AppRecommender: um recomendador de aplicativos GNU/Linux

Tássia Camões Araújo

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. Arnaldo Mandel

Durante o desenvolvimento deste trabalho a autora recebeu auxílio financeiro do CNPq

São Paulo, setembro de 2011

AppRecommender: um recomendador de aplicativos GNU/Linux

Esta dissertação trata-se da versão original
da aluna (Tássia Camões Araújo).

Resumo

A crescente oferta de programas de código aberto na rede mundial de computadores expõe potenciais usuários a inúmeras possibilidades de escolha. Em face da pluralidade de interesses destes indivíduos, mecanismos eficientes que os aproximem daquilo que buscam trazem benefícios para eles próprios, assim como para os desenvolvedores dos programas. Este trabalho apresenta o *AppRecommender*, um recomendador de aplicativos GNU/Linux que realiza uma filtragem no conjunto de programas disponíveis e oferece sugestões individualizadas para os usuários. Tal feito é alcançado por meio da análise de perfis e descoberta de padrões de comportamento na população estudada, de sorte que apenas os aplicativos considerados mais suscetíveis a aceitação sejam oferecidos aos usuários.

Palavras-chave: Sistemas de recomendação, aplicativos, pacotes Debian, filtragem colaborativa, distribuições GNU/Linux, Debian GNU/Linux.

Abstract

The increasing availability of open source software on the World Wide Web exposes potential users to a wide range of choices. Given the individuals plurality of interests, mechanisms that get them close to what they are looking for would benefit users and software developers. This work presents *AppRecommender*, a recommender system for GNU/Linux applications which performs a filtering on the set of available software and individually offers suggestions to users. This is achieved by analyzing profiles and discovering patterns of behavior of the studied population, in a way that only those applications considered most prone to acceptance are presented to users.

Keywords: Recommender systems, applications, Debian packages, collaborative filtering, GNU/Linux distributions, Debian GNU/Linux.

Sumário

Sumário vii

Lista de Figuras ix

Lista de Tabelas xi

1 Introdução 1

2 Distribuições GNU/Linux 5
2.1 Surgimento . . . 5
2.2 Empacotamento de programas . . . 6
2.3 Relação entre pacotes . . . 7
2.4 Sistemas gerenciadores de pacotes . . . 8
2.5 Seleção de programas . . . 10

3 Sistemas de recomendação 13
3.1 Contexto histórico . . . 13
3.2 O problema computacional . . . 14
3.3 Ações e desafios . . . 14
3.4 Seleção de atributos . . . 15
3.5 Estratégias de recomendação . . . 16
3.6 Técnicas . . . 19
3.7 Avaliação de recomendadores . . . 35
3.8 Riscos à privacidade de usuários . . . 39

4 Trabalhos correlatos 41
4.1 Anapop/Popsuggest . . . 41
4.2 Debommender . . . 41
4.3 Mineração de dados do Popcon . . . 42
4.4 AppStream . . . 42
4.5 *Ratings and Reviews* . . . 43
4.6 Recomendadores para dispositivos móveis . . . 43

5 AppRecommender 45
5.1 Caracterização do problema . . . 45
5.2 Escolha da plataforma . . . 45
5.3 Fontes de Dados . . . 47
5.4 Decisões de projeto . . . 52
5.5 Estratégias de recomendação . . . 57

5.6 Protótipo do AppRecommender 59

6 Validação da proposta **63**
6.1 Experimentos *offline* 63
6.2 Consulta pública 73

7 Considerações finais **77**

Lista de Figuras

2.1 Extração do conteúdo de um pacote . . . 7
2.2 Detalhes do pacote *apt* . . . 9
2.3 Captura de tela do synaptic . . . 10
2.4 Captura de tela do software-center . . . 11

3.1 Avaliação de usuário no IMDb . . . 16
3.2 Problema com a extração da média de avaliações . . . 17
3.3 Cenário de uma recomendação baseada em conteúdo . . . 18
3.4 Cenário da recomendação colaborativa . . . 18
3.5 Recomendação por associação na *Amazon* . . . 19
3.6 Eliminação de *stop words* e normalização do documento por *stemming* . . . 24
3.7 Coleção de documentos . . . 25
3.8 Conjunto das partes ilustrado por um diagrama de *Hasse* . . . 34
3.9 Geração de conjuntos candidatos pelo algoritmo Apriori . . . 34

5.1 Excerto da base do Debtags . . . 48
5.2 Lista de termos indexados para o pacote *2vcard* . . . 48
5.3 Exemplo de submissão do popcon . . . 49
5.4 Fluxo de dados no UDD [**?**] . . . 51
5.5 Exemplo consulta ao plugin BTS . . . 52
5.6 Fluxo de dados no *AppRecommender* . . . 53
5.7 Execução do recomendador para o sistema local . . . 60

6.1 Distribuição de submissões do *Popcon* por tamanho de perfil . . . 64
6.2 Aplicação de métricas para recomendação de 10 itens . . . 65
6.3 Aplicação de métricas para recomendação de 100 itens . . . 66
6.4 Registro de recomendação dos experimentos . . . 66
6.5 Curva ROC média de um recomendador `cbt` . . . 69
6.6 Curva ROC com desvios de um recomendador `cbt` . . . 70
6.7 Registro de desenho da curva ROC . . . 70
6.8 Curva ROC média para estratégia `knn` . . . 71
6.9 Curva ROC com desvios para estratégia `knn` . . . 71
6.10 Curva ROC média para estratégia `knn_plus` . . . 72
6.11 Curva ROC com desvios para estratégia `knn_plus` . . . 72
6.12 Curva ROC média para estratégia `knnco` . . . 73
6.13 Curva ROC com desvios para estratégia `knnco` . . . 73
6.14 Curva ROC de modelo anômalo . . . 74
6.15 Interface da consulta pública . . . 75

Lista de Tabelas

2.1 Papéis exercidos por desenvolvedores no Debian 7
2.2 Descrição das relações entre pacotes Debian 8

3.1 Métodos de hibridização 20
3.2 K-NN: Medidas de distância e similaridade entre objetos 21
3.3 Frequência dos termos nos documentos da coleção 25
3.4 Valores de idf_t para termos do dicionário 26
3.5 Ordenação dos documentos como resultado das consultas q_1, q_2 e q_3 26
3.6 Representação da coleção no modelo de espaço vetorial 27
3.7 Representação das queries no modelo de espaço vetorial 28
3.8 Tabela de contingência da incidência dos termos 30
3.9 Matriz de contingência de uma recomendação 36
3.10 Métricas de acurácia de sistemas preditivos 38

5.1 Descrição do formato de uma submissão *popcon* 49
5.2 Descrição das estratégias de recomendação implementadas 59
5.3 Descrição dos parâmetros ajustáveis do *AppRecommender* 60

6.1 Melhores desempenhos para 10 sugestões 68
6.2 Melhores desempenhos para 100 sugestões 68
6.3 Melhores desempenhos analisados por curvas ROC 70

Capítulo 1

Introdução

A facilidade de comunicação proporcionada pela Internet tem estimulado cada vez mais as formas de trabalho colaborativas. No âmbito do desenvolvimento de programas de computador, verificamos na prática o que muitos autores consideram como o maior exemplo deste fenômeno: *o movimento do software livre*; a construção coletiva de uma ampla gama de softwares de qualidade, em constante atualização e evolução [**?**]. Outro ponto defendido é que a qualidade do esforço coletivo aumentou ao longo do tempo devido à diversidade dos colaboradores envolvidos. Esta diversidade é refletida claramente na variedade de projetos disponíveis na grande rede, compondo um *ecossistema* que reúne os interessados por meio de fóruns na Web, listas de discussão, canais IRC, sistemas para notificação de falhas, conferências etc.

O *SourceForge*[1], popular repositório de progamas na Internet, possui hoje mais de 300.000 projetos cadastrados[2], superando a marca de 2 milhões de usuários registrados. Os benefícios deste fenômeno para o público são muitos, uma vez que, além de contar com programas de alta qualidade técnica, licenciamento livre e uso gratuito, os interessados podem também ter participação direta no processo de desenvolvimento.

Esta abundância de projetos, que por um lado oferece inúmeros benefícios, pode também implicar num fator complicador para os próprios usuários. O excessivo montante de informação pode prejudicar o processo de escolha, enredando o usuário num labirinto de resultados inesperados, possivelmente distantes do que seria sua real preferência. É comum referir-se a esse fenômeno (p. ex., [**?**]) como "mais é menos", no sentido de que o aumento da disponibilidade de escolhas pode confundir o usuário e diminuir sua satisfação.

A título de exemplo, uma pesquisa pelo termo `webserver` no *SourceForge* retorna 620 projetos cadastrados. O usuário então define seus critérios para continuar uma análise mais apurada, eliminando nesta primeira etapa o que certamente não se adequada ao seu ambiente. O administrador de um sistema crítico, por exemplo, investigaria o histórico de falhas de segurança do *software*; para um *hardware* modesto, como por exemplo vê-se nos *netbooks* e *smartphones*, daria-se preferência a aplicativos de baixo custo computacional; já um usuário de *desktop* usualmente tem interesse em aplicativos atualizados, o que demanda uma comunidade de desenvolvimento ativa; uma empresa necessita de programas com termos de licenciamento claros, de modo a evitar eventuais conflitos jurídicos; programas com alto índice de popularidade — tipicamente mensurados pela quantidade de downloads — destacam-se na preferência dos usuários de uma maneira geral.

Notamos portanto uma infinita gama de possibilidades no que diz respeito aos critérios para seleção de um *software*. Há também uma variedade de fontes a se consultar com o

[1] http://www.sourceforge.net

[2] Em 2 de setembro de 2011 o *SourceForge* registra 308.307 projetos, sendo que 180.127 estão sob licenciamento aprovado pela *Open Source Initiave*, que tratam das licenças de software livre e código aberto mais populares.

objetivo de colocar lado a lado aqueles requisitos e o que o programa de fato oferece, com o objetivo de continuar o processo de filtragem. Pode-se consultar estatísticas de *commits* de código, atividades em listas de e-mail, *bugs* não resolvidos nos sistemas de notificação de falhas (*bug tracking systems*), histórico de falhas nos boletins de segurança, últimas novidades nos *blogs* dos projetos etc. Algumas destas informações são apresentadas como opções de filtragem nos próprios repositórios, como ocorre no *SourceForge*. No entanto, após uma série de filtragens de caráter manual, como a leitura das descrições daquilo que ainda não foi filtrado, comumente o usuário ainda encara uma quantidade razoável de opções, das quais muitas são inadequadas para suas necessidades e demandam análises ainda mais minuciosas.

Até este ponto relatamos o que tipicamente ocorre com usuários dotados de certa habilidade técnica, que sabem onde e como realizar as buscas, além de serem capazes de definir seus critérios embasados em conhecimento de caráter técnico. Temos por outro lado o usuário regular de *desktop*, que utiliza programas de computador como meio para outras atividades, não tem conhecimentos técnicos aprofundados e muitas vezes é limitado pela barreira do idioma. Neste contexto, repositórios como o *SourceForge* não são de grande utilidade. Estes usuários requerem um sistema simples, no seu idioma e de preferência presente no seu ambiente doméstico de trabalho, que os auxiliem a fazer escolhas acerca de quais aplicativos instalar.

A inexistência de um sistema livre capaz de oferecer tais recomendações motivaram a concepção e desenvolvimento do *AppRecommender*. Um recomendador de aplicativos exerceria papel relevante em ambos os contextos ilustrados, pois independente da habilidade ou conhecimento do usuário, o excesso de possibilidades traz dificuldades ao processo de escolha. Ao recorrer a tal sistema os usuários poupariam tempo e recursos outrora dedicados a buscas e filtragens manuais para encontrar os aplicativos mais adequados a seu ambiente de trabalho. Em contrapartida, a comunidade de desenvolvedores tiraria proveito de um consequente aumento na utilização de seus programas que, por serem experimentados por mais usuários, certamente receberiam mais relatórios de erro, sugestões e contribuições diversas.

No caso de uso típico do *AppRecommender*, o usuário submete a lista de aplicativos instalados em seu sistema e o recomendador lhe sugere outros programas que ele não possui, mas provavelmente seriam de seu interesse. No âmbito deste trabalho foram desenvolvidas uma interface em modo texto e um serviço Web[3] para este fim. O recomendador pode ainda ser facilmente integrado a outros sistemas, como os gerenciamento de aplicativos.

A recomendação oferecida é personalizada, fundamentada nas características específicas do usuário, em alternativa a recomendações generalistas, como as baseadas em popularidade dos aplicativos. Sugestões individualizadas requerem a identificação de atributos que diferenciem o usuário do restante da população. Este é um dos grandes desafios do recomendador, que utiliza conceitos específicos do domínio de aplicação na extração de atributos para composição de um perfil, como as relações de dependência entre os programas.

A filtragem do repositório de aplicativos para um determinado usuário ocorre por meio de buscas, fundamentando-se em técnicas de recuperação da informação. Os parâmetros das consultas variam de acordo com a estratégia escolhida, podendo considerar as escolhas prévias do próprio usuário, caracterizando desta forma um perfil de características de aplicativos; as escolhas de usuários com características similares às dele, referenciados como *vizinhos* neste contexto; ou uma abordagem mista. Diferentes estratégias para composição de recomendações foram implementadas, visando uma posterior análise de desempenho, que nos auxiliaria a escolher a estratégia a ser adotada pelo serviço.

Experimentos preliminares foram realizados e uma consulta pública está em curso para coletar avaliações de usuários reais sobre as recomendações produzidas. O presente texto relata as decisões de projeto que embasaram o desenvolvimento do protótipo disponível atualmente, além dos procedimentos realizados para validar o sistema.

O desenvolvimento deste trabalho foi realizado em colaboração com desenvolvedores

[3] http://recommender.debian.net

e usuários da comunidade Debian. Diversos serviços já providos pela distribuição foram integrados na solução, apresentando-se portanto como uma vitrine para estas iniciativas, algumas das quais desconhecidas pelo público.

Esta dissertação está organizada da seguinte forma: os capítulos 2 e 3 trazem uma breve introdução sobre distribuições GNU/Linux e sistemas de recomendação; no capítulo 4 trabalhos correlatos são apresentados, enquanto o 5 propõe o *AppRecommender* como uma solução para o problema exposto. Os experimentos realizados para validar a proposta são analisados no capítulo 6. Por fim, no capítulo 7 relatamos algumas considerações que julgamos relevantes em respeito aos resultados obtidos e às perspectivas de trabalhos futuros.

Capítulo 2

Distribuições GNU/Linux

O contexto histórico no qual emergiu o que se conhece hoje por distribuições GNU/Linux é abordado a seguir. Especial atenção é dedicada aos princípios de "empacotamento" de *softwares* e seus sistemas de gerenciamento, conceitos que ganharam destaque no processo de consolidação das principais distribuições — e que guardam estreita relação com o presente trabalho.

2.1 Surgimento

Distribuições GNU/Linux, popularmente conhecidas como *distros*, são variações do sistema operacional composto pelo núcleo Linux (*kernel*) e milhares de aplicativos, cuja base foi desenvolvida pelo projeto GNU. As primeiras iniciativas neste domínio surgiram em circunstâncias que favoreciam o desenvolvimento colaborativo, abertura de código e comunicação predominantemente por meio da Internet.

O projeto GNU[1] foi criado em 1983 por Richard Stallman com o objetivo principal de desenvolver um sistema operacional livre em alternativa ao UNIX[2] – solução comercial amplamente difundida na indústria – e que fosse compatível com os padrões POSIX[3]. Nos anos 90 o projeto GNU já havia atraído muitos colaboradores, que num curto espaço de tempo desenvolveram inúmeros aplicativos para compor o sistema operacional. No entanto, o desenvolvimento do núcleo do sistema (*GNU Hurd*) não acompanhou o ritmo dos demais aplicativos.

Em outubro de 1991 o estudante finlandês Linus Torvalds, na tentativa de atrair colaboradores, publicou código do Freax, o núcleo de um sistema operacional desenvolvido por ele na universidade. Anos mais tarde Torvalds declara que não imaginava que aquele projeto desenvolvido sem grandes pretensões teria a dimensão do que hoje se conhece como Linux [**?**].

Com o anúncio de Torvalds, Stallman vislumbrou a possibilidade de acelerar o lançamento do sistema operacional livre se os aplicativos GNU que já estavam prontos fossem combinados com o núcleo recém-lançado – de fato, a primeira versão estável do GNU Hurd foi lançada apenas em 2001. Em 1992 o Linux foi licenciado sob a GNU GPL[4] e as equipes dos dois projetos começaram a trabalhar na adaptação do *kernel* Linux para o ambiente GNU. Este esforço conjunto desencadeou o surgimento das primeiras distribuições GNU/Linux.

As distros oferecem diferentes "sabores" do sistema operacional, a exemplo do Debian, Fedora, Mandriva e Ubuntu, que são constituídos por aplicativos criteriosamente selecionados

[1] hhttp://www.gnu.org

[2] http://www.unix.org/

[3] Acrônimo para *Portable Operating System Interface*. Família de normas definidas pelo IEEE com foco na portabilidade entre sistemas operacionais. http://standards.ieee.org/develop/wg/POSIX.html

[4] Acrônimo para *General Public License*, é um suporte legal para a distribuição livre de softwares.

por seus desenvolvedores. Tais iniciativas tendem a reduzir a complexidade de instalação e atualização do sistema para usuários finais [**?**]. Os desenvolvedores das distribuições atuam como intermediários entre os usuários e os autores dos aplicativos, por meio do encapsulamento de componentes de software em abstrações denominadas *pacotes*.

2.2 Empacotamento de programas

O termo empacotamento refere-se ao ato de reunir num único arquivo (o pacote), um conjunto de programas executáveis, bibliotecas, arquivos de configuração, documentação, e qualquer outro dado necessário para a utilização de um programa no sistema operacional.

As distribuições que optam por disponibilizar pacotes mantêm uma infraestrutura de servidores como fonte de distribuição de programas. Tais servidores, denominados "repositórios", podem ser mantidos oficialmente pela distribuição ou oferecidos por terceiros, considerados portanto "não oficiais". Os sistemas gerenciadores de pacotes, responsáveis por manter uma base consistente de programas no sistema, são configurados para buscar os pacotes a serem instalados em um determinado conjunto de repositórios.

2.2.1 Pacotes Debian

Debian GNU/Linux e distribuições derivadas, entre elas a *Ubuntu*[5], utilizam o formato de pacote binário *deb*, composto por arquivos executáveis, bibliotecas e documentação associados a um programa ou conjunto de programas relacionados. Idealmente, todos os dados e procedimentos necessários para instalar, configurar e remover os aplicativos de um sistema estão contidos em seu pacote.

A estrutura dos pacotes e repositórios, bem como os requisitos que um pacote deve atender para que seja distribuído oficialmente estão especificados no Manual de Políticas Debian[6]. Uma das exigências deste documento é que os pacotes estejam em conformidade com padrões que visam a interoperabilidade com outros sistemas GNU/Linux, como o *Filesystem Hierarchy Standard (FHS)*, referente à localização de arquivos no sistema, e recomendações para aplicativos gráficos estabelecidos pelo *FreeDesktop.org*[7].

A estrutura de um pacote Debian pode ser observada na figura 2.1, que exibe o conteúdo do pacote *cups* extraído pela ferramenta *ar*. O arquivo *debian-binary* contém a versão da especificação de empacotamento implementada no pacote (linhas 4 e 5). *control.tar.gz* contém scripts e arquivos de controle utilizados principalmente pelo gerenciador de pacotes no momento de instalação, configuração e remoção do pacote (linhas 8 a 16). *data.tar.gz* contém todos os binários e demais arquivos que devem ser copiados para os devidos diretórios do sistema de arquivos (linhas 19 a 21).

Para cada pacote no repositório oficial existe um desenvolvedor ou equipe responsável por sua manutenção. O mantenedor acompanha o desenvolvimento do *software* original, que neste contexto é denominado *upstream*, e incorpora as correções e atualizações dos aplicativos ao contexto do Debian. Espera-se ainda que ele interaja com o desenvolvedor principal e retribua aos projetos originais as melhorias implementadas no âmbito do Debian. A tabela 2.1 sumariza os principais papéis exercidos por desenvolvedores de *software* no ecossistema de programas empacotados para o Debian.

O acesso de escrita aos repositórios do Debian é controlado por uma rede de confiança com base em assinatura de chaves criptográficas assimétricas. Para fazer parte da rede, um colaborador precisa ter a sua chave assinada por pelo menos um membro do projeto e a

[5] http://www.ubuntu.com
[6] http://www.debian.org/doc/debian-policy/
[7] http://www.freedesktop.org/

```
1  $ ar -x cups_1.4.8-2_i386.deb
2  $ ls
3   control.tar.gz  cups_1.4.8-2_i386.deb  data.tar.gz  debian-binary
4  $ cat debian-binary
5   2.0
6  $ tar -tzf control.tar.gz
7   ./
8   ./prerm
9   ./templates
10  ./md5sums
11  ./config
12  ./control
13  ./conffiles
14  ./preinst
15  ./postrm
16  ./postinst
17 $ tar -tzf data.tar.gz
18  ./
19  ./etc/
20  ./usr/
21  ./var/
```

Figura 2.1: Extração do conteúdo de um pacote

Papel	**Descrição**
Mentor	Pessoa experiente que orienta mantenedores novatos nas atividades de empacotamento.
Mantenedor	Pessoa ou equipe que mantém o pacote Debian.
Sponsor	Desenvolvedor com acesso de escrita ao repositório oficial que revisa e envia aos servidores pacotes de outros mantenedores que não possuem esta permissão.
Upstream	Autor ou mantenedor responsável pelo desenvolvimento do *software* original.

Tabela 2.1: Papéis exercidos por desenvolvedores no Debian

troca de chaves deve necessariamente acontecer pessoalmente, mediante a conferência de um documento de identificação do proponente.

O envio de um novo pacote ao repositório oficial deve obrigatoriamente ser realizado por um desenvolvedor membro oficial do projeto (*Debian Developer (DD)*). As atualizações subsequentes podem também ser realizadas por um mantenedor oficial (*Debian Maintainer (DM)*), que não passou pelo processo de se tornar DD, mas faz parte da rede de confiança e mantém pacotes oficialmente. Colaboradores que não são DD ou DM podem igualmente exercer a função de mantenedor, porém, precisam do intermédio de um *sponsor* para enviar seus pacotes ao servidor.

A título de curiosidade, em outubro de 2010 foi aprovado em votação[8] que o Debian passaria a acolher membros não empacotadores. Em reconhecimento à importância de atividades não-técnicas para o projeto, a partir desta data, podem ser admitidos como membros oficiais (DDs) colaboradores engajados em atividades como tradução, publicidade, *design* gráfico, entre outras. A admissão destes membros não testa seu conhecimento técnico, consequentemente, eles não têm acesso de escrita aos repositórios.

2.3 Relação entre pacotes

Segundo [**?**], as relações entre pacotes podem ser caracterizadas como requisitos positivos, quando representam uma dependência, ou requisitos negativos, quando indicam um conflito. A tabela 2.2 traz a descrição de relações possíveis entre os pacotes fictícios a, b e c, que seriam declaradas no conteúdo do pacote a, em seu arquivo *control* (linha 12 da figura 2.1). Existem

[8] http://www.debian.org/vote/2010/vote_002

ainda relações entre pacotes fontes[9] que deliberadamente não foram listadas por estarem fora do escopo deste trabalho.

Relação	Descrição
a `Depends` *b*	`Dependência absoluta`: *a* não será configurado pelo gerenciador de pacotes a menos que *b* tenha sido previamente configurado.
a `Pre-Depends` *b*	`Pré-dependência absoluta`: a instalação completa de *b* deve ser realizada antes que a instalação de *a* seja iniciada.
a `Recommends` *b*	`Recomendação`: dependência forte, mas não absoluta; *a* e *b* são comumente utilizados em conjunto, apesar de não haver obrigatoriedade para tal; em geral os gerenciadores de pacotes instalam recomendações por padrão.
a `Suggests` *b*	`Sugestão`: a utilização de *a* está relacionada com o uso de *b*, que pode aumentar a utilidade de *a*, no entanto, a não instalação de *b* é perfeitamente aceitável.
a `Enhances` *c*	`Melhoria`: *a* aumenta a funcionalidade de *c*; significado equivalente a *suggests*, porém declarado no pacote que aumenta a funcionalidade do outro.
a `Breaks` *b*	`Quebra`: o gerenciador não prossegue com a instalação de *a* sem que *b* seja previamente desconfigurado.
a `Conflicts` *b*	`Conflito`: restrição maior do que a quebra, pois impede que a instalação do pacote tenha início antes da completa remoção dos indicados como conflito.
a `Provides` *b*	`Fornecimento`: o pacote provê a funcionalidade *b*, representada por um pacote virtual (*b* não existe de fato no repositório).
a `Replaces` *b*	`Substituição`: *a* substitui *b*, portanto na instalação de *a* os arquivos de *b* podem ser sobrescritos.

Tabela 2.2: Descrição das relações entre pacotes Debian

O conceito de pacotes virtuais foi criado especialmente para situações em que diversos pacotes diferentes oferecem um conjunto de funcionalidades semelhantes. Pacotes virtuais não existem fisicamente no repositório, são apenas mencionados na definição de outros pacotes ("concretos"). Por exemplo, os pacotes *ssmtp* e *postfix* oferecem a funcionalidade de um agente de transporte de mensagem (servidor de *e-mail*), portanto o arquivo *control* de ambos os pacotes conterá a informação "`Provides: mail-transport-agent`". Quando uma dependência se refere a um pacote virtual, ela pode ser satisfeita com a instalação de qualquer pacote que provê o mesmo.

Informações sobre o relacionamento de um determinado pacote com outros do repositório podem ser obtidas por meio da ferramenta `apt-cache`. A figura 2.2 apresenta detalhes do pacote *apt*, com declarações de dependências entre as linhas 10 e 15. Em cada linha, o identificador da relação é seguido por uma lista de nomes de pacotes separados por vírgulas. Quando há uma lista de pacotes alternativos que satisfazem uma relação, eles aparecem separados por uma barra vertical. No exemplo dado, se ao menos um entre os pacotes *aptitude*, *synaptic* e *wajig* estiver presente no sistema, o termo `Suggests: aptitude|synaptic|wajig` é considerado satisfeito (linha 14).

Todos os campos, com exceção de `Provides`, podem restringir sua aplicabilidade a versões particulares indicadas em parênteses juntamente com uma das relações << , <=, =, >= e >>, para estritamente menor, menor ou igual, igual, maior ou igual e estritamente maior, respectivamente.

2.4 Sistemas gerenciadores de pacotes

Os gerenciadores de pacotes são sistemas projetados para coordenar as ações de aquisição, instalação, atualização e remoção de pacotes no sistema operacional, mantendo o estado do sistema consistente. Em sua estrutura os pacotes declaram dependências e conflitos com outros pacotes, procedimentos que devem ser executados antes ou depois da instalação ou

[9] http://www.debian.org/doc/debian-policy/ch-relationships.html

```
$ apt-cache show apt
Package: apt
Status: install ok installed
Priority: important
Section: admin
Installed-Size: 6168
Maintainer: APT Development Team <deity@lists.debian.org>
Architecture: i386
Version: 0.8.15.2
Replaces: manpages-pl (<< 20060617-3~)
Provides: libapt-pkg4.10
Depends: debian-archive-keyring, gnupg
Pre-Depends: libc6 (>= 2.3.4), libgcc1 (>= 1:4.1.1), libstdc++6 (>= 4.6), zlib1g
Suggests: aptitude | synaptic | wajig, dpkg-dev, apt-doc, bzip2, lzma, python-apt
Conflicts: python-apt (<< 0.7.93.2~)
Conffiles:
 /etc/apt/apt.conf.d/01autoremove b9bbfaa2954b0499576b8d00c37d6a34
 /etc/cron.daily/apt b2f73cd2b7d6cb5a087aba504ea9f507
 /etc/logrotate.d/apt 179f2ed4f85cbaca12fa3d69c2a4a1c3
Description: Advanced front-end for dpkg
 This is Debian's next generation front-end for the dpkg package manager.
 It provides the apt-get utility and APT dselect method that provides a
 simpler, safer way to install and upgrade packages.
 .
 APT features complete installation ordering, multiple source capability
 and several other unique features, see the Users Guide in apt-doc.
```

Figura 2.2: Detalhes do pacote *apt*

remoção (*postinst*, *postrm*, etc), diretórios onde os executáveis, configurações e documentações devem ser posicionados, entre outras informações.

O tratamento de dependências e conflitos é uma das mais importantes e críticas funções de um sistema gerenciador. Ao receber uma requisição – instalação de um novo aplicativo, por exemplo – o gerenciador tenta modificar o estado do sistema satisfazendo todas as restrições indicadas pelo pacote. Promove a instalação de todas as dependências antes de instalá-lo, ao passo que não permite a instalação de pacotes que conflitam com outros já instalados no sistema. Alguns são capazes de oferecer diferentes soluções para problemas de dependências não satisfeitas ou conflitos, e cabe ao usuário escolher a que melhor lhe convier.

Atualizações parciais dos sistemas e o lançamento incessante de novas versões dos pacotes nos repositórios por vezes provocam situações de inconsistência indesejável, que o gerenciador não é capaz de resolver sem intervenção humana. Esta dificuldade em lidar de forma automatizada com este problema é uma das razões para a popularização de esquemas de instalação independentes de pacotes e sistemas gerenciadores, como na distribuição *Gentoo*[10].

A seguir são listados alguns exemplos de sistemas de gerenciamento de pacotes.

2.4.1 Advanced Packaging Tool (APT)

O gerenciamento de pacotes em sistemas Debian GNU/Linux e derivados é realizado através do *APT*[11], considerado um gerenciador de alto nível. Por meio do *APT* são realizadas ações como busca, obtenção, instalação, atualização e remoção de pacotes. Após o tratamento de dependências e definição da ordem de instalação dos pacotes, o *APT* aciona o *dpkg*[12] (nível médio) que apenas checa a satisfação de dependências e por fim aciona o *dpkg-deb* para de fato ler os arquivos de controle e extrair os demais arquivos, instalando-os nos diretórios indicados.

O *Synaptic* é um gerenciador de pacotes gráfico desenvolvido em GTK+ como uma interface amigável para o *APT* (figura 2.3).

[10] http://www.gentoo.org/main/pt_br/philosophy.xml
[11] http://wiki.debian.org/Apt
[12] http://wiki.debian.org/dpkg

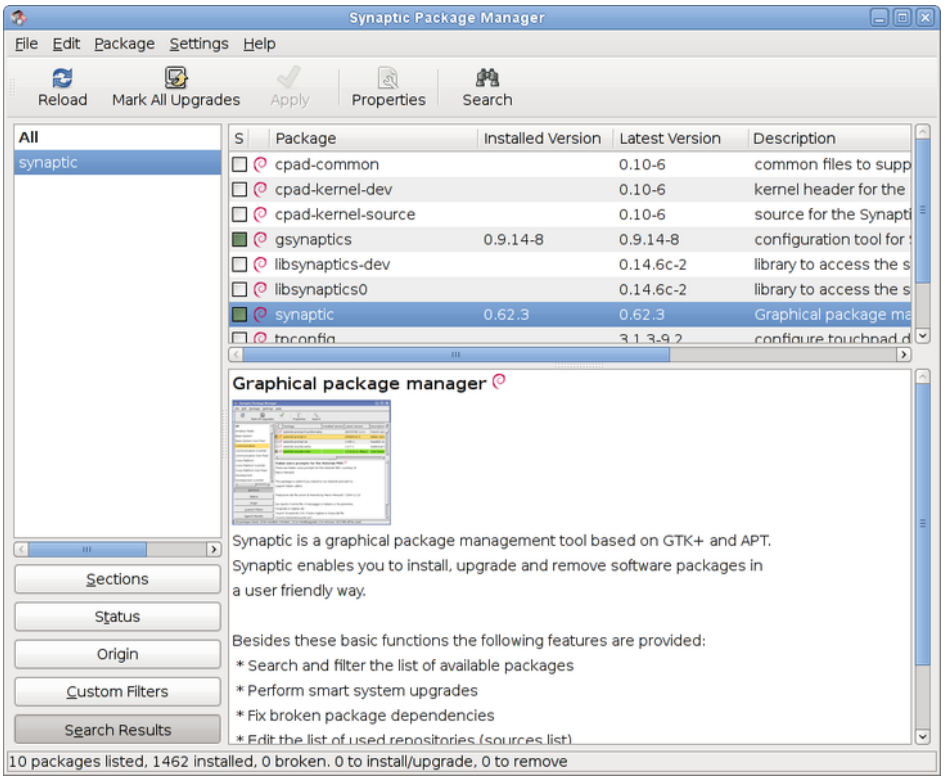

Figura 2.3: Captura de tela do synaptic

2.4.2 Ubuntu Software Center

Ferramenta inicialmente denominada *AppCenter*, foi projetada com o intuito de centralizar as atividades relativas à instalação de aplicativos em sistemas Ubuntu (navegação pelo repositório, instalação e remoção de pacotes). Foi desenvolvido com base no *gnome-app-install*, sendo escrito em Python e usa GTK+ como biblioteca gráfica. Serve como interface para o *APT*, dado que o sistema de empacotamento desta distribuição é herdado do Debian.

2.5 Seleção de programas

O conjunto de programas instalados num determinado sistema é tipicamente resultado de dois processos de seleção: o primeiro é realizado no âmbito do desenvolvimento da distribuição e o segundo faz parte da manutenção cotidiana do sistema, realizado pelo usuário e/ou administrador da máquina.

A seleção e configuração dos aplicativos básicos de uma distribuição (instalados por padrão) são de responsabilidade da equipe que a desenvolve, com diferentes níveis de interferência da comunidade. Este é um ponto bastante sensível nos projetos, dado que é um dos fatores que influenciam a escolha dos usuários finais por qual distribuição adotar, e pode revelar eventuais conflitos de interesses. Por exemplo, a decisão da Canonical que determinou, sem debates públicos, a substituição do popular gerenciador de janelas GNOME por um outro menos maduro, expôs um modelo em que a comunidade e mesmo os desenvolvedores envolvidos exercem papel limitado nos rumos do projeto [**?**].

Por outro lado, após a configuração de um sistema básico, este é geralmente personalizado para atender às demandas específicas dos usuários finais, por meio da instalação de programas adicionais. Ainda que a infraestrutura de instalação de softwares provida pelas distribuições (geralmente baseada em pacotes) simplifique a manutenção de sistemas [**?**], a seleção dos programas inevitavelmente dependerá de uma ação humana. Com o desenvolvimento deste

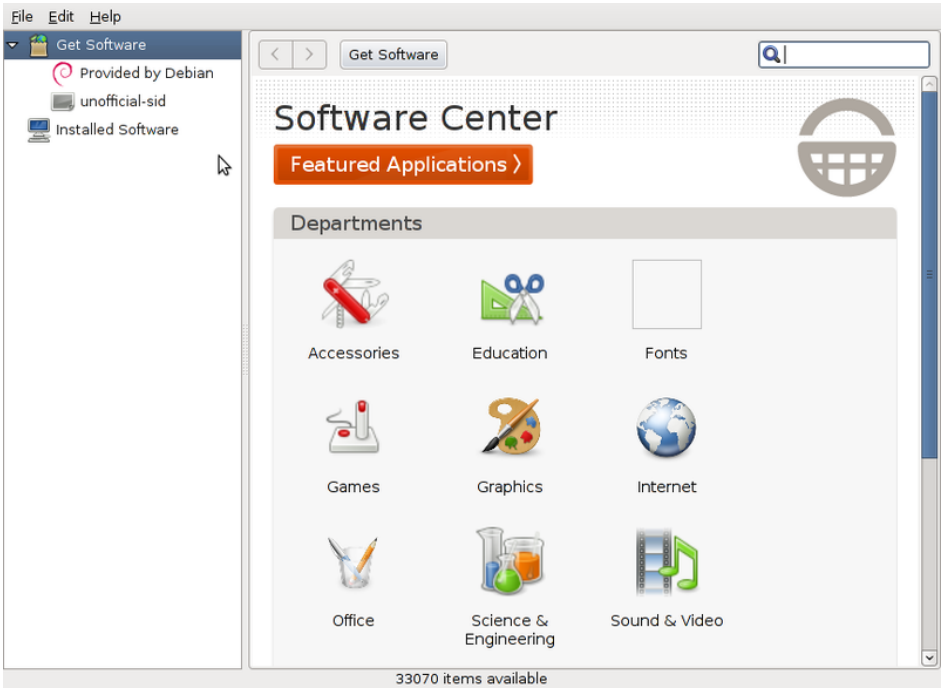

Figura 2.4: Captura de tela do software-center

trabalho pretende-se auxiliar o indivíduo nesta tarefa, especialmente no cenário em que o usuário não é dotado de experiência pessoal e habilidade suficientes para decidir qual aplicativo instalar.

Capítulo 3

Sistemas de recomendação

Este capítulo apresenta uma breve introdução ao domínio de sistemas de recomendação. Circunstâncias que motivaram o surgimento de tais sistemas, objetivos e desafios comuns no desenvolvimento, técnicas que apoiam a composição das recomendações e métricas de avaliação são alguns dos tópicos abordados a seguir.

3.1 Contexto histórico

A popularização de recursos computacionais e do acesso à Internet nas últimas décadas contribuiu para o aumento expressivo na quantidade e diversidade de conteúdo e serviços à disposição dos usuários. Um dos fatores para este aumento é que indivíduos que anteriormente limitavam-se ao papel de consumidores de conteúdo, hoje colocam-se numa posição de produtores. Surgem inúmeros casos de sucesso de serviços criados e/ou mantidos por internautas independentes, a exemplo de *blogs*, enciclopédias colaborativas como a Wikipedia[1], repositórios para compartilhamento de fotografia e vídeo, como Flickr[2] e Youtube[3], entre outros. [**?**] analisa este fenômeno, comumente referenciado como *Web 2.0*, afirmando que a maioria da população acredita que pode influenciar outras pessoas atuando no mundo através da sua força de vontade e utilizando seus próprios meios.

Recomendações, sugestões ou simples indicações do que se julga mais ou menos adequado numa determinada situação são fenômenos bastante comuns nas relações sociais. Um exemplo de recomendação tradicional são as avaliações de livros e filmes produzidas por críticos de arte e publicadas nos principais jornais e revistas especializadas.

Na Internet, que é também uma rede de interação social, refletem-se estes mesmos comportamentos. Expandem-se entretanto no mundo digital a um montante de atores e informação disponível muito mais elevados que no plano do tangível. Dada esta peculiaridade da grande rede, é natural que os processos de indicação sejam também mais sofisticados — território dos sistemas especializados em recomendação, que fundamentam-se na opinião e comportamento de usuários não especializados.

A empresa Netflix[4], locadora de filmes norte-americana, tornou-se referência na década de 90 ao utilizar preferências de usuários e histórico de compras dos usuários para produção de recomendações automatizadas.

[1] http://wikipedia.org
[2] http://flickr.com
[3] http://youtube.com
[4] http://www.netflix.com

3.2 O problema computacional

O problema da recomendação é comumente formalizado através de uma estrutura de pontuação como representação computacional da utilidade dos itens para os usuários ou clientes. A partir de avaliações feitas pelos próprios usuários do sistema, tenta-se estimar pontuações para os itens que ainda não foram avaliados pelos mesmos. Uma vez que esta estimativa tenha sido feita, pode-se recomendar os itens com maior pontuação estimada.

Todavia, a utilidade é um conceito subjetivo e difícil de mensurar, principalmente porque, em diversos contextos, a identificação dos fatores que a determinam não é uma tarefa trivial. Portanto, com a ressalva de que estas medidas não representam necessariamente a realidade, as pontuações são usadas como aproximações, pois têm como base as avaliações registradas pelos próprios usuários.

3.3 Ações e desafios

Sistemas recomendadores são implementados nos mais diversos contextos e podem ser desenvolvidos para propósitos distintos, referenciados na literatura como *ações* de sistemas de recomendação. Por exemplo, a recomendação pode se limitar a encontrar os itens mais relevantes, porém, em alguns casos, é interessante que sejam retornados todos os itens relevantes; outra possibilidade é recomendar uma sequência de itens, quando não somente os itens recomendados importam mas também a ordem em que eles são apresentados; a navegação num extenso repositório de itens também pode ser beneficiada por um recomendador que apresenta primeiramente os itens que o usuário deve se interessar [**?**].

Ações distintas não representam necessariamente a necessidade de aplicação de técnicas distintas, visto que é basicamente a apresentação dos resultados que vai ser diferenciada e não o cálculo da recomendação em si. No entanto, a avaliação de eficácia de um recomendador está diretamente relacionada com sua ação principal. Por exemplo, se apenas os mais relevantes serão apresentados ao usuário, é de extrema importância que os primeiros itens recomendados sejam acertados, enquanto que no caso de retorno de todos os relevantes o ponto chave é que nenhum item relevante seja desconsiderado, mesmo que os primeiros apresentados sejam irrelevantes.

Os desafios do desenvolvimento de tais sistemas estão relacionados a questões inerentes ao problema e sua representação computacional. As estratégias e técnicas propostas devem levar em conta tais questões, algumas das quais foram apontadas por [**?**] e são citadas a seguir.

Qualidade das recomendações

Usuários esperam recomendações nas quais eles possam confiar. Esta confiabilidade é alcançada na medida em que se diminui a incidência de falsos positivos. Em outras palavras, deve-se evitar recomendações que não interessam ao usuário.

Esparsidade

A existência de poucas relações usuário-item resulta numa matriz de relacionamentos esparsa, o que dificulta a localização de usuários com preferências semelhantes (relações de vizinhança) e resulta em recomendações fracas.

Escalabilidade

A complexidade do cálculo de recomendações cresce tanto com o número de clientes quanto com a quantidade de itens, portanto a escalabilidade dos algoritmos é um ponto importante a ser considerado.

Transitividade de vizinhança

Usuários que têm comportamento semelhante a um determinado usuário não necessariamente têm comportamento semelhante entre si. A captura deste tipo de relação pode ser desejável mas em geral esta informação não é resguardada, exigindo a aplicação de métodos específicos para tal.

Sinônimos

Quando o universo de itens possibilita a existência de sinônimos, a solução deve levar esta informação em conta para prover melhores resultados.

Primeira avaliação

Um item só pode ser recomendado se ele tiver sido escolhido por um usuário anteriormente. Portanto, novos itens precisam ter um tratamento especial até que sua presença seja notada.

Usuário incomum

Indivíduos com opiniões que fogem do usual, que não concordam nem discordam consistentemente com nenhum grupo, normalmente não se beneficiam de sistemas de recomendações.

3.4 Seleção de atributos

Uma grande quantidade de atributos a ser considerada resulta em alta complexidade computacional, além de geralmente mascarar a presença de ruídos. A fim de amenizar este problema, é comum a realização de um processo de *seleção de atributos*, que consiste em escolher algumas características dos dados e utilizar apenas este como conjunto de treinamento para a classificação, evitando assim o super-ajuste (*overfitting*). Esta seleção equivale à substituição de um classificador complexo por um mais simples. [**?**] defende que, especialmente quando a quantidade de dados de treinamento é limitada, modelos mais fracos são preferíveis.

A seleção de atributos geralmente é realizada para cada classe em separado, seguida pela combinação dos diversos conjuntos. Abaixo são apresentados alguns métodos de escolha.

Informação mútua. Análise de quanto a presença ou ausência de um atributo contribui para a tomada de decisão correta por uma determinada classe. Informação mútua máxima significa que o atributo é um indicador perfeito para pertencimento a uma classe. Isto acontece quando um objeto apresenta o atributo se e somente se o objeto pertence à classe.

Independência de eventos. Aplicação do teste estatístico χ^2 para avaliar a independência de dois eventos – neste caso, um atributo e uma classe. Se os dois eventos são dependentes, então a ocorrência do atributo torna a ocorrência da classe mais provável.

Baseado em frequência. Seleção dos atributos mais comuns para uma classe.

Os métodos apresentados acima são "gulosos", ou seja, assumem escolhas ótimas locais na esperança de serem ótimas globais. Como resultado, podem selecionar atributos que não acrescentam nenhuma informação para a classificação quando considerados outros previamente escolhidos. Apesar disto, algoritmos não gulosos são raramente utilizados em virtude do seu alto custo computacional [**?**].

3.5 Estratégias de recomendação

O presente trabalho considera uma classificação de estratégias de recomendação baseada em taxonomias propostas por diferentes autores. A peculiaridade de cada abordagem está relacionada com a fonte de dados utilizada para produzir o conhecimento do recomendador e o tipo de técnica aplicada para extrair as recomendações.

3.5.1 Reputação dos itens

Popular entre serviços de venda como livrarias, sites de leilão e lojas em geral, esta estratégia consiste no registro de avaliações dos produtos produzidas por usuários, bem como na apresentação das mesmas no momento e local apropriado [**?**].

Atualmente existem serviços especializados em reputação de produtos que apenas disponibilizam as avaliações, sem haver venda associada. Alguns exemplos são o *Trip Advisor*[5], que oferece avaliações sobre hotéis, restaurantes e recomendações em geral para viagens, e o *Internet Movie Database*[6], que armazena uma vasta coleção de informações sobre cinema (figura 3.1).

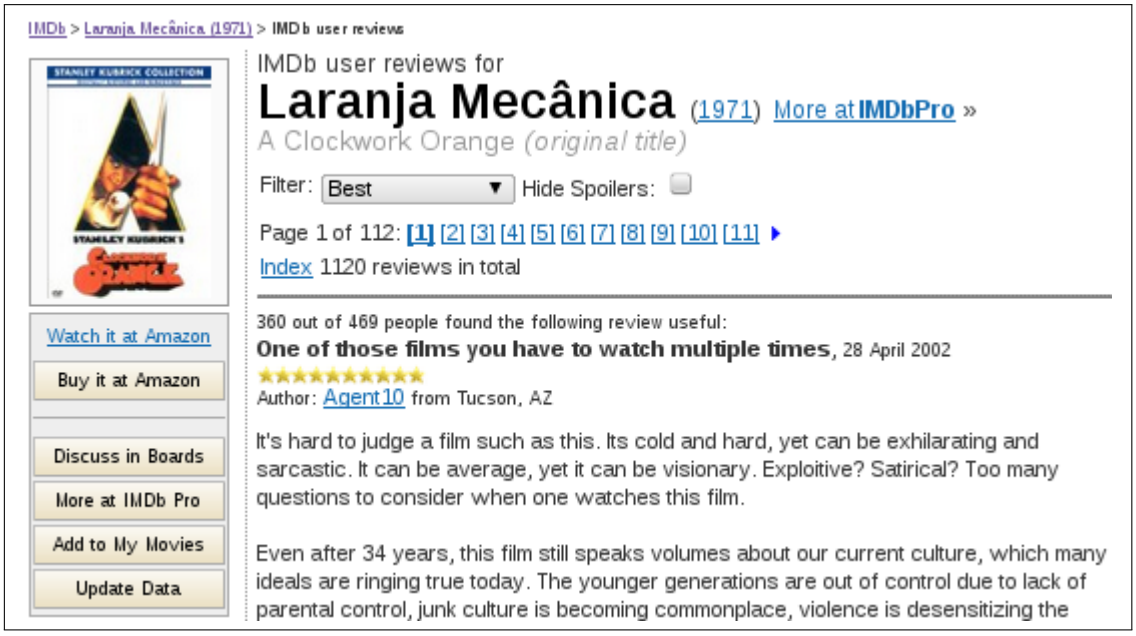

Figura 3.1: Avaliação de usuário no IMDb

A eficácia deste tipo de recomendação está diretamente relacionada com a qualidade das avaliações produzidas pelos usuários. A depender da expertise do indivíduo sobre determinado tema, ele pode se mostrar rigoroso ou permissivo demais em suas avaliações. Por este motivo, não é raro a ocorrência de avaliações conflitantes para um mesmo item. Outra dificuldade é lidar a parcialidade dos usuários em suas opiniões, que geralmente não pode ser atestada, principalmente quando tratam de questões subjetivas ou percepções pessoais. Uma maneira de lidar com estas questões é atrelar o conceito de reputação também para avaliadores ou para as próprias avaliações. Neste caso, o fato de uma avaliação ter sido bem avaliada por outros usuários tende a aumentar sua confiança. A figura 3.2 ilustra como a qualidade das avaliações produzidas por usuários pode comprometer a reputação de um item (fonte: xkcd[7]).

[5] http://www.tripadvisor.com/
[6] http://www.imdb.com/
[7] http://xkcd.com/937/

Figura 3.2: Problema com a extração da média de avaliações

Esta é uma abordagem de simples implementação visto que geralmente depende apenas da manutenção dos dados originais. Os desafios surgem quando se tenta quantificar automaticamente as avaliações, seja pelo processamento do texto e classificação entre avaliação boa ou ruim, ou ainda, quando a reputação é composta por meio de outros parâmetros, por exemplo, a quantidade de vendas, reclamações ou devoluções de um produto.

3.5.2 Recomendação baseada em conteúdo

Esta abordagem parte do princípio de que os usuários tendem a se interessar por itens semelhantes aos que eles já se interessaram no passado [**?**]. Em uma livraria, por exemplo, sugerir ao cliente outros livros do mesmo autor ou tema dos previamente selecionados é uma estratégia amplamente adotada.

O ponto chave desta estratégia é a representação dos itens por meio de suas características, por exemplo, descrever um livro pelo conjunto `{título, autor, editora, tema}`. A partir da identificação de atributos, aplica-se técnicas de recuperação da informação com o intuito de encontrar itens semelhantes ou de classificação para encontrar itens relevantes. Algumas técnicas aplicáveis neste contexto são descritas na seção 3.6.

Pelo fato de se apoiar na classificação dos itens, os resultados da recomendação são prejudicados nos casos em que os atributos não podem ser identificados de forma automatizada. Outro problema é a superespecialização, ou seja, a abrangência das recomendações fica limitada a itens similares aos já escolhidos pelos usuários [**?**].

3.5.3 Recomendação colaborativa

A recomendação colaborativa é fundamentada na troca de experiências entre indivíduos que possuem interesses em comum, portanto não exige o reconhecimento semântico do conteúdo dos itens. Esta estratégia é inspirada na técnica de classificação *K-Nearest Neighbors (K-NN)*, apresentada na seção 3.6.1.

Figura 3.3: Cenário de uma recomendação baseada em conteúdo

A vizinhança de um determinado usuário é composta pelos usuários que estiverem mais próximos a ele. O ponto chave desta abordagem é a definição da função que quantifica a proximidade entre os usuários, que também pode ser herdada de soluções em classificação e recuperação da informação. A recomendação é então produzida a partir da análise dos itens que os seus vizinhos consideram relevantes. Geralmente os itens que ocorrem com maior frequência na vizinhança compõem a recomendação. A figura 3.4 ilustra o cenário de uma recomendação colaborativa.

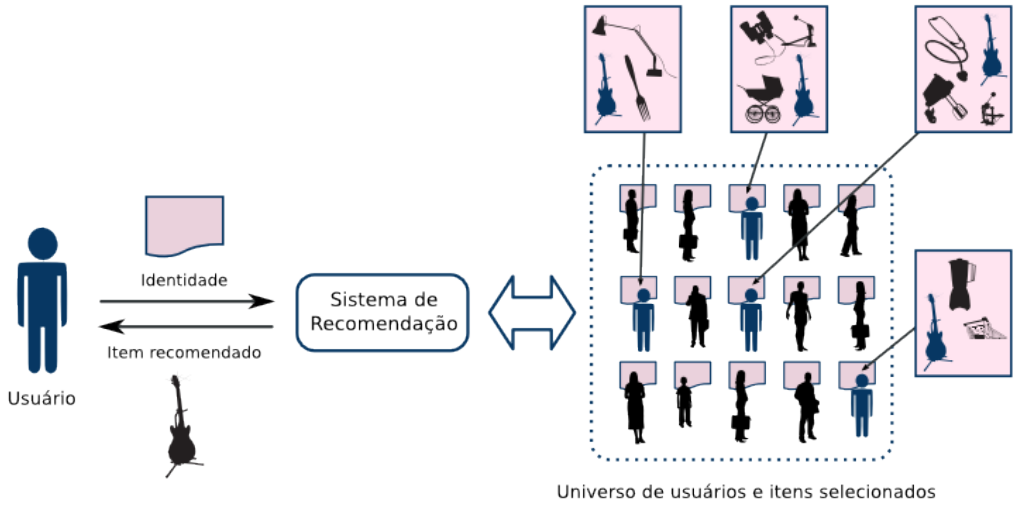

Figura 3.4: Cenário da recomendação colaborativa

Nesta abordagem o problema da superespecialização é superado, visto que a recomendação não se baseia no histórico do próprio usuário. Consequentemente itens totalmente inesperados podem fazer parte da sugestão. Outro ponto positivo é a possibilidade de formação de comunidades de usuários pela identificação de interesses semelhantes entre os mesmos [**?**].

3.5.4 Recomendação baseada em conhecimento

Esta estratégia tem como princípio a produção de recomendações a partir de um conhecimento previamente adquirido sobre o domínio da aplicação, em vez de avaliações prévias produzidas por usuários. A grande vantagem desta abordagem é que ela não depende de preferências individuais dos usuários, já que não usa a base de dados de avaliação usuário-item.

No entanto, a descoberta de conhecimento é o principal gargalo desta categoria de soluções, e por isso é mais utilizada nos casos em que já existe uma base de conhecimento disponível, por

exemplo, na forma de uma ontologia [?]. Quando não é este o caso, técnicas de aprendizado de máquina e mineração de dados podem ser utilizadas para extrair correlações e padrões frequentes no comportamento dos usuários, por meio da análise de suas escolhas ao longo do tempo.

Regras de associação são outro tipo de conhecimento formalizado, representado por regras de inferência que indicam a presença simultânea de conjuntos de itens numa determinada porcentagem dos casos conhecidos. As técnicas mais utilizadas para descoberta de tais regras são variações do algoritmo *Apriori*, apresentado na seção 3.6.6 [?]. Dado um conjunto de associações, a recomendação para determinado usuário é produzida de acordo com as regras satisfeitas pelo conjunto de itens que ele já tenha selecionado. Por exemplo, a regra $A, B, C \Rightarrow D$ seria satisfeita por usuários que possuem os itens A, B e C, resultando na indicação do item D: *"Clientes que compraram os itens A, B e C também compraram o item D"*.

Segundo [?], recomendação baseada em conhecimento é frequentemente utilizada para sugestões implícitas, por exemplo, na definição do posicionamento de produtos numa prateleira ou a realização de propagandas dirigidas. Um caso popular deste estratégia é encontrado na loja virtual da empresa *Amazon*[8] (figura 3.5).

Figura 3.5: Recomendação por associação na *Amazon*

3.5.5 Baseada em dados demográficos

A estratégia demográfica fundamenta-se na composição de perfis de usuários e identificação de nichos demográficos para produção de recomendações. Os dados pessoais geralmente são coletados de forma explícita, através de um cadastro do usuário, e podem englobar informações como idade, sexo, profissão e áreas de interesse. Dados demográficos, no entanto, são tipicamente utilizados em combinação com outras fontes de dados e técnicas diversas, como parte de uma estratégia de recomendação híbrida.

3.5.6 Estratégias híbridas

Sistemas de recomendação híbridos combinam duas ou mais estratégias, buscando obter melhor performance do que a que as estratégias oferecem individualmente. A tabela 3.1 apresenta as principais técnicas de hibridização segundo [?].

3.6 Técnicas

O desenvolvimento de sistemas de recomendação tem suas raízes em áreas distintas e o problema computacional a ser tratado está fortemente relacionado com outros problemas

[8] http://www.amazon.com/

Método	Descrição
Ponderação	Pontuações de relevância oriundas de diversas técnicas de recomendação são combinadas para compor uma única recomendação.
Revezamento	O sistema reveza entre técnicas de recomendação diversas, de acordo com a situação do momento.
Combinação	Recomendações oriundas de diversos recomendadores diferentes são apresentados de uma só vez.
Combinação de atributo	Um algoritmo de recomendação único coleta atributos de diferentes bases de dados para recomendação.
Cascata	Um recomendador refina a recomendação produzida por outro.
Acréscimo de atributo	O resultado de uma técnica é usado como atributo de entrada para outra.
Meta-nível	O modelo que um recomendador "aprendeu" é usado como entrada para o outro.

Tabela 3.1: Métodos de hibridização

clássicos, como classificação e recuperação de informação em documentos de texto.

A fim de obter a informação desejada, o usuário de uma ferramenta de busca deve traduzir suas necessidades de informação para uma consulta (*query*), que geralmente é representada por um conjunto de palavras-chave. O desafio do buscador é recuperar os documentos da coleção que são relevantes para a consulta, baseando-se nos termos que a constituem. Ademais, visto que a busca pode retornar um número excessivo de documentos, é desejável que este resultado seja apresentado ao usuário em ordem decrescente de relevância, aumentando assim as chances de a informação desejada ser encontrada com rapidez. Para tanto, cada documento da coleção deve ter uma pontuação (peso) que indique seu grau de importância para a referida *query*. Traçando um paralelo com o problema de recomendação, a identidade e/ou o comportamento do usuário representaria a consulta ao sistema de busca, que provocaria o retorno dos itens de maior peso, ou seja, com maior potencial de aceitação pelo usuário.

Na busca por informação, assume-se que as necessidades do usuário são particulares e passageiras, e por isso a reincidência de *queries* não é muito frequente [**?**]. Porém, em situações onde se observa que as mesmas consultas são aplicadas com uma certa frequência, é interessante que o sistema suporte consultas permanentes. Desta forma a computação necessária pode ser realizada previamente e apresentada sempre que a consulta for requisitada. Se a classe de documentos que satisfazem a uma dessas *queries* permanentes é tida como uma categoria, o processo de realização das consultas prévias pode ser caracterizado como uma classificação. O problema da classificação diz respeito à determinação de relacionamentos entre um dado objeto e um conjunto de classes pré-definidas.

A recomendação pode ser vista como uma classificação, na qual os itens são categorizados entre relevantes e irrelevantes – os relevantes seriam recomendados. Porém, a definição de consultas ou regras fixas para uma busca não é uma estratégia eficiente neste caso, porque a consulta estaria diretamente relacionada com a identidade do usuário e portanto deveria ser escrita especialmente para ele.

Todavia, a disciplina de inteligência artificial aborda a questão da classificação através de estratégias que não se baseiam em busca. Algoritmos de aprendizado de máquina são utilizados para a construção de modelos de classificação "inteligentes", que "aprendem" através da análise de exemplos. Os métodos supervisionados fundamentam-se na construção de um classificador que aprende na medida em que lhe são apontados exemplos de objetos classificados. São caracterizados como supervisionados porque as classes atribuídas aos objetos de treinamento são determinadas por um ser humano, que atua como um supervisor orientando o processo de aprendizado [**?**]. Por outro lado, algoritmos não supervisionados procuram identificar padrões de organização nos dados sem que haja uma classificação prévia dos exemplos.

A seguir são apresentadas algumas técnicas para tratamento destes problemas que também são utilizadas na construção de sistemas de recomendação, dando suporte às estratégias apresentadas na seção 3.5.

3.6.1 K-NN

K-nearest neighbors (k-NN), em português *k vizinhos mais próximos*, é um algoritmo de aprendizado supervisionado para classificação. Este método baseia-se no conceito de vizinhança, que representa um conjunto de objetos que estão próximos no espaço de busca.

O *K-NN* não exige nenhum treinamento prévio com os dados de exemplo, que podem ser diretamente armazenados como vetores de atributos acompanhados por suas devidas classes. A classificação de um novo objeto parte do cálculo da vizinhança do mesmo, que é composta por k objetos.

A determinação da vizinhança está diretamente relacionada com o conceito de proximidade entre objetos, que pode ser expressa em termos de similaridade ou de distância entre os mesmos (quanto maior a distância, menor a similaridade). Existem diversas medidas para mensurar estes conceitos, deve-se adotar a métrica que melhor se adeque ao domínio da aplicação e conjunto de dados. A tabela 3.2 apresenta algumas dessas medidas, onde os objetos X e Y são representados por seus vetores $\vec{X} = (x_1, ..., x_n)$ e $\vec{Y} = (y_1, ..., y_n)$. A similaridade de cosseno mede a similaridade de dois vetores através do cosseno do ângulo entre os mesmos. O coeficiente de *Pearson* é equivalente ao cosseno do ângulo entre os vetores centralizados na média. E o coeficiente de *Tanimoto* é uma extensão da similaridade de cossenos que resulta no índice de *Jaccard* para atributos binários.

Distância euclidiana	$D(X,Y) = \sqrt{(x_1\text{-}y_1)^2 + (x_2\text{-}y_2)^2 + ... + (x_n\text{-}y_n)^2}$
Similaridade de cosseno	$sim(X,Y) = \frac{\vec{X}\cdot\vec{Y}}{\lvert\vec{X}\rvert\lvert\vec{Y}\rvert} = \frac{\sum_{1\leq i\leq n} x_i y_i}{\sqrt{\sum_{1\leq i\leq n} x_i^2}\sqrt{\sum_{1\leq i\leq n} y_i^2}}$
Coeficiente de *Pearson*	$P(X,Y) = \frac{\sum_{1\leq i\leq n}(x_i\text{-}\bar{x})(y_i\text{-}\bar{y})}{\sqrt{\sum_{1\leq i\leq n}(x_i\text{-}\bar{x})^2}\sqrt{\sum_{1\leq i\leq n}(y_i\text{-}\bar{y})^2}}$
Coeficiente de *Tanimoto*	$T(X,Y) = \frac{\vec{X}\cdot\vec{Y}}{\lvert\vec{X}\rvert^2 + \lvert\vec{Y}\rvert^2\text{-}\vec{X}\cdot\vec{Y}}$

Tabela 3.2: K-NN: Medidas de distância e similaridade entre objetos

Após a definição de vizinhança, a classe mais frequente entre seus k vizinhos é atribuída ao novo objeto. Desta forma, a similaridade também pode ser entendida como grau de influência entre os objetos. Os objetos mais semelhantes a um novo objeto terão maior influência no cálculo de sua classificação.

3.6.2 Agrupamento

Agrupamento (*clustering*) é uma técnica de aprendizado de máquina não supervisionado. O algoritmo particiona a base de dados de forma a criar automaticamente grupos que reúnam usuários com comportamentos semelhantes. Para fins de recomendação o agrupamento pode ser utilizado na composição de vizinhança de um usuário na aplicação de técnicas colaborativas.

A escolha do algoritmos a ser utilizado deve depender do tipo de dados disponível e nos objetivos específicos da aplicação. É aceitável experimentar diversos algoritmos no mesmo conjunto de dados porque geralmente não se quer provar ou refutar uma hipótese pré-concebida, mas sim ver "o que os dados estão tentando nos dizer" [**?**].

Entre as técnicas mais populares de agrupamento está o *k-means*, que consiste basicamente nos seguintes passos:

1. Seleção de k elementos considerados sementes;
2. Associação de cada elemento da base de dados com a semente mais próxima a ele;
3. Cálculo de novos pontos no espaço centrais (centroides) para cada grupo, a partir dos elementos que o compõem.

O passo 3 é repetido até que não seja mais necessário calcular novos centroides.

K-Medoids é uma variação do *k-means*, onde as partições giram em torno de medoids em vez de pontos no espaço. Medoides são os elementos que acumulam menor dissimilaridade com o restante dos objetos do grupo. Esta é uma solução mais genérica do que o primeiro método porque pode ser utilizada mesmo quando um centroide não pode ser definido (a depender do tipo das variáveis), já que se apoiam em coeficientes de dissimilaridades. É utilizado também quando há interesse em preservar os objetos representativos de cada grupo, por exemplo, para fins de redução de dados ou caracterização.

3.6.3 Classificador bayesiano

Bayes ingênuo é uma solução para classificação que figura entre os algoritmos de aprendizado de máquina supervisionados. O classificador apoia-se num modelo probabilístico que aplica o teorema de Bayes com fortes suposições de independência de atributos – por esta razão o método é considerado ingênuo. Em outras palavras, a presença ou ausência de um atributo em um objeto de uma classe não estaria relacionada com a incidência de nenhum outro atributo.

A decisão acerca da classe a qual um objeto pertence é tomada de acordo com o modelo de probabilidade máxima posterior *(MAP)*, indicada na equação 3.1. Dado que C é o conjunto de classes e x objeto a ser classificado, a classe atribuída a este será a que apresentar maior probabilidade condicionada a x. $\hat{P}$ é utilizado em vez de P porque geralmente não se sabe o valor exato das probabilidades, que são estimadas a partir dos dados de treinamento.

$$c_{MAP} = \arg\max_{c \in C} \hat{P}(c|x) \tag{3.1}$$

A equação 3.2 aplica o Teorema de Bayes para probabilidades condicionadas. Na prática, apenas o numerador da fração interessa, visto que o denominador é constante para todas as classes, portanto não afeta o arg max (equação 3.3).

$$c_{MAP} = \arg\max_{c \in C} \frac{\hat{P}(x|c)\hat{P}(c)}{\cancel{\hat{P}(x)}} \tag{3.2}$$

$$= \arg\max_{c \in C} \hat{P}(x|c)\hat{P}(c) \tag{3.3}$$

É neste ponto que a independência de atributos é importante. Considera-se que um documento x pode ser caracterizado por uma série de atributos x_i – no caso de documentos de texto, os atributos são os próprios termos. Assumindo que a ocorrência de atributos acontece independentemente, tem-se que:

$$\hat{P}(x|c) = \hat{P}(x_1, x_2, ..., x_n|c) = \hat{P}(x_1|c)\hat{P}(x_2|c) \ ... \ \hat{P}(x_n|c) \tag{3.4}$$

Portanto, a função de decisão pode ser reescrita através da equação 3.5. Cada parâmetro condicional $\hat{P}(x_i|c)$ é um peso que representa a qualidade do atributo x_i como indicador da classe c, enquanto que $\hat{P}(c)$ é a frequência relativa da classe c.

$$c_{MAP} = \arg\max_{c \in C} \ \hat{P}(c) \prod_{1 \leq i \leq n} \hat{P}(x_i|c) \tag{3.5}$$

Os parâmetros são obtidos através da estimativa de maior probabilidade *(MLE)*, que corresponde ao valor mais provável de cada parâmetro de acordo com os dados de treinamento. A equação 3.6 traz a estimativa de $\hat{P}(c)$, onde N_c é o número de objetos da classe c e N é o número total de documentos.

$$\hat{P}(c) = \frac{N_c}{N} \tag{3.6}$$

As probabilidades condicionais são estimadas como a frequência relativa do atributo x em objetos que pertencem à classe c. Na equação 3.7, T_{cx} é o número de ocorrências de x em objetos de exemplo da classe c e V é o conjunto de atributos que os objetos podem apresentar.

$$\hat{P}(x|c) = \frac{T_{cx}}{\sum_{x' \in V} T_{cx'}} \tag{3.7}$$

No entanto, a estimativa *MLE* é zero para combinações atributo-classe que não ocorrem nos dados de treinamento. Considerando que as probabilidades condicionais de todos os atributos serão multiplicadas (equação 3.5), a simples ocorrência de uma probabilidade zerada resulta na desconsideração da classe na referida classificação. E de fato, dados de treinamento nunca são abrangentes o suficiente para representar a frequência de eventos raros de forma adequada [**?**]. Para eliminar *zeros*, adiciona-se 1 a cada termo da equação 3.7:

$$\hat{P}(x|c) = \frac{T_{cx} + 1}{\sum_{x' \in V} T_{cx'} + 1} \tag{3.8}$$

O classificador bayesiano também é sensível a ruídos, logo, sua performance é igualmente beneficiada pelo processo de seleção de atributos descrito na seção 3.4.

Apesar de a independência de atributos não ser verificada para a maioria dos domínios de aplicação, na prática o Bayes ingênuo apresenta resultados satisfatórios. [**?**] atribui a surpreendente boa performance deste método ao fato de que a mera existência de dependências entre atributos não prejudicaria a classificação, mas sim o modo como as dependências estão distribuídas ao longo das classes. Segundo o autor, desde que as dependências estejam distribuídas igualmente nas classes, não há problema em haver dependência forte entre dois atributos.

Variantes do modelo Bayes ingênuo

As duas principais variantes de implementação do classificador bayesiano, denominadas de modelo *multinomial* e de *Bernoulli*, diferem fundamentalmente na maneira como os objetos são representados.

O primeiro modelo utiliza uma representação que considera informações espaciais sobre o objeto. Na classificação de documentos de texto, por exemplo, o modelo gera um atributo para cada posição do documento, que corresponde a um termo do vocabulário. Já o modelo de *Bernoulli* produz um indicador de presença ou ausência para cada possível atributo (no caso de texto, cada termo do vocabulário).

A escolha da representação de documentos adequada é uma decisão crítica no projeto de um classificador, visto que o próprio significado de um atributo depende da representação. No *multinomial*, um atributo pode assumir como valor qualquer termo do vocabulário, o que resulta numa representação do documento correspondente à sequência de termos do mesmo. Já para o modelo de *Bernoulli*, um atributo pode assumir apenas os valores 0 e 1, e a representação do documento é uma sequência de 0s e 1s do tamanho do vocabulário.

3.6.4 Medida *tf-idf*

Acrônimo para *term frequency - inverse document frequency*, *tf-idf* é uma medida de peso clássica utilizada em ferramentas de busca em texto para ordenação do resultado da consulta pela relevância dos documentos.

O universo da busca é uma coleção de documentos de texto. Um documento por sua vez é uma coleção de palavras, geralmente referenciadas como *termos do documento*. O conjunto de todas as palavras presentes nos documentos da coleção é denominado *dicionário* ou *vocabulário*.

Desta forma, um documento d composto por n termos do vocabulário V pode ser representado como $d = \{t_1, t_2, ..., t_n | 1 \leq i \leq n, t_i \in V\}$.

Contudo, alguns termos do vocabulário, designados como *stop words*, são normalmente desconsiderados no cálculo de relevância dos documentos por serem muito frequentes na coleção e, em decorrência disto, pouco informativos acerca do teor dos textos. Artigos e pronomes, por exemplo, geralmente figuram nesta categoria.

Outra consideração acerca da representação dos documentos como conjuntos de termos é a realização de normalizações morfológicas. Diferentes palavras que dizem respeito ao mesmo conceito podem ser utilizadas ao longo de uma coleção, por exemplo, os termos *casa*, *casinha* e *casas*. Em certos contextos, deseja-se que a busca por uma determinada variante retorne ocorrências de todas as outras possibilidades. Neste caso, os termos devem ser tratados em sua forma normalizada, eliminando variações como plurais e flexões verbais. Os processos mais comuns de normalização são: *stemming*, que reduz a palavra ao seu radical; e *lematização*, que reduz a palavra à sua forma canônica (por exemplo, verbos no infinitivo, substantivos no singular masculino etc). A figura 3.6 apresenta um documento de texto numa coleção hipotética[9] e a sua representação após eliminação de *stop words* e procedimento de *stemming*.

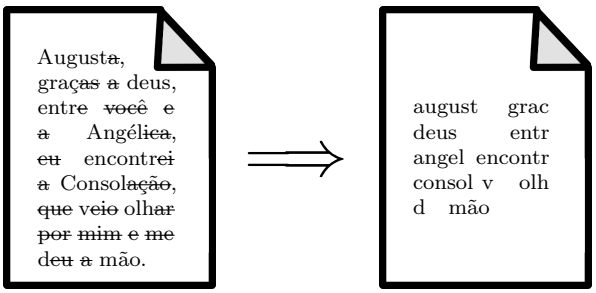

Figura 3.6: Eliminação de *stop words* e normalização do documento por *stemming*

A simples presença de um termo da *query* em um documento da coleção já é um indicativo de que o mesmo tem alguma relação com a consulta. No entanto, a quantidade de vezes que o termo ocorre é ainda mais informativo sobre sua relação com o conteúdo do documento. Intuitivamente, os documentos que referenciam os termos de uma *query* com mais frequência estão mais fortemente relacionados com a mesma, e por isso deveriam receber uma maior pontuação de relevância. O peso $tf_{t,d}$ (*term frequency*) quantifica esta noção intuitiva, relacionando documentos da coleção e termos do dicionário de acordo com a frequência destes nos documentos. Em sua abordagem mais simples, $tf_{t,d}$ é igual ao número de ocorrências do termo t no documento d.

A figura 3.7 ilustra uma coleção de documentos, cujos valores de $tf_{t,d}$ para alguns termos do dicionário são apresentados na tabela 3.3. Por exemplo, a palavra *morena* ocorre duas vezes no documento (1), por isso $tf_{moren,1} = 2$. O cálculo do tfs já considera os radicais dos termos, resultantes de um processo de *stemming*. Em razão disto, $tf_{olh,2} = 3$, pois tanto a palavra *olho* quanto as diferentes flexões do verbo *olhar* contribuem para a contagem de frequência do termo *olh*. Na tabela 3.3, os radicais dos vocábulos são seguidos por algumas variações, a título de ilustração.

O conjunto de pesos determinado pelos tfs dos termos de um documento pode ser entendido como um resumo quantitativo do mesmo. Esta visão do documento é comumente referenciada na literatura como "saco de palavras", onde a disposição das palavras é ignorada e apenas a quantidade de ocorrências para cada termo é considerada.

Uma medida de relevância baseada simplesmente na incidência dos termos da *query* nos documentos (RI) poderia ser calculada através da equação 3.9.

[9] Os textos utilizados nos exemplos desta seção são excertos de letras de música de diversos compositores brasileiros.

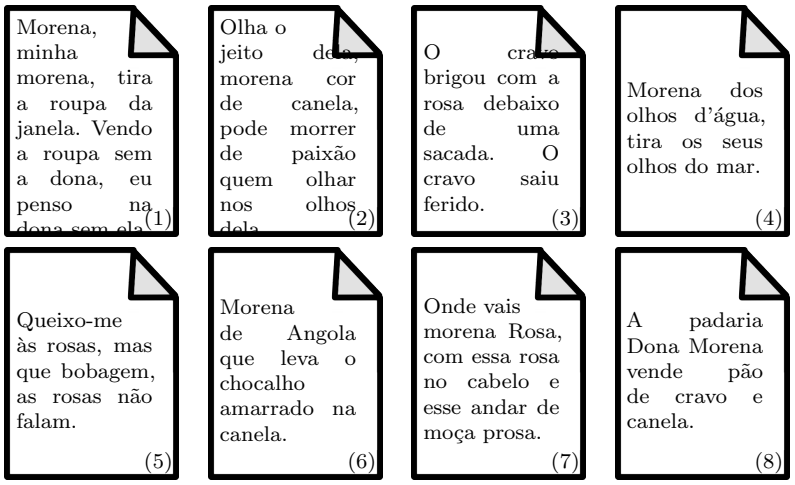

Figura 3.7: Coleção de documentos

	$tf_{t,d}$							
Termo \ ***Doc***	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**
moren {*a,o*}	2	1	0	1	0	1	1	1
roup {*a,ão*}	2	0	0	0	0	0	0	0
don {*a,o*}	2	0	0	0	0	0	0	1
crav {*o,eiro*}	0	0	2	0	0	0	0	1
canel {*a,eira*}	0	1	0	0	0	1	0	1
olh {*o,ar*}	0	3	0	2	0	0	0	0
ros {*a,eira*}	0	0	1	0	2	0	2	0
bob {*o,agem*}	0	0	0	0	1	0	0	0

Tabela 3.3: Frequência dos termos nos documentos da coleção

$$RI_{d,q} = \sum_{t \in q} tf_{t,d} \tag{3.9}$$

No entanto, alguns termos têm pouco poder de discriminação na determinação de relevância de um documento, por estarem presentes em quase todos os documentos. Ao passo que existem outros muito raros que quando presentes são forte indicativo de relevância. No contexto da coleção da figura 3.7, por exemplo, a *query* {*morena*, *bobagem*} é composta por um termo muito frequente e outro muito raro. Coincidentemente, os documentos (3) e (5), contém apenas um dos dois elementos da consulta, porém ambos apresentam $tf_{t,d} = 1$ para os respectivos termos. Todavia, enquanto esta frequência se repete ao longo da coleção múltiplas vezes para o termo *morena*, ela é única para o termo *bobagem*, o que de fato diferencia o documento (5) dos demais.

O idf_t (*inverse document frequency*) foi então introduzido para atenuar o efeito de termos muito comuns no cálculo de relevância, diminuindo o peso relacionado a um termo por um fator que cresce com sua frequência em documentos na coleção [**?**]. A equação 3.10 apresenta a forma clássica do idf, na qual N representa o número de documentos da coleção e df_t (*document frequency*) é o número de documentos que contém o termo t. É comum que o universo da busca seja uma coleção de documentos de altíssima dimensão, resultando em valores de df_t muito discrepantes. O uso do log diminui a escala de valores, permitindo que frequências muito grandes e muito pequenas sejam comparadas sem problemas.

$$idf_t = \log \frac{N}{df_t} \tag{3.10}$$

Valores de idf_t para a coleção da figura 3.7 são apresentados na tabela 3.4. Novamente os radicais dos termos são considerados: $idf_{morena} = idf_{moren} = \log \frac{8}{6} = 0.12$, enquanto $idf_{bobagem} = idf_{bob} = \log \frac{8}{1} = 0.9$.

Termo	moren	roup	don	crav	canel	olh	ros	bob
idf_t	0.12	0.9	0.6	0.6	0.42	0.6	0.42	0.9

Tabela 3.4: Valores de idf_t para termos do dicionário

A medida $tf\text{-}idf_{t,d}$ combina as definições de tf e idf (equação 3.11), produzindo um peso composto com as seguintes propriedades:

1. É alto quando t ocorre muitas vezes em d e em poucos documentos da coleção (ambos tf e idf são altos);
2. Diminui quando ocorre menos vezes em d (tf mais baixo) ou em muitos documentos da coleção (idf mais baixo);
3. É muito baixa quando o termo ocorre em quase todos os documentos (mesmo para valores altos de tf, para termos muito comuns o peso idf domina a fórmula, em decorrência do uso do log).

$$tf\text{-}idf_{t,d} = tf_{t,d} \cdot idf_t \tag{3.11}$$

A medida de relevância apresentada na equação 3.9 pode ser refinada para somar os pesos $tf\text{-}idf$ do documento d com relação aos termos da *query* q, resultando na media $R_{d,q}$ apresentada na equação 3.12 [**?**].

$$R_{d,q} = \sum_{t \in q} tf\text{-}idf_{t,d} \tag{3.12}$$

A tabela 3.5 apresenta a ordenação dos documentos da coleção como resultado do cálculo de relevância por $tf\text{-}idf$ para as consultas $q_1 = \{morena\}$, $q_2 = \{morena, bobagem\}$ e $q_3 = \{morena, dona, rosa\}$. Os valores $tf\text{-}idf_{t,d}$ foram obtidos a partir da equação 3.11, com os pesos das tabelas 3.3 e 3.4. Por exemplo, $tf\text{-}idf_{morena,1} = tf_{morena,1} \cdot idf_{morena} = 2 \cdot 0.12 = 0.24$.

doc	q_1	$R_{d,q}$
	morena	
(1)	0.24	0.24
(2)	0.12	0.12
(4)	0.12	0.12
(6)	0.12	0.12
(7)	0.12	0.12
(8)	0.12	0.12
(3)	0	0
(5)	0	0

doc	q_2		$R_{d,q}$
	morena	*bobagem*	
(5)	0	0.9	0.9
(1)	0.24	0	0.24
(2)	0.12	0	0.12
(4)	0.12	0	0.12
(6)	0.12	0	0.12
(7)	0.12	0	0.12
(8)	0.12	0	0.12
(3)	0	0	0

doc	q_3			$R_{d,q}$
	morena	*dona*	*rosa*	
(1)	0.24	1.2	0	1.44
(8)	0.12	1.2	0	1.32
(7)	0.12	0	0.84	0.96
(5)	0	0	0.84	0.84
(3)	0	0	0.42	0.42
(2)	0.12	0	0	0.12
(4)	0.12	0	0	0.12
(6)	0.12	0	0	0.12

Tabela 3.5: Ordenação dos documentos como resultado das consultas q_1, q_2 e q_3

Existem diversas variantes para o cálculo dos pesos $tf_{t,d}$ e idf_t, propostas com o intuito de aperfeiçoar o processo de busca. Por exemplo, geralmente a presença de uma palavra 20 vezes num documento não tem de fato 20 vezes mais representatividade do que uma ocorrência única. Documentos distintos podem referenciar o mesmo conceito de forma concisa ou prolixa, e simplesmente este fato não deve ser motivo para pesos muito discrepantes com relação a uma *query*, visto que o teor do texto é o mesmo. A variante denominada tf *sub-linear* incorpora o logaritmo ao cálculo do tf para atenuar o crescimento do peso para valores crescentes de frequência (equação 3.13).

$$tf\text{-}sub_{t,d} = \begin{cases} 1 + \log tf_{t,d} & \text{, se } tf_{t,d} > 0 \\ 0 & \text{, caso contrário} \end{cases} \tag{3.13}$$

Outras abordagens alternativas utilizam normalizações por diversas medidas: comprimento do documento, comprimento médio dos documentos da coleção, *tf* máximo ou médio entre os *tf*s de todos os termos do documento, entre outros.

Modelo de espaço vetorial

Uma coleção de documentos pode ser representada por um conjunto de vetores, sendo cada documento descrito como um vetor de termos do dicionário e os respectivos pesos *tf-idf* do documento. Tem-se como resultado uma visão da coleção como uma matriz de dimensões $M \times N$, na qual as linhas representam os M termos do dicionário e as colunas os N documentos da coleção. Esta representação, conhecida como *modelo de espaço vetorial*, é amplamente utilizada em soluções para recuperação da informação.

Assumindo que o vocabulário se restringe apenas aos termos para os quais os valores de *tf* e *idf* foram calculados (tabelas 3.3 e 3.4), a coleção de documentos da figura 3.7 pode ser representada no modelo de espaço vetorial pela matriz da tabela 3.6.

$tf\text{-}idf_{t,d}$								
Termo \ ***Doc***	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**
moren	0.24	0.12	0	0.12	0	0.12	0.12	0.12
roup	1.8	0	0	0	0	0	0	0
don	1.2	0	0	0	0	0	0	0.6
crav	0	0	1.2	0	0	0	0	0.6
canel	0	0.42	0	0	0	0.42	0	0.42
olh	0	1.8	0	1.2	0	0	0	0
ros	0	0	0.42	0	0.84	0	0.84	0
bob	0	0	0	0	0.9	0	0	0

Tabela 3.6: Representação da coleção no modelo de espaço vetorial

Similaridade de cosseno

Medir a similaridade entre dois documentos pode ser útil, por exemplo, para disponibilizar o recurso "mais do mesmo", onde o usuário pede indicações de itens semelhantes a um que ele já conhece. Porém, se a diferença entre os vetores de pesos de dois documentos for usada como medida para avaliação de similaridade entre os mesmos, pode acontecer de documentos com conteúdo similar serem considerados diferentes simplesmente porque um é muito maior que o outro. Para compensar o efeito do comprimento dos documentos utiliza-se como medida de similaridade o cosseno do ângulo entre os vetores que os representam (θ), apresentada na equação 3.14. O numerador representa o produto escalar dos dois vetores e o denominador a distância euclidiana entre os mesmos.

$$sim(d_1, d_2) = \cos(\theta) = \frac{V(\vec{d}_1) \cdot V(\vec{d}_2)}{|V(\vec{d}_1)||V(\vec{d}_2)|} \tag{3.14}$$

Dado um documento d, para encontrar os documentos de uma coleção que mais se assemelham a este, basta encontrar aqueles com maior similaridade de cosseno com d. Para tanto, pode-se calcular os valores $sim(d, d_i)$ entre d e os demais d_i documentos da coleção e os maiores valores indicarão os documentos mais semelhantes.

Considerando o fato de que *queries*, assim como documentos, são um conjunto de palavras, elas também podem ser representadas como vetores no modelo de espaço vetorial. A tabela

3.7 apresenta as consultas q_1, q_2 e q_3 neste espaço. Logo, a similaridade de cosseno também pode ser utilizada em buscas, considerando que os documentos mais similares a determinada *query* são os mais relevantes para a mesma (equação 3.15).

$$R_{d,q} = sim(d, q) \tag{3.15}$$

$tf\text{-}idf_{t,d}$			
Termo \ **Query**	q_1	q_2	q_3
moren	0.12	0.12	0.12
roup	0	0	0
don	0	0	0.6
crav	0	0	0
canel	0	0	0
olh	0	0	0
ros	0	0	0.42
bob	0	0.9	0

Tabela 3.7: Representação das queries no modelo de espaço vetorial

3.6.5 Okapi BM25

Okapi BM25 é o modelo probabilístico considerado estado da arte em recuperação da informação [**?**]. É amplamente utilizado no desenvolvimento de ferramentas de busca para os mais diversos domínios de aplicação. Tornou-se bastante popular em virtude de seu destaque nas avaliações do TREC[10], sendo apontado como o melhor entre os esquemas de peso probabilísticos conhecidos [**?**]. A título de ilustração, Xapian[11] e Lucene[12], bibliotecas livres para construção de motores de busca, são projetos de grande destaque na comunidade que utilizam o *BM25* como medida de pesos. O nome *Okapi* advém do primeiro sistema no qual foi implementado, denominado *City Okapi*, enquanto *BM* se refere à família de esquemas *Best Match*.

Embora seja comumente apresentado num contexto de busca em texto, o esquema não é específico para este domínio e pode ser usado na estimativa de relevância para qualquer tipo de recuperação de informação. A realização de consultas depende da descrição de itens e necessidades dos usuários, no entanto o modelo em princípio é compatível com inúmeras possibilidades de unidades descritivas [**?**]. Todavia, formalmente o modelo se refere a descrições de documentos como D e de consultas como Q, ambas podendo ser decompostas em unidades menores. Cada componente é um atributo A_i, que pode assumir valores do domínio $\{presente, ausente\}$ ou valores inteiros não negativos, representando o número de ocorrências do termo no documento ou na *query*.

A busca no modelo probabilístico fundamenta-se no *Princípio de Ordenação por Probabilidade*, segundo o qual a maior eficácia de uma consulta num conjunto de dados é obtida quando os documentos recuperados são ordenados de maneira decrescente pela probabilidade de relevância em tal base de dados. [**?**]. No entanto, o ponto chave do Princípio é que a probabilidade de relevância não é o fim em si mesma, mas um meio de ordenar os documentos para apresentação ao usuário. Portanto, qualquer transformação desta probabilidade pode ser usada, desde que preserve a ordenação pela relevância [**?**].

[10] O *Text Retrieval Conference (TREC)* é uma conferência anual realizada pelo *U.S. National Institute of Standards and Technology (NIST)* que promove uma ampla competição em recuperação da informação de grandes coleções de texto com o intuito de incentivar pesquisas na área.

[11] http://xapian.org/

[12] http://lucene.apache.org/

Modelo formal

Dado um documento descrito por D e uma *query* Q, o modelo considera a ocorrência de dois eventos: $L = \{D \text{ é relevante para } Q\}$ e $\overline{L} = \{D \text{ não é relevante para } Q\}$. Para que a ordenação por relevância seja possível, calcula-se para cada documento a probabilidade $P(L|D)$. A aplicação do teorema de Bayes permite que $P(L|D)$ seja expressa em função de $P(D|L)$ (equação 3.16).

$$P(L|D) = \frac{P(D|L)P(L)}{P(D)} \tag{3.16}$$

Para evitar a expansão de $P(D)$, a chance de $(L|D)$ é utilizada em vez da probabilidade. Na verdade, o logaritmo da chance é aplicado (equação 3.17), considerando que esta é uma transformação que satisfaz o Princípio de Ordenação [**?**]. Ademais, dado que o último termo da fórmula é igual para todos os documentos, ele pode ser desconsiderado sem que isso altere a ordenação dos documentos. Desta forma, a equação 3.18 descreve uma pontuação por relevância referenciada como primária ($\mathit{R\text{-}PRIM}_D$).

$$\begin{aligned} \log \frac{P(L|D)}{P(\overline{L}|D)} &= \log \frac{P(D|L)P(L)}{P(D|\overline{L})P(\overline{L})} \\ &= \log \frac{P(D|L)}{P(D|\overline{L})} + \log \frac{\cancel{P(L)}}{\cancel{P(\overline{L})}} \qquad (3.17) \\ \mathit{R\text{-}PRIM}_D &= \log \frac{P(D|L)}{P(D|\overline{L})} \qquad (3.18) \end{aligned}$$

Assim como o modelo de classificação *Bayes* ingênuo, o *BM25* assume que os atributos dos documentos são estatisticamente independentes de todos os outros. [**?**] justifica a suposição de independência de atributos pelos seguintes argumentos:

1. Facilita o desenvolvimento formal e expressão do modelo;
2. Torna a instanciação do modelo tratável computacionalmente;
3. Ainda assim permite estratégias de indexação e busca com melhor performance do que estratégias rudimentares, como o simples casamento de padrões aplicados a termos da *query* no documento.

De acordo com a suposição de independência, a probabilidade de um documento pode ser trivialmente derivada a partir das probabilidade de seus atributos. Logo, $\mathit{R\text{-}PRIM}_D$ poderia ser estimado como um somatório de probabilidades, cada uma relacionada a cada atributo da descrição D (equação 3.19).

$$\begin{aligned} \mathit{R\text{-}PRIM}_D &= \log \prod_i \frac{P(A_i = a_i|L)}{P(A_1 = a_1|\overline{L})} \\ &= \sum_i \log \frac{P(A_i = a_i|L)}{P(A_1 = a_1|\overline{L})} \qquad (3.19) \end{aligned}$$

No entanto, a fórmula 3.19 pressupõe a consideração de um componente para cada valor do atributo, por exemplo, para presença de um termo assim como para sua ausência. Uma alternativa mais natural seria considerar apenas valores para a presença, contabilizando a ausência como um *zero* natural. Desta forma, é subtraído da pontuação de cada documento o componente relativo a cada valor de atributo zerado (fórmula 3.20).

$$\begin{aligned} \textit{R-BASIC}_D &= \textit{R-PRIM}_D - \sum_i \log \frac{P(A_i = 0|L)}{P(A_1 = 0|\overline{L})} \\ &= \sum_i \Big(\log \frac{P(A_i = a_i|L)}{P(A_1 = a_1|\overline{L})} - \log \frac{P(A_i = 0|L)}{P(A_1 = 0|\overline{L})} \Big) \\ &= \sum_i \log \frac{P(A_i = a_i|L)P(A_1 = 0|\overline{L})}{P(A_1 = a_1|\overline{L})P(A_i = 0|L)} \end{aligned} \tag{3.20}$$

Considerando W_i como um peso para cada termo t_i do documento (equação 3.21), $\textit{R-BASIC}_D$ pode ser então reescrito em função deste peso, como na equação 3.22.

$$W_i = \log \frac{P(A_i = a_i|L)P(A_1 = 0|\overline{L})}{P(A_1 = a_1|\overline{L})P(A_i = 0|L)} \tag{3.21}$$

$$\textit{R-BASIC}_D = \sum_i W_i \tag{3.22}$$

No caso em que os atributos A_i restringem-se a exprimir a presença ou ausência do termo t_i (atributos binários), pode-se dizer que $P(A_1 = 0|L) = 1 - P(A_i = 1|L)$, o mesmo vale para $\overline{L}$. Portanto, considerando que $p_i = P(t_i \text{ ocorre } |L)$ e $\overline{p}_i = P(t_i \text{ ocorre } |\overline{L})$, a fórmula 3.21 pode ser usada como um peso para presença de termos. A pontuação de relevância para um documento seria então a soma dos pesos w_i dos termos da *query* presentes no documento.

$$w_i = \log \frac{p_i(1 - \overline{p_i})}{\overline{p_i}(1 - p)} \tag{3.23}$$

A seguir será apresentada a interpretação do modelo formal a partir de informações disponíveis sobre a coleção de documentos, com o intuito de definir funções de peso eficazes para a ordenação por relevância.

Incidência dos termos e atestação de relevância

A incidência dos termos nos documentos da coleção é uma informação que pode ser facilmente coletada e pode ser utilizada como parâmetro no cálculo da probabilidade de relevância. O popular idf_t (equação 3.10) é uma medida plausível e, apesar de ter sido proposta baseada apenas na frequência de incidência dos termos, também pode ser derivada da equação 3.23 [**?**].

No entanto, apenas a incidência dos termos é uma base fraca para a estimativa de probabilidades de relevância. As estimativas podem ser refinadas através da consideração de dados acerca da real relevância ou irrelevância de documentos, obtidos por meio de mecanismos denominados *atestação de relevância* (*relevance feedback*). Ao receber o resultado de uma consulta, o usuário avalia a lista de itens retornados, indicando quais deles são de fato relevantes, para que uma nova consulta seja realizada.

A tabela de contingência da incidência dos termos é apresentada na tabela 3.8. N representa o número total de documentos da coleção, enquanto n representa o número de documentos nos quais o termo t da *query* ocorre. Analogamente, R é a quantidade de documentos relevantes para a consulta e r a quantidade de documentos relevantes nos quais o termo ocorre.

	Relevante	Irrelevante	Incidência na coleção
t ocorre	r	$n - r$	n
t não ocorre	$R - r$	$N - n - R + r$	$N - n$
total de documentos	R	$N - R$	N

Tabela 3.8: Tabela de contingência da incidência dos termos

Portanto, a probabilidade de um termo t ocorrer num documento, dado que este é relevante para a *query* é $p = \frac{r}{R}$. Analogamente, dado que o documento não é relevante, $\overline{p} = \frac{n-r}{N-R}$. Desta forma, a equação 3.23 pode ser redefinida como a fórmula 3.24, que exprime o logaritmo da razão de chances de um termo ocorrer em documentos relevantes e irrelevantes.

$$w = \log \frac{r(N - n - R + r)}{(R - r)(n - r)} \tag{3.24}$$

Por fim, o fator de correção 0.5 é acrescido a cada termo central da fórmula para evitar que o peso seja zerado quando algum destes termos for *zero*.

$$RW_t = \log \frac{(r + 0.5)(N\text{-}n\text{-}R + r + 0.5)}{(R\text{-}r + 0.5)(n\text{-}r + 0.5)} \tag{3.25}$$

Se não houver informações provenientes da atestação de relevância, o idf_t clássico pode ser utilizado (equação 3.10), ou ainda, uma variação de RW_t a partir do estabelecimento de que $R = r = 0$ (equação 3.26).

$$RW_t = \log \frac{N\text{-}n + 0.5}{n + 0.5} \tag{3.26}$$

Distribuição dos termos nos documentos

A incidência dos termos na coleção distingue os documentos com relação aos termos da *query* no que diz respeito apenas à ocorrência ou ausência dos mesmos. Usando apenas esta medida não é possível portanto diferenciar dois documentos em relação a um termo se o mesmo ocorre em ambos. No caso em que dados de frequência dos termos são providos nas descrições dos documentos, esta informação pode contribuir para a estimativa de relevância do de um documento.

Assume-se que cada termo é associado a um conceito, ao qual um determinado documento pode estar relacionado ou não. Logo, para cada conceito existe um conjunto de documentos que dizem respeito a ele e outro conjunto que não (complementar ao primeiro). A frequência de um termo em um documento caracteriza sua ocorrência quantitativamente, porém, uma frequência maior que *zero* não significa que o documento esteja necessariamente relacionado com conceito do termo. Diante da impossibilidade de se prever esta relação conceitual, considera-se a distribuição de frequências dos termos nos documentos como uma mistura de duas distribuições, uma para cada um dos conjuntos [**?**].

Essa distribuição pode ser entendida como originada num modelo de geração de texto: o autor se depara com as posições de palavras nos documentos e escolhe termos para ocupar tais posições. Se a probabilidade de escolha de cada termo for fixa e todos os documentos forem de igual comprimento, caracteriza-se uma distribuição de *Poisson* para frequências dos termos nos documentos. Assume-se probabilidades diferentes para o conjunto de documentos relacionados ao conceito do termo e para o conjunto dos que não são – esta é a razão para a mistura de duas distribuições [**?**].

A derivação deste componente do peso é mais extensa e por esta razão foi omitida neste texto. A fórmula resultante é complexa, no entanto [**?**] examina o comportamento da mesma e propõe uma aproximação que apresenta comportamento similar à original, expressa pela equação 3.27.

$$RD_{t,D} = \frac{tf_{t,D}(k_1 + 1)}{k_1 + tf_{t,D}} \tag{3.27}$$

A constante k_1 determina o quanto o peso do documento em relação ao termo deve ser afetado por um acréscimo no valor de $tf_{t,D}$. Se $k_1 = 0$, então $RD_{t,D} = 1$, e $tf_{t,D}$ não interfere no peso final. Para valores altos de k_1, o peso passa a ter um crescimento linear com relação

a $tf_{t,D}$. De acordo com experimentos do TREC, valores entre 1.2 e 2 são os mais indicados, visto que implicam numa interferência altamente não linear de $tf_{t,D}$, ou seja, após 3 ou 4 ocorrências o impacto de uma ocorrência adicional é mínimo [**?**].

No entanto, a modelagem através distribuições de *Poisson* assume que todos os documentos têm mesmo comprimento, o que não acontece na prática. Porém, uma interpretação ligeiramente estendida do modelo permite a consideração de documentos com comprimentos distintos.

Os comprimentos dos documentos da coleção podem variar por inúmeros motivos. Todavia, nesta nova interpretação assume-se que quando dois documentos acerca do mesmo conceito têm tamanhos distintos, a razão é simplesmente que um é mais verboso que o outro. Em outras palavras, considera-se que a recorrência de palavras deve-se sempre à repetição em vez de, por exemplo, melhor elaboração do tema. Partindo desta suposição, é apropriado estender o modelo normalizando o valor de $tf_{t,D}$ em função do comprimento do documento (equação 3.28).

$$RD_{t,D} = \frac{\frac{tf_{t,D}}{NL}(k_1+1)}{k_1 + \frac{tf_{t,D}}{NL}} = \frac{tf_{t,D}(k_1+1)}{k_1 * NL + tf_{t,D}} \tag{3.28}$$

Dado que o comprimento dos documentos pode ser medido de diversas formas (quantidade de palavras, caracteres e *bytes*, considerando ou não *stop words*), considera-se a medida uniformizada para normalização, obtida pela razão entre o comprimento dos documentos e o comprimento médio dos documentos ($\frac{l_d}{l_{avg}}$). Ademais, uma normalização simples resultaria na mesma pontuação para um documento de comprimento l no qual o termo ocorre tf vezes e para outro de comprimento $2l$ que contém $2tf$ ocorrências do termo. Este comportamento pode ser indesejável por exemplo quando se considera que a recorrência de palavras está geralmente associada ao aprofundamento do conceito, em vez de mera repetição.

A fórmula proposta (3.29) permite que a normalização ocorra em diferentes graus, de acordo com o ajuste do parâmetro constante b que assume valores no intervalo $[0, 1]$. Se a configuração for $b = 1$, a normalização tem efeito completo (equivalente ao esquema *BM11*). Valores menores reduzem este efeito, e se $b = 0$ o comprimento do documento não afeta a pontuação final, (como no modelo *BM15*).

$$NL = ((1\text{-}b) + b\frac{l_d}{l_{avg}}) \tag{3.29}$$

$$RD_{t,D} = \frac{tf_{t,D}(k_1+1)}{k_1((1\text{-}b) + b\frac{l_d}{l_{avg}}) + tf_{t,D}} \tag{3.30}$$

Consultas longas

Em situações onde as consultas podem ser descritas por *queries* longas, por exemplo, o caso em que um documento pode ser utilizado como base para a consulta, a consideração da frequência do termo na *query* pode ser mais um fator contribuinte para a estimativa de relevância. O componente $RQ_{t,Q}$ também é derivado a partir da modelagem em distribuições de *Poisson*, porém aplicadas ao conjunto de *queries* em vez do conjunto de documentos. O resultado é um peso semelhante ao $RD_{t,D}$, porém com parâmetros constantes próprios (equação 3.31). Todavia, para o caso de *queries* com poucos termos, este componente do peso deve ser desconsiderado.

$$RQ_{t,Q} = \frac{(k_3+1)qtf_{t,Q}}{k_3 + qtf_{t,Q}} \tag{3.31}$$

Estimativa de relevância

Finalmente, a relevância de um documento D para uma consulta Q pode ser obtida pelo somatório dos pesos dos termos da *query* com relação a D. O peso de cada termo é obtido pelo produto dos componentes apresentados anteriormente, como indica a equação 3.32.

$$R_{D,Q} = \sum_{t \in Q} RW_t \cdot RD_{t,D} \cdot RQ_{t,Q} \tag{3.32}$$

3.6.6 Apriori

A mineração de Dados, também referenciada como descoberta de conhecimento em bases de dados, é a área da ciência da computação destinada à descoberta de correlações e padrões frequentes num conjunto de dados. Informações extraídas de uma base de dados de transações de venda, por exemplo, têm alto valor para organizações que pretendem realizar processos de *marketing* guiados por informação – modelo denominado, *análise de carrinho de compras* (*market basket analysis*). Outros domínios de aplicação que também utilizam técnicas de mineração são: detecção de intrusão através da análise de *logs* de sistemas computacionais, pesquisas na área de saúde sobre a correlação entre doenças, sequenciamento de DNA etc [**?**].

Os padrões frequentes podem ser descritos por conjuntos de itens que ocorrem simultaneamente ou por implicações na forma $X \Rightarrow Y$, denominadas de *regras de associação*, sendo X e Y conjuntos de itens disjuntos ($X \cap Y = \emptyset$). *Suporte* e *confiança* são duas métricas para quantificar a força dos padrões de acordo com a sua representatividade no banco de dados de transações. O suporte de um conjunto de itens é a frequência com a qual ele ocorre numa base de dados. Para uma regra de associação $X \Rightarrow Y$, mede-se o suporte do conjunto de itens $X \cup Y$. A confiança de uma regra é medida pela frequência de Y nos registros que contém X, representando o grau de co-ocorrência de X e Y.

O *Apriori* é um algoritmo clássico para mineração de regras de associação sustentadas por medidas mínimas de suporte e confiança numa base de dados. Este problema é comumente decomposto em dois sub-problemas:

(1) Identificação de todos os conjuntos de itens que extrapolam um valor de suporte mínimo na base de dados (denominados de *conjuntos frequentes*).

(2) Produção de regras de associação a partir dos conjuntos frequentes, selecionando apenas as que satisfazem a condição de confiança mínima. Visto que as regras são partições binárias de conjuntos de itens, uma solução trivial para este problema é: para cada subconjunto S de um conjunto frequente F, gerar a regra $S \Rightarrow F - S$ e testar seu valor de confiança.

O *Apriori* foi o primeiro algoritmo a tratar do sub-problema *(1)*, que de fato é o mais desafiador, de forma mais eficiente. Uma solução ingênua para tal problema seria: listar todos os conjuntos candidatos (conjunto das partes do universo de itens) e selecionar os conjuntos frequentes a partir do cálculo de suporte para cada um. No entanto, esta é uma estratégia extremamente custosa visto que o conjunto das partes de um conjunto com n elementos contém 2^n subconjuntos, inviabilizando o cálculo para domínios de aplicação com um universo de itens grande [**?**]. A figura 3.8 ilustra através de um diagrama de *Hasse* o conjunto das partes do universo de itens $U = \{a, b, c\}$.

A inovação do *Apriori* sobre a abordagem ingênua é a redução da quantidade de conjuntos candidatos pelo descarte de certos conjuntos que comprovadamente não são conjuntos frequentes. Desta forma o algoritmo consegue detectar todos os conjuntos frequentes sem a necessidade de calcular o suporte para todos os 2^n subconjuntos possíveis.

A descoberta de conjuntos frequentes acontece por níveis, como uma busca em largura no diagrama de *Hasse* começando pelos conjuntos unitários. Em vez de gerar os conjuntos

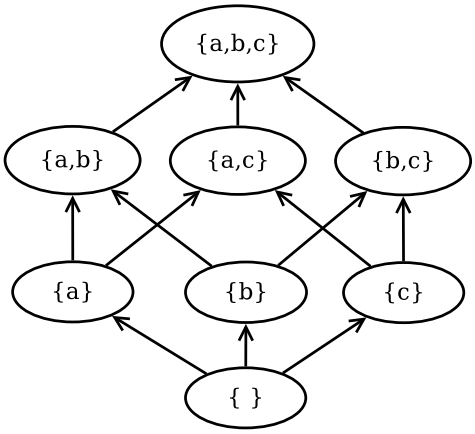

Figura 3.8: Conjunto das partes ilustrado por um diagrama de *Hasse*

candidatos a partir da base de dados, a cada nível da busca é feita uma combinação dos elementos para gerar os candidatos do nível seguinte. Neste ponto a solução se beneficia do seguinte princípio: qualquer subconjunto de um conjunto frequente também é um conjunto frequente. Portanto, só devem participar da nova combinação os elementos que apresentarem um suporte superior ao limite, pois um conjunto que não é frequente não será jamais subconjunto de um conjunto frequente [**?**].

A figura 3.9 ilustra a descoberta dos conjuntos frequentes em contraposição com o conjunto das partes do conjunto $U = \{a, b, c, d, e\}$. Neste exemplo, os subconjuntos $\{e\}$, $\{a, b\}$ e $\{b, d\}$ estão destacados por apresentarem suporte inferior ao limite. Consequentemente, todos os conjuntos dos quais estes são subconjuntos foram desconsiderados como conjuntos candidatos (nós com fundo cinza na figura). Portanto, apenas os nós com fundo branco teriam o suporte calculado.

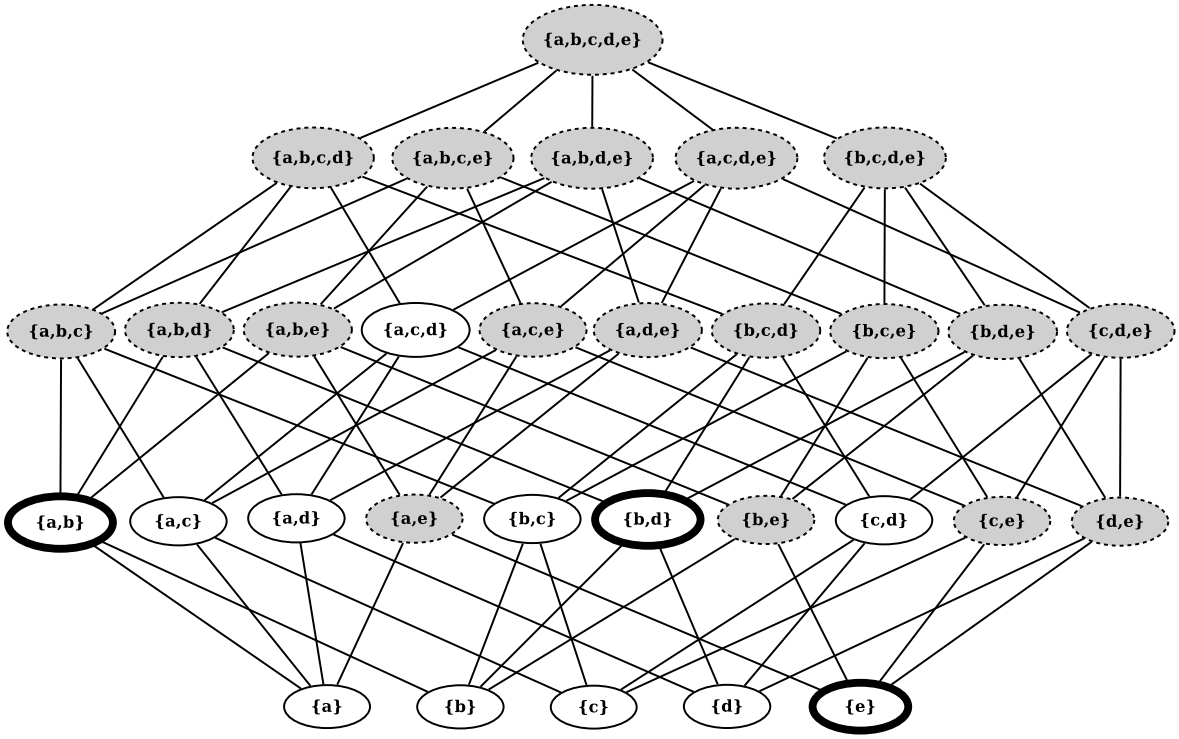

Figura 3.9: Geração de conjuntos candidatos pelo algoritmo Apriori

A introdução do *Apriori* representou um marco para o desenvolvimento de soluções para mineração de dados, motivando o surgimento de inúmeras variantes baseadas no mesmo princípio. Entre elas, surgiram algumas propostas específicas para situações onde os dados têm características adicionais conhecidas como, por exemplo, base de dados particionada, dados que satisfazem à determinadas restrições ou que fazem parte de uma taxonomia conhecida [**?**].

Apesar de apresentar um processo inovador para geração de regras de associação, o *Apriori* também apresenta fraquezas, sendo a principal delas a necessidade de percorrer a base de dados múltiplas vezes para cálculo de suporte e confiança dos conjuntos de itens. Algumas soluções alternativas fazem uso de estruturas de dados auxiliares para armazenar informações extraídas da base de dados numa única passagem, evitando desta forma repetidos acessos à mesma. Árvores de prefixos, árvores lexicográficas e matrizes binárias são algumas dessas estruturas [**?**].

3.7 Avaliação de recomendadores

A avaliação de sistemas de recomendação não é uma tarefa trivial, principalmente porque não há consenso sobre quais atributos devem ser observados e quais métricas devem ser adotadas para cada atributo [**?**]. Ademais, diferentes estratégias podem funcionar melhor ou pior, a depender com o domínio da aplicação e as propriedades dos dados. Por exemplo, algoritmos projetados especificamente para conjuntos de dados com um número muito maior de usuários do que de itens podem se mostrar inapropriados em domínios onde há muito mais itens do que usuários.

A compreensão das ações para as quais o sistema foi projetado (seção 3.3) é de fundamental importância para o planejamento dos testes e deve fundamentar as decisões metodológicas ao longo dos experimentos. Por exemplo, se a principal ação do recomendador é sugerir os n itens mais relevantes, deve-se priorizar modelos que tenham uma baixa taxa de erro entre os n primeiros itens; por outro lado, se todos os itens relevantes devem ser necessariamente retornados, o modelo ideal é o que maximiza a recuperação dos itens relevantes, independente da posição em que aparecem.

3.7.1 Seleção dos dados

[**?**] classifica procedimentos de avaliação de quanto ao conjunto de dados utilizados como (a) análises *offline*, que utilizam bases de dados previamente coletadas e (b) experimentos "ao vivo", realizados diretamente com usuários, seja num ambiente controlado (laboratório) ou em campo.

Análises *offline* geralmente são objetivas, com foco na acurácia das predições e performance das soluções [**?**]. Inicialmente os dados são particionados em porções de treinamento e de testes. Utiliza-se como base os dados de treinamento para prever recomendações para itens da porção de testes. Em seguida é feita a análise comparativa entre os resultados obtidos e os esperados. A seção 3.7.2 apresenta algumas métricas comumente utilizadas para comparar o desempenho de cada solução. No entanto, tais análises são prejudicadas em conjuntos de dados esparsos. Não se pode, por exemplo, avaliar a exatidão da recomendação de um item para um usuário se não existe uma avaliação prévia do usuário para tal item.

Por outro lado, nos experimentos "ao vivo" os recomendadores são disponibilizados para uma comunidade de usuários, cujas avaliações são coletadas na medida em que são produzidas. Neste caso, além de análises objetivas como a acurácia das soluções, pode-se avaliar fatores comportamentais como a participação e satisfação dos usuários. A esparsidade dos dados tem efeito menor neste tipo de experimento, visto que o usuário está disponível para avaliar se os itens recomendados são de fato relevantes ou não.

Quando não existem dados previamente disponíveis ou quando não são adequados para o domínio ou a ação principal do sistema a ser avaliado, pode-se ainda optar pelo uso de dados sintéticos. O uso de dados artificiais é aceitável em fases preliminares de testes, porém, tecer conclusões comparativas é arriscado uma vez que os dados produzidos podem se ajustar melhor para uma estratégia do que para outras [**?**].

3.7.2 Métricas

A utilidade prática de um sistema de recomendação pode ser avaliada a partir da observação de aspectos distintos, que comumente são combinados numa situação de comparação. Existem diversas métricas para avaliar a *acurácia* dos resultados, ou seja, o quanto que as estimativas previstas pelo sistema se aproximam das reais. Outro quesito é a *cobertura* do recomendador, que diz respeito à proporção de itens passíveis de serem recomendados entre todos os disponíveis. A satisfação do usuário ao utilizar o sistema também pode ser registrada, e informações como se ele foi surpreendido pelas recomendações pode revelar a qualidade do sistema de produzir recomendações não óbvias.

Para facilitar a percepção dos conceitos apresentados adiante, consideremos a seguinte situação. Um recomendador de aplicativos hipotético recomenda 20 programas a determinado usuário, dos quais apenas 14 são identificados por ele como de fato relevantes. O universo de aplicativos é composto por 500 itens e para participar do experimento pede-se que o usuário aprecie todos os itens e os classifique como relevantes ou irrelevantes. 150 foram apontados como relevantes.

O resultado da predição realizada pelo recomendador pode ser representado pela matriz de contingência da tabela 3.9. A quantidade de itens recomendados que de fato são relevantes é indicada pelos *verdadeiros positivos (VP)*; *falsos positivos (FP)* representam a quantidade de itens incorretamente classificados como relevantes (rejeitados pelo usuário); os que não fazem parte da recomendação mas posteriormente foram marcados como relevantes são os *falsos negativos (FN)*; e os *verdadeiros negativos (VN)* não foram recomendados nem classificados como relevantes pelo usuário.

	Predito		
Real	$VP = 14$	$FN = 136$	positivo: 150
	$FP = 6$	$VN = 344$	negativo: 350
	positivo: 20	negativo: 480	Total: 500

Tabela 3.9: Matriz de contingência de uma recomendação

Duas categorias de métricas de acurácia são consideradas por [**?**]: *acurácia de classificação*, que diz respeito à frequência com a qual o sistema classifica os itens corretamente; e *acurácia de predição*, que pondera as diferenças entre as pontuações previstas paras os itens e as reais.

Um medida simples de acurácia é quantificada pela proporção de itens classificados corretamente do total de itens do conjunto ($\frac{VP+VN}{P+N}$). Esta métrica no entanto não considera a quantidade de objetos pertencentes a cada uma das classes e por esta razão pode causar uma falsa impressão de bons resultados. Por exemplo, suponha que 90% dos itens seja da classe A. Se um classificador indica a classe A para todos os casos, ele apresenta uma acurácia de 90% mesmo sem ser útil na prática.

Algumas métricas comumente utilizadas para avaliar a eficácia de modelos preditivos são apresentadas a seguir e sumarizadas na tabela 3.10.

Precisão ou preditividade positiva

Proporção de itens relevantes entre todos os classificados como relevantes. No exemplo dado, a precisão é de 70% ($\frac{14}{20}$).

Recuperação, sensibilidade ou taxa de verdadeiros positivos

Proporção de itens apresentados como relevantes dentre todos os relevantes. Mede a capacidade do modelo de identificar resultados positivos. No exemplo, a recuperação é de 9.33% ($\frac{14}{150}$).

Medida F

A medida F (F *score*) combina numa mesma métrica os valores de precisão (p) e recuperação (r). Sua forma mais conhecida é $F_1 = \frac{2pr}{p+r}$, que representa a média harmônia entre p e r. Sua fórmula genérica é $F_\beta = (1 + \beta^2)\frac{pr}{\beta^2 p} + r$, sendo que F_2 prioriza recuperação em detrimento de precisão e $F_{0.5}$ pontua mais a precisão. No exemplo, os valores de F_1, F_2 e $F_{0.5}$ são, respectivamente, 0.16, 0.21, 0.56.

Especificidade ou taxa de verdadeiros negativos

Proporção de verdadeiros negativos entre todos os classificados como negativos. Avalia a capacidade do modelo de identificar itens irrelevantes como tal. No exemplo, a especifidade é de 98% ($\frac{344}{350}$).

Taxa de falsos positivos

Proporção de negativos que foram classificados erroneamente como positivos. Esta medida é o complemento da especificidade ($1-$*especificidade*). No exemplo, tem valor de 2%.

Curva ROC

As curvas ROC (*Receiver Operating Characteristic*) foram desenvolvidas em pesquisa para detecção de ruído em sinais de rádio. Atualmente é uma técnica bastante utilizada na definição de valores limítrofes para diagnósticos médicos.

A curva representa graficamente o poder discriminativo de um classificador binário. Cada ponto expressa a qualidade do resultado de um processo de classificação por meio da *taxa de verdadeiros positivos (tpr)* (sensibilidade) e *taxa de falsos positivos (fpr)* (complemento da especificidade). Os pontos são dispostos num gráfico com valores de *tpr* no eixo das ordenadas e *fpr* nas abcissas.

Muitas técnicas de classificação produzem como resultado uma pontuação associada a cada item, que quando superior a um determinado limiar (*ponto de corte*) causa sua categorização para um grupo ou outro. Sendo assim, pontos de corte diferentes representam modelos preditivos distintos. A identificação do limiar que produz os melhores resultados de classificação pode ser auxiliada pela análise da curva ROC produzida a partir da variação do ponto de corte.

Alguns pontos do gráfico são bastante informativos. O ponto $(0, 0)$ representa uma classificação que não produz resultados, nem positivos nem negativos; o ponto $(0, 1)$ indica que todos os positivos são corretamente identificados e não há ocorrência de falsos positivos (situação de sensibilidade e especificidade máximas do recomendador). Um modelo que classifica todos os itens como positivos é representado pelo ponto $(1, 1)$, enquanto que o $(1, 0)$ representa um modelo que sempre faz predições incorretas.

A curva ROC de um classificador perfeito é desenhada sobre o eixo das abcissas até o ponto $(0, 1)$ e segue na horizontal até o ponto $(1, 1)$. Já um modelo com comportamento aleatório é representado na diagonal ascendente que liga os pontos $(0, 0)$ e $(1, 1)$.

Uma medida comum de comparação entre duas curvas ROC é a *área sob a curva (AUC)*, que é numericamente igual à probabilidade de, dados dois exemplos escolhidos randomicamente, um positivo e outro negativo, o positivo seja melhor pontuado que o negativo [**?**].

Coeficiente de correlação de Matthews (MCC)

Resume as informações da matriz de contingência em um único valor. É geralmente utilizado para identificar o limiar com melhor resultado num curva ROC. Os pontos com melhores MCC estão localizados no quadrante superior esquerdo do gráfico ROC.

Erro absoluto e quadrático médio (MAE e MSE)

Medidas de desvio médio absoluto (MAE) e quadrático (MSE) entre pontuações previstas (p_i) e reais (r_i). A acurácia do modelo é inferida a partir da comparação numérica entre os valores preditos e pontuações reais indicadas pelo usuário, para os itens cujas medidas são conhecidas.

Métrica	Fórmula	Categoria
Precisão	$p = \frac{VP}{(VP+FP)}$	Acurácia de classificação
Recuperação	$r = \frac{VP}{(VP+FN)}$	
Medida F_1	$F_1 = \frac{2pr}{p+r}$	
Especificidade	$\frac{VN}{VN+FP}$	
Curva *ROC*	Área sob a curva (AUC) e MCC	
MCC	$MCC = \frac{(VP*VN)-(FP*FN)}{\sqrt{(VP+FP)(VP+FN)(VN+FP)(VN+FN)}}$	
MAE	$\lvert\overline{E}\rvert = \frac{\sum_{i=1}^{N} \lvert p_i - r_i \rvert}{N}$	Acurácia de predição
MSE	$\lvert\overline{E}\rvert = \frac{\sum_{i=1}^{N} \lvert p_i - r_i \rvert^2}{N}$	

Tabela 3.10: Métricas de acurácia de sistemas preditivos

3.7.3 Validação cruzada

Técnicas de reamostragem, como a *validação cruzada*, são comumente utilizadas na avaliação de modelos preditivos, principalmente quando se dispõe de uma quantidade limitada de dados para testes. Isola-se uma porção aleatória dos dados cuja classe é conhecida; treina-se o modelo com os demais dados e em seguida a porção reservada é submetida ao modelo para testá-lo. A acurácia dos resultados pode então ser medida por meio da comparação dos resultados obtidos com os esperados. A validação em rodadas (*k-fold cross-validation*) consiste basicamente nos seguintes passos:

1. O conjunto de dados original é particionado aleatoriamente em k subconjuntos;
2. Em cada uma das k rodadas:
   (a) Um dos subconjuntos é reservado para testar o modelo;
   (b) Os demais subconjuntos são passados ao modelo como dados de treinamento;
   (c) Uma predição é gerada e avaliada por meio de métricas pertinentes.
3. Ao final dos testes, os k resultados são combinados para produzir uma estimativa única.

3.8 Riscos à privacidade de usuários

Por lidar com informações pessoais, ainda que anonimizadas, sistemas de recomendação são vulneráveis a ataques que podem comprometer a privacidade dos usuários. Qualquer possibilidade de revelação de dados não públicos é considerado um vazamento de informações do recomendador.

Considere a equação $F(D, q) = R$, onde F é a função para composição das sugestões, D o conjunto de dados utilizado pelo recomendador, q a consulta e R a recomendação. Em tese, se a função F é pública, um atacante é capaz de realizar infinitas consultas a F, variando os valores de R para descobrir quais seriam os possíveis conjuntos de dados D que satisfariam a premissa $F(D, q) = R$. Quanto menos conjuntos de dados possíveis, maior é a vulnerabilidade do recomendador.

Na prática, pode-se partir de uma hipótese cuja validade é checada por meio de consultas ao recomendador. Dado que o atacante tem acesso a F, ele é capaz de inferir informações a partir da sugestão produzida. Por exemplo, dado que um usuário comprou os itens a, b e c e deseja-se saber se é provável que ele também tenha comprado d. O atacante pode realizar repetidas consultas ao recomendador, observando se d aparece na recomendação com uma dada frequência.

Resultados de pesquisa recente apresentada por [**?**] também demonstram que mudanças em recomendações ao longo do tempo podem revelar transações de usuários, no caso em que informações auxiliares sobre os mesmos sejam conhecidas. Por exemplo, suponha que um atacante tenha conhecimento sobre compras anteriores de um cliente, visto que são dados públicos: avaliações de produtos e publicações em redes sociais realizadas pelo próprio usuário. Novas compras afetam os cálculos de similaridade entre os itens novos e antigos, possivelmente causando alterações perceptíveis para recomendações relacionadas aos itens antigos. O estudo demonstra que um atacante pode descobrir quais foram os novos itens comprados por meio de análises das mudanças relacionadas aos itens antigos.

Os ataques até então apresentados são classificados como ataques passivos, dado que o conjunto de dados original não é afetado. Um exemplo de ataque ativo seria o envio de perfis falsos de usuários ao recomendador para modificar seu comportamento, aumentando assim a chance de sucesso em ataques posteriores.

Capítulo 4

Trabalhos correlatos

Nos últimos anos foram publicados, em âmbito nacional e internacional, inúmeros trabalhos acadêmicos nas áreas de mineração de dados e recuperação da informação aplicadas aos mais diversos domínios. Neste capítulo nos limitamos a apresentar trabalhos correlatos que tratam especificamente do problema de recomendação desenvolvidos no contexto de aplicativos. São também mencionados a seguir projetos desenvolvidas de maneira independente que, mesmo sem o rigor acadêmico, serviram como fontes de inspiração e referências.

4.1 Anapop/Popsuggest

Esta solução foi disponibilizada em 2007 pelo desenvolvedor Debian Enrico Zini[1] como uma ilustração das possibilidades de uso dos dados coletados pelo *Popcon* – concurso de popularidade de pacotes Debian, apresentado na seção 5.3.3.

A ferramenta *anapop* indexava previamente a base de dados integral do *Popcon*. Diante de uma lista de pacotes de determinado usuário, por meio de buscas no índice previamente criado, o *popsuggest* sugeria pacotes que usuários de perfil similar tinham instalados. Este é um exemplo de aplicação de estratégia de recomendação colaborativa, implementado de forma ingênua uma vez que nenhuma seleção de atributos era realizada.

O serviço foi disponibilizado na Web por alguns meses, porém, segundo depoimento do autor durante a Debconf11[2], foi descontinuado por falta de colaboradores interessados em evoluir o protótipo.

4.2 Debommender

O *Debommender* é um sistema de recomendação para pacotes GNU/Linux desenvolvido no âmbito de um trabalho final de graduação, apresentado em 2007 na Universidade Federal do Rio Grande do Sul [**?**]. Segundo o autor, a ferramenta foi desenvolvida como prova de conceito, não sendo portanto integrada aos serviços da distribuição.

Foram realizados experimentos com cerca de 30 usuários para avaliar a eficácia as soluções implementadas. O modelo que obteve melhores resultados utilizava uma estratégia híbrida por ponderação, onde os resultados de um componente baseado em conteúdo e outro colaborativo eram combinados de acordo com pesos estabelecidos.

A fonte de dados utilizada pelo recomendador para estratégias colaborativas era o conjunto de dados de entrada fornecidos pelos usuários participantes dos experimentos (suas listas de pacotes). Talvez por este motivo, as estratégias baseadas em conteúdo obtiveram melhor

[1] http://www.enricozini.org/2007/debtags/popcon-play/
[2] http://penta.debconf.org/dc11_schedule/events/773.en.html)

cobertura (proporção de itens disponíveis passíveis de recomendação) do que as colaborativas, pois, devido ao número reduzido de usuários, muitos pacotes disponíveis não estavam presentes em nenhum perfil e portanto não podiam aparecer nas recomendações produzidas.

4.3 Mineração de dados do Popcon

Trabalho de mestrado que experimentou a implementação de técnicas de mineração de dados na base do *popcon* para produção de regras de associação. A dissertação com título *"Projeto e criação de um sistema para produção de sugestões personalizadas para o Instalador Debian"*[3] foi defendida em agosto de 2007 na Universität Paderborn, na Alemanha [**?**].

O trabalho foi apresentado na conferência anual de Desenvolvedores Debian (DebConf7[4]), ainda em fase de desenvolvimento. Segundo relato do autor durante a DebConf11, a geração de regras de associação se mostrou bastante custosa e, diante do crescimento do repositório de pacotes desde 2007, talvez a solução tenha se tornado impraticável.

4.4 AppStream

Grande parte das distribuições GNU/Linux têm investido no desenvolvimento de interfaces para facilitar o gerenciamento de aplicativos e a forma como se obtém informações sobre os mesmos. Entre os dias 18 e 21 de janeiro 2011 aconteceu a primeira reunião acerca desta temática com a presença de desenvolvedores de distribuições variadas (*Cross-distribution Meeting on Application Installer*). O encontro teve como principais objetivos a definição de padrões entre os diferentes projetos no que diz respeito a: procedimentos de instalação de aplicações; metadados associados aos pacotes; o modo como tais informações devem ser geradas e armazenadas; protocolo para manutenção de metadados dinâmicos; e a definição de quais metadados devem ser compartilhados entre as distribuições, em detrimento de outros considerados específicos de cada projeto [**?**].

Decidiu-se que a colaboração entre os diversos serviços seria guiada pela especificação OCS (*Open Collaboration Service*), um padrão aberto projetado para dar suporte à colaboração entre serviços web, permitindo o armazenamento de avaliações de usuários e outras informações do domínio de aplicação.

Apesar de o *Software Center* não estar em conformidade com a especificação OCS, esta foi a ferramenta escolhida como plataforma base de desenvolvimento do *AppStream*, dado que há interesse por parte dos desenvolvedores de adequá-lo ao padrão. A única pendência com relação à adoção deste *software* é a exigência por parte da Canonical – empresa responsável pelo desenvolvimento do Ubuntu – de que todos os colaboradores do projeto assinem um termo de atribuição de *copyright* (*Canonical Contributor License Agreement*)[5]. Esta exigência, além causar antipatia de alguns membros da comunidade que se recusam a assinar o termo por entenderem que ele vai de encontro com a sua liberdade, de fato impossibilita a participação no projeto de desenvolvedores que são contratados por outras empresas as quais não permitem a assinatura de tal termo com terceiros. O impasse ainda não foi resolvido e atualmente o andamento dos trabalhos está suspenso[6]

[3] Tradução do título original do trabalho no idioma alemão.
[4] https://penta.debconf.org/~joerg/events/83.en.html
[5] http://www.canonical.com/contributors
[6] http://lists.freedesktop.org/archives/distributions/2011-May/000583.html

4.5 *Ratings and Reviews*

A Canonical mantém um servidor para armazenar avaliações de usuários sobre aplicativos, com pontuação e comentários (*Ubuntu Ratings and Reviews Server*[7]). Esta pode vir a ser uma poderosa fonte de dados para a produção de recomendações sobre aplicativos, alternativa à base do *popcon*, que não foi projetada para fins de recomendação. No entanto, os detalhes de implementação do servidor não são públicos e ao que tudo indica a Canonical deve manter este esforço como uma iniciativa própria.

Embora já faça parte da pauta de discussões no contexto do *FreeDesktop.org* um esquema global de armazenamento de avaliações sobre aplicativos enviadas por usuários de múltiplas distribuições, ainda não há previsão para tal solução de ser implementada [**?**].

4.6 Recomendadores para dispositivos móveis

A explosão no uso de dispositivos móveis na última década naturalmente refletiu-se no desenvolvimento em larga escala de aplicativos para estes dispositivos. A maioria destes programas não é livre e a documentação geralmente é escassa, portanto o conhecimento em detalhes das estratégias de recomendação adotadas é prejudica. Abaixo estão listados alguns recomendadores de aplicativos para os dispositivos *iPhone, iPod* e *iPad*, desenvolvidos pela *Apple*, e *Android*, desenvolvido pela *Google*.

Genius

Sistema de recomendação desenvolvido pela *Apple*, inicialmente para ser acoplado ao *iTunes* e oferecer sugestões de músicas para os usuários. A partir do iOS 3.1, o recomendador pode ser utilizado para sugerir aplicativos da loja da *Apple* (*App Store*) com base nos programas já instalados no dispositivo. Para compor recomendações, o *Genius* envia periodicamente à *Apple* informações sobre os aplicativos instalados. O histórico de compras do usuário na *App Store* e informações fornecidas por outros clientes também são utilizados para deduzir recomendações mais relevantes[8].

Appolicious

Serviço que permite a navegação por categorias de aplicativos, busca simultânea em repositórios de aplicativos para *Android* e *iPhone*, além de fornecer listas do tipo "os 10 mais"[9].

Heyzap

Recomendador para *Android* e *iPhone* especializado em jogos que utiliza estratégia colaborativa para composição de sugestões personalizadas[10].

Explor

Produz recomendações personalizadas para *iPhone* com compartilhamento de favoritos com rede de amigos[11].

[7] http://launchpad.net/rnr-server
[8] http://support.apple.com/kb/HT2978
[9] http://www.androidapps.com/
[10] http://www.heyzap.com/
[11] http://explorapp.com/

Apptitude

Sistema de recomendação de aplicativos para *iPhone* que faz interface com o *Facebook*[12] para obtenção da rede de amigos do usuário. Para inferir a lista de aplicativos dos amigos, o sistema analisa as publicações no mural dos usuários da rede social, além de itens marcados como interessantes ("curtir"), portanto não exige que os amigos também sejam usuários cadastrados[13].

Applause

Recomendador para *Android* baseado em informações de contexto, como localização do usuário, passatempo favorito, idade e compromissos agendados. O sistema ainda está em fase de desenvolvimento, no entanto o Departamento de Informática (ICS) da universidade UC Irvine já disponibilizou um *survey* para avaliar os métodos de aquisição de informações de contexto[14].

[12] http://www.facebook.com
[13] http://www.apptitu.de/
[14] http://ucistudy.ics.uci.edu/android/

Capítulo 5

AppRecommender

Este capítulo apresenta o *AppRecommender* como proposta de um recomendador de aplicativos GNU/Linux. Detalhamento sobre o problema abordado, justificativa de escolha da plataforma, fontes de dados da solução, decisões de projeto e estratégias implementadas são alguns dos tópicos abordados a seguir.

5.1 Caracterização do problema

A composição de recomendações de forma automatizada consiste na dedução de um conjunto de itens de potencial interesse para determinado usuário a partir da avaliação prévia realizada pelo usuário acerca de um conjunto de itens que ele tem conhecimento. De acordo com a estratégia de recomendação escolhida, podem ser sugeridos por exemplo itens com características semelhantes aos considerados relevantes ou, diante da disponibilidade de avaliações por outros usuários, itens bem avaliados por indivíduos com perfis semelhantes podem fazer parte da recomendação.

No contexto deste trabalho, os clientes do recomendador são instâncias de sistemas GNU/Linux. Os programas são mapeados como itens, de modo que o perfil do usuário é composto a partir da lista de aplicativos instalados no sistema. Neste modelo não existem avaliações explícitas sobre a relevância dos itens para o usuário. O perfil é definido pelo comportamento do sistema, caracterizando uma avaliação implícita, ao assumirmos que a presença de um aplicativo em sua lista de programas é um indicativo de relevância. Para o caso de pontuação multi-valorada, além da lista de aplicativos instalados, informações adicionais precisam ser disponibilizadas. Por exemplo, a indicação de utilização recente de um programa pode resultar numa pontuação superior do que a de um aplicativo que não tenha sido executado há bastante tempo.

Um caso típico de recomendação do *AppRecommender* é então caracterizado da seguinte maneira: dada a lista programas instalados em determinado sistema, o recomendador retorna uma lista de aplicativos sugeridos, que supostamente são aplicativos de potencial interesse para os usuários daquele sistema.

5.2 Escolha da plataforma

A distribuição escolhida como base para o desenvolvimento deste trabalho foi o Debian GNU/Linux. No entanto, a independência de plataforma foi sempre levada em consideração na fase de desenvolvimento com o intuito de que os resultados sejam facilmente adaptáveis para outros ambientes. As seguir estão descritos os critérios que pautaram esta escolha.

1. **Esquema consistente de distribuição de aplicativos.** O gerenciamento de pacotes

em sistemas Debian GNU/Linux é realizado através do *apt*, cujas funcionalidades foram apresentadas na seção 2.4.1. Apesar de atualmente outras distribuições oferecerem ferramentas similares, o *apt* é certamente uma das mais maduras, sendo geralmente apontada como uma das principais razões da explosão no surgimento de distribuições derivadas do Debian, herdeiras do esquema.

2. **Disponibilidade de dados estatísticos.** A base de dados do *Popcon* (seção 5.3.3) ultrapassou a marca de 100.000 colaboradores[1] em fevereiro de 2011. É certamente uma das maiores coleções de dados disponíveis atualmente sobre a utilização de pacotes Debian, compondo uma importante fonte de informação para a realização de estratégias colaborativas de recomendação.

3. **Popularidade.** O Debian é um projeto de destaque no ecossistema do software livre. Desde o lançamento da primeira versão de sua distribuição, em 1993, o projeto cresceu bastante em termos de componentes de software (atualmente provê mais de 25.000 pacotes), colaboradores e usuários. A *Distrowatch*, que tem 323 distribuições ativas em sua base de dados[2], classifica o Debian GNU/Linux entre as 10 distribuições mais populares[3]. Em suas estatísticas de páginas mais visitadas o Debian aparece na quinta posição[4]. Já o *Linux Counter apresenta o Debian como a segunda distribuição mais popular (*16% *das máquinas cadastradas que rodam o kernel Linux*[5] , ficando atrás apenas do Ubuntu[6], que é uma distribuição derivada do Debian, com 24%. Nas pesquisas da *W*[3] *Techs* sobre tecnologias para serviços web, o Debian aparece em segundo lugar, estando presente em 27% dos servidores[7] – na primeira posição está o CentOS com 31%.

4. **Maturidade do projeto.** De modo geral, quando o projeto Debian é mencionado trata-se não somente do sistema operacional, mas de toda infraestrutura de desenvolvimento e coordenação que dá suporte ao trabalho de cerca de 900 desenvolvedores oficiais[8], além de outros milhares de colaboradores ao redor do globo. O trabalho é realizado de forma colaborativa, afinado pelo objetivo comum de produzir e disponibilizar livremente um sistema operacional de qualidade para seus usuários [**?**]. A interação entre os desenvolvedores acontece majoritariamente através da Internet, por meio de canais IRC e listas de discussão públicas. Não existe uma entidade formal ou qualquer tipo de organização que concentre, coordene ou defina as atividades do projeto. O que observa-se é um modelo de governança consolidado que emergiu naturalmente ao longo de sua história [**?**].

5. **Possibilidade de integração dos resultados do trabalho.** De acordo com o *contrato social do Debian*[9], o desenvolvimento do projeto é pautado pelas necessidades dos usuários e da comunidade FOSS. Portanto as iniciativas de colaboradores individuais, sejam eles desenvolvedores oficiais ou não, serão igualmente consideradas e poderão fazer parte da distribuição desde que sigam os princípios do projeto e sejam considerados úteis para a comunidade. A autora deste trabalho colabora com o projeto desde de 2005,

[1] http://lists.alioth.debian.org/pipermail/popcon-developers/2011-February/001913.html
[2] Consulta realizada em 24 de janeiro de 2011.
[3] http://distrowatch.com/dwres.php?resource=major
[4] http://distrowatch.com/stats.php?section=popularity
[5] http://counter.li.org/reports/machines.php
[6] http://www.ubuntu.com/community/ubuntu-and-debian
[7] http://w3techs.com/technologies/history_details/os-linux
[8] http://www.perrier.eu.org/weblog/2010/08/07#devel-countries-2010
[9] http://www.debian.org/social_contract.pt.html

tendo atuado em esforços de tradução, empacotamento de programas, organização da conferência anual de desenvolvedores e atualmente faz parte da equipe do Debtags[10].

5.3 Fontes de Dados

O projeto Debian tem se destacado no universo das distribuições por suas iniciativas pioneiras no campo de gerenciamento de aplicações [**?**]. Diante da complexa e crescente estrutura do projeto, observa-se um esforço por parte dos desenvolvedores, principalmente da equipe responsável pelo controle de qualidade[11], de reunir, organizar e disponibilizar as informações ou meta-dados concernentes a esta estrutura [**?**].

Algumas destas iniciativas que serviram como fontes de dados para o *AppRecommender* são detalhadas a seguir. Importante ressaltar que todas estas foram desenvolvidas inicialmente num contexto extra-oficial e, ao passo em que se mostraram úteis e eficazes, foram absorvidas pela comunidade de usuários e desenvolvedores.

5.3.1 Debtags

Para fins de organização, o repositório oficial Debian é particionado em *seções*, que atualmente representam 53 grupos de pacotes[12]. A seção de um pacote é definida no momento do empacotamento, dado que vem declarada em seu conteúdo (arquivo *control*).

Debtags é um esquema de classificação idealizado por Enrico Zini como uma maneira de categorizar pacotes alternativa às seções [**?**]. A principal motivação desta iniciativa foi a impossibilidade de relacionar pacotes a múltiplas seções. Um navegador web, por exemplo, não poderia ser categorizado como *network* e *web* simultaneamente. O uso de *tags* (em português, rótulos) possibilitaria a criação de uma coleção estruturada de metadados que poderia ser utilizada para implementar métodos mais avançados do que os existentes para apresentação, busca e manutenção do repositório de pacotes Debian.

A proposta foi apresentada na Debconf5 e foi paulatinamente sendo adotada por desenvolvedores em suas atividades, sendo atualmente utilizada como base de inúmeras ferramentas no Debian, tendo atingido a marca de 45% de pacotes categorizados[13].

Utilizando *Debtags*, os pacotes podem ser caracterizados por múltiplos atributos, que são (propositalmente) definidos num momento posterior à concepção do pacote. Dado que a base de tags é mantida de forma independente ao repositório, as modificações ao longo do tempo não trazem impacto algum ao desenvolvimento de pacotes. A atribuição de tags a pacotes é realizada por colaboradores por meio do website do projeto[14] e revisada manualmente antes de ser incorporada à base de dados.

A base de dados é armazenada num arquivo texto segue um formato simples, conforme ilustrado na figura 5.1. O conjunto de *tags* disponível faz parte de um vocabulário controlado[15], que também recebe contribuições de colaboradores. O esquema é estruturado para permitir a classificação por diferentes pontos de vista, que caracterizam as facetas.

Ao indicar novas *tags* para um pacote, o usuário é surpreendido com sugestões de outras *tags* que geralmente são aplicadas em conjunto com as já selecionadas. Esta é uma aplicação de recomendação com base em regras de associação descobertas a partir de análise da base de dados de *tags*. O algoritmo utilizado para produção das regras é o Apriori, descrito na seção 3.6.6.

[10] http://www.ime.usp.br/~tassia/debian.html
[11] http://qa.debian.org
[12] http://packages.debian.org/unstable/
[13] Consulta realizada em junho de 2011
[14] http://debtags.alioth.debian.org/todo.html
[15] http://debtags.alioth.debian.org/vocabulary/

```
acx100-source: admin::kernel, implemented-in::c, role::source, use::driver

adduser: admin::user-management, implemented-in::perl, interface::commandline,
role::program, scope::utility

apache2: implemented-in::c, interface::daemon, network::server, network::service,
protocol::http, protocol::ipv6, role::metapackage, role::program, suite::apache,
web::server, works-with-format::html, works-with::text

apbs: field::chemistry, implemented-in::c, interface::commandline, role::program,
scope::utility

apcalc: field::mathematics, interface::shell, interface::text-mode, role::program,
scope::utility
```

Figura 5.1: Excerto da base do Debtags

O *Debtags* é uma poderosa ferramenta para a construção de estratégias de recomendação de pacotes baseadas em conteúdo. É fato que o conteúdo acerca de pacotes pode ser expresso em termos de atributos extraídos dos próprios pacotes, porém, a caracterização por meio de *tags* já fornece uma caracterização possível de ser utilizada e a baixo custo computacional.

5.3.2 Índice de informações sobre pacotes (apt-xapian-index)

O pacote *apt-xapian-index*[16] provê um conjunto de ferramentas para manutenção e busca em um índice Xapian de informações sobre pacotes Debian. *update-apt-xapian-index* permite a instalação de *plugins* no diretório `/usr/share/apt-xapian-index` para indexar qualquer tipo de informação relacionada aos pacotes, como tags, popularidade ou pontuação. A ferramenta *axi-cache*, pode ser utilizado para consultas no índice.

O índice criado é mantido em `/var/lib/apt-xapian-index`. Cada pacote é representado por um documento e as meta-informações relacionadas são mapeadas em termos dos documentos. Alguns termos são indexados com prefixos especiais para facilitar a busca, por exemplo, "XP" para o nome do pacote, "XS" para a seção do repositório, "XT" para tags e "Z" para termos lematizados. A figura 5.2 apresenta a lista de termos indexados para o pacote *2vcard*.

```
['2vcard', 'XP2vcard', 'XSutils', 'XTimplemented-in::perl', 'XTrole::program',
 'XTuse::converting', 'Za', 'Zabook', 'Zaddressbook', 'Zalia', 'Zan', 'Zand',
 'Zbalsa', 'Zby', 'Zcan', 'Zclient', 'Zconvert', 'Zcurrent', 'Zemail', 'Zeudora',
 'Zexampl', 'Zfile', 'Zfollow', 'Zfor', 'Zformat', 'Zfrom', 'Zgnomecard', 'Zis',
 'Zjuno', 'Zldif', 'Zlittl', 'Zmh', 'Zmutt', 'Zonli', 'Zperl', 'Zpine', 'Zpopular',
 'Zscript', 'Zthat', 'Zthe', 'Zto', 'Zuse', 'Zvcard', 'Zwhich', 'Zyou', 'a',
 'abook', 'addressbook', 'addressbooks', 'alias', 'an', 'and', 'balsa', 'by', 'can',
 'client', 'convert', 'currently', 'email', 'eudora', 'example', 'file', 'files',
 'following', 'for', 'format', 'formats', 'from', 'gnomecard', 'is', 'juno', 'ldif',
 'little', 'mh', 'mutt', 'only', 'perl', 'pine', 'popular', 'script', 'that', 'the',
 'to', 'use', 'used', 'vcard', 'which', 'you']
```

Figura 5.2: Lista de termos indexados para o pacote *2vcard*

5.3.3 Popularity Contest (popcon)

O *popcon* é um "concurso de popularidade" entre pacotes Debian criado pelo desenvolvedor Avery Pennarun em 1998 com o propósito inicial de auxiliar a escolha dos pacotes que devem

[16] http://www.enricozini.org/sw/apt-xapian-index/

ser incluídos no primeiro CD de instalação[17] (os mais populares são selecionados). Atualmente o repositório de pacotes Debian pode ser obtido em 52 imagens de CDs ou 8 de DVD. Dado que comumente apenas a primeira imagem é obtida por *download* – os demais pacotes podem ser obtidos diretamente do repositório por meio de uma conexão de rede – a priorização de pacotes populares na primeira imagem tende a contribuir para a satisfação dos usuários.

Na instalação de um sistema Debian o administrador é convidado a participar do concurso. Se aceitar, o software cliente do *popcon* é instalado na máquina e envia para um servidor central periodicamente a lista de pacotes instalados naquele sistema, indicando ainda quando cada pacote foi utilizado pela última vez.

A figura 5.3 apresenta um exemplo de submissão do popcon. Os campos temporais são indicados no formato *Unix time_t*[18]. A primeira linha contém um hash que identifica um sistema unicamente no concurso. Cada linha seguinte representa um pacote instalado no sistema, no formato `<atime> <ctime> <package-name> <mru-program> <tag>`, detalhado na tabela 5.1.

```
1  POPULARITY-CONTEST-0 TIME:914183330 ID:b92a5fc1809d8a95a12eb3a3c84166dd
2  914183333 909868335 grep /bin/fgrep
3  914183333 909868280 findutils /usr/bin/find
4  914183330 909885698 dpkg-awk /usr/bin/dpkg-awk
5  914183330 909868577 gawk /usr/bin/gawk
6  ...
7  ...
8  ...
9  END-POPULARITY-CONTEST-0 TIME:914183335
```

Figura 5.3: Exemplo de submissão do popcon

Campo	Descrição
<package-name>	Nome do pacote Debian que contém o arquivo <mru-program>
<mru-program>	Programa, biblioteca ou cabeçalho contido no pacote que foi utilizado mais recentemente.
<atime>	Tempo de acesso do <mru-program> no disco, atualizado pelo kernel cada vez que o arquivo é aberto.
<ctime>	Tempo de criação do <mru-program> no disco, definido no momento de instalação do pacote.
<tag>	RECENT-CTIME: indica que <atime> é muito próximo de <ctime>, geralmente quando o pacote foi recentemente instalado ou atualizado; OLD: <atime> é anterior a 30 dias atrás, portanto o pacote não foi usado no último mês; NOFILES: o pacote não contém programas, portanto <atime>, <ctime> e <mru-program> são inválidos.

Tabela 5.1: Descrição do formato de uma submissão *popcon*

A informação sobre o uso dos pacotes também tem sido utilizada como guia para o time de qualidade acerca de quais pacotes merecem atenção especial. Os times de tradução também têm considerado estes dados para ordenar sua lista de prioridades de acordo com a popularidade dos pacotes. Por outro lado, a baixa popularidade é um dos parâmetros para a remoção de um pacote do repositório (*low-popcon*). Pacotes considerados problemáticos[19] que não são populares tendem a perder a simpatia dos desenvolvedores.

[17] http://lists.debian.org/debian-devel-announce/2004/03/msg00009.html

[18] Quantidade de segundos desde meia-noite de primeiro de janeiro de 1970 no horário GMT.

[19] p. ex. pacotes órfãos (sem mantenedor) ou cujo mantenedor está inativo há bastante tempo (na terminologia do Debian, mantenedor em MIA - *Missing in Action*), contendo muitos *bugs*, especialmente se forem *bugs RC* (que impedem o lançamento)

Essa abordagem, no entanto, tem sido duramente criticada. Segundo Joey Hess[20], uma vantagem do Debian é justamente que não apenas programas populares são empacotados, mas os incomuns e específicos de um nicho de usuário também costumam estar disponíveis em pacotes. E de fato o *popcon* não mede o benefício de pacotes poucos usados estarem disponíveis no repositório, prontos para serem usados. Portanto, ao remover pacotes que aparentemente não são populares corre-se o risco de transformar o Debian numa distribuição homogênea, submetida à "tirania da maioria".

Existem ainda questões relativas a (1) representatividade desses dados, visto que alguns perfis de usuários dificilmente participam do concurso (p. ex. sistemas embarcados); e (2) acurácia de informações temporais, dado que `<atime>` e `<ctime>` podem ser inconsistentes caso o sistema de arquivos tenha sido montado com a opção `noatime`.

Todas essas ressalvas devem ser consideradas quando pretende-se utilizar os dados do *popcon*. No entanto, desde que as informações sejam manejadas de forma consciente e responsável, acredita-se que valiosas correlações possam ser reveladas após uma série de análises.

As submissões recebidas são processadas diariamente e as estatísticas geradas são disponibilizadas na página web do projeto[21]. As listas de pacotes originais submetidas não são publicadas a fim de preservar a privacidade dos usuários. No entanto, para a realização deste trabalho o acesso foi permitido mediante supervisão de desenvolvedores oficiais do projeto. Os dados "crus" (antes do processamento de estatísticas) são essenciais para a realização de estratégias de recomendação colaborativas por preservarem os relacionamentos usuários-itens, e já foram utilizados anteriormente para o mesmo fim (seções 4.1 e 4.3).

5.3.4 Ultimate Debian Database (UDD)

O UDD[22] é uma iniciativa recente do time de qualidade criada com o intuito de reunir informações de diversos aspectos do Debian numa base de dados única [**?**].

O fluxo de dados do UDD é apresentado na figura 5.4. Existe um coletor para cada fonte de dados (p. ex. o BTS, sistema de acompanhamento de *bugs*[23]) que implementa uma interface comum e esconde a complexidade e especificidade de acessar cada um dessas fontes. Existe um processo central no UDD que invoca os coletores periodicamente, provocando a inserção dos dados na base única.

A principal motivação para o desenvolvimento do UDD foi a de auxiliar a equipe de qualidade em suas atividades, além de facilitar a colaboração com distribuições derivadas do Debian. Apesar de ser possível, dificilmente usuários consultariam esta base para tomar decisões acerca de que pacotes utilizar, visto que os dados armazenados no UDD geralmente são acessíveis por outros meios. No entanto, para fins de mineração de dados ou para o desenvolvimento de um recomendador automático, a possibilidade de acesso a dados de tamanha heterogeneidade numa fonte de dados única é uma grande benefício.

5.3.5 Debian Data Export (DDE)

Informações sobre o Debian e seus pacotes são publicadas em diversos tipos de formato, por vezes específicos e obscuros, e nem sempre de fácil localização. O DDE[24] foi criado para facilitar a publicação e aquisição destas informações e a descoberta de quais informações estão disponíveis sem a necessidade de se preocupar com formatos de dados, protocolos e controle de acesso.

[20] http://kitenet.net/~joey/blog/entry/the_popcon_problem/
[21] http://popcon.debian.org
[22] http://udd.debian.org
[23] http://bugs.debian.org
[24] http://wiki.debian.org/DDE

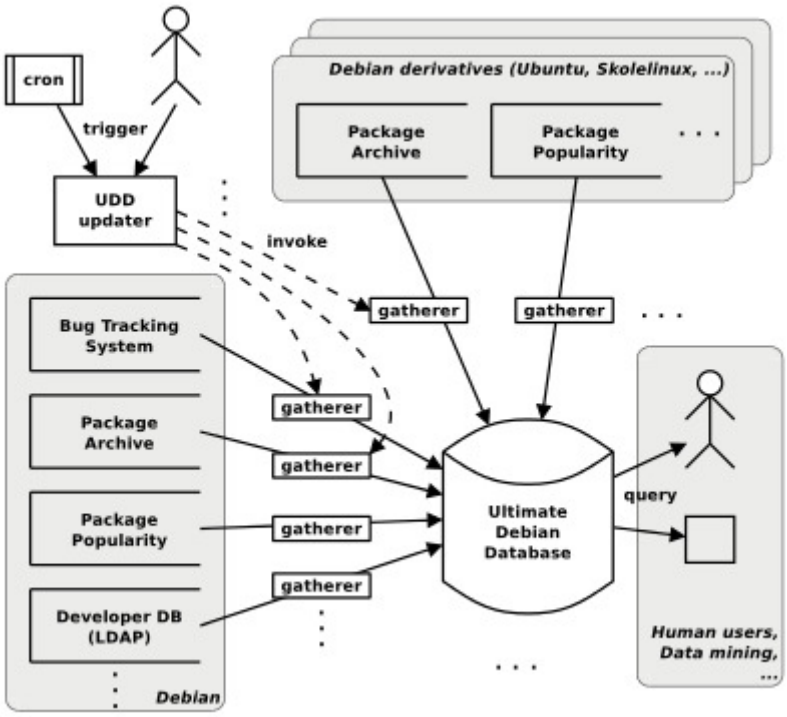

Figura 5.4: Fluxo de dados no UDD [?]

DDE e UDD são serviços complementares. Enquanto o UDD tem como meta criar um ponto central de acesso aos dados, o DDE provê o acesso de forma simples e padronizada a determinados conjuntos de dados. Os dois serviços se completam: quanto mais dados são coletados pelo UDD, mais informações são disponibilizadas via DDE, que atua como uma interface simplificada para as consultas mais úteis e populares do UDD.

Além do UDD, existem plugins para o DDE que importam dados do *apt-file*[25], *apt-xapian-index* e BTS. Atualmente os dados podem ser exportados na notação de objetos JavaScript (JSON), formato de serialização YAML, valores separados por vírgula (CSV) ou objetos Python serializados (*pickled objects*).

Os dados são representados numa grande árvore. Pode-se escolher um nó nesta árvore por meio de sua URL para adquirir todos os dados contidos na sub-árvore que contém este nó como raiz. A figura 5.5 apresenta o resultado de uma consulta ao plugin BTS pelo pacote *gnome-subtitles*, por meio da url `http://dde.debian.net/dde/q/bts/bypackage/gnome-subtitles`.

5.3.6 Screenshots

O *Screenshots*[26] é um repositório público de capturas de tela de aplicativos da distribuição Debian GNU/Linux e derivadas. O serviço foi criado para permitir que os usuários conheçam a aparência dos programas antes de instalá-los em seu ambiente de trabalho. A submissão de imagens é aberta, podendo ser realizada por qualquer usuário, no entanto uma revisão humana é realizada antes que as imagens se tornem públicas.

[25] Busca por arquivos no conteúdo de pacotes Debian
[26] `http://screenshots.debian.net/`

```
Show bugs for package gnome-subtitles

Value:

{596845: {'affects': u'',
          'archived': False,
          'blockedby': u'',
          'blocks': u'',
          'bug_num': 596845,
          'date': 1284474065,
          'done': False,
          'fixed': None,
          'fixed_date': [],
          'fixed_versions': [],
          'forwarded': u'',
          'found': None,
          'found_date': [],
          'found_versions': [u'gnome-subtitles/0.7.2-1lenny1'],
          'location': u'db-h',
          'log_modified': 1284474068,
          'mergedwith': '',
          'msgid': u'<2fltylsif06.fsf@login1.uio.no>',
          'originator': u'Petter Reinholdtsen ',
          'owner': u'',
          'package': u'gnome-subtitles',
          'pending': u'pending',
          'severity': u'normal',
          'source': u'gnome-subtitles',
          'subject': u'gnome-subtitles: Unable to find list of supported video\
                      formats',
          'summary': u'',
          'tags': [],
          'unarchived': False}}

1 children:

596845 - Details of bug #596845

Go up one level

```

Figura 5.5: Exemplo consulta ao plugin BTS

5.4 Decisões de projeto

Técnicas de busca foram selecionadas como base de implementação do *AppRecommender* devido ao desempenho das soluções com a construção prévia de índices. O desenvolvimento dos componentes de indexação e busca baseou-se na biblioteca *Xapian*.

Ao ser instanciado, o recomendador recebe como parâmetros dois índices – neste contexto denominados *repositórios* – cujo conteúdo está descrito a seguir.

(a) **Repositório de itens:** armazena todos os itens disponíveis para recomendação juntamente com seus meta-dados. Fisicamente, é um índice *xapian* onde cada pacote é representado por um documento cujos termos são os meta-dados relacionados ao pacote. O *Apt-xapian-index* pode ser utilizado como repositório de itens, porém geralmente se utiliza um índice filtrado, dado que nem todos os pacotes existentes deveriam ser considerados pelo recomendador.

(b) **Repositório de usuários:** armazena escolhas anteriores de usuários, representadas por relações to tipo *usuário* → *conjunto de itens*. Fisicamente, é um índice *xapian* onde cada sistema (usuário do recomendador) é representado por um documento cujos termos são pacotes presentes em sua submissão ao *Popcon*. Além dos nomes, são indexadas

também as *tags* de cada pacote, permitindo assim uma caracterização de conteúdo de cada documento do índice.

Além dos índices *Xapian*, a solução coleta informações do UDD por meio do DDE, utiliza captura de telas do *Screenshots* e pode coletar também informações do APT local. A figura 5.6 apresenta o fluxo de dados da solução proposta.

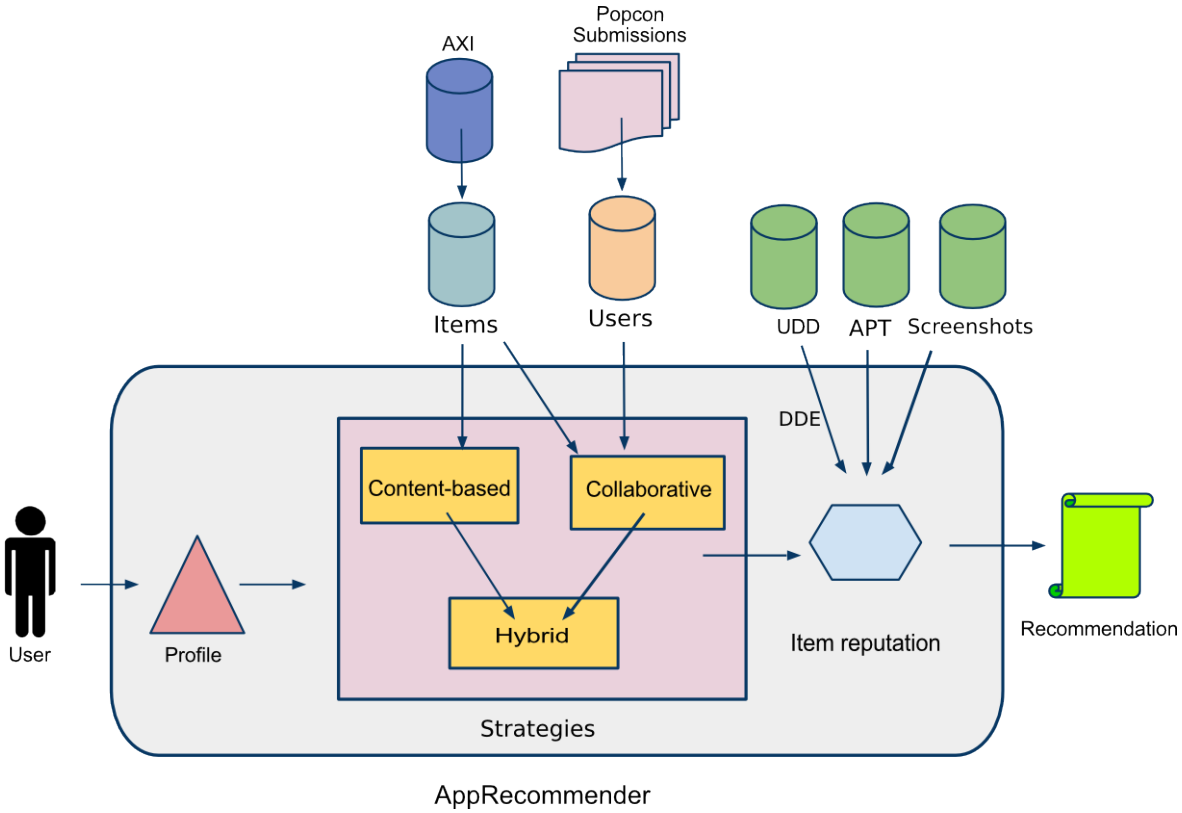

Figura 5.6: Fluxo de dados no *AppRecommender*

Foram realizados experimentos preliminares com algoritmos para mineração de regras de associação com base no *Apriori*. No entanto, devido a alta complexidade computacional destes algoritmos, seria necessário um maior planejamento e estudo de técnicas mais avançadas para o desenvolvimento de soluções computacionalmente executável. Em virtude de esta estratégia já ter sido abordada anteriormente num trabalho correlato (seção 4.3), buscou-se o reaproveitamento das regras já produzidas de sorte a integrá-las no *AppRecommender*. O autor demonstrou interesse neste sentido e pretende atualizar seu código, que apresenta-se inativo desde 2007

Houve também uma tentativa de pré-processamento da base de dados do *Popcon* através da técnica de agrupamento *K-medoides*. Contudo, a solução também demandaria um aprofundamento nas técnicas para que fosse capaz de lidar com a base de dados completa do *Popcon*. Diante da limitação de tempo para conclusão deste trabalho, optou-se focar apenas em técnicas de busca, deixando estes esforços como base para trabalhos futuros.

5.4.1 Seleção de atributos

A seguir são apresentadas peculiaridades do domínio de aplicativos que foram consideradas na modelagem dos procedimentos de seleção de atributos do *AppRecommender*. Acredita-se que tal pré-processamento dos dados de entrada proporciona ganhos na eficiência do recomendador e qualidade das sugestões produzidas, na medida em que diminui o montante de dados a ser

considerado e o ruído nos dados. A popular expressão "*Garbage in, garbage out*" pode ser utilizada neste contexto: se o sistema recebe "lixo" como entrada, independente da corretude do algoritmo, a saída provavelmente será inútil.

Pacotes *versus* aplicativos

De acordo com o conceito de empacotamento apresentado no capítulo 2, um pacote instalado num sistema GNU/Linux pode ou não representar um aplicativo. Existem pacotes no repositório que consistem apenas em documentação, dados complementares, ou ainda, bibliotecas que são requisitos para outros aplicativos.

Esta característica do pacote pode ser identificada por meio da faceta `XTrole` do *debtags*, que pode assumir valores como *program*, *data*, *shared-lib*, *examples*, entre outros. Para recomendador de aplicativos, apenas os pacotes marcados com a *tag* `XTrole::program` são considerados como itens válidos.

Mapeamento das necessidades de um usuário

No caso de uso típico do recomendador de aplicativos, determinado usuário entrega ao sistema o conjunto de aplicativos que ele possui, e a função do recomendador é sugerir, dentre os que ele não conhece, os que melhor atendem a suas necessidades. Todavia, nem sempre aquilo que o usuário busca pode ser mapeado diretamente na instalação de pacotes específicos. Necessidades do usuário dizem respeito às funcionalidades que os aplicativos oferecem. Visto que aplicativos diferentes podem executar funções similares, o mais apropriado seria representar a necessidade ou desejo do usuário como um conjunto de funcionalidades, em vez de um conjunto de aplicativos ou pacotes.

Atualmente o perfil do usuário pode ser composto por palavras extraídas das descrições de pacotes ou *debtags*. O vocabulário de *debtags* é composto por 621 *tags*, porém nem todas são úteis para fins de recomendação. Por exemplo, a linguagem em que o aplicativo foi implementada é irrelevante para a caracterização de suas funcionalidades. Portanto, todas as *tags* da faceta *implemented-in* foram desconsideradas. Após uma análise manual, o conjunto de tags válidas para o *AppRecommender* foi reduzido para 276.

Aplicativos básicos

Para um sistema operacional ou distribuição GNU/Linux, quaisquer que sejam, necessariamente haverá um conjunto de componentes que fazem parte da instalação padrão, previamente selecionados pela equipe de desenvolvimento. Considerando que os usuários do recomendador utilizam um sistema funcional, existem dois casos a considerar: (1) todo o conjunto de componentes da instalação padrão está instalado no sistema e (2) alguns componentes não estão presentes porque foram propositalmente removidos pelo usuário. Em ambos os casos a recomendação de tais pacotes certamente não interessaria ao usuário, portanto todos eles podem ser desconsiderados sem prejuízo para a recomendação.

Aplicativos instalados automaticamente

Uma característica peculiar da recomendação neste contexto é que, diferentemente de outros domínios nos quais os itens não se relacionam entre si, um aplicativo pode requisitar a presença (ou ausência) de outros aplicativos no sistema para que funcione adequadamente. Portanto, muitos programas são instalados automaticamente por serem dependências de outros, e não em decorrência de uma ação do administrador do sistema. A relevância de tais aplicativos para a composição de um perfil de usuário pode ser considerada de forma diferenciada dos outros, ou simplesmente desconsiderada.

No caso em que o recomendador é executado localmente (o cliente é o próprio sistema no qual ele é executado), têm-se acesso à base local do *APT* que guarda a informação de quais pacotes foram instalados automaticamente. Sendo assim, a seleção de atributos pode considerar apenas os que foram instalados voluntariamente.

No entanto, para o caso em que o recomendador recebe apenas a lista de pacotes instalados sem informação adicional sobre auto-instalação, uma outra estratégia é adotada. Neste cenário, a seleção de atributos desconsidera todos os pacotes que fazem parte da lista de dependências de qualquer outro pacote do perfil original. O resultado é uma lista de pacotes minimal que representa aquele sistema. Apesar de ligeiramente diferente do anterior, esta foi a abordagem que produziu resultados mais próximos da eliminação de pacotes instalados automaticamente quando não se tem acesso à base do *APT*.

Frequência de uso dos aplicativos

As informações temporais coletadas pelo *Popcon* (descritas na seção 5.3.3) podem ser utilizadas para atribuição de pesos diferenciados para pacotes do perfil, de acordo com a frequência de utilização dos mesmos. Atualmente atribui-se peso 3 para pacotes utilizados pela última vez a mais de 30 dias, peso 8 para os recentemente instalados e 10 para os recentemente utilizados. Outra seleção razoável é a filtragem apenas pelos aplicativos utilizados no último mês, visto que este período representa a atividade recente do usuário, e geralmente há grandes chances de seus interesses ainda serem os mesmos.

5.4.2 Perfis demográficos

Uma característica peculiar do *AppRecommender* é que o cliente deste recomendador é um sistema GNU/Linux com caráter multi-usuário. Seu uso é comumente compartilhado por diversas pessoas, possivelmente apresentando interesses conflitantes. O questionamento ao usuário sobre suas áreas de interesse pode não representar de fato os "interesses" do sistema cliente. Optou-se então por inferir automaticamente perfis básicos a partir da lista de aplicativos recebida como entrada, em vez de expressamente consultar o usuário.

A seguir são descritos os perfis utilizados na implementação atual e o comportamento do *AppRecommender* diante de cada um deles.

Desktop

Em conformidade com as decisões no escopo do *AppStream*, para os casos que o cliente do recomendador é um sistema utilizado como estação de trabalho, decidiu-se que a recomendação deve conter prioritariamente os pacotes que representam aplicativos *desktop*. Definiu-se então o filtro "desktop" para a identificação dos pacotes que possuem um arquivo *.desktop* em seu conteúdo (padrão estabelecido pelo FreeDesktop.org).

Testes realizados com sistemas minimais com interface gráfica, Debian e Ubuntu, apresentaram o perfil de *desktop* com 16 pacotes. Portanto, assumiu-se a caracterização deste perfil para os casos em que o sistema possui ao menos 10 aplicativos *desktop* instalados, além do sistema de janelas *X*[27].

Desktop KDE/GNOME

Sistemas gerenciadores de janelas evoluíram a tal ponto que atualmente alguns constituem ambiente *desktop* completos, com uma série de aplicativos desenvolvidos especialmente para tais ambientes, como é o caso do *KDE*[28] e *GNOME*[29]. Devido à popularidade de ambos

[27] *X Window System*, implementação de interface gráfica em sistemas GNU/Linux
[28] http://www.kde.org/
[29] http://www.gnome.org/

os projetos, alguns programas são desenvolvidos em versões específicas para um ou outro – baseados na biblioteca *Qt*[30] ou *GTK+*[31] respectivamente.

No caso em que o usuário do recomendador possui apenas um dos dois ambientes instalados, deve-se priorizar a sugestão do pacote específico para aquele gerenciador. O *AppRecommender* considera como sinônimos pacotes distintos que são versões do mesmo aplicativo, variando apenas a biblioteca gráfica na qual é baseado. Por exemplo, suponha que o pacote *autokey-gtk* faça parte da recomendação para determinado usuário. Caso apenas o *KDE* esteja instalado naquele sistema, o recomendador irá alterar a sugestão para conter o pacote *autokey-qt* em detrimento da versão para *GTK+* que fazia parte da recomendação original.

Servidor

Apesar de conceitualmente todo sistema GNU/Linux ser um servidor, esta foi a nomenclatura adotada para caracterizar sistemas cuja principal função é prover serviços, aparentemente não sendo utilizados como estações de trabalho. Neste caso, o filtro "program" é aplicado, considerando como pacotes válidos para recomendação todos aqueles marcados com a *tag* "`XTrole::program`".

Arquitetura

Os pacotes binários do Debian são automaticamente compilados para as arquiteturas de *hardware* indicadas pelo mantenedor do pacote. Se o pacote não está disponível para determinada arquitetura, muito provavelmente ele não tem utilidade para aquela categoria de *hardware*, ou simplesmente não foi portado adequadamente para ser executado naquela arquitetura. Nestes casos, o pacote deve ser desconsiderado de possíveis recomendações.

A arquitetura do cliente é um dado coletado pelo *Popcon*, indicado na primeira linha do arquivo de submissão (figura 5.3). Se o *AppRecommender* recebe este arquivo como entrada, esta informação será considerada e apenas pacotes disponíveis para a referida arquitetura serão sugeridos. Se o formato da entrada é uma lista de pacotes simples, esta informação pode ainda ser coletada pela interface do recomendador. Caso contrário, todos os aplicativos são passíveis de recomendação, sem considerar a arquitetura de *hardware*.

5.4.3 Bugs e popularidade

A quantidade de *bugs* e popularidade do pacote foram fatores inicialmente considerados para compor a reputação dos itens, resultando numa implementação de recomendação em cascata (3.5.6). Esta estratégia foi anulada por ser considerada inadequada.

Houve uma discussão recente entre desenvolvedores sobre a validade do uso dos dados do *Popcon* para fins de comparação entre pacotes. Joey Hess defende que a popularidade não deveria ser utilizada como uma medida de competição entre pacotes porque pacotes de naturezas distintas não são comparáveis[32]. Por exemplo, é um engano comparar um pacote popular como o *gnome-terminal* com um de baixa popularidade como *udhcpc*. O primeiro é instalado por padrão nos sistemas *desktop*, apesar de muitos usuários de *desktop* não fazerem uso do aplicativo. Por outro lado, o *udhcpc* é instalado apenas em sistemas embarcados e mesmo que seja absurdamente popular não aparecerá como popular nas estatísticas do *Popcon* visto que este tipo de sistema geralmente não participa do concurso de popularidade.

Tratando-se de *bugs*, pensava-se em recomendar prioritariamente pacotes com poucos *bugs*, assumindo que estes recebiam atenção constante de seu mantenedor. Contudo notou-se que tal critério seria falho, uma vez que *bugs* abertos podem também representar atividade no

[30] http://qt.nokia.com/
[31] http://www.gtk.org/
[32] http://kitenet.net/~joey/blog/entry/the_popcon_problem/

desenvolvimento e popularidade, tese corroborada por um estudo recente de [**?**]. Optou-se então por não utilizar esta informação como base para composição de reputação do item.

5.4.4 Privacidade dos usuários

O tópico privacidade no desenvolvimento do *AppRecommender* foi cuidadosamente considerado. Dado que os usuários do recomendador não são pessoas e sim sistemas GNU/Linux, uma quebra de privacidade neste contexto pode significar o comprometimento da segurança destes sistemas. Por exemplo, se a partir de um conjunto de ferramentas sabidamente instalados em um sistema pode-se deduzir que um outro conjunto de ferramentas também está presente, vulnerabilidades das ferramentas descobertas podem ser exploradas num ataque.

Aplicativos com poucas instalações reportadas pelo *Popcon* oferecem um risco de segurança neste aspecto. Considere um pacote *a* com apenas uma instalação reportada. Há grandes chances de uma consulta ao recomendador pelo aplicativo *a* resultar numa lista de pacotes instalados na máquina do mantenedor do pacote. Outra vulnerabilidade é que o *Popcon* exibe em suas estatísticas pacotes não-oficiais, muitos gerados pelo próprio usuário, inclusive revelando informações capazes de identificá-lo, como por exemplo o pacote "mec-dp-joao.da.cruz.e.sousa". Para evitar este tipo de vazamento de informações o *AppRecommender* desconsidera pacotes considerados *low-popcon*, ou seja, com menos de 20 instalações.

Vulnerabilidades referentes a mudanças nas recomendações ao longo do tempo, como mencionado na seção 3.8 não afetam este trabalho em seu estágio atual, dado que os dados do *Popcon* foram obtidos uma única vez para construção dos índices. No caso de um esquema de atualização desses dados, testes de vulnerabilidade deveriam ser implementados antes de o sistema ser disponibilizado ao público.

5.5 Estratégias de recomendação

As diferentes estratégias de recomendação implementadas no *AppRecommender* são apresentadas a seguir e sumarizadas na tabela 5.2.

5.5.1 Baseadas em conteúdo

O ponto chave das estratégias baseadas em conteúdo é a capacidade de percepção das características dos itens. Sua aplicação é bastante dificultada nos casos em que a representação de itens por conteúdo não pode ser realizada de forma automática.

O processo de recomendação pode ser sumarizado da seguinte maneira: a partir da lista de aplicativos do usuário, é composto um *perfil* em termos de conteúdo; é realizada então uma consulta no repositório de itens pelo conteúdo característico daquele usuário e os pacotes indicados como mais relevantes são recomendados (cada pacote é representado por um documento no repositório).

Estratégias baseadas em conteúdo são parametrizáveis pelo tamanho do perfil do usuário. Quanto maior a quantidade de termos considerados no perfil, mais diversificado será o resultado da busca. Perfis menores tendem a ser mais especializados, porém, corre-se o risco de não representarem suficientemente bem as características de um usuário.

As variações implementadas para a abordagem baseada em conteúdo diferem entre si em dois pontos: o tipo de conteúdo considerado e o método de composição do perfil.

Tipo de conteúdo

No âmbito deste trabalho itens são aplicativos, que fisicamente são representados por pacotes. Por princípio, quaisquer informações relacionadas a pacotes poderiam ser utilizada como característica. Considerando as questões sobre as necessidades do usuário abordadas na seção

5.4.1, a implementação atual faz uso das *tags* e descrições dos pacotes para extração de atributos. As abordagens implementadas são apresentadas a seguir.

(a) *Tags*: restringe-se a termos com prefixo `XT`, representa o conjunto de *tags* mais relevantes para determinado conjunto de pacotes.

(b) *Descrição*: restringe-se a termos com prefixo `Z` ou em texto livre, além de descartar a lista de palavras comuns *stopwords*), retorna os termos mais frequentes em descrições de seus pacotes.

(c) *Misto*: composto por *tags* e termos de descrição, sem limitação na quantidade de cada um, os que tiverem maior peso são retornados.

(d) *Meio-a-meio*: resultado da combinação dos perfis de *tags* e termos de descrição de pacotes, limitando-se a um número igual de cada tipo.

Método de composição do perfil

Foram utilizados dois métodos de busca básicos para composição do perfil de um usuário, detalhados a seguir.

(a) *Expansão de query*: é feita uma busca no repositório de itens pela lista de pacotes do usuário; cada documento retornado é marcado como relevante, e pede-se ao *Xapian* que execute uma expansão de query (`ESet`), obtendo termos representativos para aquele conjunto de documentos relevantes.

(b) *tf-idf sub-linear*: é feita uma busca no repositório de itens pela lista de pacotes do usuário; percorre-se a lista de documentos retornados acumulando todos os termos num único documento; calcula-se a pontuação *tf-idf* sub linear para cada termo deste documento, retornando os que obtiverem maior peso.

5.5.2 Colaborativas

Para esta categoria de estratégias, o perfil do usuário é composto pelos próprios nomes de pacotes instalados em seu sistema. As recomendações são baseadas no que outros usuários com interesses semelhantes aos dele têm instalado em seus sistemas e ele não tem.

O processo de recomendação pode ser sumarizado da seguinte maneira: a vizinhança é composta a partir de uma busca no repositório de usuários pela lista de pacotes do cliente da recomendação; a partir de análise das listas de pacotes dos vizinhos a recomendação é produzida.

A maneira como a análise da vizinha é realizada produz recomendações diferenciadas. A seguir são descritos os três métodos implementados.

Método de colaboração

(a) *Expansão de query*: cada documento da vizinhança é marcado como relevante (cada vizinho é representado por um documento do repositório de usuários); pede-se ao *Xapian* que execute uma de expansão de query (`ESet`), obtendo termos representativos para aquele conjunto de documentos relevantes, que neste caso são nomes de pacotes.

(b) *tf-idf sub-linear*: percorre-se a lista de vizinhos acumulando todos os nomes de pacote num único documento; calcula-se a pontuação *tf-idf* sub-linear para cada termo deste documento, retornando os que obtiverem maior peso.

(c) *tf-idf "plus"*: comportamento similar ao *tf-idf* sub-linear, porém a função de peso considera a distância de cada vizinho ao cliente da recomendação no momento de agregar todos os pacotes num só documento; quanto mais próximo do usuário o vizinho estiver, mais influência seus pacotes terão no cálculo geral.

5.5.3 Híbridas

A seguir são detalhadas as estratégias de recomendação híbridas disponíveis no *AppRecommender*, de acordo com a taxonomia apresentada na seção 3.5.6.

(a) *Acréscimo de atributo*: recomendações produzidas de forma colaborativa com base em conteúdo. Após a composição da vizinhança, os termos de conteúdo encontrados nos vizinhos são ordenados por relevância (apenas *tags* são indexadas juntamente com pacotes no repositório de usuários) ; as *tags* mais relevantes compõem um perfil por conteúdo por (expansão de *query* ou *tf-idf* sub-linear), que é então submetido à estratégia baseada em conteúdo.

(b) *Revezamento*: de acordo com a inferência de dados demográficos, se o sistema for caracterizado como *desktop*, este filtro é aplicado a todos os repositórios (de itens e usuários) e apenas aplicativos de *desktop* podem ser recomendados. Se o perfil for KDE ou GNOME, a lista de sinônimos é consultada e a recomendação é atualizada caso necessário. Caso contrário, a recomendação acontece normalmente, com os repositórios de aplicativos integral;

(c) *Combinação*: apresentação em conjunto dos resultados de múltiplos recomendadores básicos.

Estratégia	Classificação	Método	Conteúdo
cb	Baseada em conteúdo	*tf-idf*	misto
cbt			*tags*
cbd			descrição
cbh			meio-a-meio
cb_eset		ESet	misto
cbt_eset			tags
cbd_eset			descrição
cbh_eset			meio-a-meio
knn	Colaborativa	*tf-idf*	–
knn_plus		*tf-idf* "plus"	–
knn_eset		ESet	–
knnco	Híbrida	*tf-idf*	*tags*
knnco_eset		ESet	*tags*

Tabela 5.2: Descrição das estratégias de recomendação implementadas

5.6 Protótipo do AppRecommender

O protótipo do *AppRecommender* pode ser obtido a partir do repositório do projeto no github[33]. A ferramenta deve ser executada em um terminal, por meio do *script apprec.py*, como indicado na figura 5.7. Uma interface web para o recomendador foi desenvolvida utilizando o módulo *python webpy* e *layout* inspirado no *Screenshots* (seção 5.3.6).

[33] http://github.com/tassia/AppRecommender

```
$ git clone http://github.com/tassia/AppRecommender
$ cd AppRecommender/src/bin
$ ./apprec.py -s knn
INFO: Set up logger
INFO: Basic config
INFO: Loading recommender filters
INFO: Setting recommender strategy to 'knn'
INFO: Computation started at 2011-09-03 07:21:09.264660
INFO: Recommending applications for user local-2011-09-03 07:21:09.473482

 0: yate-qt4        - YATE-based universal telephony client
 1: gtkwhiteboard   - GTK+ Wiimote Whiteboard
 2: gravitywars     - clone of Gravity Force
 3: conduit         - synchronization tool for GNOME
 4: jokosher        - simple and easy to use audio multi-tracker
 5: xjump           - A jumping game for X
 6: gweled          - A "Diamond Mine" puzzle game
 7: gmpc            - Gnome Music Player Client (graphical interface to MPD)
 8: homebank        - Manage your personal accounts at home
 9: aranym          - Atari Running on Any Machine
10: gameconqueror   - GUI for scanmem, a game hacking tool
11: gwc             - Audio file denoiser
12: gquilt          - graphical wrapper for quilt and/or mercurial queues
13: xiphos          - environment for Bible reading, study, and research
14: hatari          - Emulator for the Atari ST, STE, TT, and Falcon computers
15: tecnoballz      - breaking block game ported from the Amiga platform
16: dates           - a calendar optimised for embedded devices
17: sylpheed        - Light weight e-mail client with GTK+
18: klavaro         - Flexible touch typing tutor
19: fofix           - rhythm game in the style of Rock Band(tm) and Guitar Hero(tm)

INFO: Computation completed at 2011-09-03 07:21:42.529051
INFO: Time elapsed: 10 seconds.
```

Figura 5.7: Execução do recomendador para o sistema local

Os parâmetros para instanciação do recomendador podem ser definidos por meio de opções na linha de comando ou por um arquivo de configuração. A tabela 5.3 traz a descrição dos parâmetros básicos.

Parâmetro	Descrição
Repositório de itens	Índice *Xapian* contendo informações sobre os aplicativos
Repositório de usuários	Índice *Xapian* que armazena escolhas anteriores de usuários
Estratégia	Método para composição da recomendação
Tamanho do perfil	Quantidade de termos (palavras, tags ou ambos) que caracterizam o usuário perante o recomendador
Tamanho da vizinhança	Quantidade de usuários mais próximos a considerar para estratégias colaborativas

Tabela 5.3: Descrição dos parâmetros ajustáveis do *AppRecommender*

5.6.1 Codificação

O desenvolvimento foi majoritariamente realizado na linguagem de programação *Python*[34], principalmente pela facilidade de integração com outras ferramentas do Debian também desenvolvidas nesta linguagem.

O código-fonte está licenciado pela GNU GPL e disponível em um repositório público[35].

[34] http://www.python.org/
[35] http://github.com/tassia/AppRecommender

O desenvolvimento obedeceu ao guia de estilo para código em *python*[36], fez uso de testes automatizados e padrões de projeto. A documentação é automaticamente gerada pelo *Doxygen*[37] e disponibilizada no repositório git[38].

[36] http://www.python.org/dev/peps/pep-0008/
[37] http://www.stack.nl/~dimitri/doxygen/
[38] http://www.ime.usp.br/~tassia/doc/html/index.html

Capítulo 6

Validação da proposta

Os experimentos que estão sendo realizados com o intuito de validar a ferramenta desenvolvida são descritos neste capítulo. Foram planejadas duas fases de testes: a primeira de caráter exploratório, com a variação de ajustes dos modelos, em busca da configuração mais adequada ao domínio de aplicação e conjunto de dados estudado; a segunda etapa é uma consulta pública, por meio da qual usuários reais podem avaliar a utilidade das recomendações produzidas por diferentes estratégias do *AppRecommender*.

Os resultados dos experimentos, o conjunto completo de gráficos gerados e outras atualizações podem ser acompanhados no *website da pesquisa* [1].

6.1 Experimentos *offline*

Considerando cada configuração distinta do *AppRecommender* como um modelo preditivo, a análise de desempenho foi pautada pelas métricas precisão, medida $F_{0.5}$, cobertura e curva ROC. O uso das métrica é justificado na descrição de cada experimento.

Os usuários dos experimentos são criados a partir de submissões do *Popcon* selecionadas aleatoriamente para compor a amostra de testes. Notamos que o tamanho da lista de aplicativos instalados utilizada para compor o perfil primário dos usuários causava uma certa variação nos resultados, então para cada experimento consideramos apenas usuários cujo tamanho de perfil estivesse numa determinada faixa. A distribuição dos usuários na base do *Popcon* com relação à quantidade de aplicativos no perfil está representada na figura 6.1.

A eficácia de um modelo é mensurada a partir de validação cruzada com cada usuário da amostra. Em cada rodada é extraída uma porção de aplicativos do seu perfil para compor o conjunto de testes. Um usuário artificial é criado com a lista de aplicativos restantes, que é submetida ao recomendador. Considerando a porção de testes como o conjunto de itens relevantes (positivos reais) e a recomendação como o conjunto de itens supostamente relevantes (positivos preditos), uma matriz de contingência é criada e as métricas consideradas pertinentes são aplicadas. Ao final das iterações, as médias das estimativas são calculadas para sumarização dos resultados por usuário. Para avaliar os resultados na amostra, além da média consideramos o desvio padrão das medidas.

6.1.1 Avaliação por ação

Nesta etapa de experimentos variamos os parâmetros de configuração de cada modelo e analisamos como o desempenho do recomendador é afetado. Para as estratégias baseadas em conteúdo, o parâmetro investigado é o tamanho do perfil do usuário a ser considerado (*profile*

[1] http://recommender.debian.net/experiments

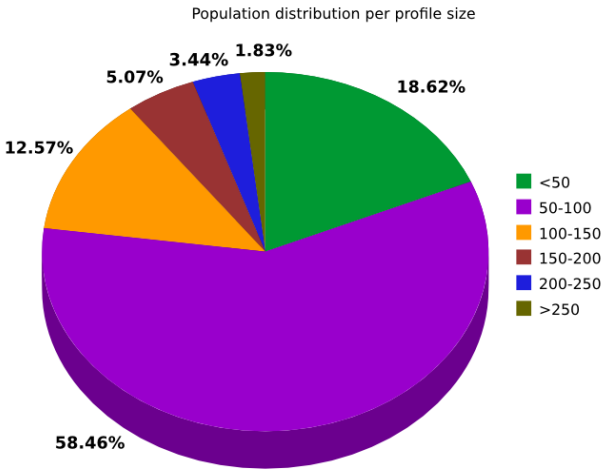

Figura 6.1: Distribuição de submissões do *Popcon* por tamanho de perfil

size); para estratégias colaborativas, o tamanho da vizinhança (*neighborhood size*); e para estratégias híbridas ambas as variações são estudadas.

A escolha das métricas para esta primeira etapa de testes foi guiada pela *ação* do recomendador em cada contexto de aplicação (ver seção 3.3).

No caso de uso típico de do *AppRecommender*, em que o usuário explicitamente requisita sugestões de aplicativos, consideramos que a oferta de 10 sugestões é razoável. A apresentação de muitos itens pode comprometer a usabilidade da interface, causando o indesejável excesso de opções para o usuário. Neste cenário deseja-se maximizar a taxa de acertos do recomendador com relação aos 10 itens recomendados, que pode ser avaliada pela *precisão* ($\frac{VP}{VP+FP}$). Outra característica analisada é a *cobertura* representada pela proporção de itens do repositório que aparecem em alguma recomendação. Cobertura baixa revela recomendadores viciados numa faixa restrita do repositório. Consideramos esta medida como um indicativo da capacidade de produzir recomendações variadas, com a ressalva de que ela depende diretamente da variabilidade dos dados de entrada dos testes.

Há situações, no entanto, em que o recomendador deve ser capaz de produzir uma maior quantidade de sugestões válidas. Por exemplo, o usuário pode solicitar mais recomendações do que o montante apresentado; ou ainda, no caso em que o recomendador é acoplado a um navegador de aplicativos que deve apresentar primeiramente as opções mais suscetíveis a aceitação. Neste cenário, além da cobertura, analisamos a variação da medida $F_{0.5}$ para recomendações de tamanho 100. Consideramos que para sugestões mais amplas a recuperação também é uma métrica importante. A medida F combina os dois conceitos, sendo que $F_{0.5}$ prioriza um pouco mais a precisão (ver seção 3.7.2).

Descrição dos testes

Os dados de entrada destes testes são a estratégia do recomendador e uma amostra de usuário. Neste momento utilizamos amostras de 50 usuários. Cada configuração da estratégia (diferentes tamanhos de perfil ou vizinhança) é testada para todos os usuários da amostra por meio de validação cruzada e os resultados são sumarizados em dois gráficos:

(a) *Limiar 10*: referente à recomendação de 10 aplicativos, com a representação de precisão média e cobertura, à medida que o tamanho do perfil ou vizinhança é variado;

(b) *Limiar 100*: referente à recomendação de 100 aplicativos, com a representação de $F_{0.5}$ médio e cobertura, à medida que os parâmetros são variados.

As figuras 6.2 e 6.3 apresentam os gráficos gerados para a estratégia `knn_eset`, aplicada à amostra de usuários com perfil entre 100 e 150 aplicativos.

Para cada ponto (x, y) das linhas de precisão ou $F_{0.5}$, a coordenada y é resultado da média dos valores obtidos para cada usuário da amostra, quando o recomendador é configurado com o parâmetro x. Por exemplo, o ponto $(20, 0.71)$ indica que a estratégia `knn_eset` com tamanho de vizinhança 20 produziu na média uma precisão de 71%. O desvio padrão da medida é indicado no gráfico pela linha vertical, neste caso com valor 0.19. Os detalhes de cada gráfico estão registrados em arquivo no formato da figura 6.4 e também disponibilizados no *site*, na visão detalhada na figura [2].

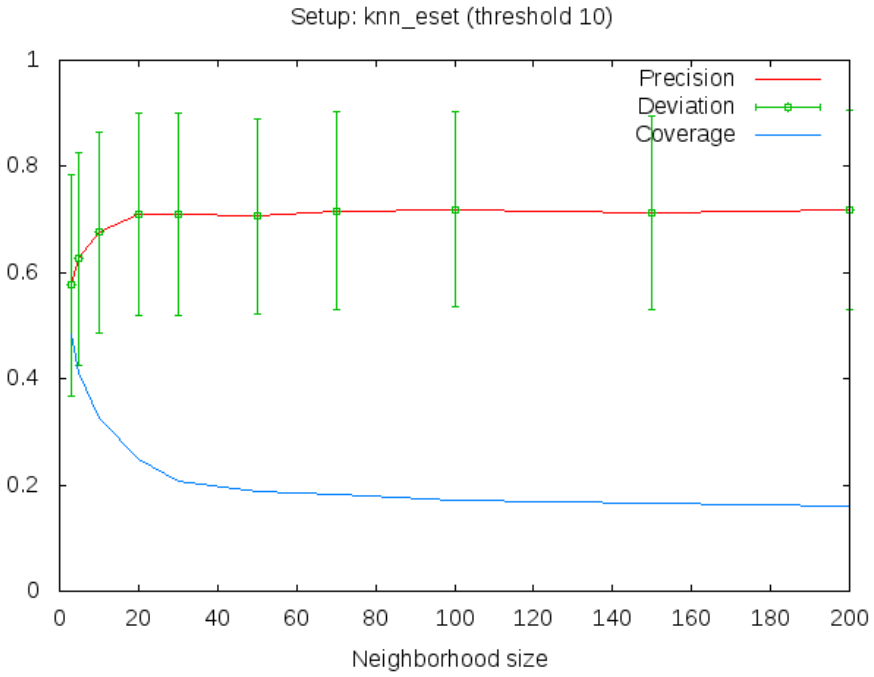

Figura 6.2: Aplicação de métricas para recomendação de 10 itens

Resultados

A cobertura é considerada métrica secundária na comparação de resultados. No entanto, nos casos em que a cobertura diminui à medida que a precisão ou $F_{0.5}$ aumenta, consideramos uma cobertura de 50% como o limite aceitável. Nos casos em que a cobertura é sempre inferior a 50%, consideramos o exemplo com maior valor de cobertura, mesmo que não represente a melhor acurácia [3].

As tabelas 6.1 e 6.2 apresentam os melhores resultados para a amostra de perfis entre 50 e 100 aplicativos, escolhida por ser a mais representativa da base do *Popcon* (58% dos usuários têm esse perfil). Para uma mesma configuração de recomendador, em geral não houve variação de comportamento entre as diferentes amostras testadas. No entanto, em termos absolutos, as amostras de usuário com perfil maior produziram melhores resultados.

A seguir estão relacionados alguns comportamentos recorrentes identificados na análise dos gráficos.

[2] http://recommender.debian.net/graphs/metrics/collaborative/sample-045-050_50/sample-045-050_50-knn_eset-10.html

[3] http://recommender.debian.net/experiments/cap6-results.html

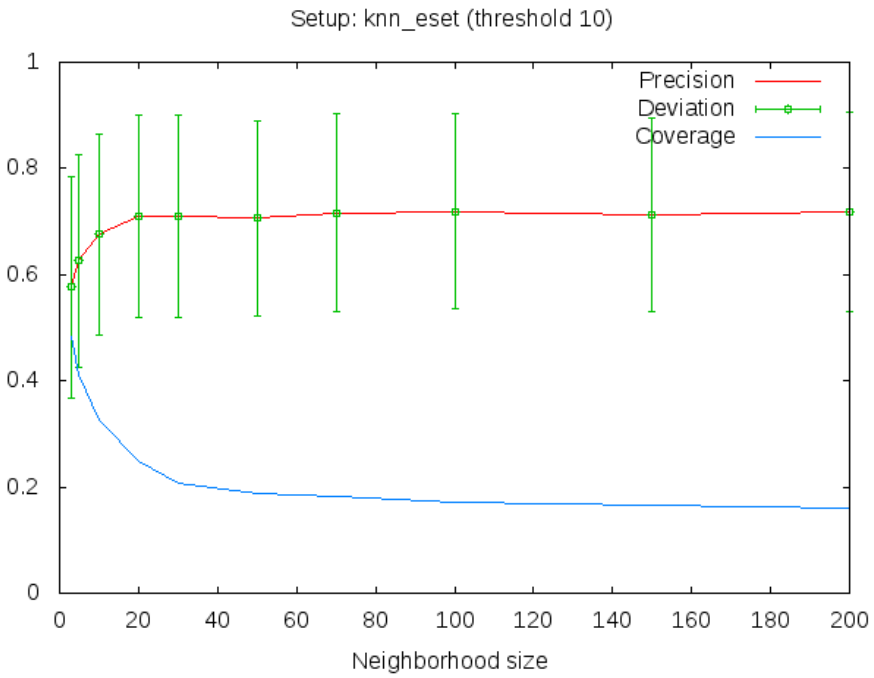

Figura 6.3: Aplicação de métricas para recomendação de 100 itens

```
# sample sample-045-050_50
# strategy knn_eset
# threshold 10
# iterations 20

# neighborhood  mean_p_10   dev_p_10    c_10

       3        0.5760      0.2088      0.4826
       5        0.6267      0.2001      0.4120
      10        0.6757      0.1897      0.3261
      20        0.7103      0.1915      0.2497
      30        0.7102      0.1897      0.2065
      50        0.7068      0.1839      0.1865
      70        0.7168      0.1856      0.1818
     100        0.7194      0.1837      0.1702
     150        0.7119      0.1826      0.1670
     200        0.7183      0.1886      0.1596
```

Figura 6.4: Registro de recomendação dos experimentos

(a) Tanto para estratégias baseadas em conteúdo quando para as híbridas, a precisão aumenta à medida que o tamanho do perfil do usuário aumenta. No entanto, dado que geralmente o aumento do perfil provoca queda na cobertura, nem sempre o maior valor de acurácia é aproveitável, visto que para melhor desempenho busca-se o equilíbrio entre estas duas medidas [4].

(b) Abordagens para composição de perfil baseadas puramente em *tags* (`cbt` e `cbt_eset`) resultam em cobertura muito baixa. Isto deve-se provavelmente ao fato de o conjunto de *tags* válidas ser limitado e de nem todos os pacotes do repositório possuírem *tags* associadas. O resultado é que as buscas no repositório aplicativos por estas consultas restritas não são capazes de retornar todos os itens disponíveis [5].

[4] http://recommender.debian.net/experiments/cap6-results-a.html
[5] http://recommender.debian.net/experiments/cap6-results-b.html

(c) O perfil das estratégias híbridas também é composto por *tags*, mas a cobertura atinge valores acima de 70%. Neste caso, a colaboração com outros usuários e contribui para a ocorrência de recomendações mais variadas [6].

(d) O desempenho das estratégias `cb` e `cbd` segue o mesmo padrão; o mesmo acontece para as estratégias `cb_eset` e `cbd_eset`, que apresentam resultados bastante similares. Notamos que, na prática, o perfil misto é praticamente dominado por termos de descrição dos aplicativos. Este tipo de perfil não impõe restrição alguma quanto à quantidade de *tags* e termos que o perfil final deve conter, e geralmente os termos livres ganham a disputa. O conjunto de termos de descrição de pacotes é ilimitado e por isso tem mais poder de caraterização dos itens, enquanto o de *tags* limita-se ao subconjunto de *debtags* que o *AppRecommender* considera como válidas. Sendo assim, é natural que não haja grandes variações entre as estratégias com perfil misto e composto exclusivamente por termos de descrição dos pacotes [7].

(e) Cobertura cai à medida que o tamanho do perfil aumenta para as estratégias `cb`, *cb_eset*, *cbd*, e *cbd_eset*. Estas são as abordagens de composição de perfil dominadas por termos livres. Perfis pequenos representam buscas especializadas e com alta variabilidade provocada pela processo de re-amostragem. Perfis grandes tendem a incluir grandes conjuntos de termos populares, que geralmente provocam recomendações dentro de um padrão [8].

(f) Em geral, estratégias baseadas em conteúdo com composição de perfil por *eset* conseguem bom desempenho com perfis menores do que suas correspondentes por *tf-idf* [9].

(g) Recomendação baseada em conteúdo para limiar 100 apresentou desempenho insatisfatório, com $F_{0.5}$ sempre abaixo de 0.2 e quase não houve variação de acordo com o tamanho do perfil do usuário [10].

(h) Para a maioria das estratégias colaborativas, tanto para 10 quanto para 100 recomendações, o melhor desempenho é alcançado com menor tamanho de vizinhança (consideramos 3 o menor tamanho) [11].

(i) A estratégia `knn_eset` para 10 recomendações apresentou um comportamento diferenciado das demais colaborativas. Neste caso houve aumento da precisão à medida que aumentamos o tamanho da vizinhança. No entanto, visto que a cobertura cai com o aumento da vizinhança, o melhor desempenho foi limitado ao tamanho 5 para vizinhança [12].

Para o limiar de 10 sugestões, a estratégia que se destacou nesta série de experimentos foi a colaborativa `knn_eset`, que de fato foi a única que apresentou precisão superior a 50%. As demais estratégias obtiveram desempenho similar no melhor caso, com exceção de `cbt` e `cbt_eset`, cuja cobertura comprometeu os resultados.

Considerando o limiar de 100 recomendações, as três com melhores resultados foram `knn`, `knn_plus` e `knnco`, que têm em comum a aplicação do *tf-idf*. As estratégias baseadas em conteúdo e a híbrida `knnco_eset` produziram resultados insatisfatórios.

O desvio padrão das medidas de acurácia foi relativamente alto, indicando que para alguns usuários os resultados foram muito bons e para outros muito ruins. Isto pode significar uma grande variância de características entre os usuários da amostra. Tal fato revela a necessidade

[6] http://recommender.debian.net/experiments/cap6-results-c.html
[7] http://recommender.debian.net/experiments/cap6-results-d.html
[8] http://recommender.debian.net/experiments/cap6-results-e.html
[9] http://recommender.debian.net/experiments/cap6-results-f.html
[10] http://recommender.debian.net/experiments/cap6-results-g.html
[11] http://recommender.debian.net/experiments/cap6-results-h.html
[12] http://recommender.debian.net/experiments/cap6-results-i.html

de um estudo mais aprofundado buscando identificar grupos de usuários com características comuns.

Estratégia	Perfil	Vizinhança	Precisão	Cobertura
cb	100	–	0.2163 ± 0.1383	0.5342
cbt	10	–	0.1818 ± 0.0737	0.2460
cbd	100	–	0.2145 ± 0.1312	0.5479
cbh	10	–	0.1817 ± 0.0939	0.5564
cb_eset	60	–	0.2306 ± 0.1282	0.5358
cbt_eset	10	–	0.2039 ± 0.0858	0.2529
cbd_eset	60	–	0.2384 ± 0.1420	0.5479
cbh_eset	10	–	0.2096 ± 0.0898	0.4800
knn	–	3	0.2338 ± 0.1701	0.7524
knn_plus	–	3	0.2086 ± 0.1766	0.7692
knn_eset	–	5	0.6451 ± 0.2070	0.5169
knnco	10	3	0.2233 ± 0.1721	0.7619
knnco_eset	10	100	0.2689 ± 0.0796	0.2329

Tabela 6.1: Melhores desempenhos para 10 sugestões

Estratégia	Perfil	Vizinhança	$F_{0.5}$	Cobertura
cb	240	–	0.0848 ± 0.0236	0.8488
cbt	100	–	0.1095 ± 0.0232	0.6017
cbd	240	–	0.0722 ± 0.0236	0.8109
cbh	40	–	0.1105 ± 0.0237	0.8793
cb_eset	10	–	0.0920 ± 0.0286	0.9557
cbt_eset	20	–	0.1099 ± 0.0245	0.6091
cbd_eset	60	–	0.0788 ± 0.0333	0.9515
cbh_eset	60	–	0.1211 ± 0.0254	0.8936
knn	–	3	0.5014 ± 0.1806	0.8841
knn_plus	–	3	0.5024 ± 0.1894	0.8836
knn_eset	–	3	0.4142 ± 0.1524	0.8820
knnco	240	3	0.5240 ± 0.1534	0.8246
knnco_eset	10	100	0.1188 ± 0.0196	0.6264

Tabela 6.2: Melhores desempenhos para 100 sugestões

6.1.2 Comparação entre modelos

Os experimentos apresentados até o momento nos permitem ter uma visão ampla da interferência da variação de parâmetro em cada modelo, mas a comparação entre os diversos modelos não é uma tarefa fácil de ser conduzida de maneira objetiva. Decidimos então realizar novos testes para analisar o comportamento dos modelos preditivos independentemente do tamanho da recomendação produzida, utilizando como suporte a plotagem de curvas ROC.

Descrição dos experimentos

Os dados de entrada desta série são novamente a estratégia do recomendador e uma amostra de usuário. Reduzimos as amostras para 20 usuários em virtude do alto poder de processamento demandado para a execução destes experimentos.

Considerando a quantidade de aplicativos recomendados como o ponto de corte de um recomendador, foram produzidas curvas ROC que variam este limiar e facilitam a percepção de quais modelos apresentam melhor poder preditivo.

Cada modelo é testado em cada ponto de corte para todos os usuários da amostra por meio de validação cruzada. As taxas de falso negativo (fpr) e verdadeiro positivo (tpr) são

registradas para produção da curva ROC. As diferentes curvas produzidas são condensadas a partir das médias de *fpr* e *tpr* para cada ponto de corte, conforme descrito em [**?**].

Uma característica das curvas produzidas é que, pelo fato de o *AppRecommender* se basear em técnicas de busca, a ordenação dos aplicativos por relevância não engloba todos os programas do repositório, mas apenas aqueles incluídos no resultado da busca. Portanto, para possibilitar a comparação entre os modelos pelo cálculo da área sob a curva, estendemos cada curva ROC com uma linha ligando seu ponto mais à direita ao ponto $(1, 1)$. Estamos assim considerando que após apresentar todos os pacotes retornados pela busca, o recomendador apresenta os pacotes restantes de forma aleatória, até que todos os itens tenham sido contemplados.

Produzimos dois gráficos por modelo, o primeiro com a representação da curva ROC média, que facilita a interpretação da curva nos moldes tradicionais. O segundo desenha os mesmos pontos médios, registrando também os desvios padrão para cada ponto, nas duas dimensões. As figuras 6.5 e 6.6 apresentam dois destes gráficos, produzidos para a estratégia de recomendação `cbt`. Os detalhes dos gráfico estão registrados em arquivo no formato da figura 6.7.

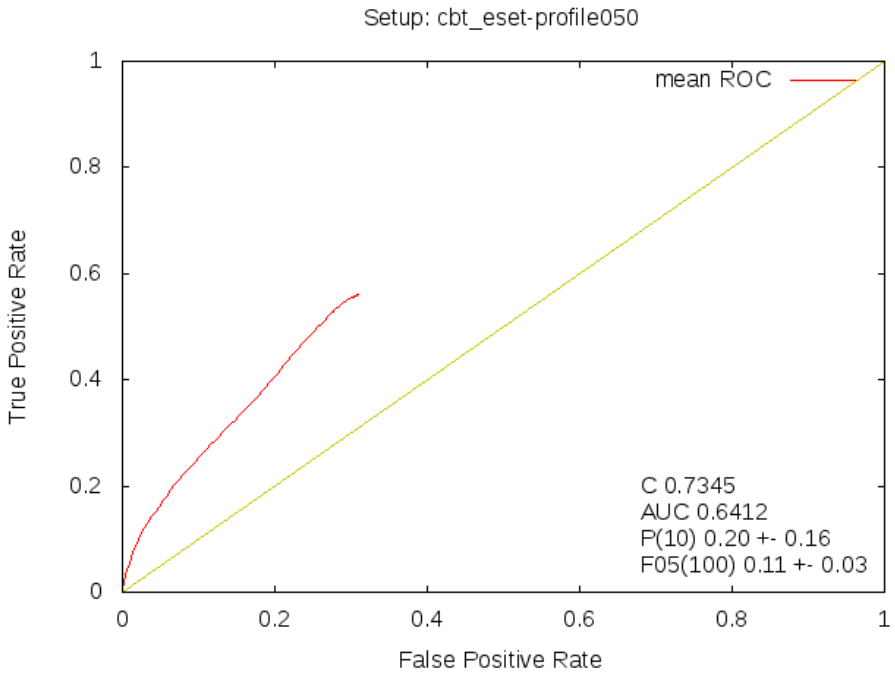

Figura 6.5: Curva ROC média de um recomendador `cbt`

Resultados

A análise gráfica das curvas ROC produzidas nos permite inferir acerca do desempenho do modelo, que é considerado mais satisfatório quando a curva se aproxima do eixo das ordenadas. A diagonal ascendente representa o comportamento de um modelo aleatório 3.7.2). Para uma análise mais objetiva, utilizamos como parâmetro a área sob a curva, e novamente a cobertura como medida secundária.

A tabela 6.3 sumariza os resultados dos experimentos, indicando os melhores desempenhos obtidos para cada modelo.

A análise gráfica das curvas indica que as estratégias `knn` e `kmm_plus` com vizinhança 10 (figuras 6.9, 6.8, 6.11 e 6.10) e `knnco` com vizinhança 3 e perfil entre 50 e 200 (6.13 e 6.12)

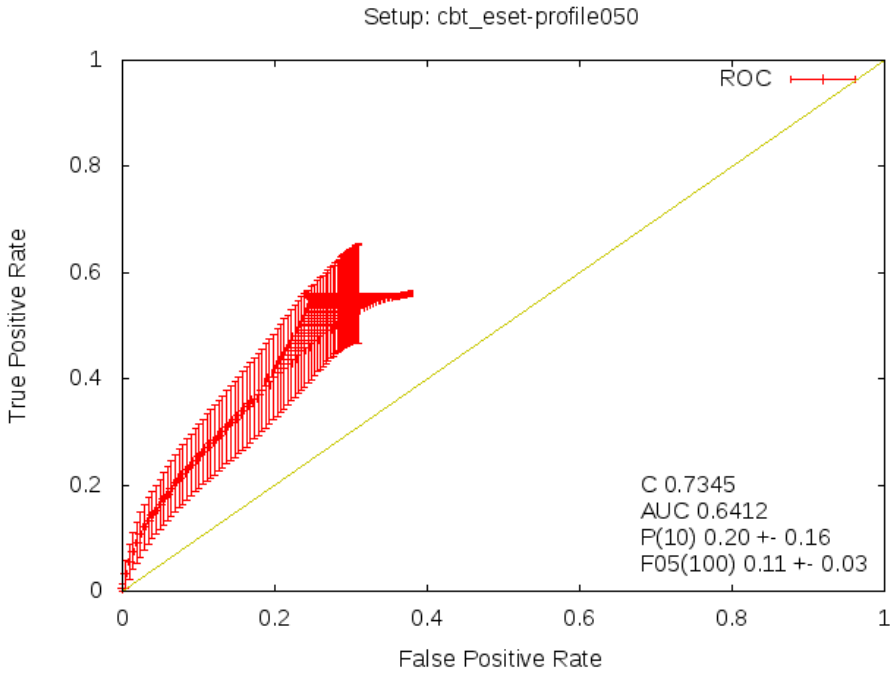

Figura 6.6: Curva ROC com desvios de um recomendador `cbt`

```
# strategy-profile
# cbt_eset-profile050

# roc AUC
0.6412

# threshold mean_fpr    dev_fpr     mean_tpr    dev_tpr     coverage
   1        0.0004      0.0003      0.0054      0.0078      0.0274
   10       0.0044      0.0009      0.0323      0.0258      0.1428
   20       0.0090      0.0011      0.0558      0.0328      0.2197
   30       0.0139      0.0012      0.0748      0.0347      0.2729
   40       0.0187      0.0013      0.0920      0.0385      0.3135
   50       0.0237      0.0015      0.1069      0.0422      0.3519
   60       0.0287      0.0015      0.1206      0.0438      0.3878
```

Figura 6.7: Registro de desenho da curva ROC

Estratégia	Perfil	Vizinhança	AUC	Cobertura
cb	100	–	0.5642	0.9557
cbt	20	–	0.6401	0.7339
cbd	100	–	0.5377	0.9557
cbh	50	–	0.6357	0.9531
cb_eset	50	–	0.5674	0.9557
cbt_eset	50	–	0.6412	0.7345
cbd_eset	10	–	0.5489	0.9557
cbh_eset	50	–	0.6118	0.9557
knn	–	10	0.8042	0.9173
knn_plus	–	10	0.8037	0.9189
knn_eset	–	100	0.7206	0.9557
knnco	200	3	0.7544	0.8161
knnco_eset	50	50	0.6653	0.7540

Tabela 6.3: Melhores desempenhos analisados por curvas ROC

são as que produzem os melhores modelos preditivos. Esta análise confirma os valores de área sob a curva calculados analiticamente (tabela 6.3).

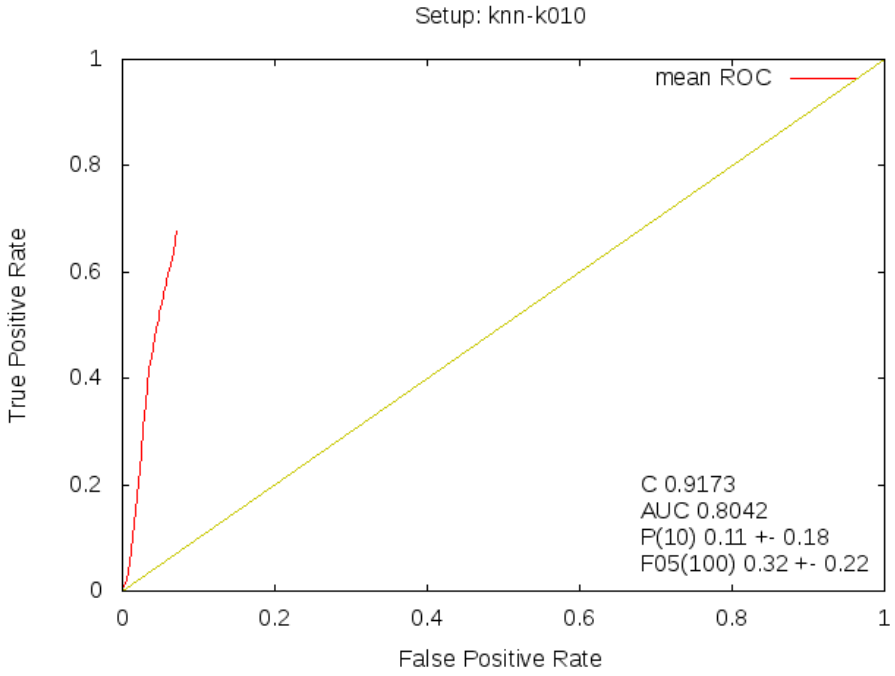

Figura 6.8: Curva ROC média para estratégia `knn`

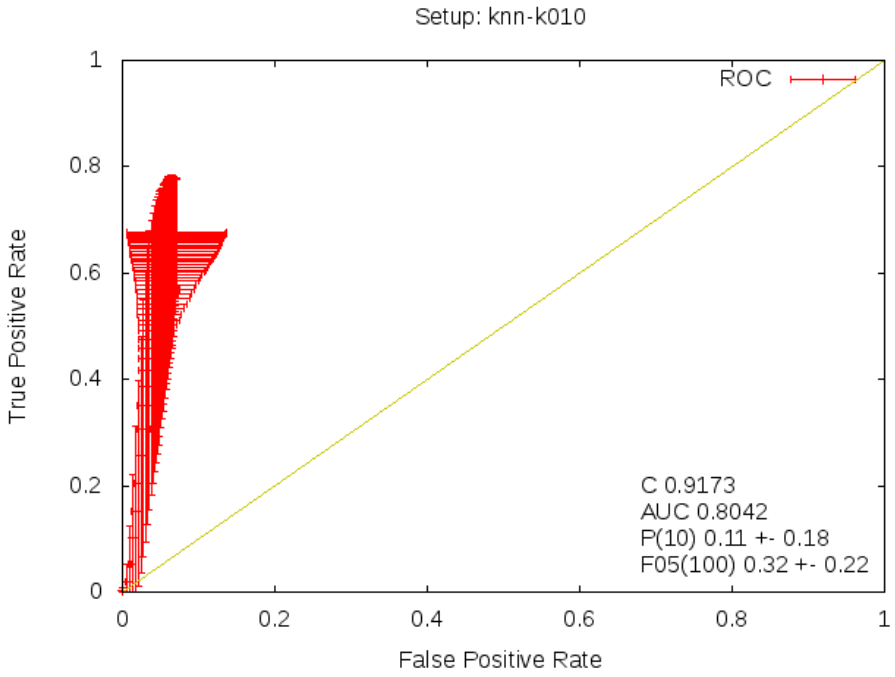

Figura 6.9: Curva ROC com desvios para estratégia `knn`

Este experimento também possibilitou o descarte de modelos inválidos, nos casos em que o desempenho era pior do que o de um modelo aleatório. Por exemplo, a implementação inicial da estratégia *knn_plus* não considerava nenhuma normalização dos pesos dos vizinhos, o que

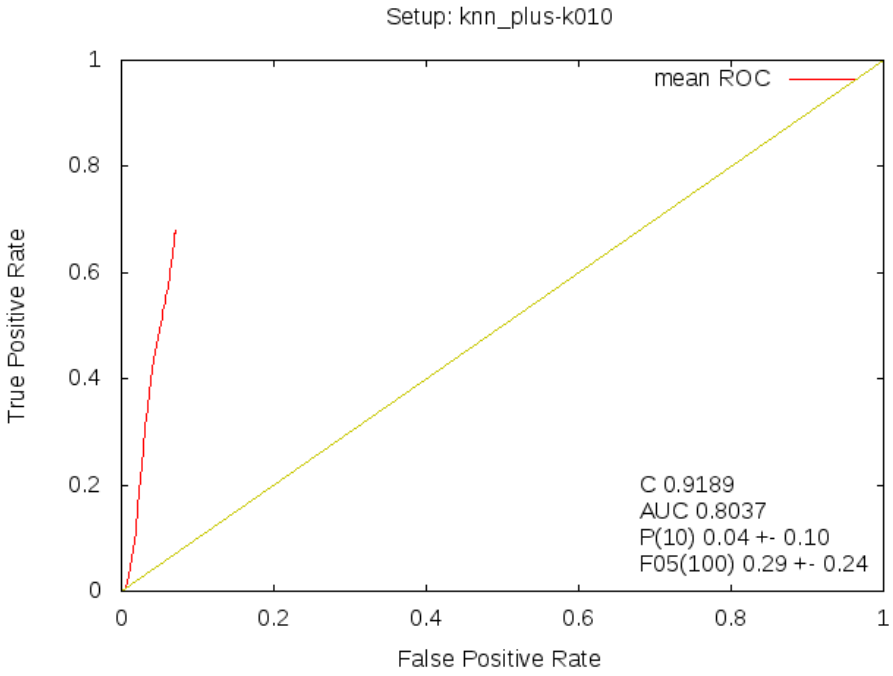

Figura 6.10: Curva ROC média para estratégia `knn_plus`

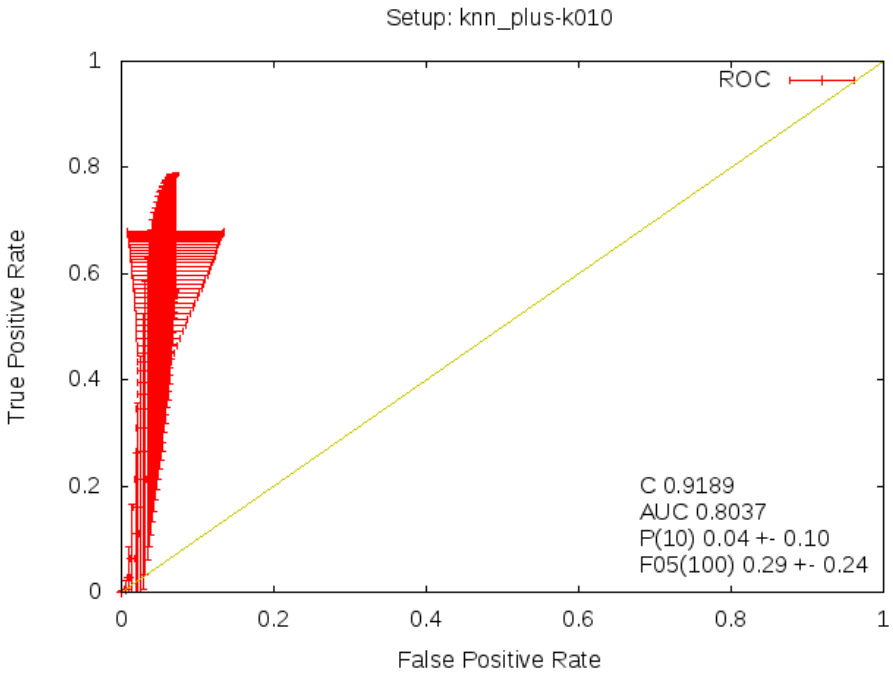

Figura 6.11: Curva ROC com desvios para estratégia `knn_plus`

provocava um comportamento anômalo do recomendador. Percebemos o problema por meio da análise das curvas ROC e evoluímos a implementação conforme orientação de [**?**], obtendo assim melhores resultados.

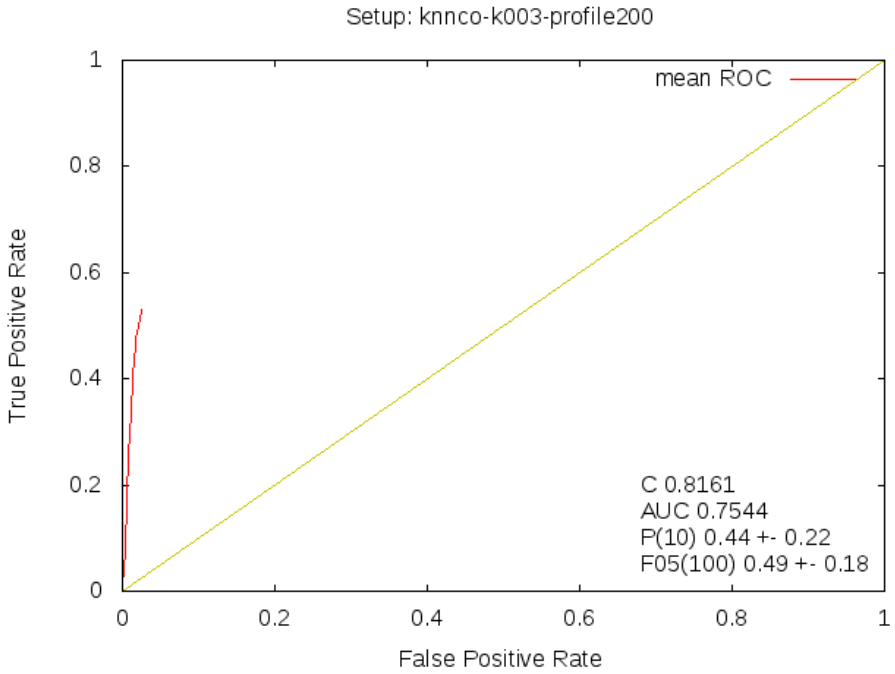

Figura 6.12: Curva ROC média para estratégia `knnco`

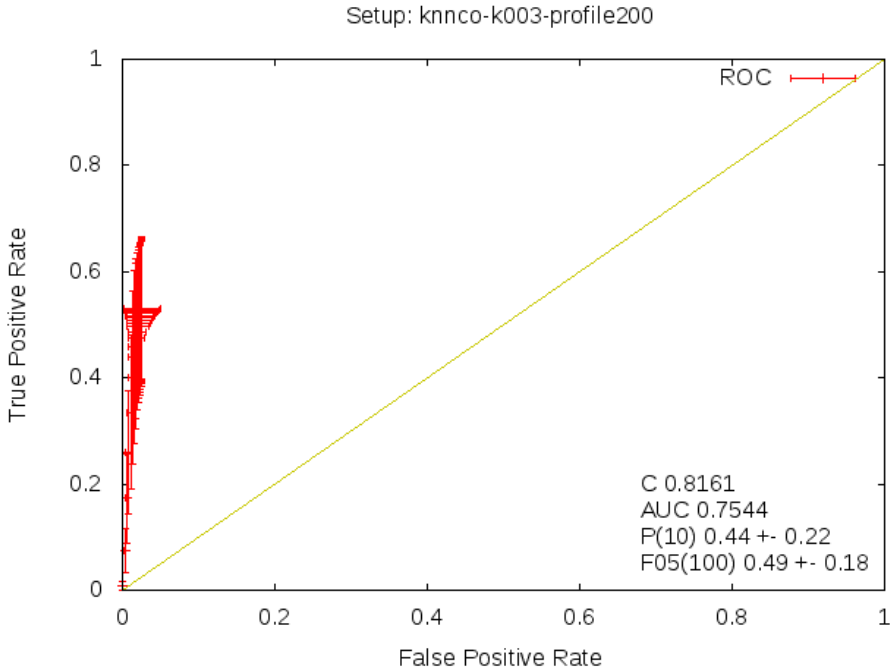

Figura 6.13: Curva ROC com desvios para estratégia `knnco`

6.2 Consulta pública

Os resultados dos experimentos *offline* não são conclusivos. As estratégias colaborativas em geral produzem resultados melhores, porém bastante variados a depender da situação de testes. E mesmo as estratégias baseadas em conteúdo, merecem ser mais investigadas, em face de sua facilidade de implementação.

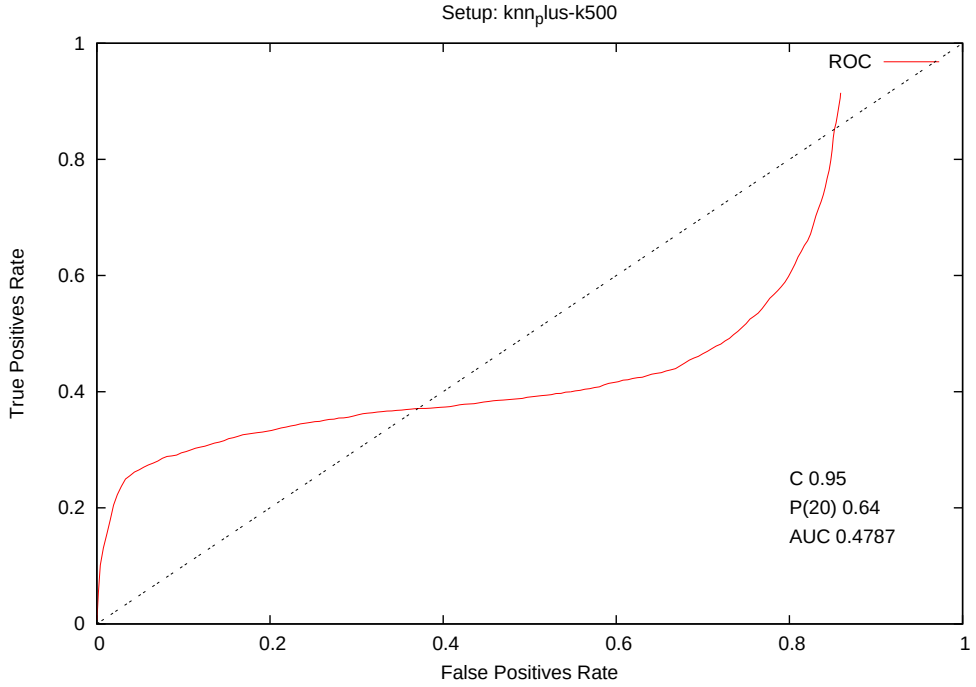

Figura 6.14: Curva ROC de modelo anômalo

Os experimentos *offline* não dependem de agentes externos à pesquisa, visto que baseiam-se em processamento automatizado de dados. No entanto, esta classe de testes não assegura um nível de confiabilidade como pode-se obter em experimentos realizados com usuários reais. Ademais, o conceito de utilidade de um aplicativo é subjetivo e apenas um indivíduo dotado de subjetividade é capaz fazer esta avaliação. Métricas como a novidade promovida por uma recomendação dificilmente seria mensurável por meio de validação cruzada.

Tais fatos motivaram a implementação de uma consulta pública *online* [13] que busca integrar avaliações de caráter subjetivo aos resultados até então obtidos pelos testes *offline*. Esta consulta é conduzida pelos seguintes passos:

(1) O participante envia a lista de pacotes instalados em seu sistema.

(2) O *AppRecommender* utiliza como estratégia primária a recomendação híbrida por revezamento a partir da análise do perfil do usuário e escolhe aleatoriamente entre um conjunto de estratégias disponíveis para aquele tipo de usuário.

(3) A computação da recomendação é realizada.

(4) As sugestões são apresentadas individualmente. Para cada aplicativo recomendado são exibidos: uma descrição curta, uma descrição longa, o *website* do *upstream*, o mantenedor do pacote e uma captura de tela. É também apresentado um quadro de avaliação, onde o usuário deve classificar a recomendação entre *surpresa boa*, *útil* ou *ruim*.

(5) O usuário seleciona uma das opções e segue para avaliar o próximo item.

[13] http://recommender.debian.net/survey

(6) Ao final de 10 avaliações, o resultado é enviado ao servidor e o usuário pode escolher se deseja realizar uma nova rodada de avaliações (uma nova estratégia será sorteada) ou finalizar o experimento.

(7) Se decidir continuar com o experimento, o passo 3 é retomado.

(8) Quando o usuário decidir finalizar sua participação, ele recebe uma mensagem de agradecimento e um formulário de preenchimento facultativo com identificação.

(9) Os resultados da validação são armazenados no servidor para posterior análise.

A figura 6.15 apresenta a interface web desenvolvida para realização da consulta pública.

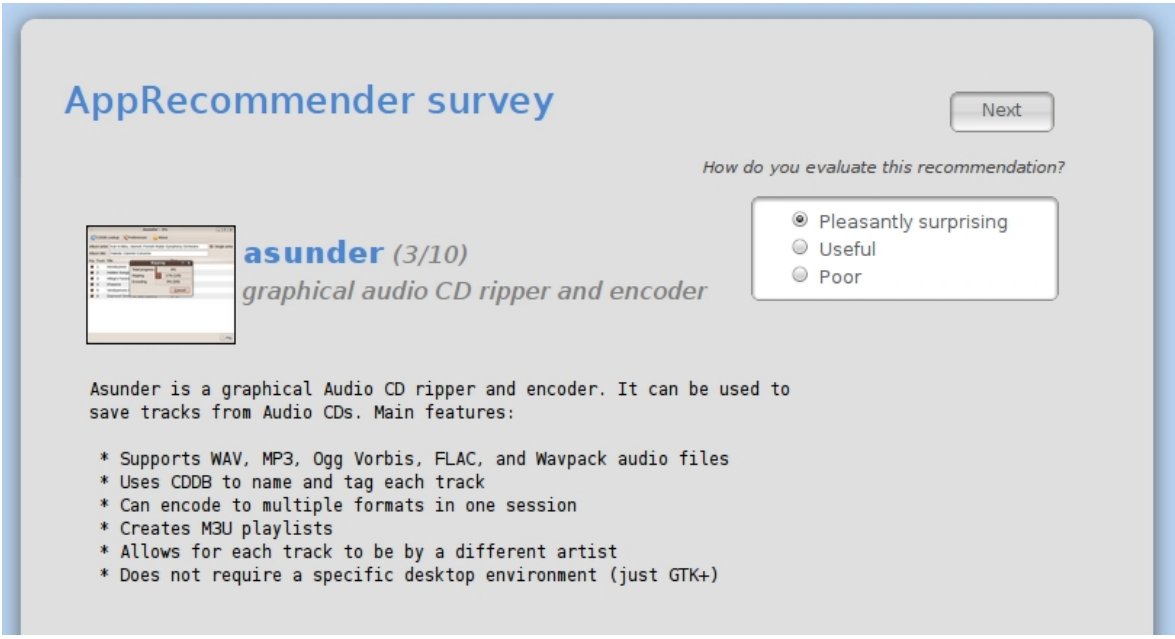

Figura 6.15: Interface da consulta pública

6.2.1 Métricas

As métricas selecionadas para avaliar os modelos nesta fase de experimentos foram as seguintes:

(a) *Precisão* ($\frac{VP}{VP+FP}$): das métricas de acurácia clássicas apresentadas na seção 3.7.2, a precisão é a única que pode ser calculada de forma objetiva após a avaliação do usuário, pois depende apenas da sua apreciação sobre os itens recomendados (preditos positivos). Outras métricas, como recuperação e medidas F_β, dependeriam da avaliação do usuário de todos os itens do repositório para que o conjunto de itens positivos (reais positivos) fosse conhecido.

(b) *Novidade*: esta é uma métrica que só pode ser mensurada com a participação de usuários reais e foi a grande motivadora para a realização do *survey*. Será mensurada de forma semelhante à precisão, como a proporção de itens marcados como *novidade* do conjunto de itens recomendados ($\frac{N}{(VP+FP)}$).

(c) *Cobertura*: a cobertura será computada ao final da fase de coleta, indicando para cada modelo a proporção do repositório de itens que fizeram parte de alguma recomendação ao longo de todo o período de experimentos.

6.2.2 Resultados

Um piloto [14] do *survey* foi lançado por meio do envio de convites para cerca de 20 usuários selecionados. No momento da escrita deste texto os últimos ajustes estão sendo realizados para que a divulgação massiva do experimento seja efetivada. Após a coleta de um número significativo de submissões para a realização de uma análise estatística, os resultados parciais do experimento serão publicados no site da pesquisa [15].

[14] http://recommender.debian.net/survey
[15] http://recommender.debian.net/experiments

Capítulo 7

Considerações finais

Apontamos como a contribuição mais relevante deste trabalho a concepção e desenvolvimento de um recomendador de aplicativos GNU/Linux, atualmente disponível para a comunidade por meio de um serviço *Web*[1]. O trabalho foi apresentado na conferência anual de Desenvolvedores Debian (*DebConf11*)[2], onde recebeu críticas e sugestões, inclusive de desenvolvedores que já trabalharam em iniciativas semelhantes no Debian (seções 4.1 e 4.3).

A validação da proposta aconteceu num primeiro momento por meio de experimentos *offline*, com a aplicação de métodos de avaliação para modelos preditivos. Os resultados dos experimentos guiaram o ajuste de parâmetros das estratégias de recomendação, além de possibilitar o descarte das que apresentaram desempenho insatisfatório. A segunda fase de validação está em curso, e consiste numa consulta pública por meio da qual usuários reais avaliam a eficácia do recomendador.

A seguir são apresentados alguns desdobramentos possíveis deste trabalho, que no nosso entendimento abrem portas para futuras colaborações entre a academia e a comunidade de desenvolvimento de programas livres, em especial o projeto Debian.

Ciclos de testes e atualizações

Os primeiros experimentos foram executados, mas a interpretação dos dados gerados ainda pode ser mais aprofundada. Ademais, ao fim de cada fase de testes, o recomendador deve ser ajustado de acordo com as configurações que obtiveram melhor desempenho nos experimentos. Após os ajustes, é importante que se realize outro ciclo de testes, já que o modelo preditivo sofreu alterações. Em seguida, outra fase de ajustes, e assim por diante.

Entendemos a calibragem dos algoritmos e estratégias como um processo contínuo, sendo esta apenas a primeira fase de ajustes. A configuração do recomendador será passível de refinamento por todo o tempo em que estiver a serviço da comunidade.

Fortalecimento de estratégias baseadas em conteúdo

De acordo com as métricas de avaliação adotadas, estratégias baseadas puramente em conteúdo não apresentaram bons resultados. Acreditamos que um investigação ainda mais detalhada na base do *Popcon* possa revelar novos perfis de usuário que auxiliem a produção de recomendações mais acertadas. Integrantes da equipe responsável por programas científicos no Debian[3] mostraram interesse em aprofundar a pesquisa sobre estes perfis, possivelmente utilizando técnicas de agrupamento.

[1] http://recommender.debian.net
[2] http://penta.debconf.org/dc11_schedule/events/773.en.html
[3] http://wiki.debian.org/DebianScience

É importante ressaltar a relevância das estratégias baseadas em conteúdo no contexto do *AppRecommender*, devido principalmente ao fato de não dependerem do acesso a uma base de dados contendo informações de outros usuários. Nesta abordagem a recomendação pode ser computada no sistema do cliente, mediante o acesso a uma base local de informações sobre os aplicativos. Esta seria uma alternativa para os usuários que optam por não enviar seus dados a servidores mantidos por terceiros.

O tema foi discutido durante a *Debconf11*, onde surgiu a ideia de disponibilizar uma versão do recomendador de aplicativos que funcionasse de forma descentralizada, e que a colaboração pudesse acontecer de um para poucos, não necessariamente entre todos os usuários. Desenvolvedores do *FreedomBox*[4] colocaram-se a disposição para colaborar com experimentos do *AppRecommender* como uma aplicação piloto do projeto.

Análise dos resultados do *survey*

A coleta de dados por meio da consulta pública ainda está em curso. Esta fase de avaliação é de fundamental relevância para o projeto, uma vez que a análise humana fornecerá uma visão acerca do quesito *novidade* trazido pela recomendação. Os resultados desta etapa serão confrontados com os experimentos *offline* e possivelmente nos permitirá ter novas percepções sobre a utilidade de cada modelo, corroborando ou enfraquecendo as conclusões dos experimentos com base em validação cruzada.

Melhorias na solução

São apresentadas a seguir algumas ideias de melhoria para o processo de composição da recomendação que surgiram ao longo do desenvolvimento e ainda não foram implementadas na versão atual.

- *Atestação de relevância*: Aplicar o conceito de atestação de relevância apresentado na seção 3.6.5 ao resultado de uma recomendação. Após receber um conjunto de aplicativos recomendados, o usuário indicaria quais são de fato relevantes para ele e o recomendador utilizaria esta informação para estender a primeira consulta realizada. Desta forma o recomendador seria realimentado com as avaliações do usuário para produzir recomendações ainda mais específicas às suas necessidades.

- *Ranking de aplicativos*: Adaptar o conceito de *page rank* utilizado em buscas na *Web* ao contexto do *AppRecommender*. Motores de busca na Internet "premiam" as páginas que possuem mais links incidentes, como um indicativo de confiabilidade do conteúdo. No caso do repositório de aplicativos, entendemos que as relações de dependência entre pacotes poderiam conduzir um parâmetro de confiabilidade análogo. Quando maior for a quantidade de programas que dependem de um determinado aplicativo, maior será a comunidade de desenvolvedores e usuários do aplicativo que se importam com seu bom funcionamento. Sendo assim, *bugs* tendem a ser resolvidos mais rapidamente e atualizações do *upstream* tendem a ser incorporadas com maior frequência.

- *Log de atividades*: Utilizar um sistema de *log* de atividades para registrar informações que forneçam um maior detalhamento acerca do perfil do usuário ou sistema. O programa *Zeitgeist*[5], por exemplo, registra informações sobre a utilização de arquivos no sistema, navegação na Internet, conversas por *chat* ou *e-mail*. A coleta destas informações é mais intrusiva que a do *Popcon*, pois as informações submetidas não limitam-se ao perfil do sistema, mas da pessoa que o utiliza. Portanto, o uso de tais dados deveria ser ainda mais criterioso e a coleta só deveria ser iniciada mediante expressa solicitação do usuário.

[4] http://freedomboxfoundation.org/learn/
[5] https://launchpad.net/~zeitgeist

Análise de riscos de privacidade

Atualmente a base do *Popcon* utilizada pelo *AppRecommender* não é atualizada constantemente, portanto, dificilmente estaria vulnerável a ataques que monitoram o comportamento do recomendador ao logo do tempo. Contudo, a frequente atualização destes dados certamente permitiria a produção de sugestões mais eficazes, dado que se basearia em informações mais "frescas". Acreditamos na necessidade de um estudo mais aprofundado dos riscos de exposição dos dados dos usuários, antes que um esquema de atualização seja proposto.

Aperfeiçoamento da interface

A interface *Web* do recomendador disponível atualmente[6], além de computar a recomendação personalizada de aplicativos, fornece uma série de informações sobre os mesmos, por meio da integração de serviços providos pelo Debian, como o *screenshots.debian.org* e UDD. No entanto, seria de utilidade para o público a integração de outras funcionalidades relacionadas com o repositório de pacotes, como a navegação por seções ou *tags*, busca, possibilidade de instalação de pacotes por meio do protocolo *AptURL*, entre outras. Um exemplo de serviço nesta linha é *AppNR*[7].

Outro ponto concernente à interface que merece atenção neste trabalho é a *internacionalização* do *AppRecommender*. Pretendemos implementar uma infraestrutura que permita a tradução dos textos de interface com o usuário para outros idiomas (processo conhecido como *localização*).

Empacotamento para o Debian

Estamos trabalhando na criação de um pacote Debian para o *AppRecommender*. Estando em conformidade com a política de empacotamento de programas para a distribuição, não encontraremos barreiras para incorporá-lo oficialmente ao projeto. A inclusão do pacote no repositório do Debian daria uma maior visibilidade ao *AppRecommender*, resultando na atração de novos usuário e colaboradores, contribuindo assim para o amadurecimento do trabalho.

Integração com outras distribuições

A integração com outras distribuições foi sempre uma meta no desenvolvimento do recomendador. Acompanhamos as discussões do projeto *AppStream* até o ponto que surgiu o impasse sobre o uso do *Software-center* – ainda não resolvido.

Uma alternativa simples para esta questão, que não necessita de infraestrutura adicional alguma, é o mapeamento da lista de pacotes do usuário para uma lista de pacotes Debian; computação da recomendação e mapeamento de volta para a referida distribuição. A função de mapeamento pode ser realizada pelo programa *distromatch*[8].

Documentação no idioma inglês

Pretendemos sumarizar o conteúdo desta dissertação num documento de referência para colaboradores externos, incluindo principalmente a arquitetura do serviço e as decisões tomadas ao longo do desenvolvimento que dizem respeito ao funcionamento do recomendador. A criação de uma documentação em inglês é de grande importância para que os desdobramento deste trabalho de fato envolvam a comunidade internacional.

[6] http://recommender.debian.net
[7] http://appnr.com
[8] http://enricozini.org/2011/debian/distromatch/

Palavras finais

A realização deste projeto não seria possível sem a colaboração da comunidade Debian que, além de disponibilizar os dados utilizados como base da composição de recomendações, possibilitou o aproveitamento de esforços anteriores e a frequente interação com desenvolvedores mais experientes nos temas abordados.

Além da dedicação contínua para que o presente trabalho ofereça contribuições no âmbito acadêmico, estamos confiantes de que fomos também capazes de retribuir um pouco ao projeto Debian os benefícios que este tem proporcionado aos seus usuários nestes 18 anos de existência. Por fim, seguimos bastante motivados a prosseguir com os trabalhos, de sorte que o *AppRecommender* siga um processo natural de aperfeiçoamento e consequente integração com outros sistemas já consolidados.