Este método é geralmente utilizado em sites de filmes, músicas etc. Alguns sites permitem que membros da comunidade publiquem suas próprias listas de recomendação.

Resumo Estatístico

Em alguns casos, quando a personalização é impraticável ou desnecessária, as aplicações de recomendação podem fornecer resumos estatísticos da opinião da comunidade. Estes resumos podem conter medidas de popularidade internas (por exemplo, percentual de pessoas que compraram ou gostaram de um item), como também resumos de avaliações (por exemplo, número de pessoas que recomendam ou

avaliação média de um item). Um exemplo de site que utiliza este tipo de aplicação é o leilão virtual *eBay*, que fornece avaliações médias de compradores e vendedores. Este método de recomendação é muito popular pois as recomendações são facilmente computadas e podem ser utilizadas em ambientes não personalizados, como em lojas físicas.

Baseado em Atributos

Este tipo de aplicação utiliza perfis de clientes baseados em atributos para recomendar produtos. Por exemplo, uma pessoa que esteja navegando na seção de música popular brasileira de uma loja de CDs e que possua vários outros discos que estão em oferta especial na sua cesta de compras pode receber recomendações de discos de música popular brasileira que estejam em promoção.

Correlação Item-a-Item

As aplicações que utilizam a correlação item-a-item são capazes de identificar itens freqüentemente encontrados em associação com itens que o cliente tenha demonstrado interesse. Na sua forma mais simples, este método pode ser usado para identificar itens correlacionados com um único item (por exemplo, uma outra roupa comumente comprada com uma determinada calça). Sistemas mais poderosos relacionam todos os itens que o cliente tem na cesta de compras para obter recomendações mais apropriadas.

A correlação item-a-item geralmente utiliza somente informações da sessão atual de compra. Por exemplo, se um cliente de um supermercado virtual colocou na cesta de compra produtos como carvão, carne e sal grosso, a aplicação de correlação item-a-item pode recomendar, baseada em padrões de compra semelhantes (como as que incluem ingredientes para churrasco), a compra de cerveja, refrigerante e farofa. Outra aplicação para este tipo de sistema é a recomendação de presentes. O cliente precisa apenas eleger alguns itens que a outra pessoa gosta para receber as recomendações destinadas a ela. Esta técnica de recomendação será detalhada mais adiante, no item 3.5 deste trabalho.

Correlação Pessoa-a-Pessoa

Os sistemas que fazem uso da correlação pessoa-a-pessoa recomendam produtos com base na correlação entre o cliente-alvo e os outros clientes que compraram

produtos no site. Esta tecnologia também é conhecida como filtragem colaborativa. Para gerar uma recomendação para uma pessoa, digamos Maria, o sistema identifica uma comunidade de clientes que tenha o mesmo comportamento de compra ou o mesmo perfil que Maria. O princípio desta técnica baseia-se no fato de que, se vários membros desta comunidade compraram o último CD da Madonna, é muito provável que Maria irá gostar e poderá comprá-lo também. Este método será explicado com mais detalhes adiante, no item 3.4.

3.3.3 Outros Aspectos de Projeto

Grau de Personalização

As recomendações produzidas por um sistema podem ter diferentes graus de personalização. Dois dos fatores que determinam este nível são a **precisão**, que determina quão correto é o sistema, e a **utilidade** da recomendação. A utilidade inclui questões como a capacidade do sistema dede gerar recomendações valiosas e inesperadas e a possibilidade de produzir recomendações distintas para pessoas diferentes. Uma aplicação que é capaz de recomendar livros muito particulares é muito mais interessante que uma aplicação que só recomenda best-sellers. Porém um sistema que produz recomendações muito particulares, mas raramente está correto, não será usado por muito tempo.

As informações que o site possui sobre o cliente podem influenciar o nível de personalização oferecido. Na primeira compra que um cliente faz são poucos os dados que se tem sobre o seu perfil, conseqüentemente pode ser interessante oferecer produtos com alta probabilidade de resposta da população em geral. Mas para clientes antigos e leais, dos quais foi possível armazenar uma grande quantidade de dados e modelar um perfil, entender suas exatas necessidades pode ser crucial (Kitts, 2000).

De acordo com estes fatores, três níveis de personalização podem ser identificados:

- **Não-personalizado**. As recomendações estão baseadas em métodos como seleção manual, resumos estatísticos, entre outros. Muitos dos exemplos de

recomendações no comércio eletrônico são não-personalizadas, sites deste tipo apresentam as mesmas sugestões para todos os clientes.

- **Efêmero**. Aplicações que fazem uso apenas das informações instantâneas proporcionam personalização efêmera. Dentro deste grau pode-se ter maior ou menor personalização de acordo com o tipo de dado utilizado. Sistemas que utilizam informações sobre toda a sessão atual produzem recomendações muito mais personalizadas que sistemas que utilizam apenas informações da página sendo visitada. Recomendações efêmeras estão geralmente apoiadas em métodos de correlação item-a-item, baseados em atributos ou ambos.

- **Persistente**. Os sistemas que oferecem recomendações altamente personalizadas pertencem a este grau. Mesmo que dois clientes estejam vendo o mesmo produto, o sistema pode gerar recomendações distintas. Este tipo de aplicação necessita de uma grande quantidade de informações, como o histórico de compras, e requer que o cliente tenha uma identidade persistente. Os métodos de correlação usuário-a-usuário, recomendação baseada em atributos usando preferências de atributos persistentes ou correlação item-a-item baseada em preferências de itens persistentes são empregados.

Apresentação

Apresentar as recomendações da maneira adequada para cada cliente é uma decisão crítica nos sistemas de recomendação de comércio eletrônico, da mesma forma como é no marketing tradicional.

Modos de apresentação das recomendações podem ser classificados em três grupos:

- **Baseados em tecnologia *push***. Nas aplicações de comércio eletrônico a tecnologia *push* mais comum é o e-mail. O envio de recomendações personalizadas, com ofertas especiais, pode fazer o cliente retornar à loja virtual.

- **Baseados em tecnologia *pull***. Aplicações que trabalham neste modo podem controlar quando as recomendações são mostradas. Uma característica

importante é que o cliente está sempre consciente de que recomendações personalizadas estão disponíveis, mas elas só são efetivamente apresentadas quando forem explicitamente requisitadas. As primeiras aplicações trabalhavam neste modo pois a computação das recomendações era cara e poderia tornar o site lento. Hoje em dia, a tecnologia *pull* é uma opção para aplicações em que alguns tipos de recomendações são considerados periféricos (por exemplo as recomendações de presentes) ao invés de integrados à aplicação.

- **Apresentação passiva**. Também chamada de "recomendação orgânica", a apresentação passiva mostra as recomendações em um contexto natural da aplicação de comércio eletrônico. As sugestões podem aparecer, por exemplo, no momento que um item relacionado está sendo visto. A vantagem deste tipo de abordagem é que as recomendações surgem no momento em que o cliente ainda está receptivo à idéia. Uma desvantagem é que elas podem não ser identificadas como sugestões personalizadas, embora não haja estudos que comprovem que a percepção explícita possa tornar mais efetivo o processo de compra.

Nas primeiras aplicações, a tecnologia *pull* era enfatizada para mostrar para o cliente que ele recebia uma atenção especial. Recentemente, a tendência é pela apresentação das recomendações de modo passivo no web site, em conjunto com técnicas *push* para promover o retorno dos clientes à loja.

3.4 Correlação Pessoa-a-Pessoa

A correlação pessoa-a-pessoa ou usuário-a-usuário, também conhecida como filtragem colaborativa, tem sido a tecnologia mais bem sucedida e utilizada pelos melhores sistemas de recomendação na web. Neste tipo de sistema, as recomendações para um determinado cliente são baseadas na opinião dos outros clientes.

Existem várias classes de algoritmos de filtragem colaborativa, entre eles estão os métodos baseados em vizinhança, redes Bayesianas (Breese, 1998), decomposição de

valor singular com classificação de rede neural (Billsus, 1998) e aprendizado de regra de indução (Basu, 1998).

Os algoritmos prevalentes são os **baseados em vizinhança**. Técnicas estatísticas são empregadas para encontrar um conjunto de clientes chamados de **vizinhos**, pessoas que possuem as mesmas preferências ou comportamento de compra. Uma vez formada a vizinhança, as recomendações podem ser produzidas por uma série de algoritmos.

3.4.1 O Processo de Obtenção de Recomendações

O processo de recomendação por filtragem colaborativa pode ser dividido em três partes (Sarwar et al, 2000): (1) representação, (2) formação da vizinhança e (3) geração da recomendação, conforme ilustrado na figura 3.2. A primeira parte, a **representação**, trata do esquema utilizado para modelar os produtos que já foram comprados pelo cliente. A **formação da vizinhança** diz respeito à tarefa de como encontrar os vizinhos. A última parte, **geração da recomendação**, trata do problema de identificar os produtos mais recomendados para um cliente a partir da sua vizinhança.

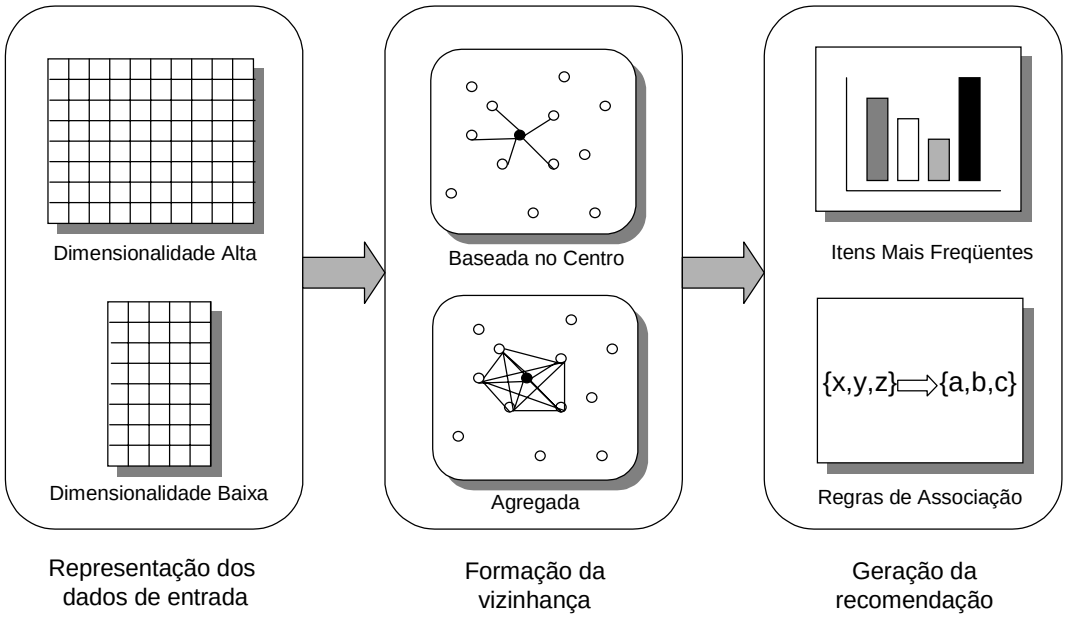


Figura 3.2 – As três partes principais de um Sistema de Recomendação

Fonte: Sarwar et al, 2000 (tradução minha)

3.4.1.1 Representação

A entrada de um típico sistema de recomendação baseado em FC é uma matriz $m \times n$, onde m representa os clientes e n representa os produtos comprados por cada cliente numa perspectiva histórica. Nesta matriz cliente-produto R, o elemento $r_{i,j}$ é igual a um se o i-ésimo cliente comprou o j-ésimo produto, ou igual a zero, caso o cliente i não tenha comprado o produto j. Outra possibilidade é normalizar a linha para que fique com a dimensão de uma unidade (ou outra norma), diferenciando assim clientes que compram poucos produtos de clientes que compram mais produtos. Ainda em outro contexto, o elemento $r_{i,j}$ poderia também representar a avaliação de um cliente i a respeito do produto j. A tabela 3.1 mostra um exemplo simplificado de uma matriz cliente-produto.

Tabela 3.1 – Matriz cliente-produto

	O Mundo de Sofia	**Os Catadores de Conchas**	**3001: A Odisséia Final**	**Sistemas Operacionais**	**Redes de Computadores**
João	1	1	1		1
Maria	1	1			
Sílvia			1	1	1
Luiz	1			1	1

Esta representação, também chamada de **representação original,** embora muito simples, impõe alguns problemas para os sistemas de recomendação:

- **Esparsidade**: nas aplicações comerciais dos sistemas de recomendação, geralmente as bases de dados são muito grandes. Hoje em dia há lojas virtuais que possuem milhares de clientes e comercializam outros milhares de produtos, tal como a Amazon.com. Nestes casos, mesmo os clientes mais ativos compram em torno de 1% de todos os produtos disponíveis e o sistema de recomendação pode não ser capaz de fazer predições. Este problema é conhecido como **cobertura reduzida** (ou *reduced coverage*). Além disso, a precisão pode ser baixa pela insuficiência da quantidade de dados.

- **Escalabilidade**: quanto maior o número de clientes e produtos, maior é o esforço computacional para gerar as predições. Como na web estes números podem ser gigantescos, a escalabilidade é um sério problema para os algoritmos de filtragem colaborativa.
- **Sinonímia**: este problema está relacionado à dificuldade dos sistemas em associar produtos com nome diferentes, mas que sejam similares.

Estes problemas originaram vários estudos explorando métodos alternativos de representação para que os sistemas de recomendação pudessem ser viáveis num cenário como a web. Entre estes estudos está a **representação dimensional reduzida**.

Esta representação usa LSI – *Latent Semantic Indexing*, para obter uma aproximação da matriz original, através de decomposição por valor singular truncada, transformando a matriz original $n \times m$ em uma matriz $n \times k$. Nesta nova matriz o problema de esparsidade é resolvido, pois todas as entradas são diferentes de zero, o que quer dizer que todos os n clientes têm opinião sobre todos os k meta-produtos. Esta abordagem também diminui o problema da escalabilidade, pois k é um número muito menor que n e o tempo de processamento e os requisitos de armazenamento caem drasticamente. Na redução do espaço, esta representação captura a associação entre produtos e clientes, resolvendo assim o problema da sinonímia.

3.4.1.2 Formação da Vizinhança

Este pode ser considerado o passo mais importante em um sistema de recomendação baseado em filtragem colaborativa. Consiste em encontrar, para cada cliente u, uma lista ordenada de l clientes $N=\{N_1,N_2,...,N_l\}$ tal que $u \notin N$ e $sim(u, N_1)$ é máxima, $sim(u, N_2)$ é o próximo máximo e assim por diante.

O algoritmo de formação da vizinhança pode ser dividido em duas partes: (1) ponderar os possíveis vizinhos ou calcular a medida de proximidade e (2) selecionar a vizinhança. Estas tarefas podem ser computadas de diversas formas. Algumas são apresentadas a seguir.

Medidas de Proximidade

- *Correlação de Pearson*: a proximidade entre dois clientes *a* e *b* é medida calculando-se a correlação de Pearson, $corr_{ab}$, dada por:

$$corr_{ab} = \frac{\sum_i (r_{ai} - \bar{r}_a)(r_{bi} - \bar{r}_b)}{\sqrt{\sum_i (r_{ai} - \bar{r}_a)^2 \sum_i (r_{bi} - \bar{r}_b)^2}} . \quad (1)$$

- *Correlação de Spearman*: é similar à correlação de Pearson, mas calcula uma medida de correlação entre classes (*ranks*) em vez de valores de avaliação (*rating values*). A proximidade entre dois clientes *a* e *b* é medida calculando-se a correlação de *Spearman*, $w_{a,b}$, dada por:

$$corr_{ab} = \frac{\sum_{i=1}^{m} (rank_{a,i} - \overline{rank}_a) * (rank_{b,i} - \overline{rank}_b)}{\sqrt{\sum_i (rank_{a,i} - \overline{rank}_a)^2 \sum_i (rank_{b,i} - \overline{rank}_b)^2}} . \quad (2)$$

- *Cosseno:* neste caso, dois clientes *a* e *b* são considerados como dois vetores no espaço-produto *m* dimensional (ou o espaço *k*-dimensional no caso de representação reduzida). A proximidade é medida calculando-se o cosseno do ângulo entre dois vetores, que é dado por:

$$\cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|_2 * \|\vec{b}\|_2} , \quad (3)$$

onde "." denota o produto interno (*dot-product*) dos dois vetores.

Nas fórmulas apresentadas acima, $\bar{r}$ denota a média das avaliações sobre os itens do cliente. A correlação é calculada somente sobre os itens que ambos usuários *a* e *b* avaliaram. O resultado da medida de proximidade para os *n* clientes é uma matriz de similaridade S ($n \times n$).

Herlocker et al (1999) faz as seguintes recomendações:

- se a escala de avaliação for um pequeno número de classes discretas (por exemplo 1–5, 1–7, ou 1–20), deve-se usar a correlação de Spearman;
- se a escala de avaliação não for discreta, mas contínua, deve-se considerar a correlação de Pearson;

- se a escala for binária ou unária, deve-se considerar uma abordagem diferente (apud Breese, Heckerman e Kadie, 1998).

Herlocker et al (1999) também demonstrou que, além destas medidas de proximidade ou similaridade, pode-se melhorar a ponderação sobre os vizinhos, medindo também **significância** e **variância**.

Na ponderação da significância, a medida de similaridade é desvalorizada quando baseada em um pequeno número de itens avaliados. Nos experimentos de Herlocker et al (1999) se dois usuários tinham menos de 50 itens avaliados em comum, um peso de significância de *n*/50 era aplicado, onde *n* era o número de itens avaliados. Se o número de itens fosse maior que 50, uma significância de peso 1 era aplicada. Empregando esta metodologia, foi obtido um considerável aumento de precisão.

Na ponderação de variância, foi hipotetizado que os itens mais distinguidos influenciavam a similaridade, melhorando a precisão das predições. Para incluir esta medida, a correlação de Pearson citada acima foi modificada para incorporar um fator de variância de itens. Porém, isto não surtiu efeito significante nos resultados (os testes foram realizados sobre uma base de dados de filmes).

Seleção da Vizinhança

É importante, tanto em termos de precisão quanto em termos de performance, selecionar apenas uma parte dos clientes (ou vizinhança) em vez de toda a base de dados para computar uma predição (Shardanand, Maes, 1995). Herlocker et al (1999) demonstrou como o erro médio absoluto das predições aumenta quando o tamanho da vizinhança é elevado. Além disso, os sistemas de filtragem colaborativa comerciais estão começando a manipular milhões de clientes, exigindo assim mais agilidade na geração das recomendações.

O tamanho da vizinhança pode ser determinado de várias maneiras e as mais utilizadas são **limiar de correlação** (*correlation-thresholding*) e **melhores *n* vizinhos**.

O limiar de correlação utilizado por Shardanand e Maes (1995) fixa um valor de correlação e seleciona todos os clientes que possuem correlação superior a este limite. Em seus experimentos, Herlocker et al (1999), observou que esta técnica não difere muito de selecionar toda a base de dados. Quando o valor fixado é muito alto a

vizinhança pode ser insuficiente para gerar a predição e do contrário, quando o valor for muito baixo, o propósito do limiar é anulado.

O segundo método, melhores *n* vizinhos, consiste em selecionar as *n* melhores correlações para um dado *n*. Ainda considerando o estudo de Herlocker et al (1999), o tamanho da vizinhança determinado desta maneira apresentou melhores resultados, embora o valor de *n* possa significativamente alterar o resultado.

Quando os algoritmos são configurados para dar maior cobertura, em ambas as técnicas as contribuições das poucas altas correlações tendem a se perder no ruído das correlações mais baixas. Entretanto, Sarwar et al (2000) em sua análise de algoritmos de recomendação para *e-commerce*, observou que em geral a qualidade aumenta quando um número maior de vizinhos é considerado, mas ressaltou que o número ótimo depende da base de dados utilizada.

Tipos de Vizinhança:

- **Baseada no centro**: forma uma vizinhança de tamanho *k*, para um cliente particular *c*, através da seleção dos *l* outros clientes mais próximos;
- **Agregada**: forma uma vizinhança de tamanho *l*, para um cliente *c*, primeiro selecionando o vizinho mais próximo de *c*. Então, os *l*-1 clientes restantes são selecionados como segue. Seja que em um certo ponto existem *j* vizinhos na vizinhança *N*, onde $j < l$. O algoritmo calcula o centróide da vizinhança. O centróide de *N* é definido como $\vec{C}$ e é calculado como $\vec{C} = \frac{1}{j}\sum_{\vec{V} \in N} \vec{V}$. Um cliente *w*, tal que $w \notin N$ é selecionado como o vizinho j+1º somente se *w* é o mais próximo do centróide $\vec{C}$. Então o centróide é recalculado para os j+1 vizinhos e o processo continua até $|N|=l$. Este esquema é benéfico para dados esparsos porque os vizinhos mais próximos afetam a formação da vizinhança.

3.4.1.3 Geração da Recomendação

A etapa final de um sistema de recomendação baseado em filtragem colaborativa é gerar as top-N recomendações da vizinhança dos clientes. Novamente, existem várias técnicas que podem ser utilizadas, entre elas pode-se citar:

- **Recomendação do item mais freqüente**: os dados de compra dos *N* vizinhos são analisados, fazendo-se uma contagem dos produtos. Depois de feita a contagem, o sistema ordena a lista de produtos por ordem dos mais freqüentes e retorna os *N* melhores produtos que ainda não foram comprados pelo cliente em questão.
- **Recomendação baseada em regra de associação**: utiliza os *l* vizinhos para gerar regras de associação. Considerada uma técnica tradicional de *data mining*, encontrar regras de associação consiste em descobrir associação entre dois conjuntos de produtos tal que a presença de alguns produtos em uma transação implique que produtos de um outro conjunto estejam também presentes na mesma transação. Já se observou que, ao considerar poucos vizinhos, pode não ser possível gerar regras de associação fortes suficientes e, como conseqüência, obter poucos produtos para recomendar. Neste caso, o algoritmo do item mais freqüente pode ser utilizado para gerar o resto das combinações se for necessário.

3.4.2 Problemas

Existe uma série de problemas que pesquisadores e desenvolvedores de aplicações comerciais têm identificado nos sistemas de recomendação baseados em filtragem colaborativa, principalmente quanto não são utilizadas avaliações implícitas. Sarwar et al (1998) citou alguns:

- **O problema dos primeiros avaliadores**. Os novos itens precisam ser avaliados para que possam ser recomendados. As primeiras pessoas que avaliam, não recebem valor em troca. Em alguns casos pode-se considerar a idéia de pagamentos ou outros tipos de benefícios para encorajar as primeiras avaliações.
- **O problema da esparsidade**. O aumento do número de usuários e de itens gera o problema da esparsidade. Pode-se estimar que poucas pessoas terão lido e formado opinião sobre 1/10 de 1% dos mais de dois milhões de livros disponíveis nas maiores livrarias. Por um lado a esparsidade é uma das razões de ser da filtragem colaborativa: a maioria das pessoas não está

interessada na maior parte dos livros. Por outro lado a esparsidade impõe um desafio computacional. Os esforços realizados para combater este problema, ainda segundo Sarwar et al (1998) incluem:

- *Particionamento*: o projeto de pesquisa GroupLens mostrou que o particionamento da base de dados de avaliações, já que nem todas as pessoas estão inscritas em todos os grupos, melhora um pouco a precisão das recomendações e a densidade (Konstan et al, 1997). Entretanto não resolve o problema.
- *Redução da dimensionalidade*: vários pesquisadores têm investigado técnicas estatísticas para comprimir a dimensionalidade da base de dados. Embora pareçam promissoras, estas técnicas ainda não foram demonstradas na resolução do problema da esparsidade.
- *Avaliações implícitas*: vários sistemas tentam aumentar o número de avaliações observando o comportamento do usuário.

Sarwar et al (1998) propôs uma nova abordagem para aumentar o número de avaliações e diminuir a esparsidade no sistema GroupLens (de recomendação de artigos para *newsgroup*): a utilização de agentes de filtragem semi-inteligentes. Nesta técnica, os agentes são incluídos no sistema como se fossem usuários comuns, para que suas avaliações ajudem a gerar as recomendações. Três tipos de agentes foram implementados:

- **SpellCheckerBot**: avalia artigos com base na proporção de erros de ortografia encontrados;
- **IncludedMsgBot**: avalia artigos com base no percentual de texto citado de outros artigos. Artigos com muito texto citado e pouco novo conteúdo recebem uma avaliação baixa;
- **LengthBot**: avalia artigos convertendo a quantidade de palavras para a escala de avaliação. Baseia-se na hipótese de que os usuários de *newgroup* preferem mensagens curtas a mensagens longas.

3.5 Correlação Item-a-Item

O intenso crescimento do número de clientes nos sites de *e-commerce* impôs o desafio da escalabilidade aos métodos de correlação pessoa-a-pessoa. A complexidade destes métodos aumenta linearmente com o número de clientes, que em uma aplicação típica pode chegar aos milhões. Visando contornar o problema da escalabilidade, começaram a ser desenvolvidas técnicas de correlação baseada em itens, que buscam identificar relações entre os diferentes itens e usar estas relações para obter uma lista de recomendações (Karypis, 2000).

Recentes pesquisas demonstraram a eficiência desta técnica, mesmo comparada à correlação pessoa-a-pessoa (Karypis, 2000; Sarwar et al, 2001). O gráfico da figura 3.3, obtida do estudo de Karypis, mostra o resultado alcançado por diferentes algoritmos na correlação item-a-item relativo ao resultado alcançado pelos algoritmos na filtragem colaborativa. Os algoritmos “cosseno” e “C-Prob” são descritos a seguir, já o algoritmo “Frequent”, recomenda os itens mais frequentes ainda não presentes no conjunto de itens do cliente ativo. Os termos “*ecommerce*”, “*catalog*”, “*ccard*”, “*skills*” e “*movielens*” representam as diferentes bases de dados utilizadas na avaliação e indicam, respectivamente, compras através de um site de comércio eletrônico, compras através de catálogo, compras através de cartão de crédito de uma loja de departamentos, habilidades presentes nos currículos de vários candidatos de um portal de empregos e avaliações de filmes obtidas do projeto de pesquisa “MovieLens”.

O trabalho de Sarwar citado acima também contém uma comparação entre os métodos de filtragem colaborativa e correlação item-a-item. Entretanto, como a base de dados e a abordagem da pesquisa diferem bastante deste, o mesmo não foi aqui incluído. Os resultados e a descrição completa do trabalho podem ser encontrados em (Sarwar et al, 2001).

Para que sejam mais eficientes, os métodos de correlação baseados em item exploram uma característica própria dos produtos: na maioria dos casos a relação entre eles é estática, ao contrário dos clientes, em que a compra de um produto particular pode mudar totalmente a “vizinhança”. Isto permite que a correlação entre eles seja determinada off-line, mantendo on-line apenas a operação de obtenção das recomendações, tornando muito menor o esforço computacional.

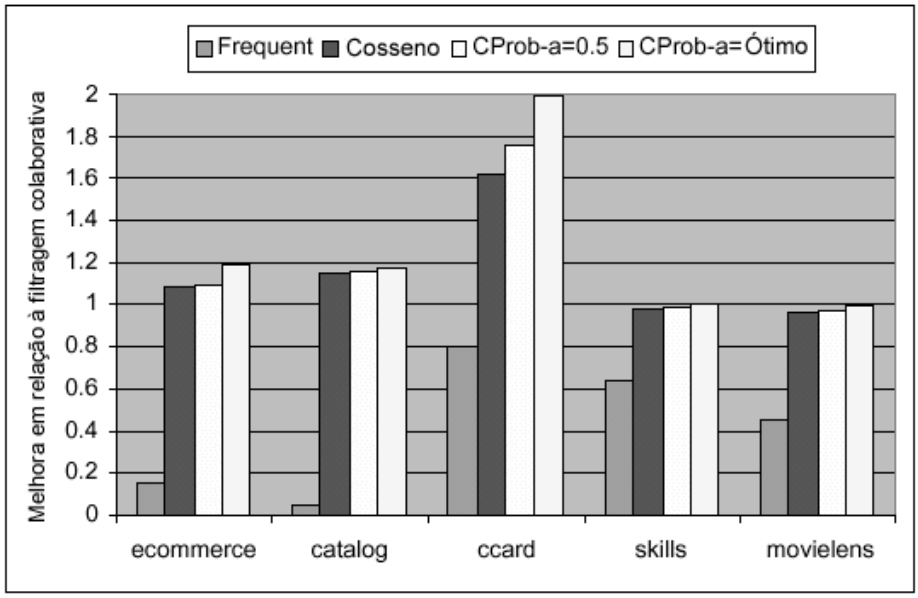


Figura 3.3 – Resultado obtido pela correlação baseada em itens relativo ao resultado obtido pela filtragem colaborativa

Fonte: Karypis, 2000, pg. 11 (tradução minha)

3.5.1 O Processo de Obtenção de Recomendações

A seguir é apresentado um processo de recomendação que utiliza a similaridade item-a-item para calcular as relações entre os produtos.

Da mesma forma como na correlação por pessoa, o processo de recomendação por correlação item-a-item pode ser dividido em três partes: (1) representação, (2) cálculo de similaridade e (3) geração da recomendação.

3.5.1.1 Representação

A representação dos dados de entrada é a mesma utilizada na correlação pessoa-a-pessoa, ou seja, uma matriz $m \times n$, onde m representa os clientes e n representa os produtos comprados por cada cliente.

3.5.1.2 Cálculo de Similaridade

O cálculo da similaridade é um processo crítico pois pode determinar a qualidade do resultado. Existem várias formas de fazê-lo, entre elas podemos citar a **similaridade baseada em cosseno**, a **similaridade baseada em correlação** e a

similaridade baseada em probabilidade condicional. Estes três modos de cálculo são descritos a seguir.

Similaridade Baseada em Cosseno

Neste caso, os itens ou produtos são considerados como vetores no espaço *m* dimensional dos clientes. A similaridade entre dois itens é medida calculando-se o cosseno do ângulo entre os dois vetores correspondentes. A equação utilizada é a mesma utilizada na filtragem colaborativa, ou seja, a similaridade entre os itens *i* e *j* é dada por:

$$sim(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\left\|\vec{i}\right\|_2 * \left\|\vec{j}\right\|_2}, \quad (4)$$

onde "." denota o produto interno dos dois vetores.

Por esta equação constata-se que a similaridade entre dois itens será grande se cada cliente que compra um dos itens compra o outro também. Além disso, uma característica importante da similaridade baseada em cosseno é que a freqüência de compra dos diferentes itens é levada em conta (devido ao denominador da equação), fazendo com que itens freqüentemente comprados sejam similares a itens freqüentemente comprados e não a itens com pouca freqüência de compra e vice-versa. Isto tende a eliminar recomendações óbvias, já que itens freqüentes somente serão recomendados se houverem outros itens similares na cesta de compras (Karypis, 2000).

Similaridade Baseada em Correlação

Neste caso, a similaridade entre dois itens *i* e *j* é medida calculando-se a correlação de Pearson, como visto anteriormente na correlação pessoa-a-pessoa. O cálculo leva em conta somente os clientes que avaliaram ambos os itens *i* e *j*. Sendo *U* o conjunto destes clientes, a similaridade de correlação é dada por:

$$sim(i, j) = \frac{\sum_{u \neq U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \neq U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \neq U} (r_{u,j} - \bar{r}_j)^2}}, \quad (5)$$

onde $r_{u,i}$ é a avaliação do cliente *u* sobre o item *i* e $\bar{r}_i$ é a avaliação média do *i*-ésimo item.

Similaridade Baseada em Probabilidade Condicional

Neste caso a similaridade é obtida usando-se uma medida baseada na probabilidade condicional de comprar um dos itens dado que outros itens foram comprados. A probabilidade de comprar o item *i*, dado que *j* já foi comprado, *P(i|j)* é o número de clientes que compraram ambos os itens *i* e *j* sobre o número total de clientes que compraram *i*. Ou:

$$P(i \mid j) = \frac{Freq(ij)}{Freq(j)}, \quad (6)$$

Esta medida de similaridade traz uma limitação: o item *j* tenderá a ter grande probabilidade condicional com itens freqüentemente comprados, ou seja, *P(i|j)* será grande como resultado de que *i* ocorre freqüentemente e não porque *i* e *j* tendem a ocorrer juntos.

Tendo em vista este problema, (Karypis, 2000) propôs uma outra equação. Primeiramente cada linha da matriz *R* deveria ser normalizada para ser de tamanho 1. Então o cálculo da similaridade seria dado por:

$$sim(i, j) = \frac{\sum_{\forall k: r_{k,i} > 0} r_{k,j}}{Freq(i) \times (Freq(j))^{\alpha}}. \quad (7)$$

O numerador da equação é a soma das entradas não-zero correspondentes à *j*-ésima coluna da matriz usuário-item. α é um parâmetro que varia entre 0 e 1.

3.5.1.3 Geração da Recomendação

Um sistema de recomendação pode, através dos cálculos de similaridade, apenas oferecer uma **predição** do valor que um cliente daria para um determinado item *i*, como também pode fornecer os ***n* melhores itens** relacionados, por exemplo, aos produtos que estão na cesta de compras.

Na primeira situação, para sistemas que consideram avaliações explícitas dos usuários, uma vez que foram isolados os itens mais similares em relação ao item *i*, do qual se deseja obter a predição, deve-se observar as avaliações dos usuários para fazer o cálculo. A soma ponderada é uma das técnicas usadas para obter predições desta maneira (Sarwar, 2001). Nela a predição da avaliação de um item *i* para um usuário *u* é calculada somando-se as avaliações realizadas por ele sobre os itens similares a *i*. Cada

avaliação r é ponderada pela similaridade $s_{i,j}$ correspondente aos itens i e j. Formalmente a predição $P_{u,i}$ pode ser dada como:

$$P_{u,i} = \frac{\sum_{todos\ os\ itens\ similares,N} (s_{i,N} * r_{u,N})}{\sum_{todos\ os\ itens\ similares,N} (| s_{i,N})}. \quad (8)$$

Para as aplicações em que se deseja obter os n melhores itens, Karypis (2001) apresentou o seguinte método, onde U representa o conjunto de itens comprados (por exemplo os que estão na cesta de compras):

1. O conjunto C dos itens recomendados candidatos é formado pela união dos k itens mais similares a cada item $j \in U$. São removidos quaisquer itens que já estejam em U;
2. Para cada item $c \in C$ é calculada sua similaridade em relação ao conjunto U como a soma das similaridades entre todos os itens $j \in U$ e c, usando somente os k itens mais similares de j;
3. Os itens em C são ordenados em ordem decrescente em relação à similaridade e os n primeiros são tomados como o conjunto dos n melhores itens.

3.6 Métricas de Avaliação

Vários tipos de métricas são utilizados para avaliar a qualidade das recomendações produzidas pelos sistemas. As mais citadas em trabalhos da área são descritas a seguir.

3.6.1 Cobertura

Cobertura é a medida do percentual de itens para os quais o sistema de recomendação pode fornecer predições. A cobertura é calculada como o percentual de itens sobre todos os usuários para o quais foi requisitada uma recomendação e o sistema teve capacidade de gerar uma predição.

Características que podem reduzir a cobertura são pequena vizinhança e pouca quantidade de usuários para encontrar vizinhos. Em muitos sistemas a cobertura diminui

como uma função da precisão – o sistema pode produzir poucas recomendações precisas ou mais recomendações imprecisas.

3.6.2 Precisão Estatística

A precisão estatística avalia a precisão de um sistema comparando as avaliações numéricas geradas pelo sistema contra as reais avaliações dos usuários para o par usuário-item em uma base de dados de teste.

Existem várias métricas para avaliar a precisão estatística como o **erro médio absoluto** (MAE, do Inglês *Mean Absolute Error*), **raiz quadrada da média dos erros ao quadrado** (RMSE, do Inglês *Root Mean Squared Error*) e **correlação entre avaliações e predições**. Segundo Good (1999), experiências mostram que todas estas métricas produzem resultados semelhantes, entretanto o MAE é o mais utilizado e o que apresenta maior facilidade de ser interpretado diretamente.

O MAE é a medida do desvio das recomendações do seu verdadeiro valor especificado pelo usuário. Para cada par avaliação-predição $<p_i, q_i>$, esta métrica trata o erro absoluto entre eles, $|p_i - q_i|$, igualmente. O MAE é calculado como a soma todos os erros absolutos dos *N* pares avaliação-predição e dividido por *N*. Formalmente,

$$MAE = \frac{\sum_{i=1}^{N} | p_i - q_i |}{N}. \quad (9)$$

Quanto menor o MAE, mais precisas são as predições produzidas pelo sistema de recomendação.

O RMSE é uma medida que tende a dar mais peso a grandes erros e menor peso a pequenos erros. A intuição por trás do RMSE é que muitas recomendações com uma diferença de 0.25 em uma escala de 5 são melhores que umas poucas com diferença de 3 ou 4. Como o MAE, um RMSE baixo indica melhor precisão. Para calcular o RMSE, os erros individuais são elevados ao quadrado, somados, divididos pelo número de erros individuais e então é extraída a raiz quadrada. Formalmente,

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (p_i - q_i)^2}{N}}. \quad (10)$$

A correlação é a medida estatística de concordância entre dois vetores de dados. Pode-se usar o coeficiente de correlação de Pearson como uma medida linear entre os dois vetores. Um maior valor de correlação indica recomendações mais precisas.

3.6.3 Precisão de Suporte a Decisão

Este tipo de métrica avalia a efetividade de um sistema de recomendação em auxiliar um usuário a selecionar itens de maior qualidade dentre o conjunto de todos os itens. Está baseada no fato de que, do ponto de vista do usuário, as recomendações são binárias: ele compra ou não compra, lê ou não lê, etc. Algumas das métricas de precisão de suporte a decisão são: **avaliação reversa**, **sensibilidade ROC** *(Receiver Operating Characteristic)* e ***recall/precision***.

A avaliação reversa é a medida da freqüência com que o sistema comete grandes erros, recomendando fortemente (acima de um limiar *H*) coisas que o usuário não gosta (avalia abaixo de um limiar *L*) ou gerando recomendações pobres (abaixo de um limiar *L*) de coisas que ele gosta muito (avalia acima de um limiar *H*).

A sensibilidade ROC é uma técnica de avaliação usada na teoria de detecção de sinais e que tem sido cada vez mais aplicada em outros campos como diagnóstico médico, aprendizado de máquina e sistemas de recuperação de informação (Macskassy et al, 2001). A medida de sensibilidade ROC indica quão efetivo o sistema é em oferecer às pessoas itens altamente recomendados e afastá-las de itens com baixa avaliação (Sarwar et al, 1998). Os gráficos ROC traçam a *sensibilidade* vs. *1-especificidade* do teste (Swets, 1988). A sensibilidade refere-se à probabilidade de um bom item selecionado aleatoriamente ser aceito pelo filtro (positivo verdadeiro) e a especificidade é a probabilidade de um item ruim selecionado aleatoriamente ser rejeitado pelo filtro (negativo verdadeiro) (Good et al, 1999). Os pontos são obtidos variando-se o limiar de recomendação acima do qual o item é aceito. A área sob a curva aumenta conforme o filtro for capaz de aceitar mais itens bons e poucos itens ruins. O intervalo da sensibilidade ROC vai de 0 a 1, onde 1 é um filtro perfeito e 0.5 é um filtro razoável. A figura 3.4 ilustra três curvas-exemplo de um gráfico ROC, onde a proporção de positivos verdadeiros está traçada contra a proporção de falsos positivos.

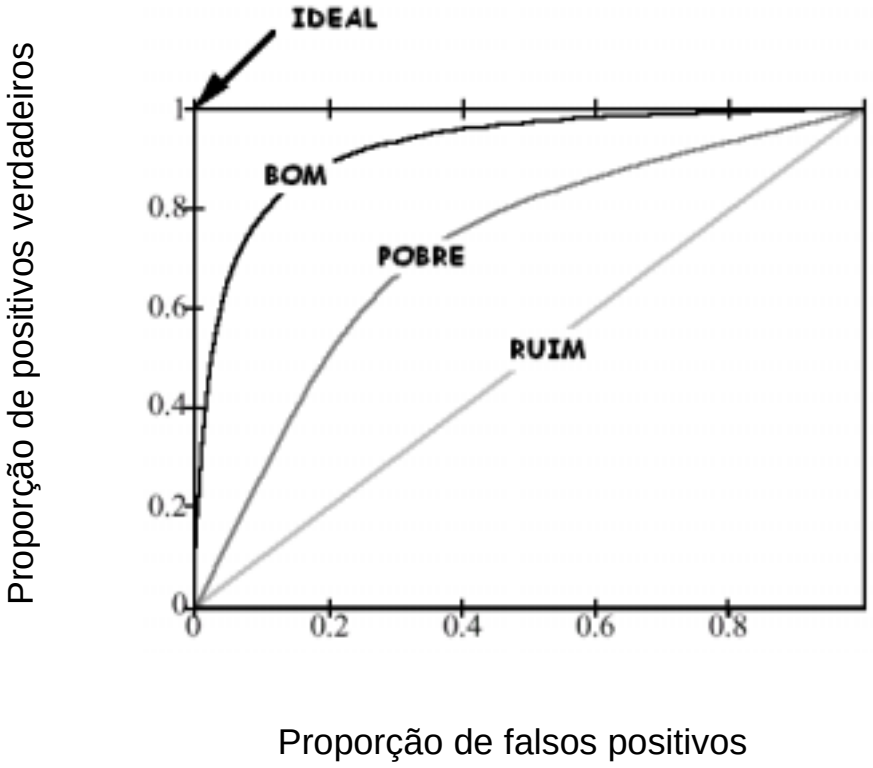


Figura 3.4 – Exemplo de gráfico ROC

Sarwar et al (2001) observou que MAE e ROC oferecem a mesma ordem de diferentes esquemas experimentais em termos de qualidade das predições.

Recall/Precision é uma métrica muito usada na área de **Recuperação de Informações** (*Information Retrieval*) e foi adaptada por Sarwar et al (2000) para avaliar sistemas de recomendação. Considerando o *hit set* como o conjunto de recomendações gerado por uma base de dados de treinamento (top-N *set*) que coincide com os dados da base de teste (*test set*), *recall* e *precision* podem ser definidos:

$$recall = \frac{tamanho\ do\ hit\ set}{tamanho\ do\ test\ set} \quad \text{ou} \quad recall = \frac{|test \cap top - N|}{|test|}, \quad (11)$$

$$precison = \frac{tamanho\ do\ hit\ set}{tamanho\ do\ top - N\ set} \quad \text{ou} \quad precision = \frac{|test \cap top - N|}{N}. \quad (12)$$

Como ambas as medidas são conflitantes por natureza – por exemplo, um aumento em *N* tende a aumentar *recall*, mas diminui *precision* – Sarwar et al (2000)

utilizou uma combinação das duas medidas: a **métrica F1** (Yang e Liu, 1999). Esta métrica dá peso igual às duas medidas e é calculada da seguinte forma:

$$F1 = \frac{2 * recall * precision}{recall + precision}. \quad (13)$$

O valor F1 é computado para cada cliente individualmente e a média geral pode ser usada como métrica para avaliar o sistema.

4 O Modelo Proposto

4.1 Introdução

Tendo sido avaliados os aspectos teóricos das soluções para os problemas de personalização, este capítulo traz a proposta para a construção de um modelo genérico que atenda às necessidades das aplicações de comércio eletrônico. Em primeiro lugar é apresentada uma visão geral do modelo de personalização, em seguida é descrito em mais detalhes o método de recomendação híbrido e, por fim, é ilustrada a arquitetura do modelo.

4.2 O Modelo da Aplicação de Recomendação

De acordo com a taxonomia dos sistemas de recomendação analisada no item 3.3, descrevem-se a seguir as questões que implicam a construção da aplicação baseada no modelo proposto.

Os **dados de entrada** do sistema são as transações efetuadas pelos clientes de um site de *e-commerce* em uma perspectiva histórica. Estes dados são utilizados para montar a matriz cliente-produto. Questões como a quantidade e a freqüência de compra dos itens estão incluídas no modelo. Em relação ao cliente ativo, para o qual se deseja fornecer recomendações em tempo real, são observados, além das informações sobre seu histórico, os produtos contidos na cesta de compras durante a interação com o site.

A **saída** do sistema consiste em uma lista das *n* melhores recomendações para um cliente, onde *n* é um dos parâmetros a ser configurado no sistema. Estes itens selecionados serão utilizados pela aplicação da loja virtual, que determina o que apresentar ao cliente e de que maneira ou em que contexto.

O **método de recomendação**, componente central de um sistema de recomendação, constitui o verdadeiro escopo deste trabalho. Ele foi desenvolvido tendo como base os dois métodos de sistemas de recomendação mais conhecidos:

- correlação item-a-item: permite prover ofertas personalizadas de acordo com o que o cliente possui em sua cesta de compras;
- correlação pessoa-a-pessoa: dá um caráter mais individualizado ao fazer uso das informações históricas do cliente para produzir sugestões.

Este método híbrido, que será descrito em detalhes na seção seguinte, foi inspirado no trabalho de Karypis (2000). Karypis afirmou, na conclusão de seu trabalho sobre algoritmos de recomendação baseados em correlação item-a-item, acreditar que os mesmos poderiam ser melhorados combinando elementos das abordagens baseadas em itens e das baseadas em pessoas. Por um lado, a correlação pessoa-a-pessoa, ao calcular vizinhos de modo dinâmico, representa de modo mais exato o comportamento dos clientes. Contudo, a correlação item-a-item tem demonstrado mais eficiência ao gerar recomendações e ao mesmo tempo manteve sua qualidade (Karypis, 2000; Sarwar et al, 2001).

De acordo com a taxonomia que está sendo analisada e como conseqüência da aplicação do método de recomendação híbrido constata-se que o **grau de personalização** do sistema é persistente, já que, além de informações da sessão – cesta de compras –, também serão utilizadas informações do histórico de compra dos clientes.

A forma de **apresentação** das recomendações fica por conta da aplicação de comércio eletrônico. Pode ser passiva, ao expor os itens durante a navegação do cliente pelo site, ou baseada em tecnologia *push*, caso a política de relacionamento da organização ou o próprio cliente permita que lhe sejam enviados e-mails ou cartas com ofertas personalizadas.

A figura 4.1 mostra um esquema do modelo proposto de acordo com as observações feitas acima.

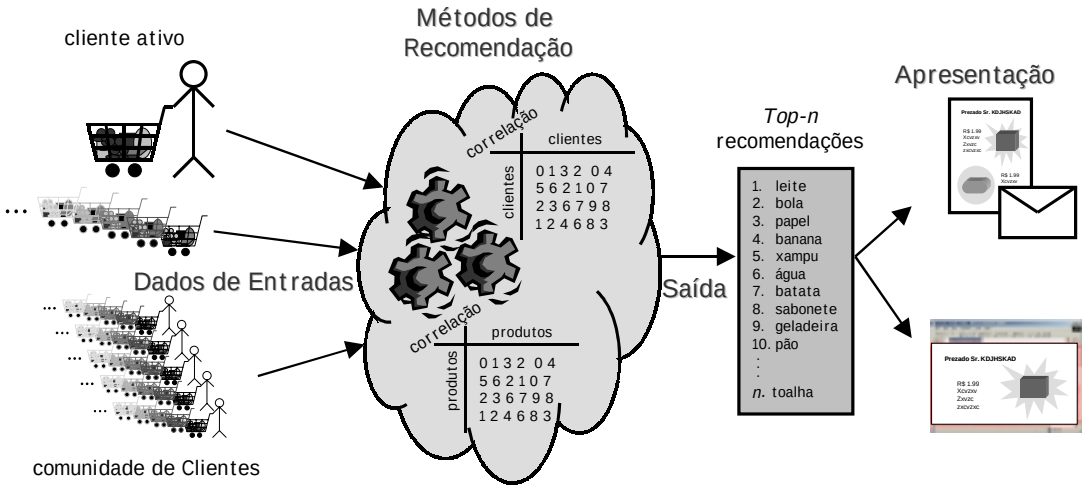


Figura 4.1 – Modelo de personalização proposto

4.3 O Método de Recomendação Híbrido

O método de recomendação proposto neste trabalho é caracterizado pela tentativa de combinação dos dois mais famosos métodos: a filtragem colaborativa e a correlação item-a-item. O desenvolvimento deste método representa um verdadeiro desafio, pois não se conhece estudo semelhante na literatura. Outra particularidade deste método é sua aplicação a casos em que os produtos são comprados de forma repetida. As pesquisas conhecidas também não tratam desta classe de problemas.

O processo de geração de recomendações pode ser dividido em três partes: (1) representação, (2) construção do modelo e (3) geração das recomendações. A primeira parte, a **representação**, trata do esquema utilizado para modelar os produtos que já foram comprados pelo cliente, bem como os produtos que eventualmente estejam no carrinho durante a compra. A **construção do modelo** diz respeito à tarefa de determinar de forma numérica as relações entre clientes e entre produtos e organizar estas informações de forma a deixá-las facilmente acessíveis para a fase seguinte. A última parte, **geração das recomendações**, trata do problema de identificar os produtos mais recomendáveis para um cliente a partir das informações de similaridade e de sua cesta de compras. A figura 4.2 ilustra como estas três partes estão relacionadas com as tarefas dos métodos de filtragem colaborativa e correlação item-a-item apresentados

anteriormente. Como pode ser observado pela área branca do esquema, no método proposto a etapa de construção do modelo – tarefa 2 – engloba, além das atividades de formação da vizinhança e cálculo de similaridade – também tarefas 2 –, a atividade de geração das recomendações do modelo de filtragem colaborativa – tarefa 3. Esta particularidade será elucidada mais adiante.

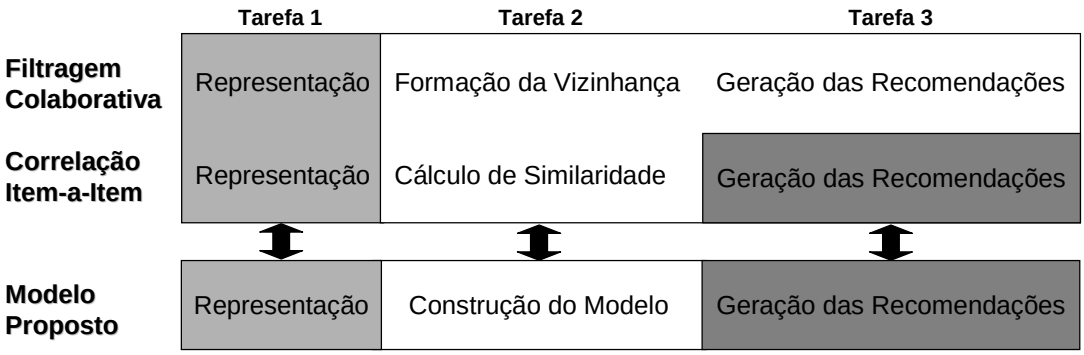


Figura 4.2 – Correspondência entre os processos dos modelos tradicionais e os processos do modelo proposto

4.3.1 Representação

No caso específico de aplicações para comércio eletrônico, as pesquisas publicadas tratam a informação de compra como um dado binário, isto é, a única informação representada no modelo é a de que o cliente comprou ou não comprou determinado produto. Em alguns casos como em Karypis (2000), adota-se o procedimento de normalizar as linhas da matriz cliente-produto, fazendo com que a soma dos elementos seja igual a um. O objetivo da normalização é fazer com que os clientes que compram freqüentemente tenham menos influência no resultado, gerando melhores recomendações. Esta representação, embora muito útil para lojas em que geralmente compra-se apenas um exemplar de cada produto, não atende a outros tipos de varejo, como supermercados, lojas de suprimentos de informática e escritório, entre outras, nas quais os produtos são comprados repetidamente.

O modelo de método de recomendação aqui proposto visa superar esta limitação, a fim de que mais aplicações possam fazer uso dos sistemas de recomendação para personalizar a interação de clientes com os sites de venda.

A representação dos dados de entrada para este modelo que compreende o histórico de compras de todos os clientes é realizada de duas maneiras distintas: a primeira é a entrada para o componente de filtragem colaborativa e a segunda é a entrada para o componente de correlação item-a-item.

4.3.1.1 Representação para a filtragem colaborativa

A representação tradicional dos dados de entrada de um sistema de recomendação eletrônico é uma matriz $m \times n$, onde m é o número de clientes e n é o número de produtos. No comércio eletrônico os elementos desta matriz, também chamada de matriz cliente-produto, armazenam apenas informação binária, ou seja, $a_{i,j}=1$ se o cliente i já comprou o produto j ou $a_{i,j}=0$, caso o cliente i ainda não tenha comprado o produto j. Como pode ser observado, esta representação é adequada para lojas onde geralmente apenas um exemplar de cada produto é comprado. Entretanto, para os casos em que os produtos são comprados repetidamente este modelo não serve, pois os sistemas atuais não recomendam um item que já foi comprado.

As soluções encontradas para resolver este problema foram: (1) utilizar na representação dos dados de entrada as informações de quantidade de produtos comprados e número de compras efetuadas ao longo do relacionamento com o cliente e (2) incluir os produtos do próprio cliente na geração das recomendações de produtos a partir da vizinhança. Esta última estratégia será detalhada a seguir, na fase de construção do modelo.

Ao utilizar os dados de quantidade de produtos e o número de compras efetuadas, a matriz cliente-produto transforma-se em uma matriz de freqüência de compra de produtos, ou seja, $a_{i,j} = \frac{quantidade}{|compras|}$, onde *quantidade* representa a quantidade comprada do produto j pelo cliente i e |compras| é o número de transações efetuadas pelo cliente i. Assim, produtos freqüentemente comprados terão sempre um peso maior na geração das recomendações, enquanto que produtos comprados apenas uma vez terão um peso menor a cada compra.

A normalização das linhas da matriz, como já justificada e empregada com sucesso (Karypis, 2000), também foi aplicada aqui. Assim espera-se que as compras mais pulverizadas exerçam menos influência no modelo.

A figura 4.3 ilustra a representação tradicional e a representação proposta.

produtos

clientes

0	1	0	0	1
1	1	0	0	1
0	0	0	0	1
1	1	1	0	0
0	1	0	1	0
0	0	0	1	1
1	0	1	0	0

(a) Representação tradicional em *e-commerce*

produtos

clientes

0,54	3,23	10,6	52,3	0
0,01	6,36	5,01	0	1
0	0	12,3	25	1,1
9,96	52	1,4	4,56	0
12	1	0	0	0
25,6	0	0	7,85	9
3,2	1,2	33,7	0	0

(b) Representação proposta para suportar recomendações repetidas

Figura 4.3 – Formas de representação

A inclusão destes valores no modelo não traz dificuldades em termos de obtenção dos dados, pois estas informações já fazem parte do banco de dados das empresas. Além disto, esta forma de representação propicia uma melhor captura do comportamento de compra dos clientes.

4.3.1.2 Representação para a correlação item-a-item

Dentro do modelo proposto, a informação gerada pela correlação item-a-item é utilizada para que, durante a interação do cliente com o site de compras, seja possível recomendar produtos de acordo com os itens que estejam no carrinho.

Sendo assim, correlacionar o total de produtos comprados pelos clientes, utilizando a matriz cliente-produto apresentada anteriormente, não representa a realidade do comportamento de compra. O mais adequado seria correlacionar os produtos em relação a cestas de compras. Esta foi a forma de representação concebida para o componente de correlação item-a-item do modelo proposto.

Neste caso, em vez da representação dos dados de compra dos clientes ser feita através de uma matriz cliente-produto $m \times n$, onde m é o número de clientes e n é o

número de produtos, ela é feita a partir de uma matriz $k*m \times n$, onde k é o número de cestas de cada cliente a serem incluídas. Chamaremos esta matriz de matriz cesta-produto. Quanto maior o valor de k, mais subsídios tem o algoritmo de correlação para calcular as relações entre os produtos.

A multiplicação do número de linhas da matriz por k não chega a trazer problemas de performance para o sistema, pois na grande maioria dos casos o número de clientes é sempre muitas vezes inferior ao número de produtos.

4.3.2 Construção do Modelo

Esta pode ser considerada a etapa mais importante de todo o processo, quando de fato acontece o processo de aprendizado ou de construção do modelo para um algoritmo de sistema de recomendação. É aqui que são identificadas e quantificadas as relações entre os clientes e entre os produtos.

O modelo proposto segue os mesmos princípios das abordagens tradicionais: os valores de similaridade para produtos e para clientes são calculados de modo independente, a combinação dos métodos só se dá na fase seguinte: na geração das recomendações. A figura 4.4 ilustra os subprocessos realizados nesta etapa do processo de geração de recomendações.

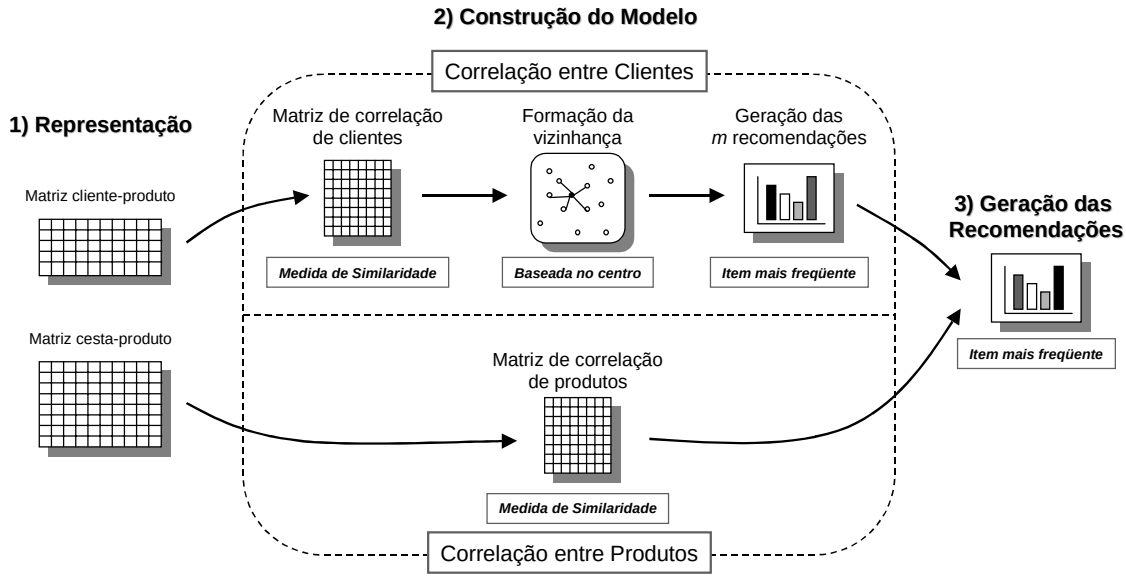
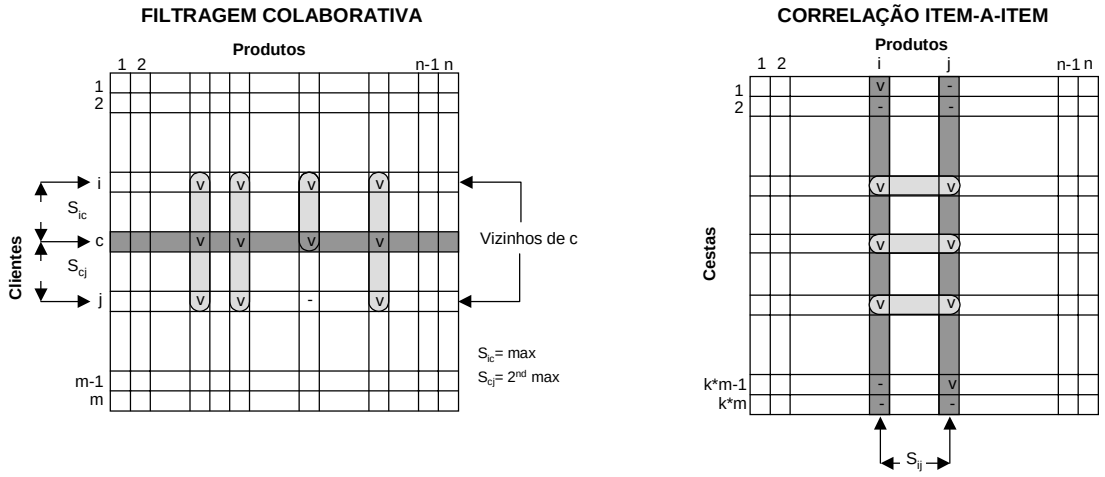


Figura 4.4 – Os subprocessos da construção do modelo

4.3.2.1 A correlação entre clientes

Esta etapa utiliza o método de filtragem colaborativa para obter uma lista dos produtos mais recomendáveis para cada cliente. O cálculo da similaridade entre os clientes determina a formação de vizinhanças, ou seja, o conjunto de clientes que servirão de referência para recomendar produtos ao cliente-alvo.

A partir da matriz cliente-produto de freqüência de compras é gerada uma matriz $m \times m$ de correlação de clientes, onde m é o número de clientes. Nesta matriz de correlação cada elemento $u_{a,b}$ representa o valor de proximidade entre os clientes a e b. A figura 4.5a ilustra o esquema de cálculo da similaridade entre clientes a partir da matriz cliente-produto.



(a) Cálculo de similaridade entre clientes

(b) Cálculo de similaridade entre produtos

Figura 4.5 – Algoritmos de filtragem colaborativa e correlação item-a-item

As duas medidas de similaridade testadas neste procedimento realizam o cálculo de correlação apenas sobre os produtos que ambos os clientes compraram. Assim, pode ocorrer de a correlação entre dois clientes resultar em um alto valor, mesmo que os dois clientes tenham apenas dois produtos em comum. Foi provado que vizinhos relacionados através de poucos itens tendem a prejudicar a recomendação de produtos (Herlocker et al, 1999). Para evitar que estes valores “mascarados” tenham demasiada influência sobre o resultado, é aplicada uma ponderação de significância sobre o valor de correlação calculado. Nos casos em que os dois clientes em questão possuem menos

de 50 produtos em comum, é aplicado um peso de significância de *n/50*, onde *n* é o número de itens comprados por ambos. Se 50 ou mais produtos em comum foram comprados pelos dois clientes, o peso de significância *1* é aplicado.

Depois de calculada a similaridade entre todos os clientes, é realizado o processo de geração dos *m* produtos mais recomendáveis para cada cliente, que compreende as tarefas de formação da vizinhança e geração das *m* recomendações.

O método utilizado para formar a vizinhança é o baseado no centro, o qual forma uma vizinhança de tamanho *k*, para um cliente particular *c*, através da seleção dos *l* outros clientes mais próximos.

Tendo formado a vizinhança, o passo seguinte é gerar os *m* produtos mais recomendáveis para cada cliente, que servirá de entrada para o processo de geração de recomendações on-line. No método de filtragem colaborativa original, este conjunto de *m* produtos seria o resultado final de todo o processamento. Porém, no modelo proposto esta é apenas uma etapa intermediária, os *m* produtos serão combinados na fase seguinte com os produtos da cesta do cliente-alvo, para então gerar as recomendações finais.

Conforme mencionado anteriormente, foram utilizados dois artifícios para possibilitar ao modelo gerar recomendações de produtos que o cliente já tenha comprado. Um deles é a representação da freqüência de compra, já citada, e o outro é a inclusão dos produtos do próprio cliente na seleção de produtos a partir da vizinhança. Para isso, após a seleção dos *k* vizinhos, o próprio cliente é inserido na vizinhança tendo seus produtos considerados na geração dos *m* melhores itens.

A técnica empregada para obter-se os *m* melhores produtos a partir da vizinhança foi a **recomendação do item mais freqüente**. Neste método os dados de compra dos *k* vizinhos e do cliente para o qual se deseja gerar as recomendações são analisados, somando-se os valores dos produtos correspondentes. Os valores dos produtos do cliente-alvo são multiplicados por um fator de significância *p*. Quanto maior o valor de *p*, mais valor será dado ao histórico do cliente, do contrário, quando *P* for igual a zero, os produtos do próprio cliente não serão somados e não influirão no resultado final. Esta última situação é a que deve ser utilizada quando a aplicação não deve recomendar itens que já foram comprados pelo cliente.

Depois de feita a contagem, o sistema ordena a lista de produtos por ordem do mais freqüente e retorna os *m* melhores. Por fim, esta lista é armazenada em um banco de dados, para que possa ser acessada posteriormente na fase de geração das recomendações.

4.3.2.2 A correlação entre produtos

Esta etapa utiliza o método de correlação item-a-item para gerar uma matriz de similaridade entre produtos. O valor da similaridade entre produtos permite identificar que produtos tendem a ser comprados em associação com outros. A proximidade entre os produtos é uma informação utilizada na geração das recomendações para correlacionar os produtos da cesta de compras com os *m* melhores produtos de cada cliente obtidos através da filtragem colaborativa.

A partir da matriz cesta-produto é gerada uma outra matriz $n \times n$, onde n é o número de produtos. Nesta matriz de correlação cada elemento $v_{a,b}$ representa o valor de proximidade entre os produtos *a* e *b*. A figura 4.5b ilustra o esquema de cálculo da similaridade entre produtos a partir da matriz cesta-produto.

Do mesmo modo que na correlação entre clientes, produtos correlacionados por apenas duas cestas em comum podem afetar negativamente as recomendações. Sendo assim, após o cálculo da similaridade entre dois produtos, o valor da correlação é desvalorizado, usando-se os mesmos pesos aplicados na correlação entre clientes.

Por fim, todos os pares de produtos para os quais foi calculado um valor de correlação são armazenados em um banco de dados para que possam ser posteriormente consultados no processo final de geração das recomendações.

4.3.3 Geração das Recomendações

Esta é a etapa final de todo o processo do sistema de recomendação e na qual o modelo definitivamente combina as informações processadas pelas técnicas de correlação item-a-item e filtragem colaborativa.

Durante a navegação dos clientes pelo site, o conjunto *U* de produtos inseridos na cesta de cada cliente é utilizado para gerar as top-N recomendações, da seguinte maneira:

1. O conjunto *C* de produtos candidatos é obtido através da recuperação dos *m* melhores itens gerados pela filtragem colaborativa e armazenados no banco de dados. Os itens que já estejam em *U* são removidos de *C*.
2. Para cada $c \in C$, é calculada a sua similaridade com o conjunto *U*, como a soma das similaridades entre todos os itens $j \in U$ e *c*. O valor da similaridade entre *c* e *U* é ponderado pelo valor correspondente de *c* gerado no processo de obtenção dos *m* melhores produtos na filtragem colaborativa.
3. Por fim, o conjunto *C* é ordenado em ordem decrescente de similaridade e os primeiros *N* itens são selecionados como o conjunto das *N* melhores recomendações.

4.4 Arquitetura do Modelo

O método desenvolvido foi dividido em duas partes com o objetivo de evitar problemas de escalabilidade e desempenho causados por grandes bases de dados. Na primeira parte, que é executada off-line, toda a base de dados de histórico dos clientes é utilizada para construir um modelo parcial de personalização para cada cliente. Estes dados parciais são então utilizados pela segunda parte, no momento em que os clientes acessam o site, para gerar as recomendações on-line. O esquema de funcionamento destas duas partes componentes do sistema e sua relação com os três processos apresentados anteriormente pode ser observado na figura 4.6.

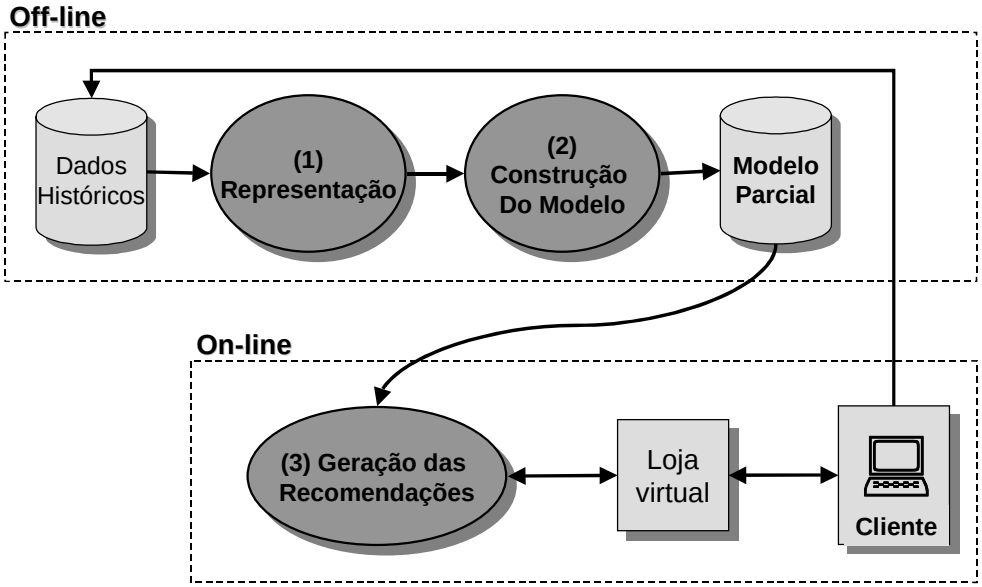


Figura 4.6 – As partes componentes do modelo

O componente off-line do modelo compreende as tarefas de (1) construir a matriz cliente-produto, (2) identificar as relações entre todos os clientes e armazenar os produtos mais recomendáveis para cada cliente e identificar as relações entre todos os produtos e armazenar os valores de correlação de todos os pares. A tarefa 2 implica uma complexidade computacional $O(m^2n)$, já que é necessário calcular $m(m-1)$ similaridades, cada uma requerendo n operações. Por este motivo, sendo executada off-line, esta tarefa poupa esforço computacional no momento da interação do cliente com o site.

Uma alternativa que pode ser utilizada para abrandar o esforço computacional no caso do componente de filtragem colaborativa é a segmentação da base de dados de clientes. Após a segmentação, apenas os clientes preferenciais, ou que trazem melhores retornos à empresa, podem ser selecionados para participar do processo de geração de recomendações personalizadas. No caso do componente de correlação item-a-item, pode-se reduzir o número de produtos passíveis de serem recomendados.

O componente on-line do modelo utiliza as informações pré-computadas anteriormente para gerar as recomendações em tempo real de acordo com os produtos que o cliente vai colocando em sua cesta de compras. O procedimento básico é utilizar os valores de similaridade de produtos armazenados para correlacionar os produtos que

estão na cesta com os *k* produtos mais recomendáveis computados pela filtragem colaborativa.

Para as aplicações que envolvem um número não tão grande de clientes, geralmente o caso de lojas virtuais que entregam seus produtos em uma área de abrangência municipal, o processo pode ser executado totalmente on-line, simplificando a operação do sistema.

5 Avaliação Experimental

5.1 Introdução

Este capítulo apresenta uma análise estatística dos resultados obtidos utilizando o modelo proposto sobre um conjunto de dados de compras reais cedido por uma empresa de comércio eletrônico. A seguir é apresentada a implementação do modelo proposto, a descrição do conjunto de dados utilizado, a metodologia e a métrica de avaliação e, finalmente, os resultados das avaliações.

5.2 Implementação do Modelo

Para fins de avaliação experimental foi desenvolvida uma aplicação de geração de recomendações baseada no modelo proposto e também uma aplicação para simular uma compra em massa realizada por todos os clientes, a fim de obter as medidas de desempenho do modelo. Ambas as aplicações foram implementadas na linguagem Java e as avaliações foram realizadas em um microcomputador PC, com processador de 800 Mhz, 512 MB de memória RAM e ambiente MS windows. O anexo 1 apresenta o código das aplicações desenvolvidas.

A primeira aplicação corresponde ao componente off-line do modelo proposto, a qual armazena os resultados em um banco de dados. A segunda aplicação está relacionada ao componente on-line e utiliza as informações resultantes da primeira para gerar recomendações a partir do modelo proposto (MP). Para realizar um teste comparativo com as abordagens tradicionais de algoritmos de recomendação nesta aplicação também foram implementados procedimentos para obter-se recomendações a partir do modelo de filtragem colaborativa (MFC), do modelo de correlação item-a-item (MCI) e do modelo de produtos mais vendidos (MMV). A saída desta última aplicação é a medida de desempenho *recall* para cada um dos métodos de recomendação. O esquema de funcionamento deste conjunto pode ser observado na figura 5.1.

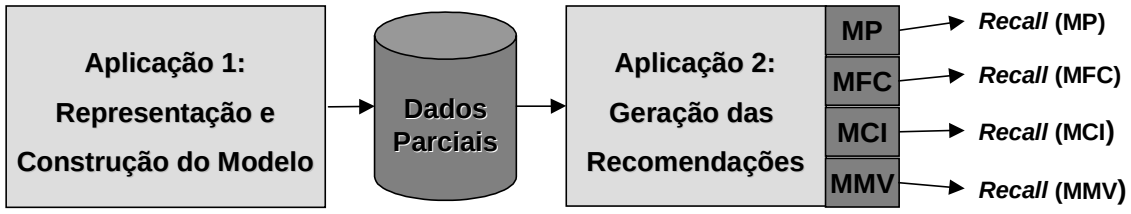


Figura 5.1 – A aplicação do modelo e a aplicação para simulação

5.3 Os Conjuntos de Dados

Os dados utilizados nesta avaliação foram cedidos pela empresa Hippo Supermercados Ltda., do ramo supermercadista, que atua tanto no varejo tradicional como no comércio eletrônico. Os dados fornecidos originaram-se de duas fontes distintas:

- **Cartão**: movimento de compras realizadas com o cartão de fidelidade Hippo;
- **Internet**: movimento de compras através do site do supermercado Hippo.

As avaliações que seguem foram conduzidas distintamente sobre estes dois conjuntos de dados, de modo a poder avaliar também os resultados de acordo com a forma de comercialização dos produtos.

Os conjuntos de dados são constituídos de um identificador único para cada cliente, o número de compras efetuadas no período e uma identificação única dos produtos comprados por cada cliente juntamente com a quantidade ao longo do período.

Algumas restrições foram colocadas para que se pudesse avaliar o modelo de forma mais precisa. Dos dados originais obtidos foram removidos os seguintes casos:

- Clientes que efetuaram apenas uma compra. É necessário que todos os clientes tenham efetuado pelo menos duas transações: uma para gerar as recomendações e outra para simular uma compra;

- Clientes que possuíam somente um produto comprado na última transação. São necessários pelo menos dois itens já que um deles deve fazer parte do conjunto de teste e outro é usado para gerar as recomendações;
- Clientes com apenas um produto comprado no histórico. Os algoritmos de correlação necessitam de pelo menos dois produtos para poder correlacionar os clientes.

As informações referentes a cada conjunto de dado, subdidivido pela aplicação das restrições acima relacionadas, podem ser observadas na tabela 5.1.

Tabela 5.1 – As características dos conjuntos de dados utilizados na avaliação

Característica	**Cartão**		**Internet**	
	Pré-filtragem	Pós-filtragem	Pré-filtragem	Pós-filtragem
Número de Clientes	1780	1465	509	269
Número de Produtos	7367	7211	4781	4225
Média de Produtos no Carrinho	13,63	13,77	36,87	37,09
Período	6 meses		1 ano	

5.4 Metodologia e Métrica de Avaliação

O objetivo dos testes realizados foi avaliar a qualidade das recomendações produzidas pelo modelo proposto. Para isso, cada conjunto de dados foi divido em duas partes: um conjunto de dados para treinamento e um conjunto de dados para teste. A base de dados de treinamento constitui-se pela massa de dados de todas as compras efetuadas por cada cliente, com exceção da última. Esta massa de dados é utilizada para gerar o modelo de dados parcial do sistema de recomendação. O conjunto de dados de teste é definido pela última compra de cada cliente, da qual um item é retirado aleatoriamente para testar as recomendações e o restante dos produtos, para efeito de simulação, é usado como a cesta de compras. O item a ser testado é retirado aleatoriamente pois, dentre as informações obtidas sobre as compras efetuadas por cada cliente através da Internet, não constava a ordem em que os produtos foram colocados no carrinho. Caso este dado estive presente, a verificação da qualidade das

recomendações poderia ser ainda mais apurada, utilizando-se como item de verificação o último adquirido.

As experiências foram realizadas da seguinte maneira: em cada teste foram obtidas *N* recomendações. A qualidade das recomendações foi medida verificando-se o número de acertos, isto é, o número de itens retirados aleatoriamente do conjunto de dados de teste que também estavam no conjunto dos *N* itens recomendáveis retornados para cada cliente. A métrica *recall* – citada do item 3.6.3 e utilizada por Karypis (2000) – foi calculada para os três modelos. Sendo *k*, o número total de clientes, o *recall* de cada modelo é computado como:

$$recall = \frac{número\ de\ acertos}{k}. \quad (14)$$

Se o valor de *recall* for igual a um, isto quer dizer o sistema conseguiu recomendar o item que havia sido aleatoriamente retirado da última compra para todos os clientes. Caso contrário, quando o *recall* se iguala a zero, significa que o modelo não foi capaz de recomendar nenhum dos itens retirados.

Este teste sobre os modelos foi executado dez vezes, alterando-se o item retirado do conjunto de dados de teste randomicamente a fim de assegurar a validade estatística dos resultados. Os resultados que seguem na próxima seção são uma média dos dez experimentos de cada modelo. Nestes experimentos foram requisitadas aos sistemas dez recomendações para cada cliente (top-10).

5.5 Resultados Experimentais

Para avaliar a qualidade das recomendações é necessário determinar a sensibilidade de alguns parâmetros antes de executar o experimento principal: a comparação dos resultados gerados pelo modelo proposto com os resultados dos outros modelos de recomendação.

Os parâmetros a serem analisados incluem o algoritmo de similaridade para o componente da filtragem colaborativa e da correlação item-a-item, o peso *p* dos produtos do cliente-alvo na geração do *m* melhores produtos e o número de melhores produtos *m* a ser considerado na geração das recomendações.

5.5.1 Efeito dos Algoritmos de Similaridade

Um aspecto crítico no modelo de sistema de recomendação proposto é quanto ao algoritmo utilizado para calcular a similaridade entre os clientes e entre os produtos. Com o intuito de verificar o comportamento do sistema frente às possibilidades de algoritmos, foi realizado um teste variando a medida de similaridade nos dois pontos de cálculo de similaridades do modelo. As duas medidas testadas nestes pontos foram a **correlação de Pearson (CP)** e o **cosseno (CO)**.

No cálculo da similaridade baseada em **cosseno** os produtos *a* e *b* são tratados como vetores *m*-dimensionais no espaço dos clientes. O cosseno entre dois vetores é dado por:

$$sim(a,b) = \cos(\vec{a},\vec{b}) = \frac{\vec{a}\cdot\vec{b}}{\left\|\vec{a}\right\|_2 * \left\|\vec{b}\right\|_2}, \quad (15)$$

onde "." denota o produto interno dos dois vetores.

A aplicação desta medida para a correlação entre produtos foi apoiada em trabalhos anteriores a respeito desta técnica (Karypis, 2000; Sarwar et al, 2001). A similaridade baseada em cosseno tem sido a medida de proximidade mais comumente utilizada nos algoritmos de correlação item-a-item pesquisados.

A outra medida de proximidade que foi utilizada é a **correlação de Pearson**, a qual é dada por:

$$sim(a,b) = corr_{ab} = \frac{\sum_i (r_{ai} - \bar{r}_a)(r_{bi} - \bar{r}_b)}{\sqrt{\sum_i (r_{ai} - \bar{r}_a)^2 \sum_i (r_{bi} - \bar{r}_b)^2}}. \quad (16)$$

A escolha desta medida de similaridade foi baseada nas recomendações de Herlocker (1999), em seu estudo sobre algoritmos de filtragem colaborativa, que indica a utilização da correlação de Pearson nos casos em que os elementos da matriz cliente-produto formem uma escala não-discreta.

A partir destas duas formas de calcular as similaridades, foram testadas quatro possibilidades, conforme mostra a tabela 5.2.

Tabela 5.2 – Testes das medidas de similaridade

Combinação	**Similaridade entre Clientes**	**Similaridade entre Produtos**
1	CP	CP
2	CP	CO
3	CO	CO
4	CO	CP

O número de produtos correlacionados com os produtos das cestas foi *m=30* e o peso do cliente-alvo utilizado foi *p*=5. O resultado deste teste pode ser analisado na figura 5.2.

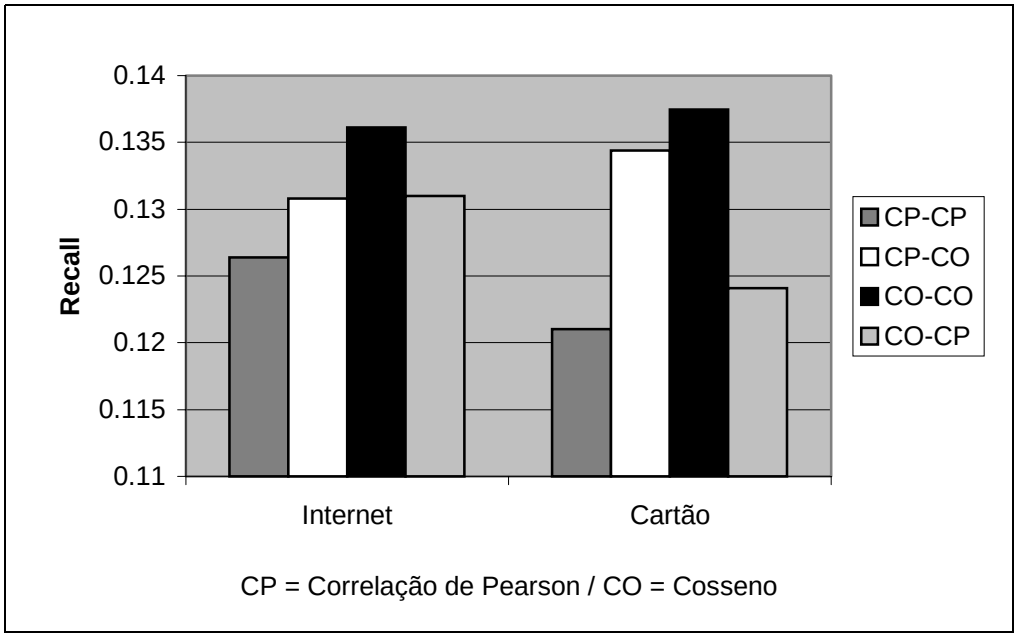


Figura 5.2 – Impacto das combinações de medidas de similaridade sobre o resultado

A combinação que obteve o melhor resultado em ambos os conjuntos de dados foi a Cosseno-Cosseno, levando uma boa vantagem sobre todas as demais abordagens. Sendo assim, nas experiências seguintes foi utilizada a combinação CO-CO como algoritmos para gerar as recomendações.

5.5.2 Sensibilidade ao Peso do Cliente-alvo

Um parâmetro importante a ser considerado no modelo proposto é o valor de *p*, o fator que multiplica os produtos do cliente-alvo no momento da seleção dos *m* melhores produtos.

O método utilizado para selecionar os *m* produtos é somar os valores de correlação dos produtos dos *k* vizinhos do cliente ativo com os seus próprios valores multiplicados por *p*. Em todos os experimentos foi considerada uma vizinhança de tamanho *k=50.*

A figura 5.3 mostra os resultados alcançados variando-se o valor de *p* de 0 até 300.

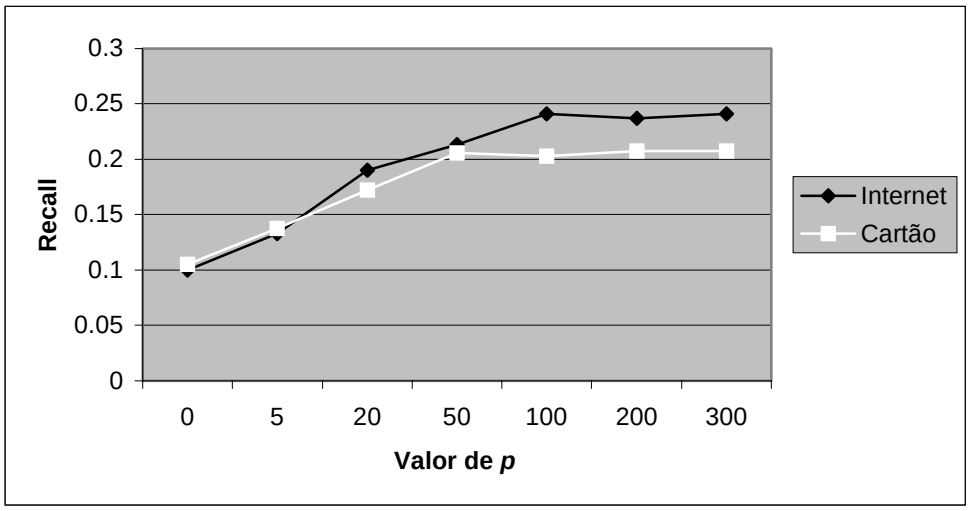


Figura 5.3 – Sensibilidade do modelo proposto ao valor de *p*

O gráfico mostra que a partir de *p*=100, o *recall* é estabilizado para os dois conjuntos de dados testados. Como são 50 os vizinhos computados, este valor de *p* significa que, dando ao próprio cliente o dobro da importância da vizinhança, atinge-se o máximo de *recall*.

É importante ressaltar que quanto maior for o fator de multiplicação dos produtos do cliente-alvo, menos recomendações de produtos ainda não comprados, mas muito bem classificados pelos seus semelhantes, ele receberá.

Em razão dos resultados obtidos, os experimentos seguintes foram conduzidos com p=100.

5.5.3 Sensibilidade ao Tamanho do Modelo

O método de recomendação proposto gera as recomendações utilizando os *m* produtos obtidos através do método de filtragem colaborativa. Para avaliar a sensibilidade do modelo em relação ao valor de *m*, foi conduzida uma experiência variando o valor de *m*. A simulação foi realizada para os valores 15, 20, 25, 30 e 40 de *m*. A figura 5.4 apresenta os resultados deste experimento.

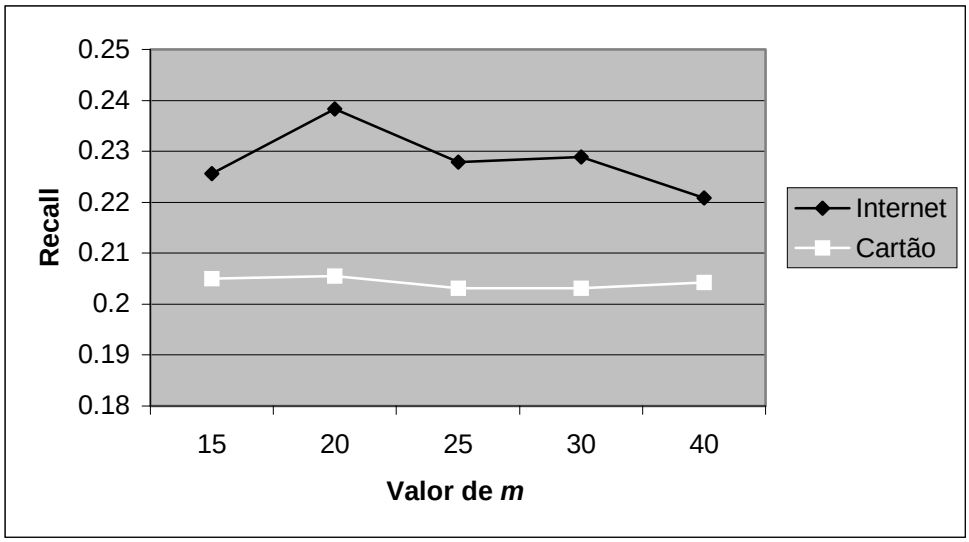


Figura 5.4 – Sensibilidade do modelo proposto ao valor de *m*

Os melhores valores foram obtidos com *m*=20 para ambos os conjuntos de dados, embora para os dados do cartão não tenha havido muita variação no *recall*.

É interessante observar que com *m* igual a dez, o resultado seria o mesmo do MFC nesta avaliação, já que a ordem das top-N recomendações não influencia o resultado.

Em vista do comportamento dos dados verificado, para a experiência seguinte o valor de *m* foi fixado em 20.

5.5.4 Comparação com outros Métodos de Recomendação

O objetivo desta avaliação é analisar a qualidade e a eficiência das top-N recomendações geradas pelo modelo proposto comparativamente aos modelos mais tradicionais de sistemas baseados exclusivamente em filtragem colaborativa e correlação de itens e a um modelo que recomenda invariavelmente os produtos mais vendidos.

O experimento foi realizado gerando dez recomendações para todos os clientes através dos quatro modelos com a mesma cesta de compras e o mesmo produto "escondido" para ser checado.

O modelo de filtragem colaborativa implementado para esta avaliação é o mesmo do componente filtragem colaborativa do modelo proposto. Sendo assim, dos *m* melhores produtos gerados por ele foram selecionados os dez primeiros produtos como as top-N recomendações.

Da mesma forma, o modelo de correlação item-a-item utiliza os dados gerados pelo componente correlação item-a-item do modelo proposto. Porém, para gerar as dez recomendações é usado o método apresentado por Karypis (2000), o qual foi descrito no item 3.5.1.3, com uma única ressalva: para Karypis a cesta de compras é formada pelos itens comprados anteriormente pelo cliente e neste caso a cesta de compras é formada apenas pelos produtos da última compra, os quais representam a cesta de compras do teste. Conseqüentemente, o passado histórico do cliente não está sendo considerado e isso pode explicar, como será mostrado mais adiante, o baixo *recall* gerado por este modelo se comparado aos demais métodos.

O modelo dos itens mais vendidos (MMV), como o próprio nome diz, corresponde à seleção dos *N* produtos mais comprados pela comunidade de clientes considerados. As recomendações são únicas para todos os clientes.

A figura 5.5 mostra os resultados obtidos neste experimento.

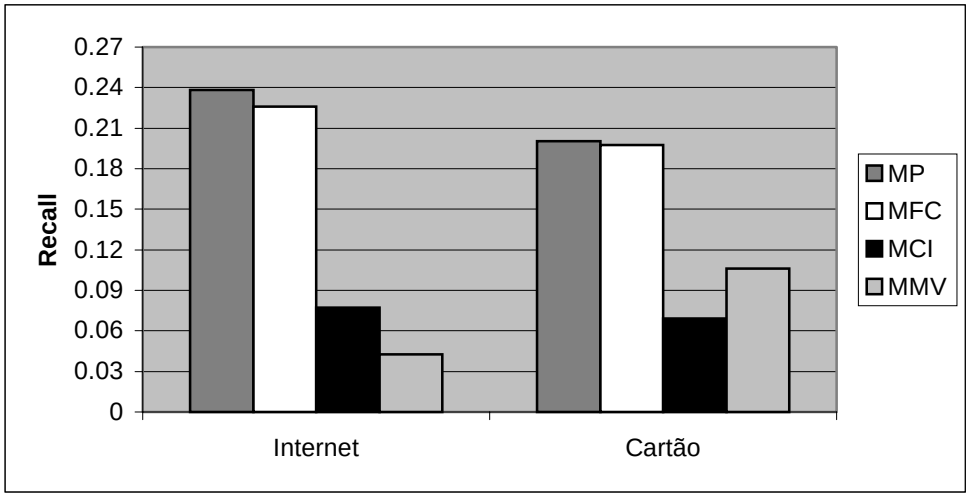


Figura 5.5 – Comparação do modelo proposto com os demais modelos

Como mostram as barras do gráfico, o modelo proposto obteve resultados ligeiramente melhores que os outros métodos de recomendação. É interessante observar que para o conjunto de dados da Internet os resultados foram melhores que para o conjunto de dados do cartão de fidelidade, mesmo este último tendo mais dados para construir o modelo. Isto talvez possa ser explicado pela natureza das transações nos ambientes de compra distintos. No caso do varejo supermercadista on-line, a cesta de compras geralmente apresenta uma maior quantidade de produtos se comparada ao varejo tradicional e o comportamento de compra é mais homogêneo, o que possibilita uma captura mais precisa do comportamento do cliente.

Uma diferença significativa entre os dois conjuntos de dados é entre o MMV e o MCI. Pelos mesmos motivos citados acima, o MCI se sai melhor que o MMV no caso das compras via internet, mas o mesmo não ocorre no varejo tradicional.

Pode-se concluir também que tanto o MP quanto o MFC são muito mais eficientes para fazer recomendações aos clientes que o MMV, além de fornecerem um serviço personalizado. Isso nos leva a ressaltar que mais do que os resultados positivos alcançados pelo MP nesta comparação, ele apresenta muitos outros benefícios além de predizer o que cada cliente quer comprar, por exemplo *cross-sell* e *up-sell*, como também as outras vantagens citadas no capítulo 2.

Os resultados superiores do MP e MFC em relação ao MMV também se prestam para observar que as estratégias utilizadas para recomendar produtos que são comprados de forma repetitiva – representação da freqüência de compras na matriz cliente-produto e inserção de um peso sobre os produtos comprados pelo cliente-alvo na geração das recomendações – realmente funcionam. Em vista destes resultados, pode-se atestar a validade da utilização de sistemas de recomendação para lojas que comercializam produtos desta forma.

Outro aspecto a ser considerado é que todos os modelos incluídos na comparação podem ser utilizados pelos sites de comércio eletrônico em diferentes situações do relacionamento com o cliente. Por exemplo, quando um prospecto visita a loja virtual, ainda não se têm informações sobre seu histórico, dessa forma, pode-se recomendar a ele os produtos mais vendidos. Após o visitante selecionar um ou mais itens e colocá-los na sua cesta de compras, pode-se indicar produtos com base na correlação item-a-item. Na sua volta à loja, o então já cliente pode receber recomendações personalizadas baseadas nos itens anteriormente comprados e nos seus vizinhos antes mesmo de colocar alguns itens na cesta. Por fim, depois de selecionados alguns itens, as recomendações poderão ser ainda mais apuradas utilizando o modelo híbrido proposto.

6 Conclusão

O objetivo do presente trabalho foi criar um modelo genérico para a implementação de sistemas de personalização baseados em sistemas de recomendação. A personalização é uma estratégia muito interessante para o CRM, pois mantém o cliente satisfeito ao mesmo tempo em que traz lucros para as organizações através de *cross-sell* e *up-sell*.

Os sistemas de recomendação têm alcançado destaque na web em decorrência do constante crescimento da quantidade de informações e produtos aos quais as pessoas são submetidas, como também pela busca de ferramentas que possam suprir as estratégias das empresas centradas no cliente.

Este trabalho apresentou uma revisão bibliográfica dos sistemas de recomendação, incluindo sua taxonomia e os dois métodos mais conhecidos e utilizados: a correlação item-a-item e a correlação pessoa-a-pessoa. Observou-se na literatura estudada que características do domínio dos negócios, como a freqüência de compra dos produtos, em geral não são consideradas.

O modelo que foi proposto utilizou uma combinação dos algoritmos de recomendação baseados em correlação de clientes e correlação de produtos para gerar ofertas personalizadas. Além do desenvolvimento deste novo método híbrido, o modelo criado incluiu aspectos como freqüência e quantidade das compras, possibilitando a implementação das técnicas de recomendação em uma gama maior de aplicações de comércio.

O plano de desenvolvimento apresentado incluiu a concepção do modelo de sistema de recomendação, a construção de uma aplicação de geração de recomendações e de uma aplicação de simulação de compras em massa e a avaliação experimental do modelo. As informações utilizadas para conduzir os testes foram obtidas através de uma

empresa proprietária de um supermercado, que forneceu dados de movimentação de compras realizadas através da internet e do cartão de fidelidade.

Os resultados alcançados pelo modelo proposto, aplicando as medidas de similaridade cosseno para o componente de filtragem colaborativa e para o componente de correlação item-a-item, superaram em média a qualidade dos resultados obtidos utilizando os métodos de filtragem colaborativa e correlação item-a-item puros. Além disso, a comparação com o modelo dos produtos mais vendidos mostrou que os sistemas de recomendação são de grande valia também para empresas e clientes que vendem/compram produtos repetidamente. As estratégias de representação de freqüências na matriz cliente-produto e inserção do próprio cliente na geração dos melhores produtos mostraram-se válidas neste propósito.

Como continuidade deste trabalho, um próximo passo poderia ser a aplicação e teste do modelo proposto sobre dados de lojas virtuais que comercializam outros tipos de produtos ou até mesmo em domínios não comerciais para verificar se os bons resultados também se confirmam em situações diversas.

Outro desafio seria avaliar se a parte off-line do modelo prejudica o resultado das recomendações num contexto mais amplo e determinar a partir de que dimensões da base de dados o modelo proposto poderia ser executado totalmente em tempo real.

7 Referências Bibliográficas

BASU, Chumki; HIRSH, Haym; COEN, William. Recommendation as Classification: Using Social and Content-Based Information in Recommendations. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (15, 1998, Madison, WI). **Proceedings**… p. 714-720. Disponível em: <http://citeseer.nj.nec.com/basu98recommendation.html>. Acesso em: 25 outubro 2001.

BILLSUS, Daniel; PAZZANI, Michael J. Learning collaborative information filters. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING (15, 1998, Madison, WI). **Proceedings**… p. 46-54. Disponível em: <http://citeseer.nj.nec.com/billsus98learning.html>. Acesso em: 25 outubro 2001.

BREESE, John S.; HERCKERMAN, David; KADIE, Carl. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In: CONFERENCE ON UNCERTANTY IN ARTIFICIAL INTELIGENCE (14, 1998, San Francisco). **Proceedings**… p. 43-52.

GOLDBERG, D.; NICHOLS, D.; OKI, B. M. et al. Using Collaborative Filtering to Weave an Information Tapestry. **Communications of the ACM**, New York, v. 35, n. 12, p. 61-70, dez. 1992.

GOOD, N.; SCHAFER, J. B.; KONSTAN, J. et al. Combining Collaborative Filtering with Personal Agents for Better Recommendations. In: CONFERENCE OF THE AMERICAN ASSOCIATION OF ARTIFICAL INTELLIGENCE (1999). **Proceedings...** p. 439-446. Disponível em: <http://www.cs.umn.edu/Research/GroupLens/research.html>. Acesso em: 16 outubro 2001.

GORDON, Ian. **Marketing de Relacionamento**: estratégias, técnicas e tecnologias para conquistar cliente e mantê-los para sempre. Tradução: Mauro Pinheiro. São Paulo: Futura, 1998.

GREENBERG, Paul. **CRM at the Speed of Light**: Capturing and Keeping Customers in Internet Real Time. Berkeley: Osborne/McGraw-Hill, 2001.

HAFNER, Arthur W. **Pareto's Principle: The 80-20 Rule**. Disponível em: <http://library.shu.edu/HafnerAW/awh-th-math-pareto.htm>. Acesso: 25 novembro 2001.

HERLOCKER, J. L.; KONSTAN, J. A.; BORCHERS, A. et al. An Algorithmic Framework for Performing Collaborative Filtering. In: ANNUAL ACM CONFERENCE ON RESEARCH AND DEVELOPMENT IN INFORMATION RETRIEVAL (22, 1999, Berkeley, CA). **Proceedings**… New York: ACM Press, 1999. p. 230-237.

HILL, Will; STEAD, Larry; ROSENSTEIN, Mark et al. Recommending And Evaluating Choices In A Virtual Community Of Use. In: CONFERENCE ON HUMAN FACTORS AND COMPUTING SYSTEMS (1995). **Proceedings**… New York: ACM Press, 1995. p. 194-201.

KARYPIS, George. **Evaluation of Item-Based Top-N Recommendation Algorithms**. Relatório Técnico CS-TR-00-46. Computer Science Dept., University of Minnesota, 2000. Disponível em: <http://citeseer.nj.nec.com/article/karypis00evaluation.html>. Acesso em 06 novembro 2001.

KITTS, B.; FREED, D.; VRIEZE, M. Cross-sell: A Fast Promotion-Tunable Customer-item Recommendation Method Based on Conditionally Intedependent Probabilities. In: ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING (6, 2000, Boston, MA USA). **Proceedings**… New York: ACM Press, 2000. p. 437-446.

KONSTAN, Joseph A. et al. GroupLens: applying collaborative filtering to Usenet news. **Communications of the ACM**, New York, v. 40, n. 3, p. 77-87, mar. 1997.

LARSEN, Steve; TUTTEROW, Sam. **Developing the Personalization-Centric Enterprise through Collaborative Filtering and Rules-Based Technologies**.

Disponível em: <http://www.crmproject.com/documents.asp?grID175&d_ID=714#>. Acesso: 13 novembro 2001.

MACSKASSY, Sofus A. et al. **Intelligent Information Triage**. Disponível em: <http://citeseer.nj.nec.com/macskassy01intelligent.html>. Acesso em 25 outubro 2001.

RESNICK, Paul et al. GroupLens: an open architecture for collaborative filtering of netnews. In: CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK (1994, Chapel Hill, NC, USA). **Proceedings**… New York: ACM Press, 1994. p. 175-186.

RESNICK, Paul; VARIAN, Hal R. Recommender Systems. **Communications of the ACM**, New York, v. 40, n. 3, p. 56-58, mar. 1997.

ROMANO, Nicholas C. Jr. Customer Relationship Management Research: An assessment of Sub Field Development and Maturity. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (34, 2001). **Proceedings**…

SARWAR, Badrul M. et al. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In: COMPUTER SUPPORTED COOPERATIVE WORK **(**1998). **Proceedings**… New York: ACM Press, 1998. p. 345-354.

SARWAR, Badrul; KARYPIS, George; KONSTAN, Joseph; RIEDL, John. Analysis of Recommendation Algorithms for E-Commerce. In: ACM CONFERENCE ON ELECTRONIC COMMERCE (2, 2000, Minneapolis, MN, EUA). **Proceedings**... ACM Press, 2000. p. 158 – 167.

SARWAR, Badrul; KARYPIS, George; KONSTAN, Joseph et al. Item-based Collaborative Filtering Recommendation Algorithms. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE (10, 2001, Hong Kong). **Proceedings**… ACM Press, 2001. p. 285-295.

SCHAFER, Ben J.; KONSTAN, Joseph; RIEDL, John. Recommender Systems in E-Commerce. In: ACM CONFERENCE ON ELECTRONIC COMMERCE (1, 1999, Denver, Colorado, EUA). **Proceedings**… New York: ACM Press, 1999. p. 158-166.

SCHAFER, Ben J.; KONSTAN, Joseph; RIEDL, John. **E-Commerce Recommendation Applications**. Disponível em: <http://citeseer.nj.nec.com/schafer01ecommerce.html>. Acesso em: 25 outubro 2001.

SHARDANAND, Upendra; MAES, Pattie. Social Information Filtering: Algorithms for Automating "Word of Mouth". In: CONFERENCE ON HUMAN FACTORS AND COMPUTING SYSTEMS (1995). **Proceedings**… New York: ACM Press, 1995. p. 210-217.

SHARMA, Arun; LAMBERT, Douglas M. Segmentation of Markets based on Customer Service. In: PAYNE, A. et al. **Relationship Marketing for Competitive Advantage**. Oxford: Butterworth, 1998, p. 137 – 153.

SINDELL, Kathleen. **Loyalty Marketing for the Internet Age**: How to Identify, Attract, Serve, and Retain Customers in an E-commerce Environment. EUA: Dearborn Trade, 2000.

SWETS, John A. Measuring the Accuracy of Diagnostic Systems. **Science**, v. 240, p. 1285 – 1293, jun. 1988.

WENRICH, Thomas A.; BECERRA , Jorge. **Online Retailing in Latin America 3.0**: Breaking Constraints. Relatório do Boston Consulting Group, 2001. Disponível em: <www.bcg.com>. Acesso em: 14 novembro 2001.

YANG, Yiming; LIU, Xin. A re-examination of text categorization methods. In: ANNUAL ACM CONFERENCE ON RESEARCH AND DEVELOPMENT IN INFORMATION RETRIEVAL (22, 1999, Berkeley, CA, USA). **Proceedings**… New York: ACM Press, 1999. p 42-49.

ANEXO 1 – Código de Implementação das Aplicações

Aplicação 1 – RECOM

```
/*********************************************************************************************/

package recom;

/**
 * Classe:    GeradorRecomendacoes
 * Descrição: Gerencia o processamento dos correlacionadores
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;
import javax.swing.SwingUtilities;


public class GeradorRecomendacoes {

  static public final int FASE0 =  0;
  static public final int FASE1 =  1;
  static public final int FASE1B = 4;
  static public final int FASE2 =  2;
  static public final int FASE3 =  3;

  protected GerenciadorBD               gerenciadorBD;
  protected CorrelacionadorClientes     correlacionadorCli;
  protected CorrelacionadorProdutosCesta correlacionadorProdCesta;
  protected MatrizClienteProduto        matrizCliProd;
  protected MatrizCestasProduto         matrizCestas;
  protected Interface                   frame;

  protected int     faseAtual;
  protected boolean processoFinalizado;

  public GeradorRecomendacoes(Interface frame) {
    this.frame = frame;
    faseAtual = FASE0;
    processoFinalizado = false;
  }

  /**
   * Inicia o processamento em uma outra linha de execução para que a Interface
   * permaneça disponível
   */
  public void inicie(boolean exec2, boolean exec3, int nVizinhos, int nMelhoresProdutos,
int nMelhoresProdProd) {

    final int n = nVizinhos;
    final int m = nMelhoresProdutos;
    final int mProd = nMelhoresProdProd;
    final boolean execTarefa2 = exec2;
    final boolean execTarefa3 = exec3;
    final SwingWorker worker = new SwingWorker() {
      public Object construct() {
         return executeProcesso(execTarefa2, execTarefa3, n, m, mProd);
      }
     };
     worker.start();
  }
```

```
  /**
   * Processa os dados de movimentação dos clientes através da correlação de
   * clientes e de produtos
   */
  public Object executeProcesso(boolean exec2, boolean exec3, int nVizinhos, int
nMelhoresProdutos, int nMProd) {

    // inicia o gerenciador do BD e cria a matriz cliente-produto
    gerenciadorBD = new GerenciadorBD();
    gerenciadorBD.conecte("sun.jdbc.odbc.JdbcOdbcDriver", "jdbc:odbc:recom");

    // Monta a matriz cliente-produto
    faseAtual = FASE1;
    matrizCliProd = gerenciadorBD.obtenhaMatrizClienteProduto();

    // Monta a matriz cestas
    frame.configureBarra1(0, gerenciadorBD.obtenhaTotalLinhasMatrizCesta());
    faseAtual = FASE1B;
    matrizCestas = gerenciadorBD.obtenhaMatrizCestasProduto();

    frame.completeTarefa1();
    frame.configureBarra2(0, matrizCliProd.retorneNClientes());
    frame.configureBarra3(0, matrizCestas.retorneNProdutos());

    // Cálculo correlação pessoa-a-pessoa
    if(exec2) {
      faseAtual = FASE2;
      correlacionadorCli = new CorrelacionadorClientes(matrizCliProd);
      correlacionadorCli.construaMatrizCorrelacao(gerenciadorBD, nVizinhos,
nMelhoresProdutos);
      frame.completeTarefa2();
    }

    // Cálculo correlação item-a-item
    if(exec3) {
      faseAtual = FASE3;
      correlacionadorProdCesta = new CorrelacionadorProdutosCesta(matrizCestas);
      correlacionadorProdCesta.construaMatrizCorrelacao(gerenciadorBD, 0, nMProd);
      frame.completeTarefa3();
    }

    // Encerra a conexao com a base de dados
    gerenciadorBD.encerreConexao();

    processoFinalizado = true;

    return new Object();
  }

  public int retorneProgressoCorrCli() {
    if(correlacionadorCli!=null) {
      return correlacionadorCli.retorneProgressoLinhas();
    }
    else {
      return 0;
    }
  }

  public String retorneStatusCorrClientes() {
    if(correlacionadorCli!=null) {
      return correlacionadorCli.retorneStatus();
    }
    else {
      return " ";
    }
  }

  public boolean processoFinalizado() {
    return processoFinalizado;
  }

  public int retorneFaseAtual() {
    return faseAtual;
```

```
  }

  public int retorneProgressoCorrItens() {
    if(correlacionadorProdCesta!=null) {
      return correlacionadorProdCesta.retorneProgresso();
    }
    else {
      return 0;
    }
  }

  public String retorneStatusCorrItens() {
    if(correlacionadorProdCesta!=null) {
      return correlacionadorProdCesta.retorneStatus();
    }
    else {
      return " ";
    }
  }

  public int retorneTamCorrItens() {
    if(correlacionadorProdCesta!=null) {
      return correlacionadorProdCesta.retorneTamanhoTarefa();
    }
    else {
      return 0;
    }
  }

  public String retorneStatusMatrizCliProd() {
    return gerenciadorBD.retorneStatus();
  }

  public int retorneProgressoMontagemMatriz() {
    return gerenciadorBD.retorneProgresso();
  }

  public int retorneTotalTarefaMontagemMatriz() {
    return gerenciadorBD.retorneTotalLinhas();
  }
}

/*******************************************************************************/

package recom;

/**
 * Classe:    Interface
 * Descrição: Implementa a interface gráfica com o usuário
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Calendar;

public class Interface extends JFrame{

  public final static int UM_SEGUNDO = 1000;

  private GeradorRecomendacoes geradorRecomendacoes;
  private Timer timer;

  JPanel pane;
  JPanel pane0;
  JPanel pane1;
  JPanel pane11;
  JPanel pane12;
  JPanel pane2;
  JPanel pane21;
  JPanel pane22;
```

```
JPanel pane23;
JPanel pane3;
JPanel pane31;
JPanel pane32;
JPanel pane33;
JPanel pane4;
JPanel pane5;

JLabel lbTitulo;

JButton bt1;
JLabel  lbTit1;
JLabel  lbSts1;
JProgressBar pBar1;

JButton bt2;
JLabel  lbTit2;
JLabel  lbSts2;
JProgressBar pBar2;
JLabel lbNViz;
JLabel lbNProd2;
JLabel lbNProd3;
JTextField tfNViz;
JTextField tfNProd2;
JTextField tfNProd3;

JButton bt3;
JLabel  lbTit3;
JLabel  lbSts3;
JProgressBar pBar3;

JLabel  lbHoraInicio;
JLabel  lbHoraFim;
JButton btExecutar;
JButton btCancelar;

Border bordaVazia5;
Border bordaVazia10;
Border bordaVazia15;
Border bordaSaliente;
Border bordaComposta;
Border bordaTarefas;
Border bordaBt;
Border bordaBt1;
Border bordaBt2;
Border bordaBt3;
Border bordaLado5;
Border bordaLado10;

Color azulClaro;
Color azulMedio;
Color azulEscuro;

Font fonteBotao;

Calendar hora;

int valorPB1;
int valorPB2;
String texto1;
String texto2;
boolean processoFinalizado;

public Interface() {

  super("Recom");

  geradorRecomendacoes = new GeradorRecomendacoes(this);

  pane = new JPanel();
  pane0 = new JPanel();
  pane1 = new JPanel();
  pane11 = new JPanel();
  pane12 = new JPanel();
```

```
pane2 = new JPanel();
pane21 = new JPanel();
pane22 = new JPanel();
pane23 = new JPanel();
pane3 = new JPanel();
pane31 = new JPanel();
pane32 = new JPanel();
pane33 = new JPanel();
pane4 = new JPanel();
pane5 = new JPanel();

pane0.setBackground(azulClaro);
pane0.setForeground(azulClaro);

lbTitulo = new JLabel("Projeto Recom - Geração das Tabelas de Correlação");

bt1 = new JButton("1");
lbTit1 = new JLabel("Montagem da matriz cliente-produto");
lbSts1 = new JLabel(" ");
pBar1 = new JProgressBar(0,0);

bt2 = new JButton("2");
lbTit2 = new JLabel("Cálculo da correlação entre clientes");
lbSts2 = new JLabel(" ");
pBar2 = new JProgressBar(0,0);
lbNViz = new JLabel("Vizinhos");
lbNProd2 = new JLabel("Produtos");
lbNProd3 = new JLabel("Produtos");
tfNViz = new JTextField("20", 4);
tfNProd2 = new JTextField("10", 4);
tfNProd3 = new JTextField("10", 4);
tfNViz.setMaximumSize(new Dimension(25, 20));
tfNProd2.setMaximumSize(new Dimension(25, 20));
tfNProd3.setMaximumSize(new Dimension(25, 20));
tfNViz.setHorizontalAlignment(JTextField.RIGHT);
tfNProd2.setHorizontalAlignment(JTextField.RIGHT);
tfNProd3.setHorizontalAlignment(JTextField.RIGHT);

bt3 = new JButton("3");
lbTit3 = new JLabel("Cálculo da correlação entre produtos");
lbSts3 = new JLabel(" ");
pBar3 = new JProgressBar(0,0);

lbHoraInicio = new JLabel();
lbHoraFim    = new JLabel();
btExecutar = new JButton("Executar");
btCancelar = new JButton("Cancelar");

azulClaro = new Color(197, 205, 212);
azulMedio = new Color(132, 155, 177);
azulEscuro = new Color(63, 99, 133);

fonteBotao = new Font("Comic Sans MS", Font.BOLD, 26);

hora = Calendar.getInstance();

pane.setLayout(new BorderLayout());
pane0.setLayout(new FlowLayout());
pane1.setLayout(new BorderLayout());
pane11.setLayout(new BorderLayout());
pane12.setLayout(new BorderLayout());
pane2.setLayout(new BorderLayout());
pane21.setLayout(new BorderLayout());
pane22.setLayout(new BorderLayout());
pane23.setLayout(new BoxLayout(pane23, BoxLayout.X_AXIS));
pane3.setLayout(new BorderLayout());
pane31.setLayout(new BorderLayout());
pane32.setLayout(new BorderLayout());
pane33.setLayout(new BoxLayout(pane33, BoxLayout.X_AXIS));
pane4.setLayout(new BoxLayout(pane4, BoxLayout.X_AXIS));
pane5.setLayout(new BoxLayout(pane5, BoxLayout.Y_AXIS));

pane0.setBackground(azulClaro);
pane1.setBackground(azulClaro);
```

```
    pane11.setBackground(azulClaro);
    pane12.setBackground(azulClaro);
    pane2.setBackground(azulClaro);
    pane21.setBackground(azulClaro);
    pane22.setBackground(azulClaro);
    pane23.setBackground(azulClaro);
    pane3.setBackground(azulClaro);
    pane31.setBackground(azulClaro);
    pane32.setBackground(azulClaro);
    pane33.setBackground(azulClaro);
    pane4.setBackground(azulClaro);
    pane5.setBackground(azulClaro);

    bordaLado5   = BorderFactory.createEmptyBorder(0,0,0,5);
    bordaLado10   = BorderFactory.createEmptyBorder(0,0,0,10);
    bordaSaliente = BorderFactory.createEtchedBorder(azulEscuro, Color.white);
    bordaVazia10  = BorderFactory.createEmptyBorder(10,10,10,10);
    bordaComposta = BorderFactory.createCompoundBorder(bordaSaliente, bordaVazia10);
    bordaTarefas  = BorderFactory.createCompoundBorder(bordaVazia10, bordaComposta);
    bordaVazia15  = BorderFactory.createEmptyBorder(15,15,15,15);
    bordaVazia5   = BorderFactory.createEmptyBorder(5,5,5,5);
    bordaBt2      = BorderFactory.createBevelBorder(BevelBorder.LOWERED, azulEscuro,
Color.white);
    bordaBt3      = BorderFactory.createCompoundBorder(bordaBt2, bordaLado10);
    bordaBt       = BorderFactory.createCompoundBorder(bordaBt2, bordaVazia5);

    lbTitulo.setForeground(azulEscuro);
    lbTitulo.setFont(new Font("Comic Sans MS", Font.BOLD, 22));

    pane0.add(lbTitulo, null);
    pane0.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

    pBar1.setValue(0);
    pBar1.setBorderPainted(true);
    pBar1.setStringPainted(true);
    pBar1.setForeground(azulMedio);
    pBar1.setMinimumSize(new Dimension(500, 20));
    bt1.setBorder(bordaBt);
    bt1.setBackground(azulMedio);
    bt1.setForeground(Color.white);
    bt1.setFont(fonteBotao);
    bt1.setMinimumSize(new Dimension(50,50));
    lbTit1.setForeground(azulEscuro);
    lbTit1.setBorder(bordaVazia5);
    lbSts1.setForeground(azulEscuro);
    lbSts1.setMinimumSize(new Dimension(50, 20));
    lbSts1.setMaximumSize(new Dimension(50, 20));
    lbSts1.setBorder(bordaLado5);

    pane12.add(lbSts1, BorderLayout.WEST);
    pane12.add(pBar1, BorderLayout.CENTER);
    pane11.add(lbTit1, BorderLayout.NORTH);
    pane11.add(pane12, BorderLayout.CENTER);
    pane11.setBorder(bordaVazia5);
    pane1.add(bt1, BorderLayout.WEST);
    pane1.add(pane11, BorderLayout.CENTER);
    pane1.setBorder(bordaTarefas);

    pBar2.setValue(0);
    pBar2.setBorderPainted(true);
    pBar2.setStringPainted(true);
    pBar2.setForeground(azulMedio);
    bt2.setBorder(bordaBt);
    bt2.setBackground(azulMedio);
    bt2.setForeground(Color.white);
    bt2.setFont(fonteBotao);
    bt2.setMinimumSize(new Dimension(50,50));
    bt2.addActionListener(new Bt2Listener());
    lbTit2.setForeground(azulEscuro);
    lbTit2.setBorder(bordaVazia5);
    lbSts2.setForeground(azulEscuro);
    lbSts2.setBorder(bordaLado5);
    lbNViz.setForeground(azulEscuro);
    lbNProd2.setForeground(azulEscuro);
```

```
lbNProd3.setForeground(azulEscuro);

pane23.add(lbTit2);
pane23.add(Box.createHorizontalGlue());
pane23.add(lbNViz);
pane23.add(Box.createRigidArea(new Dimension(2,0)));
pane23.add(tfNViz);
pane23.add(Box.createRigidArea(new Dimension(5,0)));
pane23.add(lbNProd2);
pane23.add(Box.createRigidArea(new Dimension(2,0)));
pane23.add(tfNProd2);
pane22.add(lbSts2, BorderLayout.WEST);
pane22.add(pBar2, BorderLayout.CENTER);
pane21.add(pane23, BorderLayout.NORTH);
pane21.add(pane22, BorderLayout.CENTER);
pane21.setBorder(bordaVazia5);
pane2.add(bt2, BorderLayout.WEST);
pane2.add(pane21, BorderLayout.CENTER);
pane2.setBorder(bordaTarefas);

pBar3.setValue(0);
pBar3.setBorderPainted(true);
pBar3.setStringPainted(true);
pBar3.setForeground(azulMedio);
pBar3.setMinimumSize(new Dimension(500, 20));
bt3.setBorder(bordaBt);
bt3.setBackground(azulMedio);
bt3.setForeground(Color.white);
bt3.setFont(fonteBotao);
bt3.addActionListener(new Bt3Listener());
bt3.setMinimumSize(new Dimension(50,50));
lbTit3.setForeground(azulEscuro);
lbTit3.setBorder(bordaVazia5);
lbSts3.setForeground(azulEscuro);
lbSts3.setBorder(bordaLado5);

pane33.add(lbTit3);
pane33.add(Box.createHorizontalGlue());
pane33.add(Box.createRigidArea(new Dimension(5,0)));
pane33.add(lbNProd3);
pane33.add(Box.createRigidArea(new Dimension(2,0)));
pane33.add(tfNProd3);
pane32.add(lbSts3, BorderLayout.WEST);
pane32.add(pBar3, BorderLayout.CENTER);
pane31.add(pane33, BorderLayout.NORTH);
pane31.add(pane32, BorderLayout.CENTER);
pane31.setBorder(bordaVazia5);
pane3.add(bt3, BorderLayout.WEST);
pane3.add(pane31, BorderLayout.CENTER);
pane3.setBorder(bordaTarefas);

pane5.add(pane1);
pane5.add(pane2);
pane5.add(pane3);
pane5.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

// Registra listeners para os botões
btExecutar.addActionListener(new BtExecutarListener());
btCancelar.addActionListener(new BtCancelarListener());

btExecutar.setBackground(azulMedio);
btCancelar.setBackground(azulMedio);
lbHoraInicio.setForeground(azulEscuro);
lbHoraFim.setForeground(azulEscuro);
pane4.add(Box.createHorizontalGlue());
pane4.add(lbHoraInicio);
pane4.add(Box.createRigidArea(new Dimension(5,0)));
pane4.add(lbHoraFim);
pane4.add(Box.createRigidArea(new Dimension(5,0)));
pane4.add(btCancelar);
pane4.add(Box.createRigidArea(new Dimension(5,0)));
pane4.add(btExecutar);
pane4.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
```

```
    pane.add(pane0, BorderLayout.NORTH);
    pane.add(pane5, BorderLayout.CENTER);
    pane.add(pane4, BorderLayout.SOUTH);

    this.getContentPane().add(pane);
    this.setTitle("Recom");

    //Cria o timer
    timer = new Timer(UM_SEGUNDO, new ActionListener() {
          public void actionPerformed(ActionEvent evt) {
              switch(geradorRecomendacoes.retorneFaseAtual()) {
               case GeradorRecomendacoes.FASE0 :
                 break;
               case GeradorRecomendacoes.FASE1 :
                 lbSts1.setText(geradorRecomendacoes.retorneStatusMatrizCliProd());
                 pBar1.setValue(geradorRecomendacoes.retorneProgressoMontagemMatriz());
                  break;
               case GeradorRecomendacoes.FASE1B :
                 lbSts1.setText(geradorRecomendacoes.retorneStatusMatrizCliProd());
                 pBar1.setValue(geradorRecomendacoes.retorneProgressoMontagemMatriz());
                  break;
              case GeradorRecomendacoes.FASE2 :
                 lbSts2.setText(geradorRecomendacoes.retorneStatusCorrClientes());
                  pBar2.setValue(geradorRecomendacoes.retorneProgressoCorrCli());
                 break;
              case GeradorRecomendacoes.FASE3 :
                 lbSts3.setText(geradorRecomendacoes.retorneStatusCorrItens());
                  pBar3.setValue(geradorRecomendacoes.retorneProgressoCorrItens());
                 break;
             }
              if(geradorRecomendacoes.processoFinalizado()) {
                Toolkit.getDefaultToolkit().beep();
                timer.stop();
               hora = Calendar.getInstance();
                lbHoraFim.setText("Término: "+hora.get(Calendar.HOUR)+":"+
                   hora.get(Calendar.MINUTE)+":"+hora.get(Calendar.SECOND));
                btExecutar.setEnabled(true);
                btCancelar.setText("Fechar");
              }
            }
        });

  }

    class BtExecutarListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
          btExecutar.setEnabled(false);
          boolean exec2 = bt2.getText().equals("2");
          boolean exec3 = bt3.getText().equals("3");
          geradorRecomendacoes.inicie(exec2, exec3, Integer.parseInt(tfNViz.getText()),
                              Integer.parseInt(tfNProd2.getText()),
Integer.parseInt(tfNProd3.getText()));
          lbHoraInicio.setText("Início: " + hora.get(Calendar.HOUR)+":"+
                     hora.get(Calendar.MINUTE)+":"+hora.get(Calendar.SECOND));
           btCancelar.setText("Cancelar");
          timer.start();
        }
    }

    class BtCancelarListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
           System.exit(0);
        }
    }

    class Bt2Listener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
          String  texto = e.getActionCommand();
          if(bt2.getText().equals("X")) {
              bt2.setText("2");
             bt2.setForeground(Color.white);
          }
          else {
              bt2.setText("X");
```

```
            bt2.setForeground(Color.red);
          }
        }
    }

    class Bt3Listener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
          String  texto = e.getActionCommand();
          if(bt3.getText().equals("X")) {
              bt3.setText("3");
             bt3.setForeground(Color.white);
          }
          else {
              bt3.setText("X");
             bt3.setForeground(Color.red);
          }
        }
    }

    public void finalizaAplicacao() {
      this.setVisible(false);
      System.exit(0);
    }

    public void configureBarra1(int min, int max) {
      pBar1.setMinimum(min);
      pBar1.setMaximum(max);
    }

    public void configureBarra2(int min, int max) {
      pBar2.setMinimum(min);
      pBar2.setMaximum(max);
    }

    public void configureBarra3(int min, int max) {
      pBar3.setMinimum(min);
      pBar3.setMaximum(max);
    }

    public void completeTarefa1() {
      pBar1.setValue(pBar1.getMaximum());
      lbSts1.setText(pBar1.getMaximum() + " de " + pBar1.getMaximum());
    }

    public void completeTarefa2() {
      pBar2.setValue(pBar2.getMaximum());
      lbSts2.setText(pBar2.getMaximum() + " de " + pBar2.getMaximum());
    }

    public void completeTarefa3() {
      pBar3.setValue(pBar3.getMaximum());
      lbSts3.setText(pBar3.getMaximum() + " de " + pBar3.getMaximum());
    }

  public static void main(String[] args) {

    Interface frame = new Interface();

    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });

    frame.pack();
    frame.setVisible(true);

  }
}

/*******************************************************************************/

package recom;
```

```
/**
 * Classe:    GerenciadorBD
 * Descrição: Realiza os processos de consulta e atualização de dados no BD
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;
import java.sql.*;

public class GerenciadorBD {

  protected Connection conexao;
  protected String     status;
  protected int        progresso = 0;
  protected int        totalLinhas;

  public GerenciadorBD() {
    status = " ";
  }

  public boolean conecte(String driver, String url) {
    boolean resultado;
    resultado = true;

    try {
       Class.forName(driver);
    }
    catch (Exception e) {
       System.out.println(e.getMessage());
       resultado = false;
    }

    try {
       conexao = DriverManager.getConnection(url);
    }
    catch (Exception e) {
       System.out.println(e.getMessage());
       resultado = false;
    }
    return resultado;
  }

  public void encerreConexao() {
     try {
        conexao.close();
     }
     catch (Exception e) {
        System.out.println(e.getMessage());
     }
  }

  /**
   *  Retorna a quantidade de linhas da tabela de compras
   */
  public int obtenhaTotalLinhas() {
     Statement stmt;
     ResultSet result;
     String sql;
     int retorno = 0;
     try {
        stmt = conexao.createStatement();
        sql = "SELECT COUNT(cod_cli) as cont FROM compras";
        result = stmt.executeQuery(sql);
       result.next();
       retorno = result.getInt("cont");
     }
     catch (SQLException e) {
       System.out.println(e.getMessage());
     }
     totalLinhas = retorno;
     return retorno;
  }
```

```
  /**
   * Retorna a quantidade de linhas da tabela de cestas de compras
   */
  public int obtenhaTotalLinhasMatrizCesta() {
     Statement stmt;
     ResultSet result;
     String sql;
     int retorno = 0;
     try {
        stmt = conexao.createStatement();
        sql = "SELECT COUNT(cod_cesta) as cont FROM cestas";
        result = stmt.executeQuery(sql);
       result.next();
       retorno = result.getInt("cont");
     }
     catch (SQLException e) {
       System.out.println(e.getMessage());
     }
     totalLinhas = retorno;
     return retorno;
  }

  /**
   * Retorna a matriz cliente-produto
   */
  public MatrizClienteProduto obtenhaMatrizClienteProduto() {
     MatrizClienteProduto matriz = new MatrizClienteProduto();
     HashMap cliente;
     Statement stmt;
     ResultSet result;
     float valor;
     String sql;

     try {
        stmt = conexao.createStatement();
        sql = "SELECT * FROM compras, clientes WHERE compras.cod_cli = clientes.cod_cli
ORDER BY compras.cod_cli, cod_prod";
        result = stmt.executeQuery(sql);
       progresso = 0;
        while(result.next()) {
           try {
              valor =
result.getFloat("quantidade")/result.getFloat("qtd_transacoes")/result.getFloat("soma_fr
ac");
           }
           catch (Exception e) {
              valor = 0;
              System.out.println(e.getMessage());
           }
           matriz.insiraValorProdutoCliente(result.getString("cod_cli"),
                result.getString("cod_prod"), valor);
          progresso++;
          status = progresso + " de " + totalLinhas;
        }
        result.close();
        stmt.close();
     }
     catch (Exception e) {
        System.out.println(e.getMessage());
     }

     return matriz;
  }

  /**
   * Retorna a matriz cesta-produto
   */
  public MatrizCestasProduto obtenhaMatrizCestasProduto() {
     MatrizCestasProduto matriz = new MatrizCestasProduto();
     HashMap cliente;
     Statement stmt;
     ResultSet result;
     float valor;
     String sql;
```

```
     progresso = 0;

     try {
        stmt = conexao.createStatement();

        sql = "SELECT * FROM cestas";
        result = stmt.executeQuery(sql);
       progresso = 0;
        while(result.next()) {
           try {
              valor = result.getFloat("quantidade");
           }
           catch (Exception e) {
              valor = 0;
              System.out.println(e.getMessage());
           }
           matriz.insiraValorProdutoCestaCliente(result.getString("cod_cesta"),
                result.getString("cod_prod"), valor);
          progresso++;
          status = progresso + " de " + totalLinhas;
        }
        result.close();
        stmt.close();
     }
     catch (Exception e) {
        System.out.println(e.getMessage());
     }

     return matriz;
  }

  /**
   * Armazena para um determinado cliente os m melhores produtos gerados pela
   * filtragem colaborativa
   */
  public boolean armazeneMMelhoresProdutos(Cliente cliente, String produtos) {
    Statement stmt;
    String sql;
    String valores;
    int retorno = 9999;
    boolean result = true;

    sql = "INSERT INTO melhores_produtos(cod_cli, cod_produtos) VALUES ('" +
cliente.retorneId();
    sql = sql + "', '" + produtos + "')";

    try {
      stmt = conexao.createStatement();
      try {
         retorno = stmt.executeUpdate(sql);
      }
      catch (SQLException e) {
         if(e.getErrorCode()==0) {
            sql = "UPDATE melhores_produtos SET cod_produtos = '" + produtos;
            sql = sql + "' WHERE cod_cli = '" + cliente.retorneId() + "'";
            try {
               stmt.executeUpdate(sql);
            }
            catch (SQLException eAtualizacao) {
              System.out.println("Problema na atualização dos dados -> " +
eAtualizacao.getMessage());
            }
         }
      }
      stmt.close();
    }
    catch (Exception e) {
      System.out.println(e.getMessage());
      result = false;
    }
    return true;
  }
```

```
  /**
   * Armazena determinado produto com seus produtos melhores correlacionados
   */
  public boolean armazeneMSimilaresProduto(String codProd, String produtos) {
    Statement stmt;
    String sql;
    int retorno = 9999;
    boolean result = true;

    sql = "INSERT INTO produtos_similares(cod_prod, cod_similares) VALUES ('" + codProd;
    sql = sql + "', '" + produtos + "')";

    try {
      stmt = conexao.createStatement();
      try {
         retorno = stmt.executeUpdate(sql);
      }
      catch (SQLException e) {
        if(e.getErrorCode()==0) {
           sql = "UPDATE produtos_similares SET cod_similares = '" + produtos;
           sql = sql + "' WHERE cod_prod = '" + codProd + "'";
           try {
              stmt.executeUpdate(sql);
           }
           catch (SQLException eAtualizacao) {
             System.out.println("Problema na atualização dos dados -> " +
eAtualizacao.getMessage());
           }
        }
      }
      stmt.close();
    }
    catch (Exception e) {
      System.out.println(e.getMessage());
      result = false;
    }
    return true;
  }

  /**
   * Armazena uma lista de pares de produtos com seu respectivo valor de
   * correlação
   */
  public boolean armazeneMatrizCorrelacaoItens(Vector matrizItens) {
    Statement stmt;
    String sql;
    Correlacao corr;

    for(int i=0; i<matrizItens.size(); i++) {
      corr = (Correlacao) matrizItens.get(i);
      if(corr.retorneCorrelacao() > 0) {
         sql = "INSERT INTO correlacao_produtos(cod_prod_a, cod_prod_b, correlacao)
VALUES ('";
         sql = sql + corr.retorneItemA() + "', '" + corr.retorneItemB() + "'," +
corr.retorneCorrelacao() + ")";
         try {
           stmt = conexao.createStatement();
           try {
             stmt.executeUpdate(sql);
           }
           catch (SQLException e) {
            System.out.println(e.getMessage());
           }
           stmt.close();
         }
         catch (SQLException e) {
           System.out.println(e.getMessage());
         }
      }
    }
    return true;
  }

  /**
```

```
   * Remove todos as linhas de uma tabela
   */
  public boolean removaDadosTabela(String tabela) {
    String sql;
    Statement stmt;
    boolean retorno = true;

    sql = "DELETE FROM " + tabela;
    try {
      stmt = conexao.createStatement() ;
      try {
       stmt.executeUpdate(sql);
      }
      catch (SQLException e) {
       retorno = false;
       System.out.println(e.getMessage());
      }
      stmt.close();
    }
    catch (Exception e) {
      retorno = false;
      System.out.println(e.getMessage());
    }
    return retorno;
  }

  public String retorneStatus() {
    return status;
  }

  public int retorneProgresso() {
    return progresso;
  }

  public int retorneTotalLinhas() {
    return obtenhaTotalLinhas();
  }

}

/*******************************************************************************************/

package recom;

/**
 * Classe:    CorrelacionadorClientes
 * Descrição: Implementa as funcionalidades da correlacao entre clientes
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;
import java.lang.Double;

public class CorrelacionadorClientes extends Correlacionador {

  final double PESO_SIGNIFICANCIA = 50.0;
  final int    PESO_CLIENTE       = 50;

  protected MatrizClienteProduto matrizCliProd;
  protected int progressoLinhas;
  protected int totalLinhas;
  protected String status;

  public CorrelacionadorClientes(MatrizClienteProduto matrizCliProd) {
     this.matrizCliProd = matrizCliProd;
  }

  /**
   * Calcula a similaridade através a correlacao de Pearson
   * */
  protected float retorneValorCorrelacaoPearson(String cliA, String cliB) {
     float    prodAi;
     float    prodBi;
```

```
float    mediaA;
float    mediaB;
float    somaA;
float    somaB;
float    numerador;
float    denominParA;
float    denominParB;
float    fatorSignificancia;
double   fator;
float    retorno;
int      cont;
double   raiz;
String   prodAtual;
HashMap  prodA;
HashMap  prodB;
Iterator nProd;
Float    denominador;

prodA = matrizCliProd.retorneProdutosCliente(cliA);
prodB = matrizCliProd.retorneProdutosCliente(cliB);

somaA=0;
somaB=0;
cont=0;
for(nProd = prodA.keySet().iterator(); nProd.hasNext();) {
   prodAtual = (String) nProd.next();
   if(prodB.containsKey(prodAtual)) {
    somaA = somaA + ((Float)prodA.get(prodAtual)).floatValue();
    somaB = somaB + ((Float)prodB.get(prodAtual)).floatValue();
    cont++;
   }
}

if(cont>0) {

  if(cont<PESO_SIGNIFICANCIA) {
     fator = cont/PESO_SIGNIFICANCIA;
    fatorSignificancia = (new Float(fator)).floatValue();
  }
  else {
     fatorSignificancia = 1;
  }

  mediaA = somaA/cont;
  mediaB = somaB/cont;

  numerador  = 0;
  denominParA = 0;
  denominParB = 0;
  for(nProd = prodA.keySet().iterator(); nProd.hasNext();) {
     prodAtual = (String) nProd.next();
     if(prodB.containsKey(prodAtual)) {
       prodAi = ((Float) prodA.get(prodAtual)).floatValue();
       prodBi = ((Float) prodB.get(prodAtual)).floatValue();
       numerador   = numerador   + (prodAi-mediaA)*(prodBi-mediaB);
       denominParA = denominParA + (prodAi-mediaA)*(prodAi-mediaA);
      denominParB = denominParB + (prodBi-mediaB)*(prodBi-mediaB);
     }
  }

  denominador = new Float(denominParA*denominParB);
  raiz = java.lang.Math.sqrt(denominador.doubleValue());
  denominador = new Float(raiz);
  if(denominador.floatValue()!=0) {
      retorno = numerador/denominador.floatValue()*fatorSignificancia;
  }
  else {
      retorno = -99999999;
  }
}
else {
  retorno = -99999999;
}
return retorno;
```

```
}

/**
 * Calcula a similaridade como o cosseno entre os vetores A e B
 * */
protected float retorneValorCosseno(String cliA, String cliB) {
   float valorAi;
   float valorBi;
   float numerador;
   float denominadorA;
   float denominadorB;
   float fatorSignificancia;
   double fator;
   float retorno;
   double raiz;
   int n;
   int cont;
   Float denominador;
   HashMap vetorA;
   HashMap vetorB;
   String prodAtual;

   vetorA = matrizCliProd.retorneProdutosCliente(cliA);
   vetorB = matrizCliProd.retorneProdutosCliente(cliB);

   numerador  = 0;
   denominadorA = 0;
   denominadorB = 0;
   cont = 0;
   for(Iterator nProd = vetorA.keySet().iterator(); nProd.hasNext();) {
      prodAtual = (String) nProd.next();
      if(vetorB .containsKey(prodAtual)) {
       cont++;

        valorAi = ((Float)vetorA.get(prodAtual)).floatValue();
        valorBi = ((Float)vetorB.get(prodAtual)).floatValue();

        numerador  = numerador  + valorAi*valorBi;
        denominadorA = denominadorA + valorAi*valorAi;
        denominadorB = denominadorB + valorBi*valorBi;
      }
   }

   if(numerador!=0 && cont>=2) {

     if(cont<PESO_SIGNIFICANCIA) {
         fator = cont/PESO_SIGNIFICANCIA;
         fatorSignificancia = (new Float(fator)).floatValue();
     }
     else {
        fatorSignificancia = 1;
     }

     denominador = new Float(denominadorA*denominadorB);
     raiz = java.lang.Math.sqrt(denominador.doubleValue());
     denominador = new Float(raiz);
     retorno = numerador/denominador.floatValue()*fatorSignificancia;
   }
   else {
    retorno = -99999999;
   }
   return retorno;
 }

/**
 * Calcula a correlação entre os clientes, seleciona os vizinhos e armazena
 *  os m melhores produtos para cada cliente.
 *  */
public boolean construaMatrizCorrelacao(GerenciadorBD gerBD, int nVizinhos,
                                                    int nProdutos) {
   Iterator chaves1;
   Iterator chaves2;
   HashMap   clientes;
   String      cli1;
```

```
      String      cli2;
      float       valorCorr;
      float       somaCorr;
      String      mProdutos;

      progressoLinhas = 0;
      totalLinhas = matrizCliProd.retorneNClientes();

      clientes = new HashMap();

      for(chaves1=matrizCliProd.retorneChavesClientes();chaves1.hasNext();) {
        clientes.clear();
        cli1 = (String) chaves1.next();
        somaCorr = 0;
        for(chaves2=matrizCliProd.retorneChavesClientes();chaves2.hasNext();) {
          cli2 = (String) chaves2.next();
          if(!cli1.equals(cli2)) {
             //valorCorr = retorneValorCorrelacaoPearson(cli1, cli2);
             valorCorr = retorneValorCosseno(cli1, cli2);
            if(valorCorr!=-99999999) {
                clientes.put(cli2, new Float(valorCorr));
                somaCorr = somaCorr+valorCorr;
            }
          }
        }
        normalize(clientes, somaCorr);
        mProdutos=retorneMMelhoresProdutos(cli1, clientes, nVizinhos, nProdutos);
        gerBD.armazeneMMelhoresProdutos(new Cliente(cli1), mProdutos);

        progressoLinhas++;
        status = progressoLinhas + " de " + totalLinhas;
      }
      return progressoLinhas >= totalLinhas;
    }


    /**
     * Retorna os M melhores produtos para um cliente a partir da correlacao com
     *  seus vizinhos
     */
    protected String retorneMMelhoresProdutos(String chaveCli,
                                    HashMap corrClientes, int nVizinhos, int m) {
      Vector  vizinhos;
      HashMap somaProdutos;
      HashMap produtos;
      Vector  produtosId;
      Vector  produtosValor;
      Object  prodAtual;
      float   valor;
      String  str;
      String  item;
      Vector  ordem;

      vizinhos = idValoresOrdenados(corrClientes, nVizinhos);
      vizinhos.add(chaveCli);  // adiciona o proprio cliente para que os produtos
                               // comprados por ele tenham peso tambem
      somaProdutos = new HashMap();

      if(!vizinhos.isEmpty() && vizinhos.firstElement()!=null) {
         produtos =
matrizCliProd.retorneProdutosCliente((String)vizinhos.firstElement());
        if(produtos!=null) {
          somaProdutos = (HashMap) produtos.clone();
          for(int i = 1; i<vizinhos.size(); i++) {
            String cliAtual = (String)vizinhos.get(i);
            if(cliAtual != null) {
              produtos = matrizCliProd.retorneProdutosCliente(cliAtual);
              if(produtos!=null) {
                for(Iterator j = produtos.keySet().iterator(); j.hasNext(); ) {
                  prodAtual = j.next();
                  valor = 0;
                  if(somaProdutos.containsKey(prodAtual)) {
                    valor = ((Float)somaProdutos.get(prodAtual)).floatValue();
                  }
```

```
                if(produtos.containsKey(prodAtual)) {
                  if(cliAtual.equals(chaveCli))
                      valor = valor +
((Float)produtos.get(prodAtual)).floatValue()*PESO_CLIENTE;
                  else
                      valor = valor + ((Float)produtos.get(prodAtual)).floatValue();
                }
                 somaProdutos.put(prodAtual, new Float(valor));
               }
             }
           }
         }
       }
    }

    str = "";
    ordem = idValoresOrdenados(somaProdutos, m);
    for(int i=0; i<ordem.size(); i++) {
      item = (String)ordem.elementAt(i);
      str = str + item + "=" + (Float)somaProdutos.get(item) + ", ";
    }
    str = str.substring(0,str.length()-2);
    return str;
  }

  /**
   *  Retorna um vetor com os identificadores ordenados pelos valores na tabela
   */
  protected Vector idValoresOrdenados(HashMap tabela, int n) {
     Vector listaId;
     Vector listaValor;
     Object elementoAtual;
     float valor;
     float valorOrd;
     boolean inseriu;
     int i;

     listaId = new Vector();
     listaValor = new Vector();

     try {
     for(Iterator e = tabela.keySet().iterator(); e.hasNext(); ) {
        elementoAtual = e.next();
        valor = ((Float)tabela.get(elementoAtual)).floatValue();
        inseriu = false;
        i = 0;
        while((!inseriu) && (i<listaId.size())) { //&& (i<n)
           valorOrd = ((Float)listaValor.get(i)).floatValue();
           if(valor > valorOrd) {
              listaId.insertElementAt(elementoAtual,i);
              listaValor.insertElementAt(new Float(valor),i);
              inseriu = true;
           }
           i++;
        }
        if((!inseriu) && (listaId.size() < n)) {
           listaId.addElement(elementoAtual);
           listaValor.addElement(new Float(valor));
        }
     }
     }
     catch (Exception e) {
       System.out.println(e.getMessage());
     }

     if(listaId.size() > n)
       listaId.setSize(n);
     return listaId;

  }

  public int retorneProgressoLinhas() {
    return progressoLinhas;
  }
```

```
  public int retorneTotalLinhas() {
    return totalLinhas;
  }

  public boolean processoFinalizado() {
    return progressoLinhas >= totalLinhas;
  }

  public String retorneStatus() {
    return status;
  }

  protected void normalize(HashMap lista, float total) {
    float valor;
    String item;
    for(Iterator e=lista.keySet().iterator(); e.hasNext(); ) {
      item = (String)e.next();
      try {
         valor = ((Float)lista.get(item)).floatValue();
        valor = valor/total;
        lista.put(item, new Float(valor));
      }
      catch (Exception ex) {}
    }
  }

}

/*******************************************************************************************/

package recom;

/**
 * Classe:    CorrelacionadorProdutosCesta
 * Descrição: Implementa as funcionalidades de correlação item-a-item para as
 *            cestas de compras
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;

public class CorrelacionadorProdutosCesta {

  protected MatrizCestasProduto  matrizCestas;
  protected Vector               listaCorrelacao;
  protected int                  progresso;
  protected int                  tamTarefa;
  protected String               status;

  final double PESO_SIGNIFICANCIA = 5.0;

  public CorrelacionadorProdutosCesta(MatrizCestasProduto matriz) {
    matrizCestas = matriz;
  }

  /**
   * Calcula a similaridade entre produtos através da correlação de Pearson
   */
  protected float retorneValorCorrelacaoPearson(String prodA, String prodB) {
     float    cliAi;
     float    cliBi;
     float    mediaA;
     float    mediaB;
     float    somaA;
     float    somaB;
     int      cont;
     float    numerador;
     float    denominParA;
     float    denominParB;
     float    fatorSignificancia;
     float    retorno;
     double   fator;
```

```
   double   raiz;
   String   cliAtual;
   HashMap  cliA;
   HashMap  cliB;
   Iterator nCli;
   Float    denominador;

   cliA = matrizCestas.retorneColunaProduto(prodA);
   cliB = matrizCestas.retorneColunaProduto(prodB);

   somaA=0;
   somaB=0;
   cont=0;
   for(nCli = cliA.keySet().iterator(); nCli.hasNext();) {
      cliAtual = (String) nCli.next();
      if(cliB.containsKey(cliAtual)) {
        somaA = somaA + ((Float)cliA.get(cliAtual)).floatValue();
        somaB = somaB + ((Float)cliB.get(cliAtual)).floatValue();
        cont++;
      }
   }

   if(cont>0) {

     mediaA = somaA/cont;
     mediaB = somaB/cont;

     if(cont<PESO_SIGNIFICANCIA) {
        fator = cont/PESO_SIGNIFICANCIA;
       fatorSignificancia = (new Float(fator)).floatValue();
     }
     else {
        fatorSignificancia = 1;
     }

     numerador  = 0;
     denominParA = 0;
     denominParB = 0;
     for(nCli = cliA.keySet().iterator(); nCli.hasNext();) {
       cliAtual = (String) nCli.next();
       if(cliB.containsKey(cliAtual)) {
         cliAi = ((Float) cliA.get(cliAtual)).floatValue();
         cliBi = ((Float) cliB.get(cliAtual)).floatValue();
         numerador   = numerador   + (cliAi-mediaA)*(cliBi-mediaB);
         denominParA = denominParA + (cliAi-mediaA)*(cliAi-mediaA);
        denominParB = denominParB + (cliBi-mediaB)*(cliBi-mediaB);
       }
     }

     denominador = new Float(denominParA*denominParB);
     raiz = java.lang.Math.sqrt(denominador.doubleValue());
     denominador = new Float(raiz);
     if(denominador.floatValue()!=0) {
        retorno = numerador/denominador.floatValue()*fatorSignificancia;
     }
     else {
        retorno = -99999999;
     }
   }
   else {
     retorno = -99999999;
   }
   return retorno;
}

/**
 * Calcula a similaridade como o cosseno entre os vetores A e B
 */
protected float retorneValorCosseno(String prodA, String prodB) {
   float   valorAi;
   float   valorBi;
   float   numerador;
   float   denominadorA;
   float   denominadorB;
```

```
      float   retorno;
      float   fatorSignificancia;
      double  fator;
      double  raizA;
      double  raizB;
      int     n;
      int     cont;
      Float   denominador;
      HashMap vetorA;
      HashMap vetorB;
      String  cliAtual;

      vetorA = matrizCestas.retorneColunaProduto(prodA);
      vetorB = matrizCestas.retorneColunaProduto(prodB);

      numerador  = 0;
      denominadorA = 0;
      denominadorB = 0;
      cont = 0;
      for(Iterator nCli = vetorA.keySet().iterator(); nCli.hasNext();) {
         cliAtual = (String) nCli.next();
         if(vetorB.containsKey(cliAtual)) {
          cont++;

          valorAi = ((Float)vetorA.get(cliAtual)).floatValue();
          valorBi = ((Float)vetorB.get(cliAtual)).floatValue();

          numerador  = numerador  + valorAi*valorBi;
          denominadorA = denominadorA + valorAi*valorAi;
          denominadorB = denominadorB + valorBi*valorBi;
         }
      }

      if(numerador!=0 && cont>=2) {

        if(cont<PESO_SIGNIFICANCIA) {
           fator = cont/PESO_SIGNIFICANCIA;
          fatorSignificancia = (new Float(fator)).floatValue();
        }
        else {
           fatorSignificancia = 1;
        }

        denominador = new Float(denominadorA);
        raizA = java.lang.Math.sqrt(denominador.doubleValue());
        denominador = new Float(denominadorB);
        raizB = java.lang.Math.sqrt(denominador.doubleValue());
        denominador = new Float(raizA*raizB);
        retorno = numerador/denominador.floatValue()*fatorSignificancia;
      }
      else {
       retorno = -99999999;
      }
      return retorno;
    }

    /**
     * Obtém a correlação entre todos os produtos e armazena o valor
     */
  public boolean construaMatrizCorrelacao(GerenciadorBD gerBD, int nVizinhos, int
nProdutos) {
      int         idProdutoAtual;
      Iterator    chaves1;
      Iterator    chaves2;
      HashMap     produtos;
      Vector      pares;
      float       valorCorr;
      float       somaCorr;
      Vector      listaProdutos;
      String      prodI;
      String      prodJ;
      String      mSimilares;

      listaCorrelacao = new Vector();
```

```
    pares           = new Vector();
    produtos        = new HashMap();
    progresso       = 0;
    tamTarefa       = matrizCestas.retorneNProdutos();

    gerBD.removaDadosTabela("correlacao_produtos");

   for(chaves1 = matrizCestas.retorneChavesProdutos(); chaves1.hasNext();) {
       prodI = (String) chaves1.next();
      somaCorr=0;
      prodJ = "";
      for(chaves2 = matrizCestas.retorneChavesProdutos(); chaves2.hasNext();) {
        prodJ = (String) chaves2.next();
          if(!prodI.equals(prodJ)) {
             //valorCorr = retorneValorCorrelacaoPearson(prodI, prodJ);
             valorCorr = retorneValorCosseno(prodI, prodJ);
            if(valorCorr!=-99999999) {
               produtos.put(prodJ, new Float(valorCorr));
             somaCorr = somaCorr+valorCorr;
              listaCorrelacao.add(new Correlacao(prodI, prodJ, valorCorr));
            }
          }
       }
      if(produtos.size() > 0) {
        normalize(produtos, somaCorr);
        mSimilares = retorneMSimilares(nProdutos, produtos);
        gerBD.armazeneMSimilaresProduto(prodI, mSimilares);
        if(!(pares.contains(prodI+prodJ) || pares.contains(prodJ+prodI))) {
           gerBD.armazeneMatrizCorrelacaoItens(listaCorrelacao);
          pares.add(prodI+prodJ);
        }
        produtos.clear();
        listaCorrelacao.clear();
      }
       progresso++;
       status = progresso + " de " + tamTarefa;
    }
    return progresso >= tamTarefa;
}

/**
 * Retorna os M produtos com maior valor de correlacao
 */
protected String retorneMSimilares(int nProdutos, HashMap produtos) {
  String str = "";
  String item;
  Vector ordem = idValoresOrdenados(produtos, nProdutos);
  for(int i=0; i<ordem.size(); i++) {
    item = (String)ordem.elementAt(i);
    str = str + item + "=" + (Float)produtos.get(item) + ", ";
  }
  str = str.substring(0,str.length()-2);
  return str;
}

protected void normalize(HashMap lista, float total) {
  float valor;
  String item;
  for(Iterator e=lista.keySet().iterator(); e.hasNext(); ) {
    item = (String)e.next();
    try {
       valor = ((Float)lista.get(item)).floatValue();
      valor = valor/total;
      lista.put(item, new Float(valor));
    }
    catch (Exception ex) {}
  }
}

/**
 * Retorna um vetor com os identificadores ordenados pelos valores na tabela
 */
protected Vector idValoresOrdenados(HashMap tabela, int n) {
   Vector listaId;
```

```
      Vector listaValor;
      Object elementoAtual;
      float valor;
      float valorOrd;
      boolean inseriu;
      int i;

      listaId = new Vector();
      listaValor = new Vector();

      try {
      for(Iterator e = tabela.keySet().iterator(); e.hasNext(); ) {
         elementoAtual = e.next();
         valor = ((Float)tabela.get(elementoAtual)).floatValue();
         inseriu = false;
         i = 0;
         while((!inseriu) && (i<listaId.size())) { //&& (i<n)
            valorOrd = ((Float)listaValor.get(i)).floatValue();
            if(valor > valorOrd) {
               listaId.insertElementAt(elementoAtual,i);
               listaValor.insertElementAt(new Float(valor),i);
               inseriu = true;
            }
            i++;
         }
         if((!inseriu) && (listaId.size() < n)) {
            listaId.addElement(elementoAtual);
            listaValor.addElement(new Float(valor));
         }
      }
      }
      catch (Exception e) {
        System.out.println(e.getMessage());
      }

      if(listaId.size() > n)
        listaId.setSize(n);
      return listaId;

  }

  public int retorneProgresso() {
    return progresso;
  }

  public String retorneStatus() {
    return status;
  }

  public int retorneTamanhoTarefa() {
    return tamTarefa;
  }
}

/*******************************************************************************/

package recom;

/**
 * Classe:    MatrizClienteProduto
 * Descrição: Representa a matriz cliente-produto e implementa suas
 *            funcionalidades
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;

public class MatrizClienteProduto {

  protected int     nClientes;
  protected int     nProdutos;
  protected HashMap matrizCli;
  protected HashMap matrizProd;
```

```
  protected Vector  produtos;

  public MatrizClienteProduto() {
     matrizCli = new HashMap();
     produtos  = new Vector();
  }

  /**
   * Retorna a quantidade de clientes da matriz
   */
  public int retorneNClientes() {
     return matrizCli.size();
  }

  /**
   * Retorna a quantidade de produtos da matriz para um determinado cliente
   */
  public int retorneNProdutos(String chaveCli) {
     if(matrizCli.containsKey(chaveCli)) {
        return ((HashMap)matrizCli.get(chaveCli)).size();
     }
     else {
        return -1;
     }
  }

  /**
   * Retorna os produtos de determinado cliente
   */
  public HashMap retorneProdutosCliente(String chaveCli) {
     return (HashMap) matrizCli.get(chaveCli);
  }

  public HashMap retorneClientesProduto(String chaveProd) {
    return (HashMap) matrizProd.get(chaveProd);
  }

  // retorna o vetor da matriz correspondente a um produto (chaveProd)
  public Vector retorneValoresProduto(String chaveProd) {
     HashMap linha;
     Vector vetor = new Vector();
     Object chave;

     linha = (HashMap) matrizProd.get(chaveProd);

     for(Iterator eCli = matrizCli.keySet().iterator(); eCli.hasNext();) {
       chave = eCli.next();
       if(linha.containsKey(chave)) {
         vetor.add(linha.get(chave));
       }
       else {
         vetor.add(null); // adiciona-se null para que todos os vetores tenham o mesmo
tamanho
       }
     }
     return vetor;
  }

  /**
   * Retorna os códigos de todos os clientes
   */
  public Iterator retorneChavesClientes() {
     return matrizCli.keySet().iterator();
  }

  /**
   * Retorna a média dos valores referentes aos produtos de determinado cliente
   */
  public float retorneMediaCliente(Object chaveCli) {
     float    soma;
     float    media;
     float    valor;
     int      cont;
     Float    teste;
```

```
      HashMap  produtos;
      Iterator nProdutos;

      produtos = (HashMap) matrizCli.get(chaveCli);

      soma = 0;
      cont = 0;
      for(nProdutos = produtos.keySet().iterator(); nProdutos.hasNext();) {
         teste = (Float) produtos.get(nProdutos.next());
         if(teste != null) {
            valor = teste.floatValue();
            soma = soma + valor;
            cont++;
         }
      }

      media = soma/cont;
      return media;
   }

   /**
    * Inclui produto e seu respectivo valor para um determinado cliente
    */
   public void insiraValorProdutoCliente(String chaveCli, String chaveProd, float valor)
{
      HashMap linha;
      HashMap linha2;
      Float valorProduto;
      boolean existe;

      // se cliente ja existe
      if(matrizCli.containsKey(chaveCli)) {
         linha = (HashMap) matrizCli.get(chaveCli);
         linha.put(chaveProd, new Float(valor));
      }
      // se cliente nao existe na matriz
      else {
         linha = new HashMap();
         linha.put(chaveProd, new Float(valor)); // insere produto
         matrizCli.put(chaveCli, linha);         // insere cliente
      }
   }
}

/*******************************************************************************/

package recom;

/**
 * Classe:    MatrizCestaProduto
 * Descrição: Representa a matriz cesta-produto e implementa suas
 *            funcionalidades
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;

public class MatrizCestasProduto {

   protected HashMap matrizCestas;

   public MatrizCestasProduto() {
      matrizCestas = new HashMap();
   }

   /**
    * Retorna a coluna da matriz relativa ao um determinado produto
    */
   public HashMap retorneColunaProduto(String codProd) {
     return (HashMap)matrizCestas.get(codProd);
   }

   /**
```

```
   * Inclui um produto e seu respectivo valor para uma determinada cesta
   */
  public void insiraValorProdutoCestaCliente(String chaveCli, String chaveProd, float
valor) {
     HashMap linha;
     HashMap linha2;
     Float valorProduto;
     boolean existe;

     // se produto ja existe
     if(matrizCestas.containsKey(chaveProd)) {
        linha2 = (HashMap) matrizCestas.get(chaveProd);
        linha2.put(chaveCli, new Float(valor));
     }
     // se produto nao existe na matriz
     else {
        linha2 = new HashMap();
        linha2.put(chaveCli, new Float(valor)); // insere cesta
        matrizCestas.put(chaveProd, linha2);    // insere produto
     }
  }

  /**
   * Retorna os códigos dos produtos da matriz
   */
  public Iterator retorneChavesProdutos() {
    return matrizCestas.keySet().iterator();
  }

  /**
   * Retorna a quantidade de produtos
   */
  public int retorneNProdutos() {
    return matrizCestas.size();
  }
}

/*********************************************************************************/

package recom;

/**
 * Classe:    Cliente
 * Descrição: Representa um cliente de uma loja virtual
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

public class Cliente {

  protected String id;
  protected int numeroTransacoes;
  protected String nome;
  protected CestaCompras cestaCompras;

  public Cliente() {
  }

  public Cliente(String id) {
     this.id = id;
  }

  public String retorneId() {
    return id;
  }
}

/*********************************************************************************/
```

Aplicação 2 – SIMULA RECOM

```
/*******************************************************************************************/

package simularecom;

/**
 * Classe:    Simulador
 * Descrição: Simula o processo de geração de recomendação de produtos para
 * todos os clientes e verifica a qualidade do resultado
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;
import javax.swing.SwingUtilities;

public class Simulador {

  static final int FASE0 = 0;
  static final int FASE1 = 1;
  static final int FASE2 = 2;
  static final int FASE3 = 3;
  static final int FASE4 = 4;

  static final int n = 10;

  protected Interface frame;
  protected GerenciadorBD gerBD;
  protected float recallMP;
  protected float recallMFC;
  protected float recallMCI;
  protected float recallMMV;
  protected int faseAtual;
  protected int i1, i2, i3, i4;
  protected boolean processoFinalizado;

  protected ModeloProposto mp;
  protected ModeloFiltragemColaborativa mfc;
  protected ModeloCorrelacaoItens mci;
  protected ModeloProdutosMaisVendidos mmv;
  protected MatrizCorrelacaoProdutos matrizCorrProd;


  public Simulador(Interface frame) {
    this.frame = frame;
    faseAtual = FASE0;
    processoFinalizado = false;

    gerBD = new GerenciadorBD();
    gerBD.conecte("sun.jdbc.odbc.JdbcOdbcDriver", "jdbc:odbc:simula_recom");
    //matrizCorrProd = null;
    matrizCorrProd = gerBD.obtenhaTabelaCorrItens();
    gerBD.obtenhaProdutosSimilares(50);
    gerBD.obtenhaMelhoresProdutos();
  }

  /**
   * Inicia o processo de simulação em uma segunda linha de execução,
   * deixando a interface disponível para o usuário
   */
  public void inicie(int nVezes, int valM) {
    final int vezes = nVezes;
    final int m = valM;
    final SwingWorker worker = new SwingWorker() {
      public Object construct() {
         return processe(vezes, m);
      }
     };
     worker.start();
  }

  /**
```

```
 * Processa a simulação para os modelos proposto, filtragem colaborativa,
 * correlação item-a-item e produtos mais vendidos
 */
public Object processe(int nVezes, int valM) {
  faseAtual = FASE0;

  Vector cesta;
  Vector clientes;
  Vector topn;
  String itemEscondido;
  String codCli;
  int acertos;
  int nSim;

  clientes = gerBD.retorneListaClientes();

  // Prepara Interface
  frame.pBar1.setMinimum(0);
  frame.pBar2.setMinimum(0);
  frame.pBar3.setMinimum(0);
  frame.pBar4.setMinimum(0);
  frame.pBar1.setMaximum(clientes.size());
  frame.pBar2.setMaximum(clientes.size());
  frame.pBar3.setMaximum(clientes.size());
  frame.pBar4.setMaximum(clientes.size());
  i1 = 0;
  i2 = 0;
  i3 = 0;
  i4 = 0;

  mp = new ModeloProposto(gerBD, valM , matrizCorrProd);
  mfc = new ModeloFiltragemColaborativa(gerBD);
  mci = new ModeloCorrelacaoItens(gerBD);
  mmv = new ModeloProdutosMaisVendidos(gerBD);

  for(nSim=1; nSim<=nVezes; nSim++) {
    frame.lbVezes.setText("Execução " + nSim + " de " + nVezes);
    frame.pBar1.setValue(0);
    frame.pBar2.setValue(0);
    frame.pBar3.setValue(0);
    frame.pBar4.setValue(0);
    recallMP = 0;
    recallMFC = 0;
    recallMCI = 0;
    recallMMV = 0;
    i1=0;
    i2=0;
    i3=0;
    i4=0;

    for(int i=0; i<clientes.size(); i++) {
      faseAtual = FASE0;
      codCli = (String) clientes.elementAt(i);
      cesta = gerBD.obtenhaItensCesta(codCli);
      itemEscondido = removaItemCesta(cesta);

     // Modelo proposto
      if(frame.chk1.isSelected()) {
        faseAtual = FASE1;
        topn = mp.obtenhaRecomendacoes(n, valM, codCli, cesta);
       if(topn!=null)
        if(topn.contains(itemEscondido)) {
          recallMP++;
         frame.chk1.setText("Modelo Proposto ( " + recallMP +" )");
        }
       i1++;
      }

     // Modelo filtragem colaborativa
      if(frame.chk2.isSelected()) {
        faseAtual = FASE2;
        topn = mfc.obtenhaRecomendacoes(n, codCli, cesta);
       if(topn!=null)
        if(topn.contains(itemEscondido)) {
```

```
          recallMFC++;
          frame.chk2.setText("Modelo Filtragem Colaborativa ( " + recallMFC +" )");
         }
        i2++;
       }

      // Modelo correlação item-a-item
       if(frame.chk3.isSelected()) {
         faseAtual = FASE3;
         topn = mci.obtenhaRecomendacoes(n, codCli, cesta);
         if(topn.contains(itemEscondido)) {
           recallMCI++;
          frame.chk3.setText("Modelo Correlaçao de Itens ( " + recallMCI +" )");
         }
        i3++;
       }

      // Modelo produtos mais vendidos
       if(frame.chk4.isSelected()) {
         faseAtual = FASE4;            // k=50
         topn = mmv.obtenhaRecomendacoes(n, codCli, cesta);
         if(topn.contains(itemEscondido)) {
           recallMMV++;
          frame.chk4.setText("Modelo Mais Vendido ( " + recallMMV +" )");
         }
        i4++;
       }

     }
     frame.mostreResultados(nSim, recallMP, recallMFC, recallMCI, recallMMV);
   }
   processoFinalizado = true;
   gerBD.encerreConexao();
   return new Object();
}

public float retorneRecallMP() {
   return recallMP;
}

public float retorneRecallMFC() {
   return recallMFC;
}

public float retorneRecallMCI() {
   return recallMCI;
}

public float retorneRecallMMV() {
   return recallMMV;
}

/**
 * Remove aleatoriamente e retorna um item da cesta para que sirva como
 * teste das recomendações
 */
protected String removaItemCesta(Vector cesta) {
   int numero = Math.abs(new Random().nextInt() % cesta.size());
   return (String) cesta.remove(numero);
}

public int retorneFaseAtual() {
   return faseAtual;
}

public int retorneProgressoFase1() {
   return i1;
}

public int retorneProgressoFase2() {
   return i2;
}

public int retorneProgressoFase3() {
```

```
    return i3;
  }

  public int retorneProgressoFase4() {
    return i4;
  }

  public boolean processoFinalizado() {
    return processoFinalizado;
  }

  public void reset() {
    processoFinalizado = false;
  }
}

/*******************************************************************************/

package simularecom;

/**
 * Classe:    Interface
 * Descrição: Implementa a interface gráfica do usuário
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;

public class Interface extends JFrame {

  static final int UM_SEGUNDO = 1000;

  protected Simulador simulador;
  protected Timer timer;
  protected int nVezes;
  protected int valM;

  JPanel pane;
  JPanel paneSul;
  JPanel paneCentro;
  JPanel paneLeste;
  JPanel paneValM;
  JButton btExecutar;
  JButton btCancelar;
  JTextArea taResultados;
  JTextField tfVezes;
  JLabel lbVezes;
  JLabel lbRecallMP;
  JLabel lbRecallMFC;
  JLabel lbRecallMCI;
  JLabel lbExecucao;
  JLabel lbSts1;
  JLabel lbSts2;
  JLabel lbSts3;
  JProgressBar pBar1;
  JProgressBar pBar2;
  JProgressBar pBar3;
  JProgressBar pBar4;
  JCheckBox chk1;
  JCheckBox chk2;
  JCheckBox chk3;
  JCheckBox chk4;
  JLabel lbValM;
  JTextField tfValM;

  public Interface() {

    super("Simulador Recom");

    simulador = new Simulador(this);
```

```
pane = new JPanel(new BorderLayout());
paneSul = new JPanel();
paneCentro = new JPanel();
paneLeste = new JPanel();
paneValM = new JPanel();

taResultados = new JTextArea(40, 40);
tfVezes = new JTextField("1");
lbVezes = new JLabel("Execuções ");
lbRecallMP = new JLabel("Recall MP ... ");
lbRecallMFC = new JLabel("Recall MFC ... ");
lbRecallMCI = new JLabel("Recall MCI ... ");

lbVezes = new JLabel("Nro. de Execuções");
lbSts1 = new JLabel("Modelo Proposto");
lbSts2 = new JLabel("Modelo Filtragem Colaborativa");
lbSts3 = new JLabel("Modelo Correlacao de Itens");
pBar1 = new JProgressBar();
pBar2 = new JProgressBar();
pBar3 = new JProgressBar();
pBar4 = new JProgressBar();
chk1 = new JCheckBox("Modelo Proposto", true);
chk2 = new JCheckBox("Modelo Filtragem Colaborativa", true);
chk3 = new JCheckBox("Modelo Correlação de Itens", true);
chk4 = new JCheckBox("Modelo Mais Vendidos", true);
lbValM = new JLabel("Valor de M");
tfValM = new JTextField("30", 7);

btExecutar = new JButton("Executar");
btExecutar.addActionListener(new BtExecutarListener());
btCancelar = new JButton("Cancelar");
btCancelar.addActionListener(new BtCancelarListener());

taResultados.append("Simulação\t");
taResultados.append("Recall MP\t");
taResultados.append("Recall MFC\t");
taResultados.append("Recall MCI\t");
taResultados.append("Recall MMV\n");
paneCentro.add(taResultados);

paneValM.add(lbValM);
paneValM.add(tfValM);

pBar1.setBorderPainted(true);
pBar2.setBorderPainted(true);
pBar3.setBorderPainted(true);
pBar4.setBorderPainted(true);
pBar1.setStringPainted(true);
pBar2.setStringPainted(true);
pBar3.setStringPainted(true);
pBar4.setStringPainted(true);
paneLeste.setLayout(new BoxLayout(paneLeste, BoxLayout.Y_AXIS));
paneLeste.add(lbVezes);
paneLeste.add(Box.createRigidArea(new Dimension(0, 20)));
paneLeste.add(chk1);
paneLeste.add(paneValM);
paneLeste.add(pBar1);
paneLeste.add(Box.createRigidArea(new Dimension(0, 10)));
paneLeste.add(chk2);
paneLeste.add(pBar2);
paneLeste.add(Box.createRigidArea(new Dimension(0, 10)));
paneLeste.add(chk3);
paneLeste.add(pBar3);
paneLeste.add(Box.createRigidArea(new Dimension(0, 10)));
paneLeste.add(chk4);
paneLeste.add(pBar4);
paneLeste.add(Box.createVerticalGlue());
paneLeste.add(Box.createVerticalGlue());
paneLeste.add(Box.createVerticalGlue());
paneLeste.add(Box.createVerticalGlue());
paneLeste.add(Box.createVerticalGlue());
paneLeste.add(Box.createVerticalGlue());
paneLeste.add(Box.createVerticalGlue());
```

```
    paneLeste.add(Box.createVerticalGlue());

    paneSul.setLayout(new BoxLayout(paneSul, BoxLayout.X_AXIS));
    paneSul.add(Box.createHorizontalBox());
    paneSul.add(lbVezes);
    paneSul.add(Box.createRigidArea(new Dimension(5,0)));
    paneSul.add(tfVezes);
    paneSul.add(Box.createRigidArea(new Dimension(10,0)));
    paneSul.add(btCancelar);
    paneSul.add(Box.createRigidArea(new Dimension(5,0)));
    paneSul.add(btExecutar);

    pane.add(paneCentro, BorderLayout.CENTER);
    pane.add(paneSul, BorderLayout.SOUTH);
    pane.add(paneLeste, BorderLayout.EAST);

    this.setTitle("Simulação dos Modelos de Recomendação");
    this.getContentPane().add(pane);

      // Cria o timer
      timer = new Timer(UM_SEGUNDO, new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                 pBar1.setValue(simulador.retorneProgressoFase1());
                  pBar2.setValue(simulador.retorneProgressoFase2());
                  pBar3.setValue(simulador.retorneProgressoFase3());
                  pBar4.setValue(simulador.retorneProgressoFase4());
              if(simulador.processoFinalizado()) {
                Toolkit.getDefaultToolkit().beep();
                btExecutar.setEnabled(true);
                btCancelar.setText("Fechar");
                timer.stop();
              }
            }
       });
  }

    class BtExecutarListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
          btExecutar.setEnabled(false);
          try {
             nVezes = Integer.parseInt(tfVezes.getText());
             valM   = Integer.parseInt(tfValM.getText());
          }
          catch (Exception exp) {
            nVezes = 0;
            valM   = 0;
          }
          simulador.reset();
          simulador.inicie(nVezes, valM);
          timer.start();
        }
    }

    class BtCancelarListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
          System.exit(0);
        }
    }

  public void mostreResultados(int nSim, float recallMP, float recallMFC, float
recallMCI, float recallMMV) {
    lbRecallMP.setText("Recall MP: " + recallMP);
    lbRecallMFC.setText("Recall MFC: " + recallMFC);
    lbRecallMCI.setText("Recall MCI: " + recallMCI);
    taResultados.append(nSim + "\t");
    taResultados.append(recallMP + "\t");
    taResultados.append(recallMFC + "\t");
    taResultados.append(recallMCI + "\t");
    taResultados.append(recallMMV + "\n");
  }

  public static void main(String[] args) {
    Interface frame = new Interface();
```

```
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });

    frame.pack();
    frame.setVisible(true);
  }
}

/*******************************************************************************/

package simularecom;

/**
 * Classe:    GerenciadorBD
 * Descrição: Realiza os processos de consulta e atualização de dados no BD
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.sql.*;
import java.util.*;
import java.io.*;

public class GerenciadorBD {

  protected Connection conexao;
  protected PreparedStatement prepStmt;
  protected HashMap similares;
  protected HashMap melhoresProdutos;
  protected HashMap corrItens;
  protected Vector chaves;
  protected Vector corr;

  public GerenciadorBD() {
  }

  public boolean conecte(String driver, String url) {
    String sql;
    boolean resultado;
    resultado = true;

    try {
       Class.forName(driver);
    }
    catch (Exception e) {
       System.out.println(e.getMessage());
       resultado = false;
    }

    try {
       conexao = DriverManager.getConnection(url);
        prepStmt = conexao.prepareStatement("select correlacao from correlacao_produtos
where cod_prod_a = ? AND cod_prod_b = ? ");
    }
    catch (Exception e) {
       System.out.println(e.getMessage());
       resultado = false;
    }

    return resultado;
  }

  public void encerreConexao() {
     try {
        conexao.close();
     }
     catch (Exception e) {
        System.out.println(e.getMessage());
    }
  }
```

```
  /**
   * Retorna a lista de todos os clientes
   */
  public Vector retorneListaClientes() {
    Statement stmt;
    Vector lista;
    String sql;
    ResultSet result;

    lista = new Vector();

    try {
        stmt = conexao.createStatement();
        sql = "SELECT cod_cli FROM clientes WHERE (qtd_transacoes  > 0) ORDER BY
cod_cli";
        result = stmt.executeQuery(sql);
        while(result.next()) {
           lista.add(result.getString("cod_cli"));
        }
        result.close();
        stmt.close();
     }
     catch (Exception e) {
        System.out.println(e.getMessage());
     }
     return lista;
  }

  /**
   * Retorna a cesta de produtos de um determinado cliente
   */
  public Vector obtenhaItensCesta(String codCli) {
    Statement stmt;
    Vector lista;
    String sql;
    ResultSet result;

    lista = new Vector();

    try {
        stmt = conexao.createStatement();
        sql = "SELECT cod_prod FROM cesta_compras WHERE cod_cli='" + codCli +"'";
        result = stmt.executeQuery(sql);
        while(result.next()) {
           lista.add(result.getString("cod_prod"));
        }
        result.close();
        stmt.close();
     }
     catch (Exception e) {
        System.out.println(e.getMessage());
     }
     return lista;
  }

  /**
   * Retorna os m melhores produtos de um determinado cliente
   */
  public Vector obtenhaMMelhoresProdutos(int m, String codCli) {
     Statement stmt;
     ResultSet result;
     String sql;
     String resultado;
     Vector retorno;
     StringTokenizer stVirg;
     String item;
     Float valor;
     int cont;

     retorno = new Vector();
     try {
        stmt = conexao.createStatement();
        sql = "SELECT cod_produtos FROM melhores_produtos WHERE cod_cli = '";
       sql = sql + codCli + "'";
```

```
      result = stmt.executeQuery(sql);
      result.next();
      resultado = result.getString("cod_produtos");
      stVirg = new StringTokenizer(resultado,",");
      cont = 0;
      while(stVirg.hasMoreElements() && cont<m) {
        retorno.add(stVirg.nextToken());
        cont++;
      }
      result.close();
      stmt.close();
    }
    catch (SQLException e) {
      System.out.println(e.getMessage());
    }
    return retorno;
}

/**
 * Retorna os m melhores produtos para um determinado cliente,
 * desconsiderando os que já estão na cesta de compras
 */
public Vector obtenhaMMelhoresProdutos(int m, String codCli, Vector cesta) {
  String prod;
  Vector retorno;
  HashMap produtos = (HashMap)melhoresProdutos.get(codCli);
  if(produtos==null) return null;
  retorno = idValoresOrdenados(produtos, m+cesta.size());
  for(int i=0; i<cesta.size(); i++) {
    prod = (String) cesta.elementAt(i);
    if(retorno.contains(prod))
      retorno.remove(retorno.indexOf(prod));
  }
  retorno.setSize(m);
  return retorno;
}

/**
 * Retorna os m melhores produtos juntamente com os respectivos pesos de
 * similaridade para um determinado cliente, desconsiderando os que já estão
 * na cesta de compras
 */
public HashMap obtenhaMMelhoresProdutosComPesos(int m, String codCli, Vector cesta) {
  String prod;
  float[] valores = new float[m];
  float soma;
  HashMap retorno = new HashMap();
  Vector ordem = obtenhaMMelhoresProdutos(m, codCli, cesta);
  HashMap produtos = (HashMap)melhoresProdutos.get(codCli);
  soma = 0;

  for(int i=0; i<ordem.size(); i++) {
    prod = (String) ordem.elementAt(i);
    try {
       valores[i] = ((Float)produtos.get(prod)).floatValue();
    }
    catch (Exception e) {
     valores[i] = 0;
    }
    soma = soma + valores[i];
  }
  for(int i=0; i<ordem.size(); i++) {
    retorno.put(ordem.elementAt(i), new Float(valores[i]/soma));
  }
  return retorno;
}

/**
 * Retorna um vetor com os identificadores ordenados pelos valores na tabela
 */
protected Vector idValoresOrdenados(HashMap tabela, int n) {
   Vector listaId;
   Vector listaValor;
   Object elementoAtual;
```

```
   float valor;
   float valorOrd;
   boolean inseriu;
   int i;

   listaId = new Vector();
   listaValor = new Vector();

   try {
   for(Iterator e = tabela.keySet().iterator(); e.hasNext(); ) {
      elementoAtual = e.next();
      valor = ((Float)tabela.get(elementoAtual)).floatValue();
      inseriu = false;
      i = 0;
      while((!inseriu) && (i<listaId.size())) { //&& (i<n)
         valorOrd = ((Float)listaValor.get(i)).floatValue();
         if(valor > valorOrd) {
            listaId.insertElementAt(elementoAtual,i);
            listaValor.insertElementAt(new Float(valor),i);
            inseriu = true;
         }
         i++;
      }
      if((!inseriu) && (listaId.size() < n)) {
         listaId.addElement(elementoAtual);
         listaValor.addElement(new Float(valor));
      }
   }
   }
   catch (Exception e) {
     System.out.println(e.getMessage());
   }

   if(listaId.size() > n)
     listaId.setSize(n);
   return listaId;

}

/**
 * Carrega a estrutura melhoresProdutos para ser acessada por outros métodos
 */
public void obtenhaMelhoresProdutos() {
   Statement stmt;
   ResultSet result;
   String sql;
   String resultado;
   String cod;
   StringTokenizer stVirg;
   StringTokenizer stIg;
   HashMap linha;
   String item;
   Float valor;

   melhoresProdutos = new HashMap();
   try {
      stmt = conexao.createStatement();
      sql = "SELECT * FROM melhores_produtos";
      result = stmt.executeQuery(sql);
     while(result.next()) {
       cod = result.getString("cod_cli");
       resultado = result.getString("cod_produtos");
       stVirg = new StringTokenizer(resultado,",");
       linha = new HashMap();
       while(stVirg.hasMoreElements()) {
         stIg = new StringTokenizer(stVirg.nextToken(),"=");
         item = stIg.nextToken().trim();
         try {
           valor = Float.valueOf(stIg.nextToken());
         }
         catch (Exception e) {
           valor = new Float(0);
         }
         linha.put(item, valor);
```

```
        }
        melhoresProdutos.put(cod, linha);
      }
      result.close();
      stmt.close();
    }
    catch (SQLException e) {
      System.out.println(e.getMessage());
    }
}

/**
 * Retorna o valor da correlação entre dois produtos
 */
public float retorneCorrelacaoProdutos(String prodA, String prodB) {
    ResultSet result;
    String sql;
    float retorno = 0;
    try {
       prepStmt.setString(1, prodA);
       prepStmt.setString(2, prodB);
       result = prepStmt.executeQuery();
      if(result.next()) {
         retorno = result.getFloat("correlacao");
      }
      else {
         prepStmt.setString(1, prodB);
         prepStmt.setString(2, prodA);
         result = prepStmt.executeQuery();
        if(result.next()) {
           retorno = result.getFloat("correlacao");
        }
        else {
          retorno = 0;
        }
      }
      result.close();
    }
    catch (SQLException e) {
      System.out.println(e.getMessage());
    }
    return retorno;
}

/**
 * Retorna os produtos similares a cada um dos produtos da cesta
 */
public HashMap obtenhaProdutosSimilares(Vector cesta) {
  HashMap retorno = new HashMap();
  String itemCesta;
  for(int i=0; i<cesta.size(); i++) {
    itemCesta = (String) cesta.elementAt(i);
    if(similares.containsKey(itemCesta)) {
     retorno.put(itemCesta, (HashMap)similares.get(itemCesta));
    }
  }
  return retorno;
}

/**
 * Retorna os produtos similares juntamente com o valor da
 * correlação para um determinado produto
 */
public HashMap obtenhaProdutosSimilares(String codProd) {
  return (HashMap)similares.get(codProd);
}

/**
 * Carrega uma estrutura com os todos produtos juntamente com seus k melhores
 * correlacionados produtos
 */
public void obtenhaProdutosSimilares(int k) {
  HashMap kItens;
  Statement stmt;
```

```
  ResultSet result;
  String sql;
  String item;
  String prod;
  Float valor;
  String resultado;
  StringTokenizer stVirg;
  StringTokenizer stIg;
  int cont;

  similares = new HashMap();
  try {
    stmt = conexao.createStatement();
    sql = "SELECT * FROM produtos_similares";
    result = stmt.executeQuery(sql);
    while(result.next()) {
     item = result.getString("cod_prod");
     resultado = result.getString("cod_similares");
     stVirg = new StringTokenizer(resultado,",");
     cont = 0;
      kItens = new HashMap();
     while(stVirg.hasMoreElements() && cont<k) {
       stIg = new StringTokenizer(stVirg.nextToken(),"=");
       prod = stIg.nextToken().trim();
       try {
         valor = Float.valueOf(stIg.nextToken());
       }
       catch (Exception e) {
         valor = new Float(0);
       }
       kItens.put(prod, valor);
       cont++;
     }
     similares.put(item, kItens);
    }
    result.close();
    stmt = conexao.createStatement();
  }
  catch (SQLException e) {
      //System.out.println(e.getMessage());
  }
}

/**
 * Retorna uma lista com os produtos mais vendidos
 */
public Vector obtenhaProdutosMaisVendidos(int n) {
  Statement stmt;
  Vector lista;
  String sql;
  ResultSet result;
  String cod;
  int cont;

  lista = new Vector();

  try {
      stmt = conexao.createStatement();
      sql = "SELECT cod_prod FROM soma_produtos ORDER BY quantidade DESC";
      result = stmt.executeQuery(sql);
     cont=0;
      while(result.next() && cont<n) {
       cod = result.getString("cod_prod");
       lista.add(cod);
       cont++;
      }
     result.close();
     stmt.close();
  }
  catch (Exception e) {}
  return lista;
}

/**
```

```
   * Retorna a matriz de correlação de produtos
   */
  public MatrizCorrelacaoProdutos obtenhaTabelaCorrItens() {
    Statement stmt;
    String sql;
    ResultSet result;
    float valor;

    MatrizCorrelacaoProdutos retorno = null;

    try {

        retorno = new MatrizCorrelacaoProdutos(1218722);
        stmt = conexao.createStatement();
        sql = "SELECT * FROM correlacao_produtos";
        result = stmt.executeQuery(sql);
       while(result.next()) {
         valor = result.getFloat("correlacao");
            retorno.insiraCorrelacaoProdutos(result.getString("cod_prod_a"),
result.getString("cod_prod_b"), valor);
       }
       result.close();
       stmt.close();
    }
    catch (Exception e) {
      System.out.println(e.getMessage());
    }
    return retorno;
  }
}

/*****************************************************************************/

package simularecom;

/**
 * Classe:    ModeloProposto
 * Descrição: Implementa as funcionalidades de geração de recomendações do
 *            modelo proposto
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;

public class ModeloProposto {

  protected GerenciadorBD gerBD;
  protected HashMap melhoresProd;
  protected MatrizCorrelacaoProdutos matrizCorrProd;

  public ModeloProposto(GerenciadorBD gerBD, int m, MatrizCorrelacaoProdutos matriz) {
    this.gerBD = gerBD;
    matrizCorrProd = matriz;
  }

  /**
   * Gera "n" recomendações, a partir de "m" melhores produtos, para um
   * determinado cliente
   */
  public Vector obtenhaRecomendacoes(int n, int m, String codCli, Vector cesta) {
    Vector recomendacoes = new Vector();
    Vector correlacao;
    Vector chavesMelhores;
    HashMap mMelhores;
    HashMap correlacoes;
    float corr;
    float valor;
    float peso;
    String codI;
    String codJ;

    mMelhores = gerBD.obtenhaMMelhoresProdutosComPesos(m, codCli, cesta);
    chavesMelhores = new Vector(mMelhores.keySet());
```

```
    correlacao = new Vector();

    if(chavesMelhores!=null) {
      for(Iterator i=mMelhores.keySet().iterator(); i.hasNext(); ) {
       codI = (String)i.next();
       peso = ((Float)mMelhores.get(codI)).floatValue();
        corr = 0;
        for(int j=0; j<cesta.size(); j++) {
          //corr = corr + gerBD.retorneCorrelacaoProdutos(codI,
(String)cesta.elementAt(j));
          corr = corr + matrizCorrProd.retorneCorrelacaoProdutos(codI,
(String)cesta.elementAt(j));
        }
        correlacao.add(new Float(corr*peso));
      }
    }

    recomendacoes = obtenhaNMaiores(n, chavesMelhores, correlacao);

    return recomendacoes;
  }

  /**
   * Ordena a lista de produtos e retorna os "n" maiores
   */
  protected Vector obtenhaNMaiores(int n, Vector produtos, Vector valores) {
    Vector retorno = new Vector();
    float maior;
    int   indiceMaior;
    if(produtos.size() < n)
       n = produtos.size();
    for(int i=0; i<n; i++) {
      maior = -999999999;
      indiceMaior = 0;
      for(int j=0; j<produtos.size(); j++) {
       if(((Float)valores.elementAt(j)).floatValue() > maior) {
         maior = ((Float)valores.elementAt(j)).floatValue();
         indiceMaior = j;
       }
      }
      retorno.add(produtos.elementAt(indiceMaior));
      valores.remove(indiceMaior);
      produtos.remove(indiceMaior);
    }
    return retorno;
  }
}

/*******************************************************************************/

package simularecom;

/**
 * Claase:    ModeloFiltragemColaborativa
 * Descrição: Implementa as funcionalidades de geração de recomendações do
 *            modelo de filtragem colaborativa
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;

public class ModeloFiltragemColaborativa {

  GerenciadorBD gerBD;

  public ModeloFiltragemColaborativa(GerenciadorBD gerBD) {
    this.gerBD = gerBD;
  }

  /**
   * Retorna as "n" melhores recomendações para um determinado cliente
   */
```

```
  public Vector obtenhaRecomendacoes(int n, String codCli, Vector cesta) {
    return (Vector) gerBD.obtenhaMMelhoresProdutos(n, codCli, cesta);
  }
}

/*******************************************************************************/

package simularecom;

/**
 * Classe:    ModeloCorrelacaoItens
 * Descrição: Implementa as funcionalidades de geração de recomendações do
 *            modelo de correlacao item-a-item
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;

public class ModeloCorrelacaoItens {

  GerenciadorBD gerBD;
  HashMap matrizCorr;

  public ModeloCorrelacaoItens(GerenciadorBD gerBD) {
    this.gerBD = gerBD;
  }

  /**
   * Retorna as "n" melhores recomendações para um determinado cliente
   */
  public Vector obtenhaRecomendacoes(int n, String codCli, Vector cesta) {
    Vector recomendacoes = new Vector();
    Vector cjtoSim;
    Vector correlacao;
    HashMap similares;
    Float corr;
    float fcorr;
    float valor;

    similares = gerBD.obtenhaProdutosSimilares(cesta);
    cjtoSim = retorneConjuntoSimilares(similares);
    correlacao = new Vector();

    for(int i=0; i<cjtoSim.size(); i++) {
      valor = 0;
      for(int j=0; j<cesta.size(); j++) {
       try {
          corr =
((Float)((HashMap)similares.get(cesta.elementAt(j))).get(cjtoSim.elementAt(i)));
         fcorr = corr.floatValue();
       }
       catch (Exception e) {
         fcorr = 0;
       }
       valor = valor + fcorr;
      }
      correlacao.add(new Float(valor));
    }
    recomendacoes = obtenhaNMaiores(n, cjtoSim, correlacao);

    return recomendacoes;
  }

  /**
   * Ordena uma lista de produtos em ordem decrescente de valor e retorna os
   * "n" primeiros
   */
  protected Vector obtenhaNMaiores(int n, Vector produtos, Vector valores) {
    Vector retorno = new Vector();
    float maior;
    int   indiceMaior;

    if(n > produtos.size()) {
```

```
      n = produtos.size();
    }

    if(produtos.size() > 0) {
      for(int i=0; i<n; i++) {
        maior = -999999999;
        indiceMaior = 0;
        for(int j=0; j<produtos.size(); j++) {
         if(((Float)valores.elementAt(j)).floatValue() > maior) {
           maior = ((Float)valores.elementAt(j)).floatValue();
           indiceMaior = j;
         }
        }
        retorno.add(produtos.elementAt(indiceMaior));
       valores.remove(indiceMaior);
       produtos.remove(indiceMaior);
      }
    }
    return retorno;
  }

  /**
   * Retorna o conjuntos dos produtos similares aos produtos da tabela
   */
  protected Vector retorneConjuntoSimilares(HashMap tabela) {
    Vector retorno = new Vector();
    HashMap linha;
    String item;
    for(Iterator e1=tabela.keySet().iterator(); e1.hasNext();) {
      linha = (HashMap)tabela.get(e1.next());
      for(Iterator e2=linha.keySet().iterator(); e2.hasNext();) {
       item = (String)e2.next();
       if(!retorno.contains(item)) {
         retorno.add(item);
       }
      }
    }
    return retorno;
  }
}

/*******************************************************************************/

package simularecom;

/**
 * Classe:    ModeloProdutosMaisVendidos
 * Descrição: Implementa as funcionalidades de geração de recomendações do
 *            modelo de produtos mais vendidos
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.Vector;

public class ModeloProdutosMaisVendidos {

  protected GerenciadorBD gerBD;
  protected Vector maisVendidos;

  public ModeloProdutosMaisVendidos(GerenciadorBD gerBD) {
    this.gerBD = gerBD;
    maisVendidos = gerBD.obtenhaProdutosMaisVendidos(100);
  }

  /**
   * Retorna as "n" melhores recomendações para um determinado cliente
   */
  public Vector obtenhaRecomendacoes(int n, String codCli, Vector cesta) {
    Vector recomendacoes = new Vector();
    int cont;

    cont=0;
    while(cont<n && cont<maisVendidos.size()) {
```

```
      if(!cesta.contains(maisVendidos.elementAt(cont)))
        recomendacoes.add(maisVendidos.elementAt(cont));
      cont++;
    }
    return recomendacoes;
  }
}

/*******************************************************************************/

package simularecom;

/**
 * Classe:    MatrizCorrelacaoProdutos
 * Descrição: Representa a matriz de correlação de produtos
 * Autora:    Elaine Venson
 * Versão:    1.0
 */

import java.util.*;

public class MatrizCorrelacaoProdutos {

  HashMap matriz;

  public MatrizCorrelacaoProdutos(int n) {
    matriz = new HashMap();
    produtos = new Vector();
  }

  /**
   * Inclui na matriz o valor da correlação entre dois produtos
   */
  public void insiraCorrelacaoProdutos(String prodA, String prodB, float correlacao) {
     HashMap linha;
     // se produto "A" ja existe
     if(matriz.containsKey(prodA)) {
        linha = (HashMap) matriz.get(prodA);
        linha.put(prodB, new Float(correlacao));
     }
     // se produto "B" ja existe
     else if(matriz.containsKey(prodB)) {
        linha = (HashMap) matriz.get(prodB);
        linha.put(prodA, new Float(correlacao));
     }
     // se nenhum dos produtos existe na matriz
     else {
        linha = new HashMap();
        linha.put(prodB, new Float(correlacao));
        matriz.put(prodA, linha);
     }
  }

  /**
   * Retorna o valor da correlação entre dois produtos
   */
  public float retorneCorrelacaoProdutos(String prodA, String prodB) {
    float retorno = 0;
    Float corr = new Float(0);
    HashMap linha;
    boolean achou = false;

    if(matriz.containsKey(prodA)) {
      linha = (HashMap) matriz.get(prodA);
      if(linha.containsKey(prodB)) {
       corr = (Float)linha.get(prodB);
       achou = true;
      }
    }

    if(!achou) {
      if(matriz.containsKey(prodB)) {
        linha = (HashMap) matriz.get(prodB);
       if(linha.containsKey(prodA)) {
```

```
          corr = (Float)linha.get(prodA);
        }
      }
    }

    return corr.floatValue();
  }
}

/*****************************************************************************/
```