Ana Catarina de Pinho Miranda

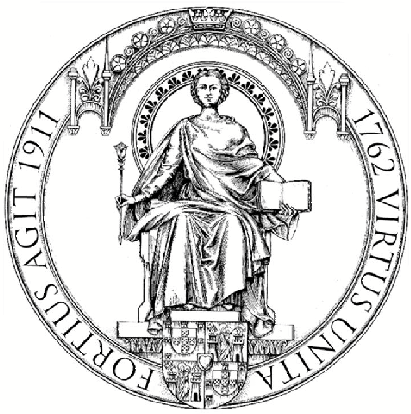
Filtragem Colaborativa Incremental para recomendações automáticas na *Web*

Faculdade de Economia da Universidade do Porto

2008

Ana Catarina de Pinho Miranda

Filtragem Colaborativa Incremental para recomendações automáticas na *Web*



Dissertação submetida à Faculdade de Economia da Universidade do Porto
para obtenção do grau de Mestre em Análise de Dados e Sistemas de Apoio à Decisão

Dissertação realizada sob supervisão
do Professor Alípio Mário Jorge
da Faculdade de Economia da Universidade do Porto

Faculdade de Economia da Universidade do Porto
2008

Para todos os que me deram força para a conclusão deste trabalho.

Breve Biografia

Ana Catarina de Pinho Miranda, nasceu a 3 de Novembro de 1983, na freguesia de Avanca, concelho de Estarreja, distrito de Aveiro. Concluiu o 12º ano do curso Geral de Científico-Natural na Escola Secundária de Estarreja, no ano lectivo 2000/2001.

Obteve a licenciatura em Matemática Aplicada e Computação pela Universidade de Aveiro, em 2005. No ano lectivo de 2006/2007 completou a parte escolar do Mestrado de Análise de Dados e Sistemas de Apoio à Decisão, na Faculdade de Economia do Porto.

Desde Março de 2006 que é Bolseira de Investigação Científica no Projecto Matemática Ensino, PmatE, da Universidade de Aveiro.

É autora de publicações científicas na área do *e-learning* e Sistemas de Recomendação.

Agradecimentos

A Deus, pela vida.

Ao meu orientador, Prof. Doutor Alípio Jorge pelo apoio, força, confiança e paciência. A orientação dada foi preciosa para o enriquecimento desta dissertação.

Aos revisores anónimos dos artigos submetidos em conferências pelas críticas úteis.

Ao LIADD-INESC Porto: *Laboratory of Artificial Intelligence and Decision Support* (LIAAD).

Ao projecto Site-O-Matic: *Web site Automation*.

À Fundação para a Ciência e Tecnologia POSC/EIA/ 58367/2004/ Site-o-Matic e PTDC/EIA/81178/2006 Rank! Projectos co-financiados por FEDER.

Ao coordenador do Projecto Matemática Ensino, Prof. Doutor António Batel Anjo, pelo tempo disponibilizado para que esta dissertação fosse possível e à coordenadora científica do mesmo, Prof. Doutora Paula Oliveira, pelo apoio e encorajamento.

Aos familiares, namorado e amigos pela compreensão da minha ausência.

Aos colegas de trabalho pela enorme colaboração.

Resumo

O uso de Sistemas de Recomendação por Filtragem Colaborativa na *Web* é normalmente feito em ambientes onde os dados estão constantemente a alterar-se e a surgirem novos clientes e produtos. Neste trabalho, propõe-se uma versão incremental dos Sistemas de Recomendação por Filtragem Colaborativa baseados nos itens para dados implícitos (binários). Esta é comparada com uma versão não-incremental, bem como com uma abordagem incremental mas baseada nos utilizadores. Pretende-se também estudar o uso de técnicas de trabalho com matrizes esparsas nestes algoritmos. Todas as versões são implementadas em R e são avaliadas empiricamente em cinco conjuntos de dados diferentes onde se faz variar o número de utilizadores e/ou itens. Observa-se que a medida de *Recall* estudada tende a melhorar quando se adiciona continuamente informação ao modelo de recomendação e que o tempo gasto para a recomendação não se degrada. O tempo gasto na actualização da matriz de semelhança (necessária à recomendação) é relativamente baixo e motiva a utilização da abordagem incremental baseada nos itens.

Abstract

The use of collaborative filtering recommenders on the Web is typically done in environments where data is constantly flowing and new customers and products are emerging. In this work, it is proposed an incremental version of item-based Collaborative Filtering for implicit binary ratings. It is compared with a non-incremental one, as well as with an incremental user-based approach. It is also study the use of techniques for working with sparse matrices on these algorithms. All the versions are implemented in R and are empirically evaluated on five different datasets with various number of users and/or items. It is observed that the measure of Recall used tend to improve when we continuously add information to the recommender model and that the time spent for recommendation does not degrade. Time for updating the similarity matrix (necessary to the recommendation) is relatively low and motivates the use of the item-based incremental approach.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

Numa sociedade globalizada como aquela em que vivemos, tornam-se imprescindíveis ferramentas com capacidade de resposta às exigências na busca de informação. Essa busca tem-se tornado cada vez mais comum na vida das pessoas, deixou de ser um simples prazer, tornando-se um hábito indispensável para a vida. Na maioria das vezes, as grandes novidades da ciência e da tecnologia impulsionam o ritmo do comportamento humano, fazendo com que a informação se propague pelo mundo. A *Web* é uma grande aliada nesse processo de difusão da informação. Através dela, pessoas de todo o mundo têm acesso a diversos tipos de informação [32][34]. A *Web* tornou-se a maior base de conhecimento que alguma vez existiu. No entanto, sem a representação apropriada desse conhecimento e sem algoritmos apropriados para a descoberta do mesmo, a *Web* é como um ser humano com uma memória extraordinária mas sem habilidade para pensar e raciocinar [11].

Com milhões de páginas criadas, *World Wide Web* é uma base de conhecimento extremamente rica. O conhecimento vem não apenas do conteúdo das próprias páginas, mas também das características únicas da *Web*, tais como a sua estrutura de hiper-

ligações e a sua diversidade de conteúdos e linguagens. A avaliação destas características revela muitas vezes padrões interessantes e novo conhecimento. Tal conhecimento pode ser usado para melhorar a eficiência e eficácia na pesquisa de informação na *Web* e também para outras aplicações que não estão directamente relacionadas com a *Web*, tais como o apoio na tomada de decisão ou gestão do negócio [11][32][47].

Apesar dos sítios *Web* inicialmente focarem-se na exactidão e disponibilidade, muitos sítios estão já a trabalhar no sentido de diferenciar os seus serviços para que possam aumentar a sua lista de clientes e mantê-los afastados dos seus competidores. Desenhar e implementar sítios personalizados surge como uma componente essencial desta estratégia de diferenciação de serviços [46].

Para que a personalização seja feita da melhor forma possível, esta tem de satisfazer as três entidades envolvidas num sítio *Web* [22]: Utilizador, Editor e Dono. O Utilizador é a pessoa que acede a um sítio *Web* e navega nele. O Editor é a pessoa responsável por criar, actualizar e eliminar conteúdos nesse sítio, podendo ser o criador (autor) ou não. O Dono é a pessoa ou organização que é proprietária do sítio *Web* e que gere a actividade do Editor [22]. Do ponto de vista do Utilizador, a *Web* é, cada vez mais, demasiado grande, dinâmica e desconhecida. Do ponto de vista do Editor, a *Web* é uma procura incessante por nova informação e actualizações da informação existente. Para além disso, o Editor não tem apenas de gerir os conteúdos do sítio, mas precisa também de gerir a estrutura de navegação, para que esta permita ao Utilizador satisfazer as suas necessidades e ao Dono atingir os seus objectivos. Do ponto de vista do Dono, as exigências em termos de força de trabalho especializada implicam custos financeiros e pessoais muito altos. O projecto Site-O-Matic (SOM) [22] surge com o objectivo de desenvolver uma plataforma e uma metodologia para automatizar várias tarefas de gestão de um sítio *Web*, tais como a captação de conteúdos relevantes, a monitorização e gestão dos conteúdos existentes e a sua organização estrutural, bem como a geração de recomendações e personalização da estrutura. Esta automatização deverá ter em conta o comportamento dos Utilizadores e também os objectivos do

Dono. Como consequência da automatização tem-se a redução do esforço do Editor e, assim, dos custos do Dono, bem como o aumento da satisfação do Utilizador conseguida à custa da personalização e adaptação do sítio *Web* às necessidades específicas deste [22].

1.2 Objectivos

Para proceder à automatização de um sítio *Web*, podem-se usar Sistemas de Recomendação. Estes podem facilitar a navegação na página e/ou apresentar produtos do interesse do cliente [45]. O tipo de Sistemas de Recomendação mais conhecido é a Filtragem Colaborativa [45]: itens são recomendados segundo um processo de "passa-a-palavra"[48], isto é, os itens são recomendados aos utilizadores tendo por base a avaliação feita por outras pessoas com gostos semelhantes [51].

Dada a dinâmica da *Web*, com novos utilizadores e itens a aparecerem e as preferências dos primeiros sempre a mudar, os modelos de recomendação têm de ser constantemente actualizados. Isto é possível actualizando o modelo periodicamente (o modelo é construído todo de novo) ou continuamente (cada vez que surge uma nova sessão, o modelo é alterado de acordo com essa nova informação).

Este trabalho tem como principal objectivo desenvolver um algoritmo incremental para os Sistemas de Recomendação por Filtragem Colaborativa que seja eficiente e que apresente bons resultados em termos de precisão. Pretende-se também comparar as abordagens incremental e não incremental, bem como as baseadas nas semelhanças entre itens e entre utilizadores (nas suas versões incrementais). Estuda-se também o desempenho de três *packages* do R [42] que permitem trabalhar com matrizes esparsas com o intuito de reduzir a quantidade de memória necessária.

Nesta dissertação foram utilizados dados pré-processados no âmbito do trabalho de Marcos Domingues [14][15].

1.3 Contribuições

Neste trabalho apresenta-se um algoritmo incremental de um sistema de recomendação por Filtragem Colaborativa. Com a incrementalidade dos algoritmos ganha-se em termos de custos computacionais aquando da sua utilização. A partir das experiências feitas conclui-se que os sistemas incrementais de recomendação por Filtragem Colaborativa baseados nos itens são, em termos de capacidade predictiva, tão bons ou melhores que os baseados nos utilizadores.

Devido ao uso do programa R [42] para implementar os algoritmos propostos neste trabalho, também se tiraram algumas conclusões relativamente às suas *packages* para trabalhar com matrizes esparsas. Concluiu-se que a *package spam* é mais rápida que as outras *packages* consideradas (*SparseM* e *Matrix*).

Foram aceites para publicação dois artigos com base neste trabalho, um no *Workshop Web and Text Intelligence 08 (WTI 08)* e outro na Conferência *2008 IEEE/WIC/ACM International Conference on Web Intelligence (WI-08)*. Em ambos os artigos, foram usados resultados obtidos durante o estudo prévio e preparação para este trabalho. *WTI 08* é um *Workshop* co-localizado com *SBIA08* e que se realiza de 26 a 30 de Outubro de 2008 em Salvador (Bahia - Brasil). Mais informações podem ser consultadas em $http://www.liaad.up.pt/\sim wti08/$. O artigo submetido e aceite foi *"Experiments with incremental collaborative filtering for implicit binary ratings"* [29]. Em relação à Conferência (*WI-08*), esta realiza-se conjuntamente com *2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-08)* de 9 a 12 de Dezembro de 2008. Mais informações podem ser consultadas em $http://datamining.it.uts.edu.au/conferences/wi08/$. O artigo submetido e aceite foi *"Incremental collaborative filtering for binary ratings"* [30].

1.4 Organização

Estruturou-se a dissertação da seguinte forma: estado da arte em *Web Mining* e Sistemas de Recomendação, nos Capítulos 2 e 3; estudo global sobre Sistemas de Recomendação Incrementais, no Capítulo 4; apresentação dos algoritmos implementados, no Capítulo 5; resultados da avaliação feita, no Capítulo 6. Por fim, apresentam-se as conclusões e trabalho futuro (Capítulo 7). Como forma de melhor exemplificar o que é exposto, sempre que possível, são apresentados gráficos ilustrativos e respectivas observações. Em anexo, encontra-se o código R usado na implementação dos algoritmos apresentados (Anexo A). Optou-se, também, por anexar os gráficos que, por mostrarem conclusões muito semelhantes às apresentadas no Capítulo 6, não foram apresentados no corpo deste trabalho (Anexo B).

Capítulo 2

Web Mining

Com o constante crescimento de fontes de informação na Internet, torna-se necessário utilizar ferramentas automatizadas para encontrar, extrair, filtrar e avaliar a informação disponibilizada e os recursos usados. Para além disso, com a transformação da *Web* na principal ferramenta para o comércio electrónico, é imperativo, para as organizações e companhias que investem milhões na Internet e tecnologias de Intranet, pesquisar e analisar padrões nos acessos dos utilizadores. Com isto surge a necessidade de criar sistemas inteligentes que extraiam conhecimento tanto do lado do cliente como de quem presta o serviço [12].

Web Mining é o uso de técnicas de *Data Mining* em informação (documentos ou serviços) existente na *Web* [12][16]. Podemos distinguir três categorias, que constituem as áreas de interesse de onde se pode extrair a informação/conhecimento [11][45][52]:

- *Web Content Mining* - descobrir informação útil a partir do conteúdo da *Web*;
- *Web Structure Mining* - descobrir informação útil a partir da estrutura de ligações da *Web*;
- *Web Usage Mining* - descobrir informação útil a partir dos dados gerados pelos acessos dos utilizadores às páginas *Web*.

Srivastava, J. et al [52] separaram os dados que podem ser usados na extracção de conhecimento da *Web*. Apresentaram a seguinte divisão quanto à categoria dos dados:

- **Conteúdo:** Dados reais das páginas *Web*, isto é, das páginas projectadas para os utilizadores. Estes dados geralmente são constituídos por textos e gráficos;
- **Estrutura:** Dados que descrevem a organização dos conteúdos. A estrutura interna das páginas inclui um conjunto de *tags* HTML ou XML. A principal estrutura de informação entre páginas constitui-se nas hiperligações que ligam uma página à outra;
- **Uso:** Dados que descrevem os padrões de uso de páginas *Web*, tais como o endereço IP, páginas acedidas e a data e hora de acesso;
- **Perfil do utilizador:** Dados que fornecem informação sobre os utilizadores de um sítio *Web*.

Indicam também a separação dos conjuntos de dados com base nas várias fontes que originam esses dados:

- **Recolha do lado do servidor:** O arquivo de *log* de um servidor *Web* constitui uma fonte muito importante de informação para a extracção de conhecimento do uso da *Web*. Isto acontece porque estes arquivos apresentam registos da navegação dos utilizadores num determinado sítio. No entanto, os dados do uso do sítio armazenados por arquivos de *log* podem não ser inteiramente confiáveis, por exemplo, as *views* de páginas (número de vezes que a página foi requisitada, e não carregada ou actualizada) não são gravadas no arquivo de *log* do servidor;
- **Recolha do lado do cliente:** Esta informação pode ser implementada usando programas remotos como os implementados com a linguagem baseada em objectos - *javascript*, ou com os *applets* da linguagem orientada a objectos - *Java*;

- **No proxy dos servidores:** O *proxy Web* actua a um nível intermediário entre o *Browser* do cliente e o servidor *Web*. O *proxy caching* pode ser utilizado para diminuir o tempo de carga das páginas *Web*. Através deste tipo de informação é possível identificar as páginas mais requisitadas por um grupo de utilizadores anónimos.

Como já foi dito anteriormente, *Web Mining* pode dividir-se em três técnicas: *Web Content Mining*, *Web Structure Mining* e *Web Usage Mining* [11][45] que se diferenciam nos dados em que se baseiam para a extracção de conhecimento e consequentemente nas técnicas aplicadas.

2.1 *Web Content Mining*

Web Content Mining é o processo de extracção de informação do conteúdo de um documento *Web*. Conteúdo esse que diz respeito ao que a página *Web* se destinou a informar aos seus utilizadores. Pode consistir em textos, imagens, áudio, vídeo ou registos estruturados, tais como listas ou tabelas [11][45]. Existem dois pontos principais quando se fala de *Web Content Mining*: o ponto de vista da "Recuperação de Informação"(RI) e o ponto de vista da "Base de Dados"(BD). O objectivo, sob o ponto de vista da RI, é auxiliar o utilizador no processo de busca ou filtragem de informação. É o que realiza os principais mecanismos de busca na Internet ao procurar atender da melhor maneira possível às solicitações feitas pelos utilizadores através de palavras-chave. Sob o ponto de vista da BD, o objectivo é modelar os dados da *Web* e integrá-los de tal modo que se possa fazer consultas mais sofisticadas do que simples consultas baseadas em palavras-chave. Isto pode ser realizado ao descobrir-se os padrões dos documentos na *Web*, construindo-se *Web Warehouses*, uma base de conhecimento da *Web* ou até mesmo uma base de dados virtual [45].

2.2 *Web Structure Mining*

Web Structure Mining tenta descobrir o modelo subjacente à estrutura das ligações da *Web* [11][45]. Se com *Web Content Mining* o interesse está na estrutura dentro dos documentos *Web* (estrutura intra-documentos), com *Web Structure Mining* encontra-se na estrutura das hiperligações dentro da própria *Web* (estrutura inter-documentos). Este conhecimento pode ser utilizado para classificar páginas *Web* e é útil para gerar informação tal como a semelhança ou relação entre diferentes sítios *Web*. Esta categoria de extracção de informação na *Web* pode ser utilizada para, por exemplo, se descobrirem sítios cujas ligações aparecem frequentemente em outros sítios (designados por sítios de autoridade - *authority sites*) [45].

2.3 *Web Usage Mining*

Web Usage Mining usa técnicas de *Data Mining* para analisar ou procurar padrões nos caminhos percorridos pelos utilizadores quando estão a navegar pela *Web* [11][45][52]. Enquanto *Web Content Mining* e *Web Structure Mining* utilizam os dados reais presentes nos documentos da Internet, *Web Usage Mining* utiliza dados secundários derivados da interacção do utilizador com a *Web*. Tais dados incluem registos de *log* dos servidores de acesso à *Web*, registos de *log* de servidores *proxy*, perfis, transacções e consultas do utilizador, dados do arquivo dos Favoritos, entre outros [11][45].

Através da análise dos dados relativos ao uso da *Web* (*Web usage data* - também referido como análise de *clickstream*) os sistemas de *Web Mining* podem descobrir conhecimento útil sobre as características do uso e os interesses dos utilizadores. Este conhecimento tem várias aplicações, exemplo disso temos a personalização das páginas *Web* e a colaboração em sistemas baseados na *Web* (*marketing*, intenção do sítio *Web*, avaliação do sítio *Web*, tomada de decisão). Um dos maiores objectivos de *Web Usage Mining* é revelar padrões interessantes que, frequentemente, podem proporcionar

conhecimentos importantes sobre os clientes de uma companhia ou utilizadores de um sistema. Devido ao crescimento exponencial da *Web*, a quantidade de dados nos *logs* dos servidores aumentou. Os *logs Web* usualmente consistem em dados de mais do que um utilizador. *Web Usage Mining* pode ajudar a identificar utilizadores que acederam a páginas *Web* semelhantes e essa informação é importante para a recomendação [11].

Os padrões de acesso à *Web* são cada vez mais dinâmicos, não apenas devido à dinâmica no conteúdo e estrutura do sítio, mas também devido às alterações dos interesses dos utilizadores e seus padrões de navegação na *Web*. Pode-se observar alteração dos padrões de acesso consoante a altura do dia, dia da semana e de acordo com padrões pertencentes à estação do ano ou outros eventos externos. Sendo assim os dados dependem de quando são recolhidos. Um sistema inteligente de *Web Usage Mining* deve conseguir aprender continuamente perante a presença de tais condições, sem paragens, reconfigurações ou recomeços improvisados [35].

2.4 Pré-processamento dos dados utilizados

Neste trabalho foram usados dados originários de vários sítios *Web*. Para serem usados tiveram de sofrer um pré-processamento. A *Data Warehouse* usada provém de estudos anteriores feitos por membros do LIAAD - INESC Porto L.A., nomeadamente Marcos Domingues (ver [14] e [15]), e trabalha com dados das páginas *Web*, acessos *Web* e bases de dados externas e é constituída por três passos: extracção, transformação/pré-processamento e carregamento dos dados (*load*). Na extracção é criada uma versão local do sítio *Web* e dos *logs* de acesso. Durante a transformação a versão local do sítio e os *logs* são transformados/pré-processados. Este processo lê os ficheiros HTML, XHTML e XML e devolve-os após normalizar e limpar as *tags* (retirar *tags* desnecessárias, fechar as *tags* que são inicializadas mas não são fechadas,...). A transformação/pré-processamento dos *logs* de acesso (guardados num ficheiro de *logs* ou em outro lugar, como por exemplo a base de dados) consiste em remover pedidos (*requests*) irrelevantes

ou repetidos, resolver problemas de IP, definir sessões e remover sessões *robot* da versão local de *logs* de acessos. Por fim, no passo do carregamento dos dados (*load*), estes já pré-processados do sítio *Web*, *logs* de acesso e base de dados externa são passados para uma *Data Warehouse*. Algumas pequenas alterações também são aí efectuadas (por exemplo para normalizar e/ou juntar campos dos dados). A *Data Warehouse* usada é explicada em [14] e [15] e suporta a automatização e monitorização de sítios *Web*. Esta *Data Warehouse* armazena os dados que descrevem o sítio *Web* e a sua actividade (acessos dos utilizadores e acções do editor). Estes dados são independentes do sítio *Web* e podem ser usados para suportar ferramentas de *Web Mining*, personalização e gestão de um sítio. Tais ferramentas irão tornar possível a reorganização adaptativa da estrutura de ligações e/ou conteúdos das páginas de forma a reflectir o uso actual, a identificação de comportamentos comuns entre os utilizadores e a personalização das páginas.

2.5 Resumo

Neste capítulo definiu-se *Web Mining*, bem como as três técnicas em que se divide, *Web Content Mining*, *Web Structure Mining* e *Web Usage Mining*. Estas diferenciam-se pelos dados que usam para a extracção de conhecimento e consequentemente nas técnicas aplicadas. Neste trabalho usaram-se técnicas de *Web Usage Mining* pois a recomendação é feita a partir dos dados gerados pelos acessos dos utilizadores às páginas *Web*.

De seguida são apresentados os vários Sistemas de Recomendação que fazem uso do conhecimento obtido com uma ou várias técnicas de *Web Mining*.

Capítulo 3

Sistemas de Recomendação

Nos últimos anos tem havido um crescimento explosivo do volume de informação disponível, crescimento esse que é mais rápido que a nossa capacidade para o processar [48][51]. O desenvolvimento da tecnologia também é responsável por esse aumento pois reduz as barreiras da publicação e distribuição de informação [48]. A tecnologia deve ajudar o utilizador a encontrar a informação desejada e necessária, bem como eliminar aquela que não corresponde ao seu interesse [48][51].

Cada vez mais se torna necessário fazer escolhas sem experiência pessoal suficiente das alternativas. Todos os dias confia-se em recomendações de outras pessoas ou que nos chegam por simples "passa a palavra", cartas de recomendação, críticas de livros e filmes que aparecem nos jornais. Os sistemas de recomendação auxiliam e ampliam este processo natural do crescimento da informação [45]. Estes sistemas usam as opiniões de uma comunidade de utilizadores de forma a ajudar indivíduos da mesma comunidade a identificar de um modo mais eficaz conteúdos que lhes interessem a partir de um conjunto enorme de possibilidades [9][45][47].

Sistemas de recomendação aplicam técnicas de análise e extracção de conhecimento de dados para ajudar os utilizadores a encontrar produtos que estejam realmente interessados em ver ou adquirir. Estes sistemas estão cada vez mais a ter sucesso no comércio

electrónico [47][49]. Neste domínio levantam-se três desafios a estes sistemas: produzir recomendações com grande qualidade, permitir a obtenção de muitas recomendações por segundo para milhares de utilizadores e produtos e alcançar uma alta cobertura (*coverage*) face a dados esparsos [49].

Existem três tipos de Sistemas de Recomendação: Baseados no Conhecimento, Baseados no Conteúdo e Filtragem Colaborativa. Pode-se ainda ter um Sistema Híbrido, o qual é a junção de dois ou mais dos sistemas de recomendação indicados anteriormente [9].

3.1 Sistemas de Recomendação Baseados no Conhecimento

Um Sistema de Recomendação Baseado no Conhecimento usa o conhecimento sobre os utilizadores e produtos para inferir as necessidades/preferências do utilizador de forma a fazer-lhe a recomendação [8][9][10]. Aqui os utilizadores fazem parte do processo de descoberta de conhecimento e ao longo da sua interacção com o sistema vai sendo elaborada uma "lista"das suas necessidades de informação. Este sistema não apresenta problemas de inicialização pois não depende de uma base inicial de resultados de outros utilizadores. Também não necessita de recolher informação de um utilizador em particular porque as suas recomendações são independentes dos gostos individuais. Estas características tornam o sistema útil quando usado sozinho,como também complementar a outros tipos de sistemas de recomendação [8][9].

Em [9] são apresentados três exemplos deste sistema de recomendação: *Entree*, *Recommender.com* e *PickAFlick*. No primeiro, o utilizador começa com um determinado restaurante que gosta numa dada localidade e o sistema encontra um restaurante semelhante na localidade pretendida pelo utilizador; outras características podem ser dadas ao sistema para além do local e tipo de restaurante (por exemplo os preços mais

baixos). No segundo, o utilizador começa por inserir o nome de um filme que gostou e o sistema recomenda outros que são semelhantes ao introduzido. *PickAFlick* também é um sistema de recomendação de filmes que com base no filme que o utilizador escolheu recomenda outros filmes com base em três estratégias: género, actores e director.

3.2 Sistemas de Recomendação Baseados no Conteúdo

Sistemas de Recomendação Baseados no Conteúdo recomendam itens que estejam de acordo com os conteúdos que fazem parte do perfil do utilizador e outros itens semelhantes aos que o utilizador viu no passado [1][9][51], ou seja, os itens recomendados ao utilizador são definidos como itens interessantes para o utilizador pelas características associadas a cada um deles [10]. Em oposição aos Sistemas de Recomendação Baseados no Conhecimento, estes sistemas não interagem com os utilizadores, pois estes não fornecem ao sistema qualquer informação sobre as suas necessidades/preferências, apenas são conhecidos os itens que viu em sessões anteriores. Neste tipo de sistema é necessária uma medida para a utilidade de um determinado item para o utilizador [1]. Por exemplo, no caso da recomendação de filmes a um utilizador u, o sistema tenta entender o que existe em comum entre os filmes que u viu no passado e aos quais deu uma avaliação (*rate*) elevada (actores, directores, género, assunto), sendo-lhe posteriormente apenas recomendados os filmes semelhantes às suas preferências [1]. No caso da recomendação de artigos, o sistema tenta relacioná-los com base na presença (ou ausência) de palavras-chave de outros artigos que o utilizador tenha visto e gostado [51].

No entanto este sistema apresenta algumas limitações. Segundo Robin Burke ([9]), apresenta um problema de inicialização e, de acordo com as características de *Machine Learning*, não pode ser apreendido até que o utilizador tenha classificado muitos itens. Outras limitações são apresentadas em [51], entre elas a necessidade de ter *à priori* informação sobre o conteúdo dos vários itens (algo por vezes apenas possível se for

feito de forma manual) e de os itens terem de ser da mesma forma (por exemplo texto). Com a tecnologia actual, som, fotografias, arte, vídeo ou itens físicos não podem ser analisados automaticamente para obtenção de informações sobre os atributos relevantes. Segundo os mesmos autores, este sistema também não tem capacidade de surpreender, pois recomenda itens com o mesmo conteúdo de outros que o utilizador já tenha visto no passado, não explorando novas categorias de itens. Também é limitativo o facto de este sistema não ter em conta a qualidade, estilo ou ponto de vista dos itens (por exemplo, não distingue entre um artigo bem escrito e outro mal escrito pois apenas tem em conta se em ambos aparecem as mesmas palavras-chave).

Armstrong et al [3] desenvolveram *WebWatcher*, um sistema que apoia o utilizador a encontrar na *World Wide Web* a informação que necessita a partir de palavras-chave indicadas pelo próprio utilizador. Para tal sugere hiperligações e recebe a avaliação das mesmas por parte do utilizador. Este sistema foi concebido para servir todos os utilizadores, reunindo informação e partilhando-a com os outros utilizadores. Por exemplo, se um utilizador relaciona uma página *Web* com as palavras-chave que tinha introduzido no sistema no início da procura, isto pode ser útil para qualquer outro utilizador que faça uma procura de informação semelhante [3][31]. Dunja Mladeni, em [31], reformula o sistema *WebWatcher* dando origem ao *Personal WebWatcher*, o qual evita que o utilizador se envolva no processo de aprendizagem do sistema ao não lhe pedir palavras-chave nem opiniões sobre as páginas *Web*. Ou seja, o sistema original foi estruturado de forma a especializar-se para um determinado utilizador modelando-se de acordo com os seus interesses. Outro exemplo de Sistema de Recomendação Baseado no Conteúdo é apresentado em [25], *InfoFinder*. Este procura documentos que satisfaçam os interesses dos utilizadores e envia-os. Em [39] é feita uma pequena apresentação do sistema *Syskill and Webert*, que foi pensado para ajudar os utilizadores a encontrar, sobre um determinado tópico, páginas *Web* interessantes. O critério utilizado para determinar se uma página é interessante depende do assunto, pois o perfil apreendido varia de acordo com o tópico a avaliar. Tem-se também que utilizadores diferentes

podem não chegar a acordo sobre o interesse de uma mesma página. Henry Lieberman [27] introduz um agente, *Letizia*, que opera em paralelo com um navegador da *Web*. O agente monitoriza o comportamento da navegação do utilizador e tenta antecipar os itens que possam ser de interesse para o mesmo.

3.3 Filtragem Colaborativa

A Filtragem Colaborativa é o sistema de recomendação mais promissor segundo vários autores ([9][20][24][44][49][51]). Este sistema automatiza o processo de recomendações por "passa a palavra": os itens são recomendados aos utilizadores baseando-se em valores atribuídos pelas outras pessoas com gostos similares [51]. Ou seja, a Filtragem Colaborativa utiliza informações de outras pessoas trabalhando com a ideia de que se os interesses do utilizador x são similares aos interesses do utilizador y, os itens preferidos pelo utilizador y podem ser recomendados ao utilizador x [1][7][47][55]. Este sistema agrega dados sobre os hábitos ou preferências dos itens adquiridos pelos utilizadores e faz recomendações a outros utilizadores baseando-se na semelhança dos padrões gerais das "compras"destes [9]. É assim feita uma base de dados de preferências dos produtos pelos consumidores. Este sistema funciona do seguinte modo: um novo utilizador, u_a, é confrontado com a base de dados para descobrir os seus vizinhos (utilizadores que historicamente têm gostos semelhantes a u_a). Os itens que os vizinhos viram e que u_a nunca viu antes são recomendados a u_a, assumindo que ele também vai gostar desses itens [48]. O objectivo da Filtragem Colaborativa é de prever a utilidade dos itens para um determinado utilizador (o chamado utilizador activo, u_a) baseando-se numa base de dados de votos de outros utilizadores [7]. Esta base de dados consiste num conjunto de votos $v_{u,i}$ que corresponde ao voto do utilizador u no item i.

Apresentam-se agora alguns exemplos de algoritmos de Filtragem Colaborativa encontrados na bibliografia consultada. *GroupLens* [44] é um sistema de Filtragem Colaborativa de *netnews*, para ajudar as pessoas a encontrar os artigos que gostam

dentro dos artigos disponíveis. Este sistema baseia-se na heurística de que as pessoas que concordaram no passado vão provavelmente concordar novamente. Em [20] é apresentado um sistema generalista de recomendação avaliado no caso da recomendação de vídeos. Shardanand et Maes [51] comentam e avaliam a implementação de um sistema designado por *Ringo* que faz recomendações personalizadas de álbuns de música e artistas e são apresentados quatro algoritmos diferentes de Filtragem Colaborativa. *Amazon.com* [28] é um exemplo de loja *on-line* que usa Sistemas de Recomendação por Filtragem Colaborativa para personalizar a página pessoal de cada cliente. Neste caso em particular são usados os sistemas baseados nos itens (apresentada a sua definição na Secção 3.3.2.2).

Os algoritmos de Filtragem Colaborativa estão divididos em duas categorias [7][48]: Algoritmos Baseados em Modelos e Algoritmos Baseados na Memória. Os primeiros usam o conjunto de dados para estimar ou apreender um modelo que é depois usado para fazer as previsões, já os segundos utilizam directamente o histórico para fazer as previsões. Sistemas de Recomendação por Filtragem Colaborativa são frequentemente distinguidos por operarem tanto sobre votos implícitos como explícitos [7]. Votos explícitos referem-se a quando os utilizadores expressam conscientemente a sua preferência por um item (usualmente numa escala numérica e discreta). Por exemplo, o sistema *GroupLens* de Resnick et al. [44] usa a escala de um (mau) a cinco (bom) para os utilizadores avaliarem artigos que estão *on-line* depois de os lerem. Votos implícitos referem-se a interpretar o comportamento do utilizador para atribuir uma preferência. Estes votos podem basear-se nos dados de navegação (por exemplo de aplicações *Web*), histórico de compras (lojas tradicionais ou *on-line*) ou outro tipo de informação de padrões de acesso [7].

3.3.1 Algoritmos Baseados em Modelos

De uma perspectiva probabilística, o objectivo da Filtragem Colaborativa pode ser visto como o cálculo do valor esperado do voto, tendo em consideração o que se conhece do utilizador [7]. O princípio dos algoritmos baseados em modelos está em prever-se os votos sobre os itens ainda não vistos pelo utilizador activo. Assumindo-se que os votos são valores inteiros de 0 a m e que $v_{u,i}$ indica o voto do utilizador u no item i então tem-se [7]:

$$p_{u_a,i} = E(v_{u_a,i}) = \sum_{j=0}^{m} j \times P(v_{u_a,i} = j | v_{u_a,k}, k \in I_{u_a}) \tag{3.1}$$

onde a expressão representa a probabilidade do utilizador activo u_a dar um determinado voto num item i, tendo em conta os itens previamente observados, I_{u_a}.

Como exemplos de modelos probabilísticos para a Filtragem Colaborativa podem ser indicados os modelos de Regras de Associação [47], *Cluster* e Redes Bayesianas [7].

3.3.2 Algoritmos Baseados na Memória

Estes algoritmos trabalham com toda a base de dados dos votos dos utilizadores. Seja I_u o conjunto de itens que o utilizador u já votou, então pode-se definir a média dos votos do utilizador u como:

$$\overline{v_u} = \frac{1}{\#I_u} \sum_{i \in I_u} v_{u,i} \tag{3.2}$$

Os algoritmos baseados na memória podem ser distinguidos em: baseados nos utilizadores (*user-based*) e baseados nos itens (*item-based*). A partir dessa distinção podem ser definidas as várias medidas usadas em cada um dos algoritmos.

3.3.2.1 Algoritmos Baseados nos Utilizadores

Algoritmos baseados nos utilizadores procedem olhando para os N utilizadores na base de dados que são mais semelhantes ao utilizador activo (u_a) - vizinhança de u_a. Os itens, que são preferidos por estes vizinhos, são recomendados a u_a [51]. Por norma estes algoritmos consideram-se simplesmente baseados na memória visto serem os primeiros a terem sido implementados.

De seguida definem-se duas medidas de semelhança presentes na bibliografia estudada e a forma como é feita a computação da previsão.

Correlação de *Pearson*: A semelhança entre dois utilizadores u_a e w segundo esta medida é dada por [7]:

$$sem(u_a, w) = \frac{\sum_x (v_{u_a,x} - \overline{v_{u_a}})(v_{w,x} - \overline{v_w})}{\sqrt{\sum_x (v_{u_a,x} - \overline{v_{u_a}})^2 \sum_x (v_{w,x} - \overline{v_w})^2}} \tag{3.3}$$

onde as somas em x dizem respeito aos itens que tanto u_a como w avaliaram.

Medida do Cosseno: A semelhança entre dois utilizadores u_a e w segundo esta medida é dada por [7][38]:

$$sem(u_a, w) = \sum_x \frac{v_{u_a,x}}{\sqrt{\sum_{k \in I_{u_a}} v_{u_a,k}^2} \sqrt{\sum_{k \in I_w} v_{w,k}^2}} \tag{3.4}$$

onde as somas em x dizem respeito aos itens que tanto u_a como w avaliaram. Os quadrados que aparecem no denominador servem para normalizar os votos de modo que utilizadores que votem num número maior de itens não tenham *à priori* uma semelhança maior quando comparada com utilizadores que votaram em menos itens [7].

Computação da Previsão: Os votos de u_a são previstos baseando-se na informação passada do utilizador que está na base de dados e de um conjunto de semelhanças

calculadas a partir da restante base de dados. Breese et al., em [7], assumem que o voto previsto de u_a para um item i, $p(u_a, i)$, é dado pela soma pesada dos votos dos restantes utilizadores:

$$p(u_a, i) = \overline{v_{u_a}} + k \sum_{q=1}^{n} sem(u_a, w)(v_{w,i} - \overline{v_w}) \tag{3.5}$$

onde n é o número de utilizadores com semelhanças não nulas relativamente a u_a e k é um factor de normalização de forma a que a soma dos valores absolutos dos pesos dê 1. Mas pode-se simplificar esta equação. Na equação de dados implícitos e tomando que o voto de u sobre i, $v_{u,i}$, é 1 quando u vê i e 0 caso contrário (o que acontece no caso sobre o qual serão apresentados resultados), tem-se então uma equação mais simplificada apresentada em [36]:

$$p(u_a, i) = \frac{1}{n} \sum_{w \in U} sem(u_a, w) \tag{3.6}$$

onde U é o conjunto dos utilizadores com semelhanças não nulas relativamente a u_a.

3.3.2.2 Algoritmos Baseados nos Itens

Nos algoritmos baseados nos itens olha-se para os N itens que são semelhantes aos itens da sessão activa (sessão de u_a - itens vistos e avaliação dada). Esta pequena diferença entre os algoritmos baseados nos utilizadores e baseados nos itens acarreta consequências computacionais importantes. Segundo [28] e [48], os algoritmos baseados nos itens são capazes de obter resultados com a mesma qualidade dos baseados nos utilizadores mas com menos computação *on-line*. Segundo Sarwar et al. [48] o objectivo principal destes algoritmos é analisar a matriz dos utilizadores-itens e depois usar essas relações para prever a avaliação de um utilizador sobre um dado item não visto anteriormente. Com isto, um utilizador estaria interessado em ver itens similares aos que viu antes e em evitar aqueles de que não gostou do passado. Estas técnicas não

requerem a identificação de utilizadores semelhantes cada vez que é solicitada uma recomendação, como resultado tendem a produzir recomendações mais rápidas. É necessário ter em atenção que, em geral, as páginas Web ou lojas que usam estas técnicas apresentam um número de itens inferior ao número de utilizadores.

De seguida definem-se duas medidas de semelhança presentes na bibliografia estudada bem como se apresenta a forma como é feita a computação da previsão.

Correlação de *Pearson*: Seja U o conjunto de utilizadores que avaliaram tanto i_a (item activo) como j. A semelhança entre estes dois itens, i_a e j, segundo a medida de Correlação de *Pearson* é dada por [48]:

$$sem(i_a, j) = \frac{\sum_{u \in U}(v_{u,i_a} - \overline{v_{i_a}})(v_{u,j} - \overline{v_j})}{\sqrt{\sum_{u \in U}(v_{u,i_a} - \overline{v_{i_a}})^2 \sum_{u \in U}(v_{u,j} - \overline{v_j})^2}} \tag{3.7}$$

onde $\overline{v_j}$ é a média dos votos no item j.

Medida do Cosseno: Considere-se n o número de utilizadores e m o número de itens na base de dados. A semelhança entre dois itens i_a e j segundo a medida do Cosseno é dada por [48]:

$$sem(i_a, j) = cos(\overrightarrow{i_a}, \overrightarrow{j}) = \frac{\overrightarrow{i_a} \cdot \overrightarrow{j}}{\| \overrightarrow{i_a} \| \times \| \overrightarrow{j} \|} \tag{3.8}$$

onde "$\cdot$"representa o produto interno entre dois vectores (neste caso de dimensão n) e $\| \overrightarrow{a} \|$ representa a norma do vector $\overrightarrow{a}$ ($\| \overrightarrow{a} \| = \|(a_1, a_2, ..., a_n)\| = \sqrt{a_1^2 + a_2^2 + ... + a_n^2}$).

Computação da Previsão - soma pesada: De seguida apresenta-se a fórmula utilizada para determinar a previsão da avaliação do utilizador activo u_a sobre um determinado item i ($p_{u_a,i}$) segundo o trabalho de Sarwar et al. em [48]:

$$p_{u_a,i} = \frac{\sum_{j \in \{\text{Todos os itens semelhantes a } i\}}(sem(i,j) \times v(u_a, j))}{\sum_{j \in \{\text{Todos os itens semelhantes a } i\}} |sem(i,j)|} \tag{3.9}$$

Com esta fórmula tenta-se prever como o utilizador activo avaliaria i. A soma pesada usa os termos com as semelhanças e os votos de u_a para garantir que a previsão fica dentro dos valores possíveis (para os votos).

3.3.3 Limitações

Os sistemas de recomendação por Filtragem Colaborativa não possuem algumas das lacunas que têm os sistemas baseados no conteúdo. Uma vez que os sistemas de Filtragem Colaborativa usam a informação de outros utilizadores, estes podem lidar com qualquer tipo de conteúdo e recomendar quaisquer itens. No entanto, estes sistemas têm as suas próprias limitações [5][26] que se expandem em três dimensões chave: o Problema da Escalabilidade (*the Scalability Problem*), o Problema da Esparsidade (*the Sparsity Problem*) e o Problema da Inicialização (*the Cold-Start Problem*).

3.3.3.1 O Problema da Escalabilidade (*the Scalability Problem*)

Os sistemas de recomendação normalmente são executados num sítio *Web* e podem ser usados por um número muito grande de utilizadores. As recomendações precisam de ser feitas em tempo real e muitas delas podem ser solicitadas ao mesmo tempo [26]. Com milhões de clientes e de produtos, os algoritmos existentes sofrerão problemas sérios em termos de escalabilidade [47]. Consequentemente, para que os algoritmos de recomendação por Filtragem Colaborativa tenham sucesso na *Web* e forneçam recomendações com um atraso aceitável, são necessárias estruturas de dados sofisticadas e arquitecturas avançadas [36].

Para lidar com este problema, Breese et al. em [7] usam redes Bayesianas e Ungar e Foster em [53] usam técnicas de *Clustering*, enquanto que Sarwar et al. [49] aplicam a Decomposição em Valores Singulares *(Singular Value Decomposition (SVD))* para reduzir a dimensão da matriz de utilizadores-itens. *SVD* é uma técnica que aparece

muitas vezes na bibliografia estudada quando se fala em factorização de matrizes [49]. Também é possível lidar com este problema através da redução dos dados ou técnicas de focagem nos dados. Yu et al. [55] e Zeng et al. [56] adoptaram a selecção de exemplos com o intuito de remover os exemplos irrelevantes ou redundantes.

3.3.3.2 O Problema da Esparsidade (*the Sparsity Problem*)

O número de utilizadores e itens na maior parte dos sítios que fazem uso dos sistemas de recomendação é enorme [28]. Em qualquer sistema de recomendação, o número de avaliações já obtidas é usualmente muito pequeno quando comparado com o número de avaliações que necessitam de ser previstas, por isso é importante conseguir-se essas previsões de uma forma eficaz. Como exemplo disso existe um sistema de recomendação de filmes. Muitos filmes podem ter sido avaliados por poucas pessoas e sendo assim serão recomendados muito raramente (mesmo que as poucas avaliações dadas sejam altas). O mesmo acontece com utilizadores cujos gostos são pouco usuais quando comparados com o resto da população e, não tendo utilizadores semelhantes (no caso de sistemas de recomendações baseados nos utilizadores), vai-se ter uma recomendação mais pobre [5]. Este problema, designado por Problema da Esparsidade, tem um impacto negativo na eficácia de um algoritmo de Filtragem Colaborativa.

Uma forma de resolver este problema é usar informação do perfil do utilizador quando se determina a semelhança entre utilizadores. Por exemplo, dois utilizadores podem ser considerados semelhantes não apenas por terem avaliado os mesmos filmes, mas também por pertencerem ao mesmo segmento demográfico [1]. Em [40] é usado o sexo, idade, código da área de localização, educação e informação do emprego dos utilizadores na recomendação de restaurantes. Esta extensão das técnicas tradicionais de Filtragem Colaborativa é por vezes designadas por "Filtragem Demográfica"[40]. Outro algoritmo que também usa as semelhanças entre utilizadores foi proposto em [21], o qual explora as associações entre utilizadores através das suas transacções

passadas e *feedback*. Uma forma diferente de lidar com este problema é usando a redução da dimensão das matrizes das avaliações (SVD), como se pode ver em [6] e [49], técnica que ajuda no Problema da Esparsidade ao remover utilizadores ou itens não representativos ou sem significado de forma a condensar a matriz. Outras soluções passam pela utilização de semelhanças entre itens (algoritmos baseados nos itens), apresentadas a título exemplificativo em [41] e [48].

3.3.3.3 O Problema da Inicialização (*Cold-Start Problem*)

Para ser mais eficiente nas suas previsões, o sistema primeiro tem de conhecer as preferências do utilizador, pois quando o sistema não mostra rápidos progressos, o utilizador pode perder o interesse e deixar de usar o sistema [26].

Problema do Novo Utilizador: De forma a poder fazer recomendações, primeiro o sistema tem de conhecer as preferências do utilizador e para tal é necessário que este tenha avaliado um número suficiente de itens [1]. Sendo assim, um novo utilizador (que tem poucos itens já avaliados ou até mesmo nenhum) não vai conseguir obter recomendações precisas. A próxima secção descreve Sistemas de Recomendação Híbridos (Secção 3.4) que combina várias técnicas de forma a combater esta e outras limitações. Uma solução alternativa é apresentada em [43] e [54], onde várias técnicas são exploradas para determinar o melhor item (isto é, o que traz mais informação ao sistema de recomendação) para um utilizador novo avaliar. Estas técnicas usam estratégias baseadas na popularidade dos itens e entropia entre eles, perfil do utilizador e combinações entre eles [43][54].

Problema do Novo Item: Novos itens vão sendo adicionados regularmente ao sistema de recomendação. O sistema apenas vai estar possibilitado de recomendar novos itens quando estes forem avaliados por um número substancial de utilizadores [1][50]. À semelhança do anterior, este problema normalmente é resolvido usando Sistemas de Recomendação Híbridos (Secção 3.4). Destes temos o exemplo do uso da

informação sobre o conteúdo para preencher a falha entre itens já existentes e os novos, inferindo o valor da semelhança entre eles e assim permitir ao sistema de Filtragem Colaborativa a recomendação ao utilizador em causa de itens semelhantes aos novos que ele avaliou [50].

3.4 Sistemas Híbridos

Os Sistemas Híbridos de Recomendação, são a combinação de duas ou mais técnicas de recomendação de forma a aumentar o seu desempenho, isto é, combater as limitações que estas técnicas apresentam quando usadas individualmente [10][32].

O que acontece com maior frequência é a combinação da Filtragem Colaborativa com outra qualquer técnica de forma a evitar o problema inicial de quando não se têm ainda dados suficientes [10]. Um exemplo desse facto é a junção da Filtragem Colaborativa (versão baseada nos utilizadores) com sistemas de recomendação baseados no conteúdo de modo a combater a incapacidade do primeiro de recomendar itens novos [32]. Em [5] pode-se encontrar a explicação de *Fab*, um sistema que combina esses dois métodos. Para a construção de *Fab* foi usado o perfil dos utilizadores baseado na análise de conteúdo e estes perfis são directamente comparados de forma a determinar utilizadores semelhantes para o método de Filtragem Colaborativa. Assim, são recomendados itens tanto dentro do seu perfil como de acordo com os perfis dos utilizadores que lhes são semelhantes. Outro exemplo de Sistema Híbrido é *EntreeC* [10] que combina a recomendação baseada em conhecimento com a Filtragem Colaborativa para recomendar restaurantes. Esta é uma versão híbrida do sistema *Entree* já apresentado na Secção 3.1. *Tapestry* [19] é um sistema de correio desenvolvido no *Xerox Palo Alto Research Center* que assenta na ideia de que a filtragem de informação pode ser mais eficaz quando os seres humanos estão envolvidos no processo de filtragem. *Tapestry* foi projectado para usar sistemas de recomendação baseados no conteúdo e Filtragem Colaborativa, o que implica que as pessoas colaborem para se ajudarem mutuamente,

pois ao realizarem a filtragem podem associar as suas reacções aos documentos que lêem. As reacções são chamadas de anotações, pois estas podem ser consultadas pelos filtros de pesquisa de outras pessoas.

3.5 Resumo

Neste capítulo apresentam-se os vários Sistemas de Recomendação: Baseados no Conhecimento, Baseados no Conteúdo, Filtragem Colaborativa e Híbridos (mistura de dois ou mais dos anteriores). Os Sistemas de Filtragem Colaborativa podem der divididos em Sistemas Baseados em Modelos e Sistemas Baseados na Memória. Este último, por sua vez, pode ainda ser dividido em Algoritmos Baseados nos Utilizadores (a recomendação é feita com base na semelhança entre utilizadores) e Algoritmos Baseados nos Itens (a recomendação é feita com base na semelhança entre itens). Apresentam-se ainda as limitações dos vários sistemas de recomendação e respectivas soluções encontradas na bibliografia estudada, bem como exemplos dos sistemas de recomendação enunciados. Neste trabalho implementaram-se quatro algoritmos de Filtragem Colaborativa (baseados na memória), em que um é baseado nos utilizadores e os restantes são baseados nos itens (com variações entre cada um deles).

De seguida apresenta-se uma nova abordagem incremental. Esta permite que o algoritmo de recomendação não tenha de ser refeito de cada vez que surge informação nova, bastando apenas actualizá-lo.

Capítulo 4

Sistemas de Recomendação Incrementais

A generalidade dos sistemas de recomendação apresenta custos computacionais, e a dificuldade de implementação cresce de forma não linear com o número de utilizadores e de itens na base de dados. Por isso, de forma a existirem sistemas de recomendação na *Web* com um desempenho aceitável, são necessárias estruturas de dados mais sofisticadas e arquitecturas avançadas [37].

4.1 *Data Streams*

Data streams são um fluxo contínuo de dados [2][4]. Na extracção de conhecimento de tais dados deve-se ter em conta que [2][4]:

- Os dados chegam de modo *on-line*;
- O sistema não tem controle sobre a ordem em que os dados chegam para serem tratados;

- Não se pode rever os exemplos anteriores, pois estes são processados à medida que vão chegando e não podem ser copiados de modo fácil, salvo se forem explicitamente armazenados na memória, que normalmente é pequena quando comparada com o tamanho dos *data streams*;

- Idealmente, produz um modelo semelhante ao obtido no caso dos algoritmos tradicionais de extracção de conhecimento.

Data streams são grandes bases de dados que chegam de forma ininterrupta de tal modo que os dados apenas podem ser analisados sequencialmente e numa única passagem [18]. Como a maioria dos *data streams* desencadeiam dados numa ordem não arbitrária, estes acabam por estar ligados a um aspecto temporal. Consequentemente, os padrões dos dados que poderiam ser descobertos dos dados seguem tendências dinâmicas e, sendo assim, são diferentes dos conjuntos de dados estáticos tradicionais que são muito grandes. Tem-se que a ordem da chegada dos dados influencia a criação de padrões. Por estas razões, mesmo as técnicas que são escalonáveis para conjuntos de dados enormes podem não ser a resposta para extracção de conhecimento de *data streams*, pois esforçam-se sempre para trabalhar no conjunto inteiro de dados sem fazer nenhuma distinção entre dados novos e dados antigos. As apertadas restrições impostas por "apenas o consegues ver uma vez"conduzem ao uso de modelos computacionais e metodologias de validação diferentes, as quais se podem distanciar das usadas na extracção de conhecimento de dados tradicional [35].

Algoritmos que processam *data streams* retornam soluções aproximadas, permitindo uma resposta mais rápida usando menos memória. Estes algoritmos passam da necessidade de uma resposta exacta para uma aproximada com um erro pequeno. Em geral, à medida que o erro diminui a necessidade de recursos computacionais aumenta [18].

Uma vez que o desafio dos modelos de decisão de *Data Mining* é mantê-los permanentemente precisos, então requer-se que os algoritmos de aprendizagem possam modificar o modelo actual sempre que surjam novos dados. Mais ainda, estes algoritmos devem

esquecer informação antiga quando esta está desactualizada. A aprendizagem a partir de *data streams* requer algoritmos de aprendizagem incrementais que tenham em conta a mudança de conceito [18].

É necessário, mas não suficiente, poder actualizar o modelo de decisão sempre que surge nova informação. É também essencial que o modelo tenha a capacidade de esquecer informação passada [23].

A capacidade de actualização pode melhorar a flexibilidade do algoritmo no ajustamento dos dados, mas tem alguns custos inerentes, geralmente medidos em termos de recursos (tempo e memória) [18].

Em [17] pode-se encontrar o sistema *FACIL*, um sistema de classificação incremental baseado em regras de decisão que guarda a informação desactualizada para evitar revisões desnecessárias quando estão presentes mudanças nos dados. Os exemplos são rejeitados quando não descrevem uma decisão de fronteira.

Em [13] apresenta-se o método *Hoeffding*, um método de aprendizagem que usa árvores de decisão de forma a contornar os custos da actualização. Estas podem ser actualizadas em tempo constante para cada exemplo, embora sendo este quase idêntico ao resultado produzido pelas árvores de decisão convencionais na presença de um número suficiente de exemplos. A probabilidade de as árvores *Hoeffding* e as convencionais virem a escolher testes diferentes num dado nó diminui exponencialmente com o aumento do número de exemplos.

4.2 Filtragem Colaborativa Incremental

Filtragem Colaborativa Incremental é um sistema de recomendação de Filtragem Colaborativa baseado na actualização das semelhanças o que possibilita a recomendação de itens de uma forma mais rápida que a Filtragem Colaborativa clássica mantendo a qualidade da recomendação [38] e perante um fluxo contínuo de dados (*data streams*).

No sistema de recomendação por Filtragem Colaborativa Incremental baseado nos utilizadores apresentado em [38], as operações sobre vectores (que apresentam custos computacionais elevados) são substituídas por operações escalares, o que permite acelerar a computação de matrizes utilizador-item de grandes dimensões. Sempre que um utilizador se submete a uma nova avaliação ou actualiza o valor de avaliações anteriores, o seu índice de semelhança para com os restantes utilizadores deve ser recalculado. Este modelo faz esta actualização de forma incremental, usando sempre o valor antigo do índice para o cálculo do novo. Neste trabalho, apresentar-se-á, também, um algoritmo de Filtragem Colaborativa Incremental baseado nos itens. Este funciona com base no mesmo princípio explicado anteriormente mas pensado em termos da semelhança entre itens sem a necessidade de guardar sempre em memória todas as sessões dos utilizadores anteriores.

4.3 Algoritmo Incremental de M. Papagelis

Nesta secção é apresentado o método de Filtragem Colaborativa Incremental baseado nos utilizadores desenvolvido por Papagelis et al. em [38].

Passe-se então à explicação de alguns conceitos usados neste método, bem como do próprio método.

A medida de semelhança entre dois utilizadores u_x e u_y é dada pela equação já apresentada na secção 3.3.2.1:

$$sem(u_x, u_y) = \frac{\sum_{h \in H}(v_{u_x,i_h} - \overline{v_{u_x}})(v_{u_y,i_h} - \overline{v_{u_y}})}{\sqrt{\sum_{h \in H}(v_{u_x,i_h} - \overline{v_{u_x}})^2}\sqrt{\sum_{h \in H}(v_{u_y,i_h} - \overline{v_{u_y}})^2}} \tag{4.1}$$

onde H representa o subconjunto dos índices dos itens que tanto u_x como u_y avaliaram.

Sempre que um utilizador u_x submete uma nova avaliação ou actualiza uma avaliação já feita anteriormente, os valores da semelhança com os restantes utilizadores têm de ser novamente calculados. Papagelis et al. determinaram a nova semelhança partindo do

valor antigo. Assim, adoptaram assim a seguinte notação para a medida de semelhança da equação 4.1:

$$A = \frac{B}{\sqrt{C}\sqrt{D}} \Rightarrow A = sem(u_x, u_y), B = \sum_{h \in H}(v_{u_x,i_h} - \overline{v_{u_x}})(v_{u_y,i_h} - \overline{v_{u_y}}),$$

$$C = \sum_{h \in H}(v_{u_x,i_h} - \overline{v_{u_x}})^2, D = \sum_{h \in H}(v_{u_y,i_h} - \overline{v_{u_y}})^2 \tag{4.2}$$

Sejam B', C', D' os novos valores dos factores B, C, D, ou seja, os valores dos factores B, C, D no caso da nova semelhança A', então considere-se:

$$A' = \frac{B'}{\sqrt{C'}\sqrt{D'}} \Rightarrow A' = \frac{B+e}{\sqrt{C+f}\sqrt{D+g}}, B' = B + e, C' = C + f, D' = D + g \tag{4.3}$$

onde e, f, g são os incrementos a serem calculados após a submissão de uma nova avaliação ou da actualização de uma avaliação já existente. B', C' e D' são expressos usando os valores anteriores de B, C e D e os respectivos incrementos e, f e g. No entanto, para que seja possível a computação incremental dos novos valores da semelhança com operações com menos custos, houve a necessidade de aplicar uma estrutura de dados apropriada. Definiram um esquema de *cache* usando tabelas na base de dados, o que lhes permitiu guardar os valores de B, C e D para todos os pares de utilizadores. Além disso, guardaram, também a média das avaliações e o número de itens que cada utilizador avaliou. A informação guardada necessita de ser actualizada depois da submissão de uma nova avaliação ou da actualização de uma avaliação já feita anteriormente, sendo sempre necessário o conhecimento das sessões dos utilizadores anteriores para a actualização do que é salvo em *cache*.

4.4 Resumo

Neste capítulo, apresentou-se o princípio que está na base da Filtragem Colaborativa Incremental.

Sendo o desafio dos modelos de decisão de *Data Mining*) mantê-los permanentemente precisos, então torna-se necessário que os algoritmos de aprendizagem possam modificar o modelo actual sempre que surgem novos dados. Estes algoritmos devem também esquecer informação antiga quando esta está desactualizada. A aprendizagem a partir de *data streams* requer algoritmos de aprendizagem incrementais que tenham em conta a mudança de conceito [18].

Segundo H. Chen e M. Chau, em [33], "nos cenários de dados contínuos, um sistema de recomendação deve lidar com um fluxo enorme de dados do utilizador sob memória e tempo restritos".

Neste capítulo foi também apresentado o estudo feito por Papagelis et al. em sistemas de recomendação por Filtragem Colaborativa Incremental baseados nos utilizadores.

De seguida, são apresentados cada um dos quatro algoritmos testados, nas versões incremental e não incremental, baseados na semelhança entre itens ou entre utilizadores e fazendo ou não uso de técnicas de trabalho com matrizes esparsas disponíveis no *software* R.

Capítulo 5

Um Sistema de Recomendação Incremental

Quer sejam baseados nos utilizadores ou baseados nos itens (algoritmos baseados na memória), a Filtragem Colaborativa tem por base um conjunto de avaliações (*ratings*) dadas pelos utilizadores a determinados itens. Estas avaliações podem assumir valores num determinado conjunto (por exemplo quando os utilizadores atribuem opinião sobre filmes que viram). No entanto, na realidade dos sistemas de recomendação para a *Web*, a única informação disponível é se um dado utilizador acedeu a este ou àquele item (por exemplo uma página *Web*). Estes dados são processados como dados implícitos binários. Neste trabalho, usa-se apenas estes dados binários de acessos *Web*.

Neste capítulo, apresentam-se os algoritmos em estudo. O código R implementado pode ser consultado no Anexo A.

Todos os algoritmos usados começam com um conjunto de dados D de pares $< u, i >$, onde cada par significa que o utilizador u viu o item i. Todos os algoritmos requerem a computação da distância entre cada par de itens ou distâncias entre cada par de utilizadores. Em cada caso, tal informação é guardada numa matriz designada por matriz de semelhanças S. Sendo assim, todos os algoritmos apresentados começam

construindo S. É necessário definir-se determinados parâmetros, entre eles: $Neib$, o número de vizinhos a serem encontrados para um dado utilizador/item, e N, o número de recomendações que vão ser feitas. Torna-se necessário, também, salvar o conjunto de $Utilizadores$ conhecidos e o conjunto de $Itens$ conhecidos.

No caso dos algoritmos baseados nos itens, a equação 3.8, que define a medida do Cosseno para a semelhança entre dois itens, pode ser simplificada, ao trabalhar-se com dados binários implícitos, para:

$$sem(i,j) = S_{i,j} = \frac{\#(U_i \cap U_j)}{\sqrt{\#U_i} \times \sqrt{\#U_j}} \tag{5.1}$$

onde U_i (respectivamente U_j) designa o conjunto de utilizadores que avaliaram i (respectivamente j) e $\#U_i$ (respectivamente $\#U_j$) designa a cardinalidade do conjunto U_i (respectivamente U_j).

De igual modo, e partindo da equação 3.4, que define a medida do Cosseno para a semelhança entre dois utilizadores, obtém-se

$$sem(u,q) = S_{u,q} = \frac{\#(I_u \cap I_q)}{\sqrt{\#I_u} \times \sqrt{\#I_q}} \tag{5.2}$$

onde I_u (respectivamente I_q) designa o conjunto de itens que avaliaram u (respectivamente q) e $\#I_u$ (respectivamente $\#I_q$) designa a cardinalidade do conjunto I_u (respectivamente I_q).

5.1 Algoritmo baseado nos itens

Este é o sistema de recomendação de Filtragem Colaborativa não incremental baseado nos itens (**FCBI**) e que serve de base (ou referência) para os restantes aquando da avaliação dos resultados. Este segue os seguintes passos:

- Construir matriz de semelhanças entre itens S

- Dada uma nova sessão (do utilizador activo u_a)

  - Determinar o peso de activação (Equação 5.3) de cada item que nunca foi visto por u_a

  - Recomendar a u_a os N itens com maior peso de activação

O peso de activação de um item é calculado determinando os $Neib$ itens mais semelhantes (vizinhos) a i. O peso de activação de i é:

$$W(i) = \frac{\sum_{\text{itens da vizinhança vistos pelo user activo}} S[i,.]}{\sum_{\text{itens da vizinhança}} S[i,.]} \tag{5.3}$$

Se o conjunto de vizinhos de i for vazio, então o peso de activação de i é 0. O mesmo acontece quando não existe qualquer item da vizinhança que tenha sido visto por u_a na sessão activa.

5.2 Trabalhar com matrizes esparsas

Este algoritmo (**FCBIE**) é semelhante a FCBI mas usa estruturas de dados que gastam menos memória aquando do trabalho com matrizes esparsas. Isto é motivado pela necessidade de escalabilidade. O uso de estruturas simples de matrizes não é sustentável para um número grande de utilizadores/itens.

5.3 Algoritmo incremental baseado nos itens

O algoritmo que aqui se apresenta, **FCBIEInc**, é a versão incremental de FCBIE. Todas as matrizes são guardadas em estruturas adequadas para matrizes esparsas. Neste algoritmo incremental, para além da matriz de semelhança S também é salvo em memória a matriz Int com a frequência observada para cada par de itens, isto é, $Int_{i,j}$ é o número de utilizadores que viram ambos os itens i e j. A diagonal principal

desta matriz representa o número de utilizadores que avaliaram cada item. Esta matriz auxiliar é necessária (e suficiente) para actualizar S incrementalmente. Este algoritmo pode ser descrito do seguinte modo:

- Construir matriz de semelhanças entre itens S
- Construir a matriz das frequências de cada par de itens em $Itens$, Int
- Para cada utilizador de teste (utilizador activo, u_a)

  - Determinar o peso de activação (Equação 5.3) de cada item nunca visto por u_a

  - Recomendar a u_a os N itens com maior peso de activação

  - Actualizar a matriz de semelhanças e a matriz Int

De modo análogo ao apresentado anteriormente, se o conjunto de vizinhos de i for vazio, então o peso de activação de i é 0. O mesmo acontece quando não existe qualquer item da vizinhança que tenha sido visto por u_a na sessão activa.

A actualização da matriz de semelhanças no algoritmo baseado nos itens e incremental FCBIEInc é feita do seguinte modo:

- Seja I o conjunto de itens que u_a avaliou na sessão activa
- Adicionar u_a à lista de $Utilizadores$
- Adicionar a $Itens$ os itens em I que não pertenciam a $Itens$ (isto é, os itens novos vistos por u_a)
- Adicionar, para cada novo item, uma linha e uma coluna a Int e a S
- Para cada par de itens em I, (i, j), actualizar $Int_{i,j}$ para $Int_{i,j} + 1$

- Para cada item i_a em I actualizar a linha (coluna) correspondente de S, usando a Equação 5.4 obtida a partir da Equação 5.1:

$$S_{i_a,\cdot} = \frac{Int_{i_a,\cdot}}{\sqrt{Int_{i_a,i_a}} \times \sqrt{Int_{\cdot,\cdot}}} \tag{5.4}$$

Posto isto, podemos concluir que, para os itens já existentes em S antes desta sessão, apenas se actualiza o valor da semelhança. Já para os itens novos, é necessário acrescentar uma linha e uma coluna por cada novo item e calcular a sua semelhança tendo apenas em conta a sessão actual (visto que é a primeira vez que foram vistos). Para os pares de itens que não pertencem a I a semelhança é 0.

5.4 Algoritmo incremental baseado nos utilizadores

O algoritmo que agora se apresenta, **FCBUEInc**, também é incremental e faz uso das estruturas de dados apropriadas para trabalhar com matrizes esparsas mas é baseado nos utilizadores. É inspirado no algoritmo proposto em [38] (ver Secção 4.3), mas, no âmbito deste trabalho, houve a necessidade de fazer adaptações de forma a poder lidar com dados binários (visto que em [38] trabalha-se com dados explícitos). Para além da matriz de semelhanças S também se guarda em memória a matriz $Int.u$, com o número de itens vistos por cada par de utilizadores, e a base de dados D. De modo análogo ao algoritmo baseado nos itens, a diagonal principal de $Int.u$ representa o número de itens que são avaliados por cada utilizador. O algoritmo implementado segue os seguintes passos:

- Construir a matriz de semelhanças entre utilizadores S
- Construir a matriz das frequências de cada par de utilizadores em $Utilizadores$, $Int.u$
- Dada uma nova sessão (do utilizador activo, u_a)

- Actualizar $Int.u$ e a matriz de semelhanças S

- Determinar o peso de activação (Equação 5.6) de cada item nunca visto por u_a

- Recomendar a u_a os N itens com maior peso de activação

A actualização de $Int.u$ e S é feita do seguinte modo:

- Seja I o conjunto de itens na sessão activa
- Adicionar u_a a $Utilizadores$
- Adicionar a sessão activa a D
- Se u_a é um novo utilizador, adiciona-se uma linha e uma coluna a $Int.u$ e a S
- A linha e a coluna de $Int.u$ que correspondem a u_a são actualizadas usando a nova base de dados D
- Adicionar a $Itens$ os itens em I que não pertenciam a $Itens$ (isto é, os itens novos vistos por u_a)
- Actualizar a linha (respectivamente coluna) de S que corresponde ao utilizador u_a, usando a Equação 5.5, obtida a partir da Equação 5.2:

$$S_{u_a,.} = \frac{Int.u_{u_a,.}}{\sqrt{Int.u_{u_a,u_a}} \times \sqrt{Int.u_{.,.}}} \tag{5.5}$$

Note-se que, para a actualização da coluna de S, tem-se uma equação análoga à anterior mas considerando $S_{.,u_a}$

O peso de activação de um item para o algoritmo baseado nos utilizadores é [36]:

$$W(i) = \frac{\sum_{\text{todos os utilizadores na vizinhança de } u_a \text{ que avaliaram } i} S[u_a, .]}{\sum_{\text{todos os utilizadores na vizinhança de } u_a} S[u_a, .]} \tag{5.6}$$

Se o conjunto de vizinhos de u_a for vazio, então o peso de activação de qualquer item é 0. Esta também é 0 quando não existe qualquer utilizador da vizinhança de u_a que tenha visto o item i.

5.5 Complexidade Computacional

A complexidade computacional para a fase de recomendação de todos os algoritmos baseados nos itens (FCBI, FCBIE e FCBIEInc) é $O(n.k)$, em que n é o número de itens e k é o tamanho da sessão activa. Para cada um dos k itens na sessão activa é calculado o peso de activação. A recomendação no caso do algoritmo FCBUEInc (baseado nos utilizadores) tem uma complexidade computacional de $O(max(m, n))$, em que m é o número de utilizadores. Dado um utilizador activo u_a, processam-se os m utilizadores para localizar os vizinhos de u_a e depois os n itens para calcular o seu peso de activação. Assim, este processo tem uma complexidade que é dominada pelo maior dos dois, m e n.

No que diz respeito à actualização, no algoritmo baseado nos itens FCBIEInc, para actualizar Int é necessário fazer $O(k^2)$ operações pois actualiza-se a semelhança entre cada um dos k itens da sessão activa e os n itens do conjunto $Itens$. Isto leva a uma complexidade computacional de $O(k(k + n))$. Esta é igual a $O(n.k)$ se se assumir que $k << n$, o que é razoável. Para actualizar a matriz de semelhanças entre utilizadores, é necessário actualizar D com a informação da nova sessão ($O(m.k)$) e $Int.u$. Para actualizar esta última matriz considera-se cada utilizador e actualiza-se a frequência da intersecção que diz respeito ao utilizador activo. Esta operação tem novamente uma complexidade de $O(m.k)$.

5.6 Implementação

Todos os algoritmos foram implementados em R [42] e o código implementado pode ser consultado no Anexo A. Para as matrizes esparsas tentaram-se três diferentes *packages* do R disponíveis para trabalhar com este tipo de matrizes: *SparseM*, *spam* e *Matrix*. Apenas se mostram os resultados obtidos com a *package spam*, pois foi a que apresentou tempos de processamento mais curtos. Assume-se que o uso de outra

linguagem de programação iria manter a proporção dos tempos usados pelos diferentes algoritmos.

5.7 Resumo

Neste capítulo apresentaram-se os quatro algoritmos a partir dos quais foram realizadas as experiências descritas no capítulo seguinte e das quais se tirou algumas conclusões. Estes são sistemas de recomendação por Filtragem Colaborativa nas suas duas vertentes de algoritmos baseados na memória: baseados nos itens e baseados no utilizadores.

De seguida são apresentadas as avaliações experimentais feitas bem como os resultados obtidos e conclusões mais relevantes para o estudo em causa.

Capítulo 6

Avaliações Experimentais e Resultados

Neste capítulo, descreve-se a avaliação experimental feita com o intuito avaliar tanto o desempenho computacional como a capacidade predictiva dos quatro algoritmos apresentados na Secção 5. Pretende-se tirar conclusões sobre o impacto da incrementalidade, analisar a recomendação baseada nos utilizadores *vs.* baseada nos itens e discutir o uso de técnicas que manipulam matrizes esparsas. Usou-se, de projectos associados ao LIAAD, cinco conjuntos de dados com acessos *Web*. Estes dados vêm de quatro fontes diferentes: um sítio de *e-learning* de um curso universitário de computadores (ZP), um portal para profissionais com interesse em Economia e Gestão (PE), um sítio *Web* português que promove o desenvolvimento dos recursos humanos portugueses (EGOV) e um sítio *Web* de uma loja de informática *on-line* portuguesa (ECOM). Nos conjuntos de dados do ZP e do PE, numa primeira fase, reduziu-se o volume de dados ao considerar-se apenas os itens mais vistos pelos utilizadores. De seguida, procurou-se ter subconjuntos de dados com diferentes proporções de utilizadores e itens. Os dados ZP originais têm 682 utilizadores/sessões e 500 itens. Destes retirou-se as sessões com apenas um acesso e com mais de 20. A partir dos dados de PE (originalmente

com 1580 utilizadores/sessões e 8818 itens) construiu-se dois subconjuntos de dados mais pequenos (PE200 e PE802). Os conjuntos de dados EGOV (originalmente com 38371 utilizadores/sessões e 173 itens diferentes) e ECOM (originalmente com 10864 utilizadores/sessões e 2387 itens) foram também trabalhados. Nestes dados, à semelhança dos dados ZP, retirou-se as sessões com apenas um acesso e com mais de 20. As características de cada um dos conjuntos de dados encontram-se na Tabela 6.1.

Conj. de dados	**#Utilizadores**	**#Itens**	**#Transacções**	**Tamanho médio das transacções**	**Densidade (%)**
ZP	509	295	2646	5,2	1,8 %
PE200	200	199	2042	10,2	5,1 %
PE802	802	100	6070	7,6	7,6 %
EGOV	1244	133	4047	3,3	2,4 %
ECOM	413	335	1409	3,4	1 %

Tabela 6.1: Descrição dos conjuntos de dados utilizados.

- **# Transacções :** Número diferente de pares $< utilizador, item >$ considerados (nos conjuntos de dados considerados não existe repetição de itens, pois cada sessão é considerada um novo utilizador);
- **Tamanho médio das transacções :** Número médio de itens que cada utilizador viu/avaliou;
- **Densidade (%) :** Considere-se a matriz $utilizador \times item$ em que cada entrada da matriz pode tomar valor "1"ou "0"consoante *utilizador* tenha visto/avaliado *item* ou não, respectivamente. Considerou-se a densidade do conjunto de dados como sendo a proporção de "1"no conjunto total de entradas na matriz. Nos conjuntos de dados trabalhados, considera-se que a Densidade é dada, em percentagem, por $\frac{\#\ Transacções}{\#Utilizadores \times \#Itens} \times 100\%$.

Conj. de dados	***Split***	**#Utilizadores iniciais do conjunto de Treino**	**#Itens iniciais do conjunto de Treino**
ZP	0,2	101	149
ZP	0,4	203	211
ZP	0,6	305	258
ZP	0,8	407	282
PE200	0,2	40	186
PE200	0,4	80	199
PE200	0,6	120	199
PE200	0,8	160	199
PE802	0,2	160	100
PE802	0,4	320	100
PE802	0,6	481	100
PE802	0,8	641	100
EGOV	0,2	248	117
EGOV	0,4	497	122
EGOV	0,6	746	125
EGOV	0,8	995	128
ECOM	0,2	82	151
ECOM	0,4	165	246
ECOM	0,6	247	288
ECOM	0,8	330	315

Tabela 6.2: Descrição dos conjuntos de dados utilizados para os vários *Splits* estudados.

Na Tabela 6.2 pode-se consultar, para cada conjunto de dados e *Split* (percentagem que separa o conjunto inicial de dados em conjunto de treino e teste), o número inicial de utilizadores e itens existente no conjunto de treino. A diferença entre # *Utilizadores* e # *Utilizadores iniciais do conjunto de Treino* indica o número de utilizadores de teste, isto é, os utilizadores novos aos quais foram feitas recomendações (após o conhecimento de parte da sessão "percorrida"pelos mesmos). Quanto à diferença no número de itens, esta permite saber quantos itens novos surgem com os utilizadores de teste e suas sessões. Note-se que por cada utilizador (respectivamente item) novo é acrescentada nova linha e coluna à matriz de semelhanças do algoritmo incremental baseado nos utilizadores, FCBUEInc (respectivamente nos itens, FCBIEInc).

O código R, implementado e usado para obtenção dos resultados dos quais se tiram conclusões neste trabalho, pode ser visto no Anexo A.

6.1 Metodologia de Avaliação

Cada conjunto de dados foi separado em conjunto de treino, conjunto de teste e conjunto oculto seguindo o protocolo de *all-but-one* [7]. As percentagens de divisão (*Splits*) usadas para a separação do conjunto de dados inicial em dados de treino e dados de teste foram *0,8* (80 por cento das sessões vão para o conjunto de treino e as restantes *20* por cento são sessões de teste), *0,6*, *0,4* e *0,2*. De cada sessão de teste um item é ocultado. O teste é feito comparando os itens ocultados com as recomendações feitas por cada um dos algoritmos . O número de vizinhos *Neib* usado foi de *3,5* e *10*.

6.2 Medidas

Para comparar a eficiência computacional dos algoritmos, mediu-se o tempo real de recomendação e também se comparou este (tempo relativo) com o Algoritmo FCBI (não

incremental e sem estruturas especiais de tratamento de matrizes esparsas). Mediu-se, igualmente, o tempo de actualização (para o caso dos algoritmos incrementais) e o tempo de construção da matriz inicial das semelhanças. Para estudar e comparar a capacidade predictiva dos algoritmos usou-se as medidas de *Recall* e *Precision*.

Seja *conj.acertos* o conjunto com os acertos do sistema de recomendação para um dado utilizador u; *conj.oculto* o conjunto com os pares $< u, item >$ que estão ocultos na sessão de u; e *conj.rec.* o conjunto das recomendações feitas a u pelo sistema de recomendação.

- ***Recall*** $= \frac{\#\text{ conj.acertos}}{\#\text{ conj.oculto}}$ mede a capacidade de recomendar tudo o que é relevante para o utilizador (**Sensibilidade**).
- ***Precision*** $= \frac{\#\text{ conj.acertos}}{\#\text{ conj.rec.}}$ mede a capacidade de recomendar apenas o que é relevante para o utilizador, deixando de fora o que ele provavelmente não irá ver/avaliar (**Precisão**).
- **Tempo**: Mede-se o tempo necessário para construir a matriz de semelhanças em cada método e o tempo médio por utilizador para fazer uma recomendação e para a actualização da matriz de semelhanças.

Neste estudo, para cada N, considerou-se *Recall* e *Precision* como sendo a média dos valores obtidos para cada utilizador de teste.

No trabalho em questão, tem-se que *Recall* é $N \times Precision$, em que N é o número de recomendações feitas a cada utilizador, pois para cada utilizador de teste apenas se esconde um item por ele visto, ou seja $\#conj.rec = N \times 1 = N \times \#conj.oculto$. Sendo assim, apenas se apresenta os resultados obtidos para a medida *Recall*.

De seguida, apresenta-se alguns dos gráficos relativos aos resultados que se obteve durante este trabalho e que envolvem as várias medidas estudadas. Estes são importantes para as conclusões finais, uma vez que comprovam/reforçam o que se enuncia nos capítulos anteriores.

6.3 Tempo

Mostra-se resultados do tempo médio de recomendação (o tempo que o algoritmo leva, em média, para fornecer recomendações dada uma sessão inicial) e de actualização ou construção da matriz de semelhanças. A construção da matriz é feita apenas uma vez no início de cada algoritmo, ao passo que a actualização é feita apenas nos algoritmos incrementais (FCBIEInc e FCBUEInc) de cada vez que surge uma nova sessão (sessão referente ao utilizador activo - u_a).

Os tempos reais medidos foram obtidos com o computador a funcionar em exclusivo nestas experiências. Esta é uma estimativa pessimista mas que permite simular o tempo real de resposta.

De seguida, apresenta-se as conclusões relativas ao impacto no tempo, de recomendação e actualização, do uso de técnicas de manuseamento de matrizes esparsas e a incrementalidade nos sistemas de recomendação por Filtragem Colaborativa, bem como o impacto da variação do conjunto de dados e das variáveis *Split* e *Neib*.

6.3.1 Tempo de Recomendação

6.3.1.1 Trabalhar com matrizes esparsas: tempo relativo

Em relação ao tempo de recomendação estuda-se tanto o tempo real como a proporção em relação ao tempo de recomendação de FCBI (tempo relativo). Os tempos apresentados dizem respeito aos obtidos usando o R, pelo que estes podem sofrer alterações no tempo com o uso de outro *software*. Em todas as figuras, pode-se verificar que o uso de *packages* de tratamento de matrizes esparsas tem um custo elevado em termos do tempo de recomendação. Das Figuras 6.1 e 6.2, pode-se tirar conclusões gerais em relação aos tempos relativos de recomendação. O Algoritmo FCBIE (algoritmo não incremental e que usa as técnicas de trabalho com matrizes esparsas) é 3 a 7 vezes

mais lento que FCBI (não incremental e não usa as *packages* de tratamento de matrizes esparsas) quando está a recomendar. Este é um preço a pagar-se devido à limitação de memória. Note-se que em geral estas matrizes são de grandes dimensões, pelo que é necessária muita memória para as guardar, o que nem sempre pode acontecer. Além disso, melhores implementações de técnicas de tratamento de matrizes esparsas podem reduzir estas diferenças de tempos de recomendação.

6.3.1.2 Impacto da incrementalidade: tempo relativo

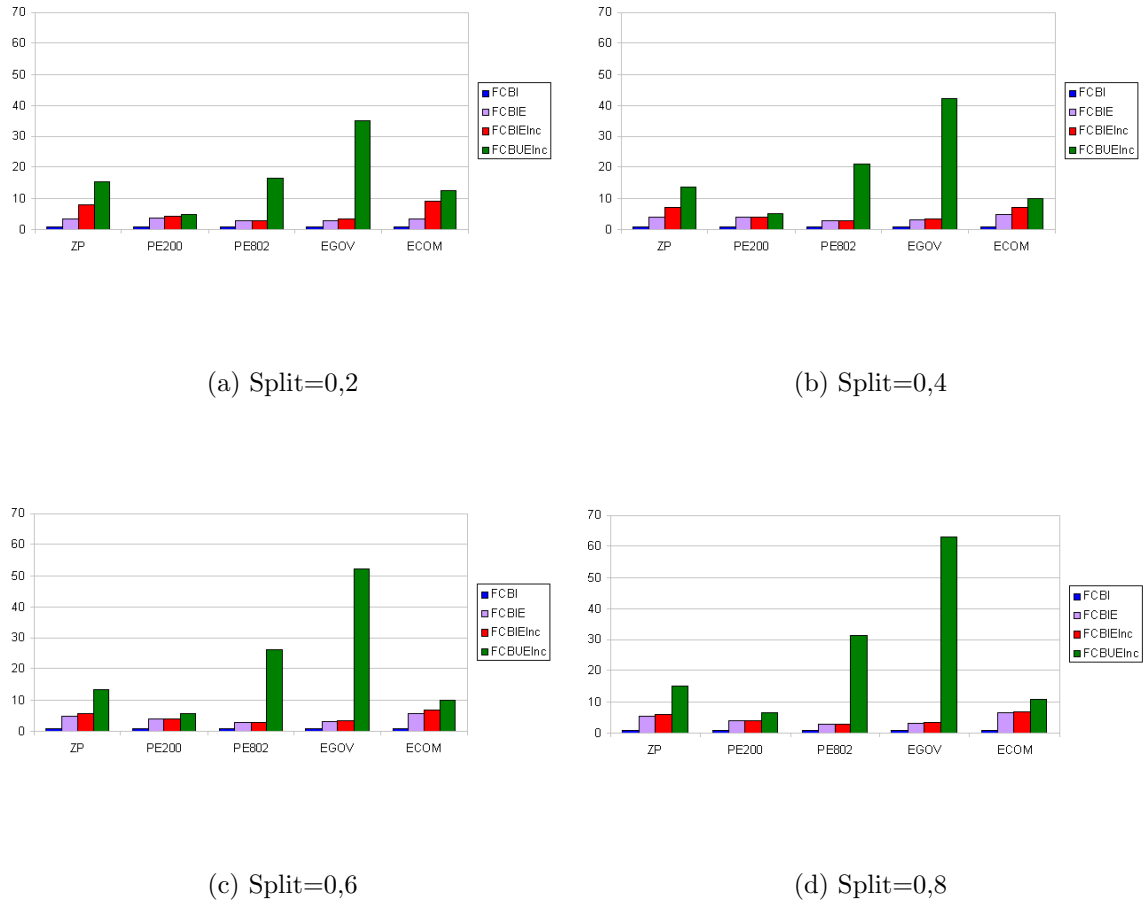
Comparando a versão incremental (FCBIEInc) com a não incremental (FCBIE) do algoritmo baseado nos itens, pode-se verificar que o tempo relativo de recomendação do primeiro é semelhante ou ligeiramente superior ao segundo. Em relação aos algoritmos incrementais, baseado nos itens (FCBIEInc) e baseado nos utilizadores (FCBUEInc), verifica-se maiores diferenças na proporção de tempo relativo de recomendação, sendo o de FCBUEInc superior ao de FCBIEInc. Isto é mais notório para ZP, PE802 e EGOV (dados com um número de utilizadores superior ao número de itens), ao passo que, para os conjuntos de dados com um número similar de utilizadores e itens (PE200 e ECOM), as diferenças na proporção de tempo relativo já são bem menores. Note-se que, como se disse na Secção 5.5, o tempo de recomendação para o algoritmo baseado nos utilizadores é dominado pelos grandes valores de m (número de utilizadores) e n (o número de itens), enquanto que o tempo de recomendação do algoritmo baseado nos itens é dominado apenas por n.

6.3.1.3 Variação com o *Split*

Da Figura 6.1 pode-se tirar ainda conclusões sobre a variação do tempo relativo de recomendação com o *Split*. Este estudo é importante para avaliar o impacto da variação da dimensão do conjunto inicial de dados. Com o aumento do *Split* verifica-se um aumento significativo do tempo relativo do algoritmo FCBUEInc para os dados

PE802 e EGOV (conjuntos de dados com maior número de utilizadores e transacções e, logo, maior conjunto de teste), não sofrendo grandes alterações no caso dos restantes conjuntos de dados e algoritmos. Por conseguinte, a variação do *Split* apenas tem impacto nos conjuntos de dados com muitos utilizadores e apenas para o algoritmo (incremental) baseado nos utilizadores.

Os gráficos referentes aos restantes valores de $Neib$ encontram-se no Anexo B na Secção B.1.



(a) Split=0,2

(b) Split=0,4

(c) Split=0,6

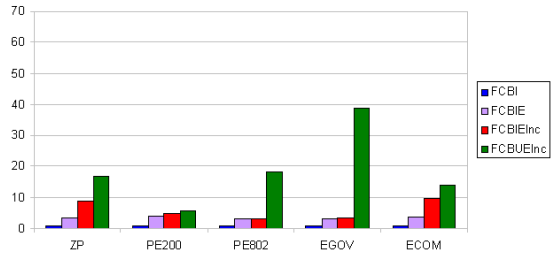
(d) Split=0,8

Figura 6.1: *Tempo (relativo) de recomendação usando o tempo de FCBI como referência para Neib=5*

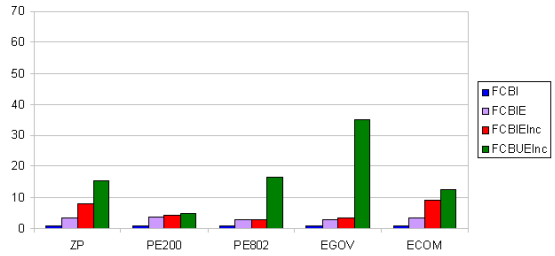
6.3.1.4 Variação com o *Neib*

Quando se estuda a variação do tempo relativo de recomendação com o número de vizinhos, *Neib* (Figura 6.2), depreende-se que este diminui com o aumento do *Neib* (não considerando FCBI pois é o tempo de referência).

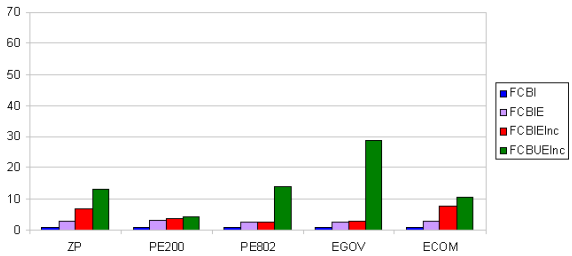
Os gráficos referentes aos restantes valores de *Split* encontram-se no Anexo B na Secção B.2.



(a) Neib=3



(b) Neib=5



(c) Neib=10

Figura 6.2: *Tempo (relativo) de recomendação usando o tempo de FCBI como referência para Split=0,2*

6.3.1.5 Variação com o Conjunto de Dados

De seguida, avalia-se os resultados tendo em conta o tempo médio real de recomendação (em segundos). Observando a Figura 6.3, pode-se ter uma visão geral por conjunto de dados desse mesmo tempo. Para cada algoritmo o tempo real de recomendação aumenta de acordo com o aumento do *Split*, sendo o aumento diferente consoante o conjunto de dados bem como o *Split*. Conclui-se, então, que, quanto maior for a matriz de semelhanças (inicial e/ou actualizada), maior é o tempo de recomendação para cada um dos algoritmos. No entanto verifica-se que para FCBI o tempo mantém-se praticamente constante. O mesmo acontece para FCBIE e FCBIEInc para PE200 (Figura 6.3(b)), PE802 (Figura 6.3(c)) e EGOV (Figura 6.3(d)). Desta forma, pode-se concluir que com o aumento do *Split* o tempo médio real de recomendação mantém-se ou aumenta (o que já era esperado), nunca diminuindo. Comparando os tempos para cada conjunto de dados, verifica-se que o algoritmo baseado nos utilizadores (FCBUEInc) tende a ser mais lento que os restantes aquando da recomendação. Esta diferença não é tão notória no conjunto de dados PE200 e ECOM, onde o número de utilizadores é idêntico ao número de itens (o que não é realista, pois, em geral, tem-se mais utilizadores do que itens). Verifica-se também que o tempo médio real de recomendação por utilizador dos algoritmos baseados nos itens (incremental e não incrementais) apresenta maior variação no caso de ZP e ECOM (Figuras 6.3(a) e 6.3(e), respectivamente), onde a densidade é menor, ou seja, a matriz de semelhanças tem mais zeros pelo que os conjuntos dados são mais esparsos. Pode-se ainda afirmar que, ao usar técnicas de manuseamento de matrizes esparsas, o tempo médio real de recomendação aumenta, o que indica, mais uma vez, que, ao ganhar-se na memória perde-se no tempo. Cabe, assim, ao proprietário da página *Web* ter em conta qual o aspecto mais importante, tempo ou memória, bem como o que ganha ao prescindir de um em favor do outro.

6.3.1.6 Trabalhar com matrizes esparsas: tempo real

Na Figura 6.4(a) avalia-se o que se ganha/perde com a incrementalidade nos algoritmos baseados nos itens e verifica-se que para todos os conjuntos de dados, o tempo de FCBIE é inferior ao de FCBIEInc, sendo que com o aumento do *Split* o tempo médio real de recomendação de FCBIE tende a aproximar-se do tempo de FCBIEInc. Ou seja, quanto menor for o *Split* o tempo médio real de recomendação no algoritmo incremental vai sendo maior que o tempo do algoritmo não incremental. Isto deve-se ao facto de a redução do *Split* implicar a redução do conjunto de treino e, consequentemente, um número maior de novas sessões.

6.3.1.7 Impacto da incrementalidade: tempo real

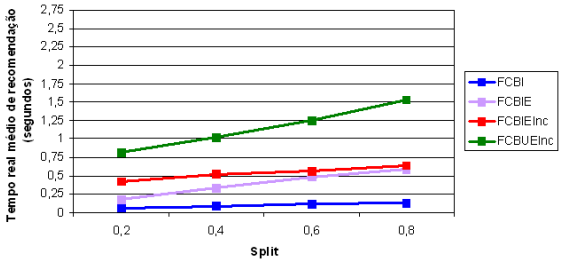
Ao comparar-se os algoritmos incrementais baseado nos itens e baseado nos utilizadores (gráfico 6.4(b)), pode-se concluir que FCBUEInc apresenta piores resultados para EGOV e PE802 (conjuntos de dados com mais utilizadores). Pode-se ainda afirmar que com o aumento do *Split* o tempo médio real de recomendação de FCBUEInc tende a afastar-se cada vez mais do tempo de FCBIEInc para os conjuntos de dados já indicados anteriormente.

6.3.1.8 Número de utilizadores e número de itens

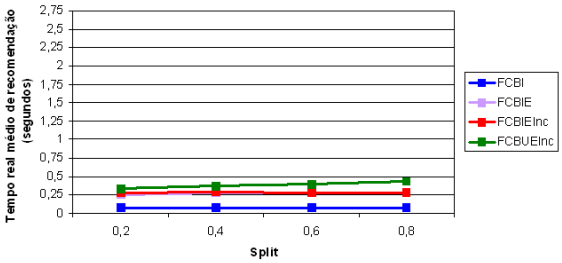
Ao tomar-se $Split$=0,4 e $Neib$=5, construiu-se os gráficos da Figura 6.5 nos quais se relaciona o número total de utilizadores e o número total de itens para cada algoritmo, com o tempo médio real, por utilizador, para a recomendação de 5 itens. Nos gráficos, o raio da bola é proporcional ao respectivo tempo de recomendação. Para FCBI, o tempo considerado está em centésimos de segundo e, para os restantes, está em décimos de segundo. Isto deve-se ao facto de o tempo do primeiro ser inferior aos restantes. Usando a mesma unidade de tempo que os algoritmos seguintes, percebeu-se

que não ia ser possível visualizar, com nitidez, o respectivo gráfico. Analisando agora os gráficos obtidos, conclui-se o mesmo que se enuncia na Secção 5.5 sobre a complexidade computacional dos algoritmos. Verifica-se que, para FCBUEInc (algoritmo incremental baseado nos utilizadores - Figura 6.5(d)), o tempo de recomendação varia tanto de acordo com o número de utilizadores como com o número de itens, sendo maior o tempo para conjuntos de dados com números maiores de utilizadores e/ou itens. No entanto, no caso dos algoritmos baseados nos itens (incremental ou não), Figuras 6.5(a), 6.5(b) e 6.5(c), verifica-se apenas uma variação do tempo médio real de recomendação com o número total de itens, em que quanto maior é o número de itens, maior é o respectivo tempo.

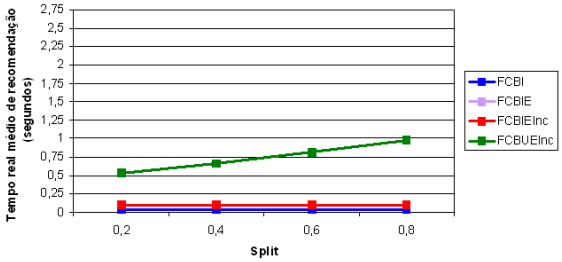
Os gráficos referentes aos restantes valores de *Split* encontram-se no Anexo B na secção B.3.



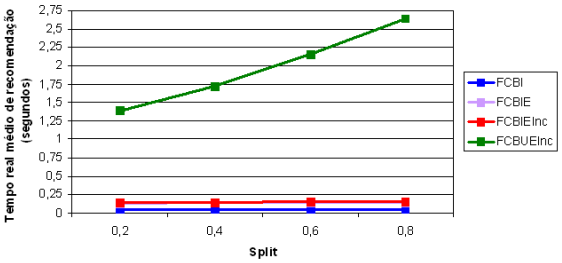
(a) ZP



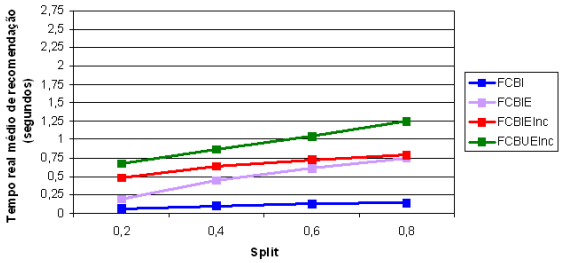
(b) PE200



(c) PE802

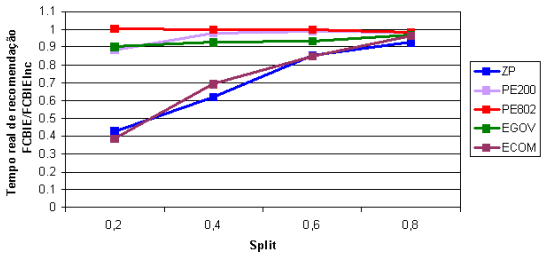


(d) EGOV

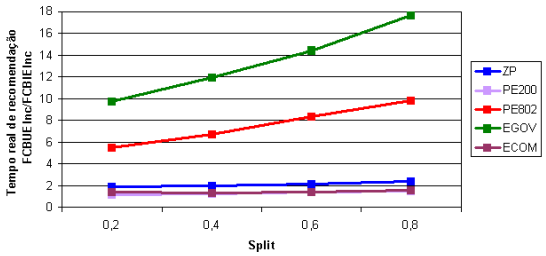


(e) ECOM

Figura 6.3: *Tempo real de recomendação de cada conjunto de dados por Split para cada algoritmo*

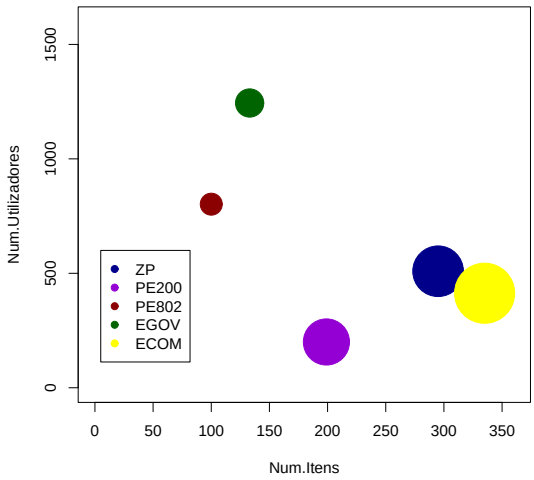


(a) FCBIE/FCBIEInc

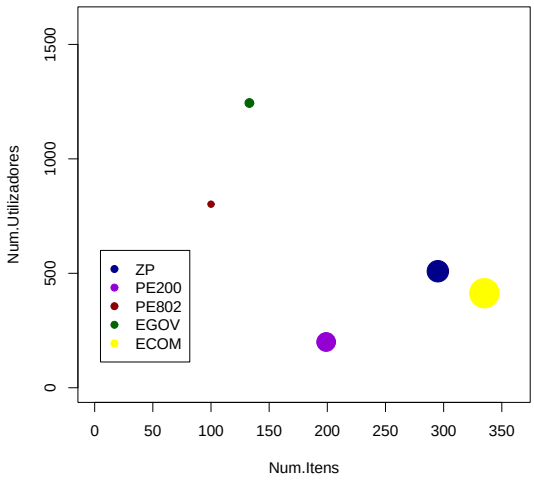


(b) FCBUEInc/FCBIEInc

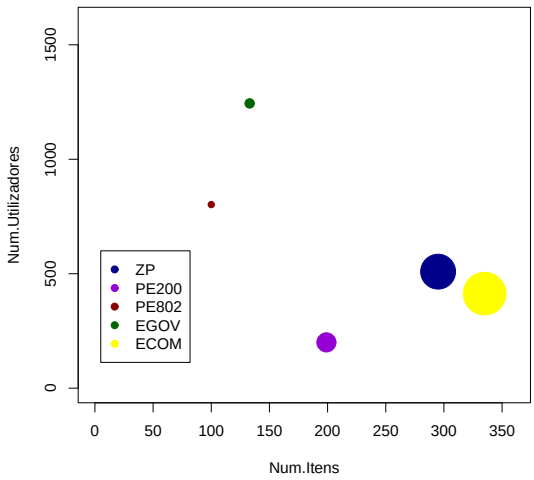
Figura 6.4: *Proporção existente no tempo médio real de recomendação entre algoritmos*



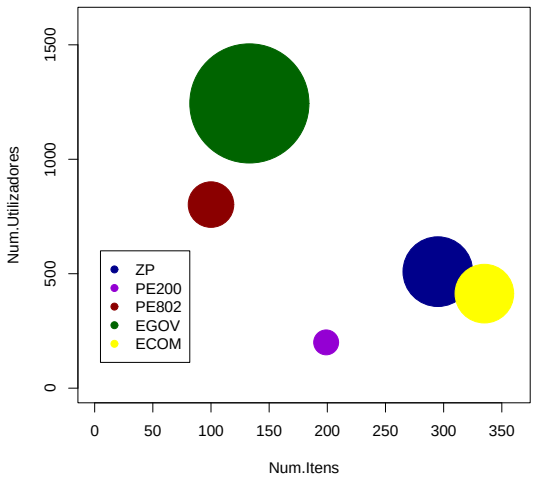
(a) FCBI tempo em centésimos de segundo



(b) FCBIE tempo em décimos de segundo



(c) FCBIEInc tempo em décimos de segundo



(d) FCBUEInc tempo em décimos de segundo

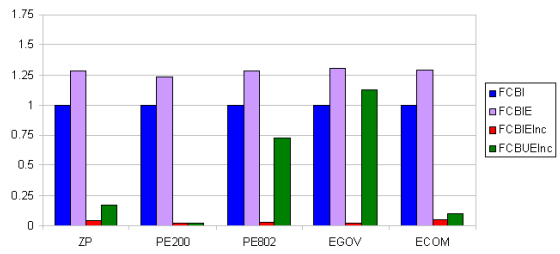
Figura 6.5: *Relação entre o número de utilizadores e o número de itens para Split=0,4, o raio da bola é proporcional ao tempo médio real de recomendação*

6.3.2 Tempo construção/actualização

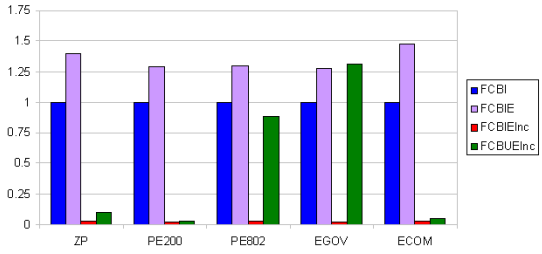
Avaliando agora o tempo relativo de construção/actualização da matriz de semelhanças, na Figura 6.6 pode-se afirmar que, para FCBIEInc, este é muito menor que o tempo de construção para FCBIE, o que significa que se gasta muito menos tempo a actualizar o modelo do que a reconstruí-lo do início de cada vez que aparece informação nova. O tempo relativo de actualização para FCBIEInc e FCBUEInc é semelhante quando o número de utilizadores não é muito maior que o número de itens (ZP, PE200 e ECOM). No entanto, quando se tem mais utilizadores, o tempo de actualização do algoritmo baseado nos utilizadores aumenta consideravelmente. De novo, está em concordância com a Secção 5.5.

6.3.2.1 Variação com o Conjunto de Dados

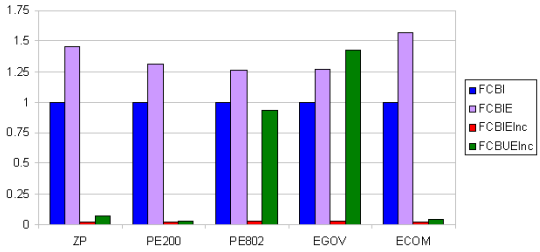
De seguida, apresenta-se resultados tendo em conta o tempo médio real (em segundos) de actualização da matriz de semelhanças (no caso dos algoritmos incrementais - FCBIEInc e FCBUEInc) ou de construção da matriz (algoritmos não incrementais - FCBI e FCBIE). Ao observar-se a Figura 6.7 pode-se ter uma visão geral por conjunto de dados desse mesmo tempo. Para cada algoritmo o tempo real de actualização/construção da matriz de semelhanças aumenta de acordo com o aumento do *Split*, sendo o aumento diferente consoante o conjunto de dados bem como o *Split*. No entanto, verifica-se que este aumento praticamente não se nota para o algoritmo incremental baseado nos itens (FCBIEInc), isto é, o tempo de actualização da matriz de semelhanças é praticamente constante para todos os conjuntos de dados quando se varia o *Split*. Com isto pode-se afirmar que o tempo de actualização para este algoritmo não sofre grandes actualizações ao variar-se o tamanho do conjunto de treino (e automaticamente do conjunto de teste - número de sessões de teste). O tempo de actualização para FCBUEInc (algoritmo incremental e baseado nos utilizadores) é sempre superior ao tempo de actualização para FCBIEInc, sendo menor esta diferença



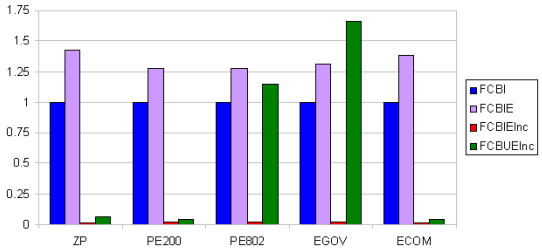
(a) 0,2



(b) 0,4



(c) 0,6



(d) 0,8

Figura 6.6: *Tempo (relativo) de construção/actualização da matriz de semelhanças, tendo como referência o tempo de construção em FCBI, para cada conjunto de dados*

quando estamos perante os conjuntos de dados PE802 e EGOV, Figuras 6.7(c) e 6.7(d) respectivamente, em que o número de novos itens é bastante inferior ao número de utilizadores de teste, como se pode observar na Tabela 6.2.

Em relação aos algoritmos não incrementais baseados nos itens (FCBI e FCBIE), verifica-se, mais uma vez, que o uso das *packages* de manipulação de matrizes esparsas afecta o tempo que demora a construir a matriz de semelhanças do algoritmo FCBIE, sendo este sempre superior ao tempo de construção no caso do algoritmo FCBI (não faz uso dessas *packages*). Esta diferença é maior no caso do conjunto de dados ZP

(Figura 6.7(e)) e ECOM (Figura 6.7(e)), pois estes são os que têm menor densidade, ou seja, são os dados mais esparsos, conforme se pode ver na Tabela 6.1.

6.3.2.2 Impacto da incrementalidade

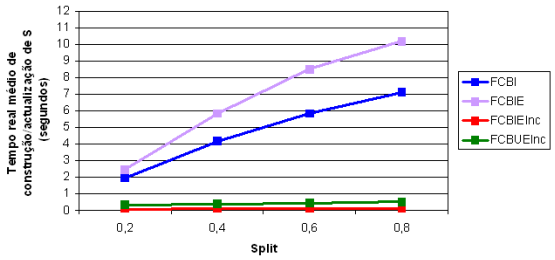
Comparando agora os tempos obtidos para os algoritmos incrementais e não incrementais verifica-se que os primeiros são menores que os segundos, à excepção dos conjuntos de dados PE802 e EGOV (conjuntos de dados com mais utilizadores e logo mais sessões) para FCBUEInc (este tem tempos de actualização próximos aos de construção da matriz de semelhanças para os algoritmos não incrementais). Com tudo isto, conclui-se que se ganha em termos de tempo de actualização/construção da matriz de semelhanças (e de *Recall*, conforme se pode ver na secção seguinte - Secção 6.4) com a incrementalidade do algoritmo baseado nos itens (em geral) e baseado nos utilizadores (o ganho no tempo não é sentido para PE802 e EGOV).

6.3.2.3 Baseado nos Itens vs Baseado nos Utilizadores

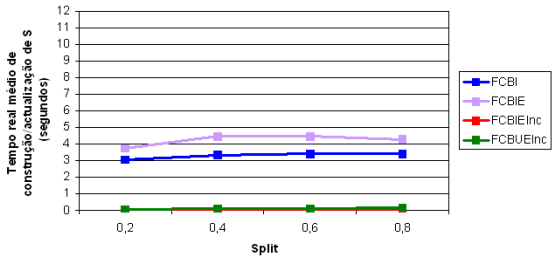
Comparando os algoritmos incrementais, baseado nos itens e baseado nos utilizadores, Figura 6.8, pode-se tirar conclusões semelhantes à Figura 6.4(b). FCBUEInc é o que apresenta piores resultados para EGOV e PE802 (conjuntos de dados com mais utilizadores). Com o aumento do *Split* o tempo médio real de actualização de FCBUEInc tende a afastar-se cada vez mais do tempo de FCBIEInc para os mesmos conjuntos de dados indicados anteriormente.

6.3.2.4 Número de utilizadores e número de itens

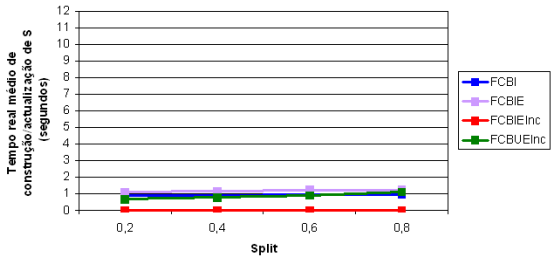
De modo análogo à Figura 6.5, nos gráficos da Figura 6.9, relaciona-se o número total de utilizadores e o número total de itens de cada algoritmo, mas, desta vez com o tempo médio real, por utilizador, de actualização da matriz de semelhanças. O raio da



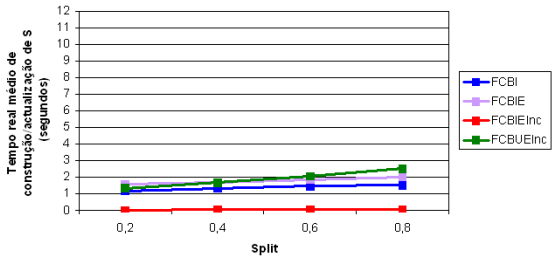
(a) ZP



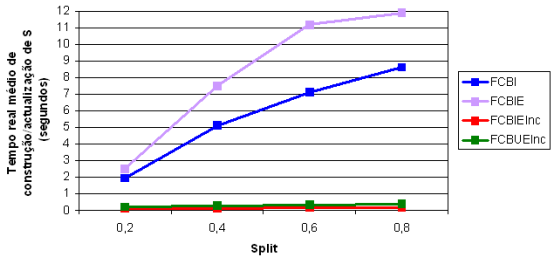
(b) PE200



(c) PE802



(d) EGOV



(e) ECOM

Figura 6.7: *Tempo real de actualização/construção da matriz de semelhanças de cada conjunto de dados por Split para cada algoritmo*

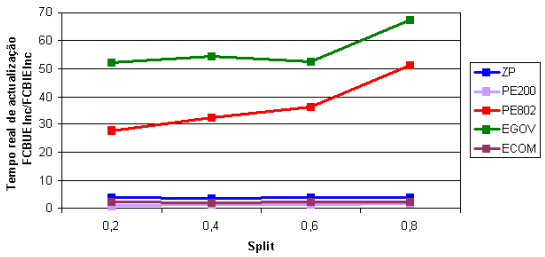
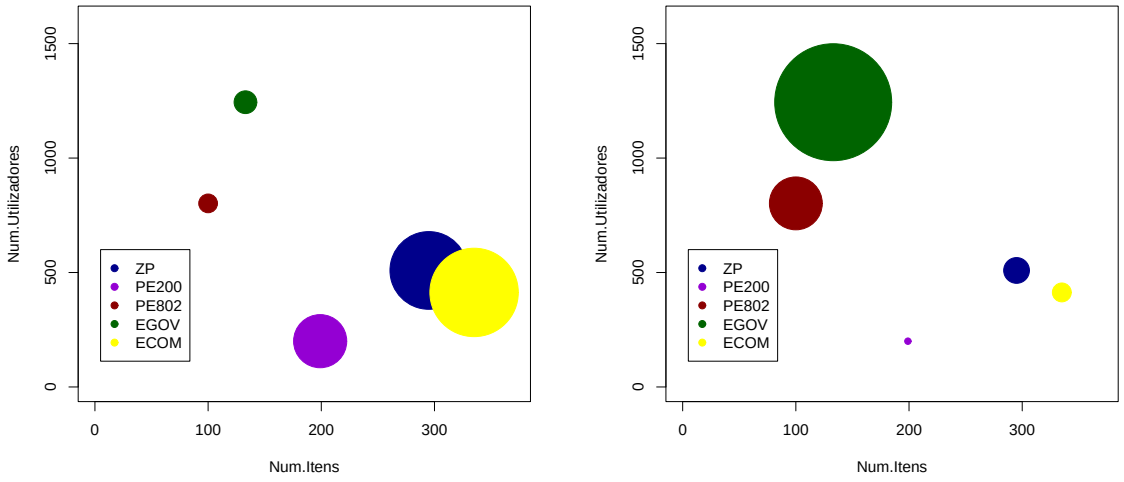


Figura 6.8: *Tempo real de actualização entre os algoritmos incrementais: FCBIEInc e FCBUEInc*

bola para cada algoritmo e conjunto de dados é proporcional ao respectivo tempo de actualização. Ao analisar-se os gráficos obtidos, verifica-se que a actualização tem um comportamento semelhante à recomendação em termos de complexidade computacional dos algoritmos, ou seja, nota-se que, nos algoritmos baseados nos utilizadores o tempo de actualização varia tanto de acordo com o número de utilizadores como com o número de itens (sendo maior para conjuntos de dados com números maiores de utilizadores e/ou itens); e nos algoritmos baseados nos itens o tempo médio real de actualização varia apenas com o número total de itens (quanto maior é o número de itens, maior é o respectivo tempo).

Os gráficos referentes aos restantes valores de *Split* encontram-se no Anexo B na Secção B.4.



(a) FCBIEInc tempo em décimos de segundo

(b) FCBUEInc tempo em décimos de segundo

Figura 6.9: *Relação entre o número de utilizadores e o número de itens para Split=0,4, o raio da bola é proporcional ao tempo médio real de actualização*

6.4 Capacidade Predictiva

6.4.1 Variação com N, $Split$ e $Neib$

Nesta secção, mostra-se as curvas do *Recall* com o aumento do N (número máximo de recomendações permitidas ao algoritmo) - Figura 6.10, $Split$ (variável usada no método usado para a separação em conjunto de treino e de teste) - Figura 6.11 e $Neib$ (número de vizinhos considerados) - Figura 6.12. Conforme se pode constatar pelos gráficos apresentados, FCBI e FCBIE têm exactamente os mesmos resultados, como deveriam, uma vez que eles diferem apenas nas estruturas de dados para armazenar e trabalhar com as matrizes esparsas. Os algoritmos baseados nos itens (FCBI, FCBIE e FCBIEINC) têm desempenho semelhante, em termos de *Recall*. O *Recall* para FCBIEInc é semelhante ou superior ao *Recall* para FCBI (e FCBIE), à excepção de ZP para $N < 5$ e EGOV para $N > 10$.

6.4.2 Impacto da incrementalidade

Pode-se afirmar que, na generalidade, a incrementalidade ajuda (ou pelo menos não faz perder) na medida do *Recall*, ou seja, com a incrementalidade tem-se melhores resultados (ou iguais) em termos das recomendações feitas. O conjunto de dados PE802 (Figura 6.10(c)) é o que apresenta maior diferença entre os resultados para FCBIEInc e FCBI (e FCBIE), sendo o algoritmo incremental o que tem valor maior de *Recall*. Note-se que para este conjunto de dados, para qualquer $Split$, todos os itens fazem parte do conjunto de treino, ou seja, com o uso da incrementalidade apenas se altera os valores das semelhanças entre os itens não se acrescentando nenhuma coluna e/ou linha nova à matriz de semelhanças (no caso do algoritmo incremental baseado nos itens). Com base nos valores apresentados na Figura 6.10(c), pode-se concluir que, quando não se altera o conjunto de itens, isto é, quando não existem itens novos, a recomendação do sistema incremental baseado nos itens (FCBIEInc) tem maior capacidade predictiva

(pelo menos para o caso PE802, único conjunto de dados usado no qual não existem itens novos).

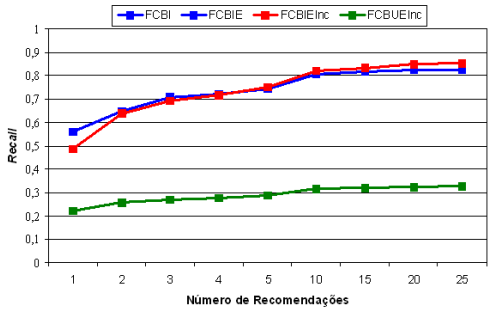
6.4.3 Baseado nos Utilizadores vs Baseado nos Itens

Compara-se agora os resultados obtidos para sistemas incrementais de recomendação baseados nos utilizadores e itens. O *Recall* para FCBUEInc é sempre inferior ao de FCBIEInc, o que prova que o algoritmo incremental baseado nos itens apresenta maior capacidade predictiva que o algoritmo incremental baseado nos utilizadores.

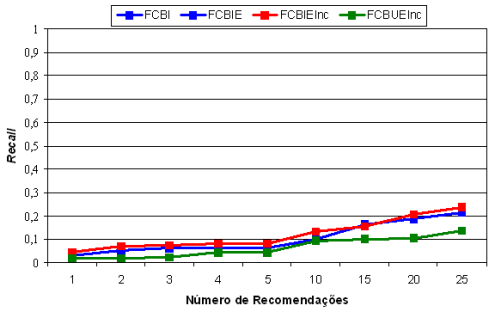
6.4.4 Variação com o Conjunto de Dados

Os gráficos apresentados na Figura 6.10 permitem concluir que, para todos os conjuntos de dados e algoritmos, a medida *Recall* aumenta com o aumento do número de recomendações (N), como seria de esperar.

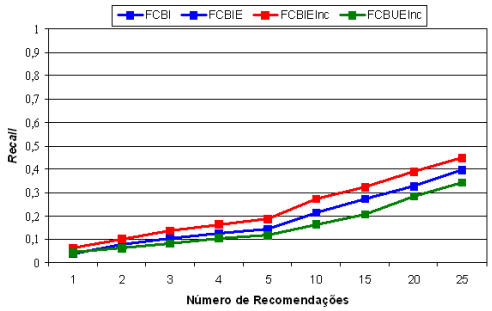
Os gráficos referentes às restantes combinações possíveis de *Split* e *Neib* encontram-se no Anexo B na Secção B.5.



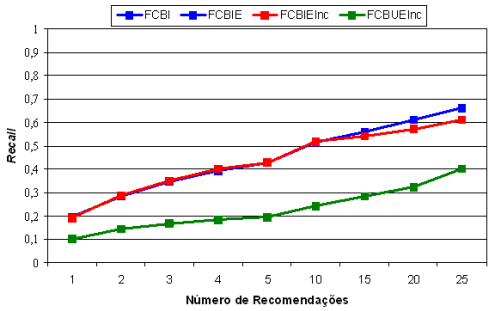
(a) ZP



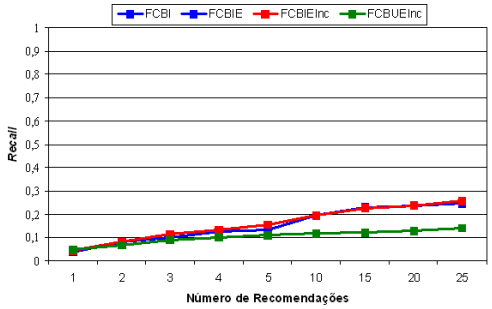
(b) PE200



(c) PE802



(d) EGOV



(e) ECOM

Figura 6.10: *Recall obtido para Split=0,2 e Neib=5*

As Figuras 6.11 e 6.12 dizem respeito aos gráficos da variação do valor do *Recall* para os *Splits* e $Neib$ considerados, respectivamente, para cada conjunto de dados. Para tal fixou-se o valor de N como sendo 5, ou seja, apenas se considerou os dados obtidos aquando da recomendação de 5 itens a cada utilizador. Isto porque, nos gráficos da Figura 6.10, percebe-se que a partir desse valor de N a medida de *Recall* tende a estabilizar. Na Figura 6.11, verifica-se que, ao fixar o tamanho da vizinhança, a medida de *Recall* aumenta com o aumento do *Split* (excepto ZP, para os algoritmos baseados nos itens, que apresenta uma pequena descida quando se passa do $Split = 0,6$ para $Split = 0,8$). Isto está de acordo com o que se disse antes, pois, quando se inicia o algoritmo com um conjunto de dados pequeno (para se construir S), se depois não se faz mais nenhuma actualização dessa matriz, as recomendações começam a perder qualidades. Em relação ao FCBUEINC continua-se a verificar que este apresenta uma qualidade inferior nas recomendações que faz. Similarmente à Figura 6.10, da Figura 6.11, conclui-se que FCBIEInc é o que apresenta maior capacidade predictiva, à excepção do conjunto de dados PE200 (mesmo número de utilizadores e itens), em que é FCBI (e, logo, também FCBIE) que tem maior valor de *Recall*. Verifica-se, igualmente, que ZP e EGOV são os conjuntos de dados que apresentam maiores diferenças entre o *Recall* dos algoritmos baseados nos itens e algoritmo baseado nos utilizadores.

Os gráficos referentes aos restantes valores de *Neib* encontram-se no Anexo B na Secção B.6.

Quanto à Figura 6.12, ao que já se conclui atrás pode-se apenas acrescentar que para FCBUEInc o *Recall* aumenta com o aumento do tamanho da vizinhança ($Neib$), sendo que, para os restantes algoritmos, o valor do *Recall* é praticamente constante (pelo menos quando comparada com FCBUEInc), ou seja, o tamanho da vizinhança afecta, principalmente, o algoritmo baseado nos utilizadores.

Os gráficos referentes aos restantes valores de *Split* encontram-se no Anexo B na Secção B.7.

6.5 Resumo

Neste capítulo, apresenta-se as várias experiências feitas para os vários algoritmos, assim como os resultados obtidos para as medidas adoptadas e as conclusões que se foi possível retirar dos mesmos.

As principais conclusões obtidas são:

- Algoritmo incremental (baseado nos itens) apresenta

  - baixos tempos de actualização

  - tempos de recomendação idênticos à versão não incremental

  - melhores (ou iguais) valores de *Recall* quando comparado com a versão não incremental

- Algoritmo incremental baseado nos itens apresenta melhores resultados, em termos de tempo de recomendação e actualização e capacidade predictiva, do que o algoritmo incremental baseado nos utilizadores

- O uso de técnicas que trabalham com matrizes esparsas faze aumentar o tempo de recomendação

- Com o aumento do número de sessões de treino (aumento do *Split*), em geral, o tempo de recomendação diminui e o de construção/actualização aumenta

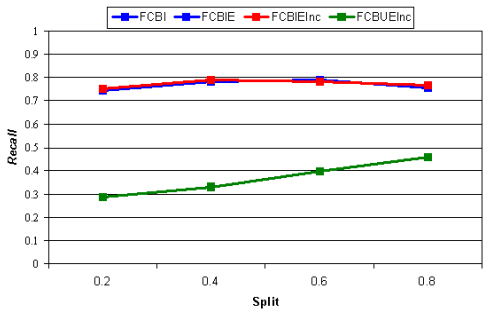
- Aumento de *Split* origina aumento do *Recall*

- *Recall* aumenta com o aumento do número de recomendações, N

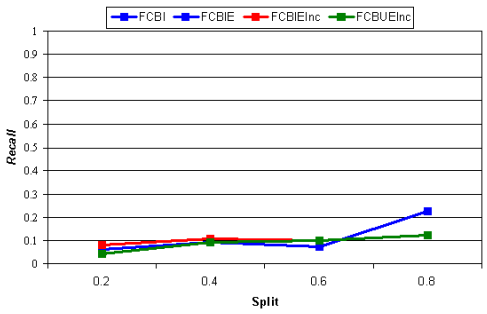
- Quanto maior é o número de itens, maior é o tempo de recomendação e o tempo de actualização dos algoritmos baseados nos itens

- Para o algoritmo baseado nos utilizadores, o tempo de recomendação e o tempo de actualização aumentam com o aumento do número de utilizadores e/ou itens.

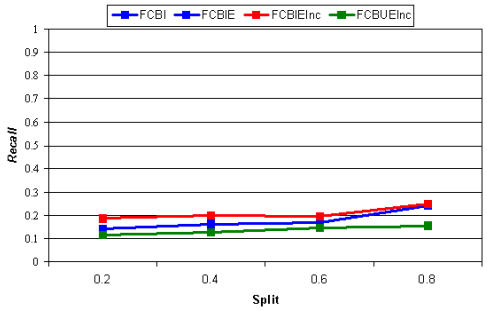
Seguidamente, são apresentadas as principais conclusões, bem como as limitações deste trabalho e o trabalho a desenvolver no Futuro.



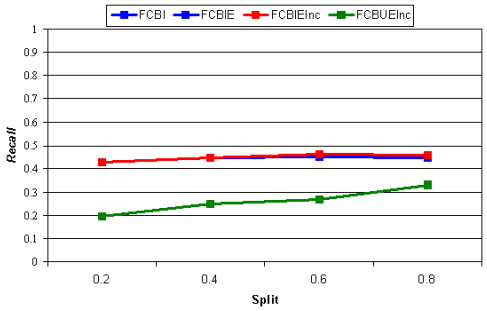
(a) ZP



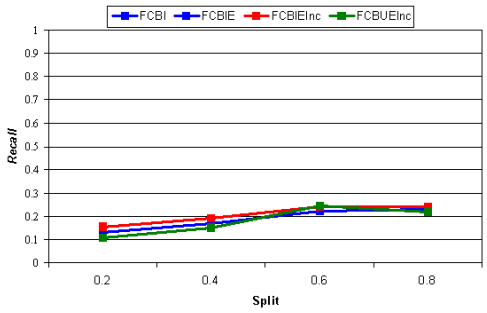
(b) PE200



(c) PE802

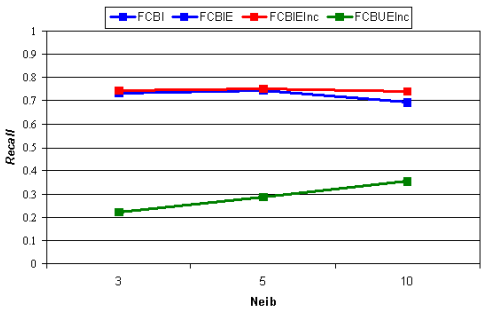


(d) EGOV

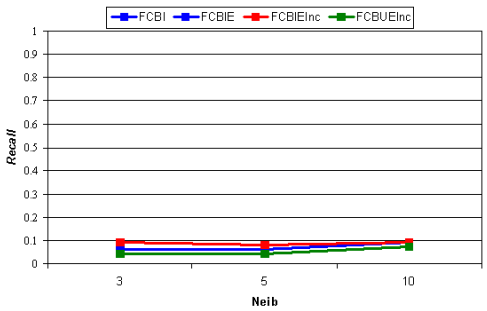


(e) ECOM

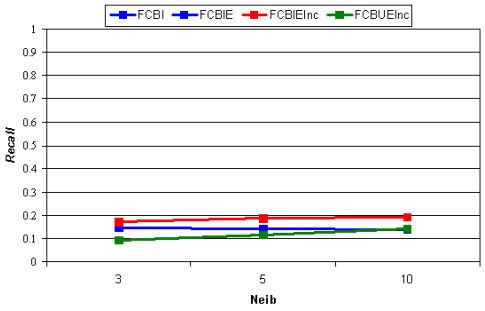
Figura 6.11: *Recall obtido para Neib=5 considerando que se está a fazer uma recomendação de 5 itens por utilizador de teste*



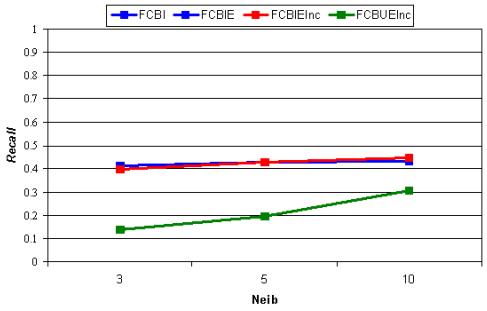
(a) ZP



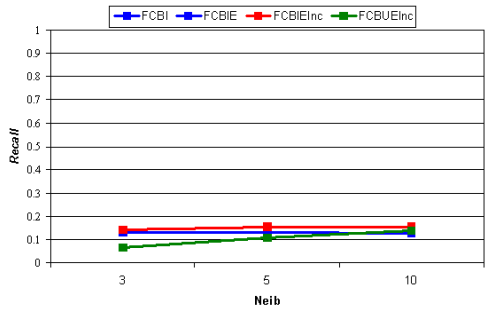
(b) PE200



(c) PE802



(d) EGOV



(e) ECOM

Figura 6.12: *Recall obtido para Split=0,2 considerando que se está a fazer uma recomendação de 5 itens por utilizador de teste*

Capítulo 7

Conclusões

Com este trabalho, apresentou-se um algoritmo incremental de Filtragem Colaborativa baseado nos itens para dados binários. Comparou-se esse com um algoritmo não incremental, mas também baseado nos itens, e com outro incremental, mas baseado nos utilizadores. Estudou-se, igualmente, as alterações no tempo e na capacidade predictiva quando se usa técnicas de manuseamento de matrizes esparsas.

7.1 Resultados

Os resultados obtidos com cinco conjuntos de dados de características diferentes mostram que a incrementalidade nos algoritmos baseados nos itens apresenta baixos tempos de actualização e um tempo de recomendação idêntico à versão não incremental, sem descurar as recomendações feitas. O algoritmo incremental apresenta melhores (ou iguais) valores de *Recall* quando comparado com a versão não incremental. Esta versão incremental do algoritmo baseado nos itens apresenta melhores resultados, em termos de tempo e capacidade predictiva, do que o algoritmo análogo baseado nos utilizadores. Esta diferença deve-se à manutenção de uma segunda estrutura com as mesmas dimensões da matriz de semelhanças (histórico das sessões) no algoritmo

baseado nos utilizadores. Esta segunda estrutura é necessária para a actualização da matriz de semelhanças entre utilizadores, o que não acontece no caso do algoritmo baseado nos itens. Logo, tem-se uma actualização mais rápida das semelhanças entre itens, pois não há a necessidade de revisitar a base de dados. A situação mais realista é a existência de um maior número de utilizadores, comparativamente ao número de itens, pelo que o algoritmo baseado nos itens responde melhor que o algoritmo baseado nos utilizadores. Usou-se, também, estruturas especiais que trabalham com matrizes esparsas para lidar com a escalabilidade (*scalability*), e tentou-se três diferentes *packages* do R. Concluiu-se que a package *spam* produz programas R mais rápidos, apesar do preço relativamente alto por se usar técnicas que trabalham sobre matrizes esparsas, como seria esperado. É o preço a pagar devido à limitação de memória.

Pode-se também tirar conclusões sobre as alterações provocadas nos resultados pelos diferentes valores tomados pelas variáveis. Sendo assim, com o aumento do número de sessões de treino (aumento do *Split*), em geral, o tempo de recomendação diminui e o de construção/actualização aumenta. Isto porque este aumento implica o aumento do conjunto de treino e, por conseguinte, matrizes com maiores dimensões desde o início, levando a um tempo médio de actualização maior e um tempo médio de recomendação menor. Este aumento do conjunto de treino também origina um aumento da medida do *Recall*. Pois, ao ter-se um conjunto de treino maior, tem-se também mais informação sobre os itens ou utilizadores e, logo, as recomendações vão mais ao encontro das necessidades do utilizador activo. Como já era esperado, verifica-se, também, um aumento de *Recall* com o aumento do número de recomendações (N). Já o número de vizinhos considerados na recomendação ($Neib$) afecta essencialmente o valor de *Recall* para o algoritmo incremental baseado nos utilizadores (*Recall* aumenta com o aumento de $Neib$). Verificou-se, ainda, que, quanto maior é o número de itens, maior é o tempo de recomendação e actualização dos algoritmos baseados nos itens. Já no caso dos algoritmos baseados nos utilizadores, o tempo de recomendação e actualização varia (aumenta) com o aumento do número de utilizadores e/ou itens (conforme se enunciou

na Secção 5.5.

Em suma, com base nos resultados obtidos e considerando a situação mais realista, em que o número de utilizadores é muito maior que o número de itens, o algoritmo baseado nos itens apresenta melhores resultados do que o algoritmo baseado nos utilizadores.

7.2 Limitações e Trabalho Futuro

No Futuro, tenciona-se investigar o comportamento dos algoritmos com conjuntos de dados maiores e/ou num ambiente de fluxo contínuo de dados, assim como alargar estes algoritmos de forma a poder trabalhar com dados explícitos ($< utilizador, item, rate >$). Pretende-se, também, que o sistema tenha a possibilidade de esquecer a informação mais antiga ou que dê um peso inferior à informação anterior e maior à informação mais recente no cálculo e/ou actualização da semelhança entre dois utilizadores ou itens.

Neste trabalho, nos conjuntos de dados usados, não havia o reconhecimento do utilizador de cada vez que ele iniciava uma nova sessão. Apenas se tinha o caminho percorrido pelo mesmo, independentemente de ser a primeira vez que acede à página Web ou não. No entanto, esse reconhecimento não acontece em grande parte das páginas Web pelo que é necessário obter a melhor forma de o fazer e de obter tais dados, bem como do melhor modo de os armazenar. Outro aspecto a melhorar é estudar novas formas de trabalhar com o algoritmo baseado nos utilizadores sem necessitar de ter sempre em memória todo o conjunto de dados (no caso incremental).

Apêndice A

Código R usado

```
library(spam)

##############################

# DATA SETUP FOR EVALUATION #

##############################

# data: basket.data

# Split:percentage

# Value: structure(train: basket.data, test.ids:vector, test:
    basket.data, hidden: basket.data)

# splits the initial basket data set into train, test and hidden.

data.setup <- function(data) {
    OUT<-NULL
    data.ids <- as.vector(unique(data[,1]))
    max <- length(data.ids)
    set.seed(20)
    sample <- sample(1:max,Split*max)
    train.ids <- data.ids[sample]
    data.train.idx<-NULL
    for(i in train.ids) data.train.idx <- c(data.train.idx,which(data[,1]==i))
    data.train <- data[data.train.idx,]
    OUT$train<-data.train
    test.ids <- data.ids[-sample]
    OUT$test.ids <- test.ids
    data.test.idx<-NULL
    data.hidd.idx <- NULL
    for(i in test.ids){
        totest <- which(data[,1]==i)
        hidsample<-sample(1:length(totest),1)
        totest.uptodate<-setdiff(totest,totest[hidsample])
        data.test.idx <- c(data.test.idx,totest.uptodate)
        data.hidd.idx <- c(data.hidd.idx,totest[hidsample])
        }
```

```
    data.test <- data[data.test.idx,]
    OUT$test<-data.test
    data.hidd <- data[data.hidd.idx,]
    OUT$hidden<-data.hidd
    OUT
}

# File:string

# Split:percentage

# data are read in the <id,item> format, fields separated by space

# no titles

# follows all-but-one protocol

# the Value is a structure as defined in data.setup

# unique on data: to avoid equal pairs <id,item>

load.data.and.setup.unique <- function() {
    data<-unique(read.table(File,sep="\t")) #read the File of the dataset (format File.txt)
    data<-data[,c(1,2)]
    data[,1]<-as.factor(data[,1])
    data[,2]<-as.factor(data[,2])
    data.setup(data)
}

##############################################

# SIMILARITY to item-based algorithm (FCBI) #

##############################################

# calculation of similarity matrix

# cosine similarity :

    # I and J are vectors representing item sets

    #(I * J) / (sqrt(sum(I))*sqrt(sum(J)))

simmatrix1<-function() {
    i.u <- tapply(Train[,1],Train[,2],function(x){x})
    B<-matrix(0,nitems.in,nitems.in)
    colnames(B)<-items.in
    rownames(B)<- items.in
    triang.m <-lapply(1:(length(i.u)-1), function(ind, i.u){
        sapply((ind+1):length(i.u), function(ind2, i.u, ind1){
            (length(intersect(i.u[[ind1]],i.u[[ind2]])))/(sqrt(length(i.u[[ind1]]))*sqrt(length(i.u[[ind2]])))
        },i.u,ind)
    }, i.u)
    cat("item...")
    for (j in 1:(nrow(B)-1)){
        cat(j," - ")
        t.m <- sapply(1:length(triang.m[[j]]), function(i){if(is.na((triang.m[[j]])[i])) 0 else (triang.m[[j]])[i]})
        B[j,(j+1):ncol(B)] <- t.m[]
    }
    cat(nrow(B)," - \n")
    B <- B + t(B)
    diag(B) <- 1
    B
}
```

```
############################################################

# SIMILARITY to item-based algorithm with package "spam" #

#  that work with sparse matrices (FCBIE and FCBIEInc)   #

############################################################

# calculation of similarity matrix

# cosine similarity

simmatrix2_spam<-function() {
    i.u <-tapply(Train[,1],Train[,2],function(x){x})
    B.s<-as.spam(matrix(0,nitems.in,nitems.in))
    triang.m <-lapply(1:(length(i.u)-1), function(ind, i.u){
        sapply((ind+1):length(i.u), function(ind2, i.u, ind1){
            (length(intersect(i.u[[ind1]],i.u[[ind2]])))/(sqrt(length(i.u[[ind1]]))*sqrt(length(i.u[[ind2]])))
        },i.u, ind)
    }, i.u)
    cat("item...")
    for (j in 1:(nrow(B.s)-1)){
        cat(j," - ")
        t.m <- sapply(1:length(triang.m[[j]]), function(i){if(is.na((triang.m[[j]])[i])) 0 else (triang.m[[j]])[i]})
        B.s[j,(j+1):ncol(B.s)] <- t.m[]
    }
    cat(nrow(B.s)," - \n")
    B.s <- B.s + t(B.s)
    diag(B.s) <- 1
    B.s
}

############################################################

# CACHE to incremental item-based algorithm (FCBIEInc) #

############################################################

# OUTPUT - Int: matrix

    # Int[pos[i],pos[j]] number os users that saw the pair of items (i,j)

    # Int[pos[i],pos[i]] number of users that saw item i

cache_spam<-function() {
    i.u <- tapply(Train[,1],Train[,2], function(x){x}) # users that evaluate each item (output: list)
# calculation of the number os users that saw each item (output: list)
    cont.i<-sapply(items.in, function(i){length(i.u[[i]])})
    Int<-as.spam(matrix(0,nitems.in,nitems.in)) # to initialize the matrix with zeros
    for(i in items.in[-nitems.in]){
        row_i<-sapply(items.in[pos.in[items.in]>pos.in[i]],function(j){ length(intersect(i.u[[i]],i.u[[j]]))})
        Int[pos.in[i],(pos.in[i]+1):nitems.in]<-row_i
    }
    Int<-Int+t(Int)
    diag(Int)<-cont.i
    Int
}

############################################################

# CACHE to incremental user-based algorithm (FCBUEInc) #

############################################################

# OUTPUT - Int.u: matrix
```

```
    # Int.u[pos.users[u],pos.users[w]] number of itens seen in common by the pair of users (u,w)

    # Int.u[pos.users[u],pos.users[u]] number of seen by the user u

cache.u_spam<-function() {
    OUT<-NULL
    u.i <- tapply(Train[,2],Train[,1], function(x){x}) # itens evaluated by each user (output: list)
    OUT$u.i<-u.i
# calculation of the number os items seen by each user (output: list)
    cont.u<-sapply(Users.in, function(u){length(u.i[[u]])})
    Int.u<-as.spam(matrix(0,nusers.in,nusers.in)) # to initialize the matrix with zeros
    for(u in Users.in[-nusers.in]){
        row_u<-sapply(Users.in[pos.users.in[Users.in]>pos.users.in[u]],function(w){length(intersect(u.i[[u]],u.i[[w]]))})
        Int.u[pos.users.in[u],(pos.users.in[u]+1):nusers.in]<-row_u
    }
    Int.u<-Int.u+t(Int.u)
    diag(Int.u)<-cont.u
    OUT$Int.u<-Int.u
    OUT
}

#####################################################

#  SIMILARITY: user-based algorithm with package   #

# "spam" that work with sparse matrices (FCBUEInc) #

#####################################################

# calculation of similarity matrix

# cosine similarity :

    # U and W are vectors representing users sets

    #(I * J) / (sqrt(sum(I))*sqrt(sum(J)))

simmatrix4.2_spam<-function() {
    u.i <- tapply(Train[,2],Train[,1], function(x){x})
    B.s<-as.spam(matrix(0,nusers.in,nusers.in))
    triang.m <- lapply(1:(length(u.i)-1), function(ind, u.i) {
        sapply((ind+1):length(u.i), function(ind2, u.i, ind1) {
            (length(intersect(u.i[[ind1]], u.i[[ind2]])))/(sqrt(length(u.i[[ind1]]))*sqrt(length(u.i[[ind2]])))
        }, u.i, ind)
    }, u.i)
    cat("iuser...")
    for (j in 1:(nrow(B.s)-1)){
        cat(j," - ")
        t.m <- sapply(1:length(triang.m[[j]]), function(i){if(is.na((triang.m[[j]])[i])) 0 else (triang.m[[j]])[i]})
        B.s[j,(j+1):ncol(B.s)] <- t.m[]
    }
    cat(nrow(B.s)," - \n")
    B.s <- B.s + t(B.s)
    diag(B.s) <- 1
    B.s
}

#################################################

# RECOMMENDATION to item-based algorithm (FCBI) #

#################################################

# Determination of the neighbors of item i
```

```
# S: similarity matrix

# Neib: Number of considered neighbors

neighbors1<-function(i,S){
    M<-S[i,-i] # it is used only the rows and columns of S that correspond to item i
    my.x<-M
    neighs <- M[sapply(1:Neib, function(dummy) {my.max <- which.max(my.x); my.x[my.max] <<- -1; my.max})]
    neighs
}

# Determination of the activation weight of item i to the active
    user

# R: observed items by the active user

wsumitem1 <- function(R,S,i) {

# items in the neighborhood of i that they had never been seen by
# the active user
    N<-intersect(names(Neighb<-neighbors1(pos.in[i],S)),R)
    num<- sum(Neighb[N])
    den<- sum(Neighb)
    if (den==0) 0
    else num / den
}

# Determination of the activation weights of all the items to the
    active user

wsum1 <- function(R,S) {
    idx<-c(1:ncol(S))
    names(idx)<-colnames(S)
    sdf<-idx[setdiff(names(idx),R)] # observed items will not be recommended
    n <- length(sdf)
    B <- sapply(1:n, function(i){wsumitem1(R,S,sdf[i])})
    names(B)<-names(sdf)
    B
}

# Outputs a vector with the maximum activation weights of the "Neib"
    items

topNrec1 <- function(R,S,N) {
    recs<-wsum1(R,S)
    maxrecs <- length(recs)
    recs<-recs[maxrecs:1]
    my.x<-recs
    OUT<-recs[sapply(1:(min(N,maxrecs)), function(dummy) {my.max <- which.max(my.x); my.x[my.max] <<- -1; my.max})]
    OUT[(min(N,maxrecs)):1]
}

# Determination of the top N recommendations for all the users in
    the Test Set

#Output: average time of recommendation and the recommendations
    that had been made

topNrec.batch1 <- function(S,N) {
    Usr<-as.vector(Test.Ids)
    uids<-NULL
    recs<-NULL
    count<-1
    aux_time_recA1<-matrix(data = NA, nrow = length(Usr), ncol = 3, byrow = FALSE)
```

```
    OUT<-NULL
    for(u in Usr) {
        begin_Rec<-Sys.time()
        urec <- names(topNrec1(Test[Test[,1]==u,2],S,N))
        end_Rec<-Sys.time()
        #aux_time_rec<-c(aux_time_rec,end_Rec-begin_Rec)
        aux_time_recA1[count,]<-c(u, end_Rec-begin_Rec, ncol(S))
        count<-count+1
        recs <- c(recs,urec)
        uids <- c(uids,rep(u,length(urec)))
    }
    OUT$recs<-as.data.frame(matrix(c(uids,recs),length(uids),2))
    OUT$Av_Time_Rec<-sum(as.double(aux_time_recA1[,2]))/length(Usr) # average time of recommendation
    OUT
}

#################################################################

# RECOMMENDATION to item-based algorithm with package "spam" #

#    that work with sparse matrices (FCBIE and FCBIEInc)     #

#################################################################

# Determination of the neighbors of item i

# S: similarity matrix

# Neib: Number of considered neighbors

neighbors2<-function(i,S,items,pos){
    M<-as.vector(as.matrix(S[pos[i],])) # because S don't have names in the columns
    names(M)<-items # to put names in the positions
    M[pos[i]]<--1 # i can't be recommended
    my.x<-M
    neighs <- M[sapply(1:Neib, function(dummy) {my.max <- which.max(my.x); my.x[my.max] <<- -1; my.max})]
    neighs
}

# Determination of the activation weight of item i to the active
    user

# R: observed items by the active user

wsumitem2 <- function(R,S,i,items,pos) {

# items in the neighborhood of i that they had never been seen by
# the active user
    Nw<-intersect(names(Neighb<-neighbors2(i,S,items,pos)),R)
    num<- sum(Neighb[Nw])
    den<- sum(Neighb)
    if (den==0)0
    else num/den
}

# Determination of the activation weights of all the items to the
    active user

wsum2 <- function(R,S,items,pos) {
    idx<-c(1:ncol(S))
    names(idx)<-items
    sdf<-idx[setdiff(names(idx),R)] # observed items will not be recommended
    n <- length(sdf)
    B <- sapply(1:n, function(j){wsumitem2(R,S,names(sdf[j]),items,pos)})
    names(B)<-names(sdf)
```

```
    B
}

# Outputs a vector with the maximum activation weights of the
    "Neib" items

topNrec2 <- function(R,S,N,items,pos) {
    recs<-wsum2(R,S,items,pos)
    maxrecs <- length(recs)
    recs<-recs[maxrecs:1]
    my.x<-recs
    OUT<-recs[sapply(1:(min(N,maxrecs)), function(dummy) {my.max <- which.max(my.x); my.x[my.max] <<- -1; my.max})]
    OUT[(min(N,maxrecs)):1]
}

# Determination of the top N recommendations for all the users in
    the Test Set to FCBIE

#Output: average time of recommendation and the recommendations
    that had been made

topNrec.batch2 <- function(S,N) {
    Usr<-as.vector(Test.Ids)
    uids<-NULL
    recs<-NULL
    #aux_time_rec<-NULL
    OUT<-NULL
    items<-items.in
    pos<-pos.in
    count<-1
    aux_time_recA2<-matrix(data = NA, nrow = length(Usr), ncol = 3, byrow = FALSE)
    for(u in Usr) {
        begin_Rec<-Sys.time()
        urec <- names(topNrec2(Test[Test[,1]==u,2],S,N,items,pos))
        end_Rec<-Sys.time()
        aux_time_recA2[count,]<-c(u, end_Rec-begin_Rec, ncol(S))
        count<-count+1
        recs <- c(recs,urec)
        uids <- c(uids,rep(u,length(urec)))
        }
    OUT$recs<-as.data.frame(matrix(c(uids,recs),length(uids),2))
    OUT$Av_Time_Rec<-sum(as.double(aux_time_recA2[,2]))/length(Usr) # average time of recommendation
    OUT
}

# Determination of the top N recommendations for user1 to FCBIEInc

#Output: recommendations that had been made

topNrec.batch3 <- function(user1,S,N,items,pos) {
    uids<-NULL
    recs<-NULL
    urec <- names(topNrec2(Test[Test[,1]==user1,2],S,N,items,pos))
    recs <- c(recs,urec)
    uids <- c(uids,rep(user1,length(urec)))
    matrix(c(uids,recs),length(uids),2)
}

#########################################################

# RECOMMENDATION to user-based algorithm with package #

#   "spam" that work with sparse matrices (FCBUEInc)  #

#########################################################
```

```
# Determination of the neighbors of item u

# S: similarity matrix

# Neib: Number of considered neighbors

# pos.users and Users are vectors that are updated after the
    recommendation to each user

neighborsuser2<-function(u,S,pos.users,Users){
    M<-as.vector(as.matrix(S[pos.users[u],])) # because S don't have names in the columns
    names(M)<-Users # to put names in the positions
    M[pos.users[u]]<--1 # u can't be on the neighborhood of u
    my.x<-M
    neighs <- M[sapply(1:Neib, function(dummy) {my.max <- which.max(my.x); my.x[my.max] <<- -1; my.max})]
    neighs
}

# Determination of the activation weight of item i to the active
    user

wsumuseritem2 <- function(i,u,S,pos.users,Users) {
    Neighb<-neighborsuser2(u,S,pos.users,Users)
    users.neighb<-unique(Train[is.element(Train[,1],names(Neighb))&Train[,2]==i,1])
    num<-sum(Neighb[as.character(users.neighb)])
    den<-sum(Neighb)
    if (den==0) 0
    else num/den
}

# Determination of the activation weights of all the items to the
    active user

wsum4.2 <- function(u,S,pos.users,pos,Users,items) {
    R<-Test[Test[,1]==u,2] # observed items by the active user
    idx<-c(1:length(items))
    names(idx)<-items
    sdf<-idx[setdiff(names(idx),R)] # observed items will not be recommended
    n <- length(sdf)
    B <- sapply(1:n, function(j){wsumuseritem2(names(sdf[j]),u,S,pos.users,Users)})
    names(B)<-names(sdf)
    B
}

# Outputs a vector with the maximum activation weights of the
    "Neib" items

topNrec4.2 <- function(u,S,N,pos.users,pos,Users,items) {
    recs<-wsum4.2(u,S,pos.users,pos,Users,items)
    maxrecs <- length(recs)
    recs<-recs[maxrecs:1]
    my.x<-recs
    OUT<-recs[sapply(1:(min(N,maxrecs)), function(dummy) {my.max <- which.max(my.x); my.x[my.max] <<- -1; my.max})]
    OUT[(min(N,maxrecs)):1]
}

# Update of the system and determination of the top N
    recommendations for each user in the Test Set to FCBUEInc

#Output: average time of update and recommendation, and the
    recommendations that had been made

topNrec.batch4.2.s<- function(S,N,Int.u,u.i) {
    uids<-NULL
```

```
    recs<-NULL
    rate<-NULL
    aux_Update<-NULL
    count<-1
    aux_time_recA4<-matrix(data = NA, nrow = length(Test.Ids), ncol = 3, byrow = FALSE)
    OUT<-NULL
    pos.users<-pos.users.in
    Users<-Users.in
    items<-items.in
    pos<-pos.in
    for(u in Test.Ids) {
        begin_Update<-Sys.time()
        aux<-update.u.u2.s(S,u,Int.u,u.i,pos.users,Users,items,pos)
        end_Update<-Sys.time()
        aux_Update<-c(aux_Update,end_Update-begin_Update)
        S<-aux$S
        Int.u<-aux$Int.u
        items<-aux$items
        Users<-aux$Users
        pos.users<-aux$pos.users
        pos<-aux$pos
        u.i<-aux$u.i
        begin_Rec<-Sys.time()
        rate1 <- topNrec4.2(u,S,N,pos.users,pos,Users,items)
        end_Rec<-Sys.time()
        aux_time_recA4[count,]<-c(u, end_Rec-begin_Rec, ncol(S))
        count<-count+1
        irec <- names(rate1)
        recs <- c(recs,irec)
        uids <- c(uids,rep(u,length(irec)))
    }
    OUT$recs<-as.data.frame(matrix(c(uids,recs),length(uids),2))
    OUT$Av_Time_Update<-sum(aux_Update)/length(Test.Ids) # average time of update
    OUT$Av_Time_Rec<-sum(as.double(aux_time_recA4[,2]))/length(Test.Ids) # average time of recommendation
    OUT
}

########################################

# HITS PER USER in the recommendations #

#       that had been done             #

########################################

# R: recommendations that been done to u

# Hid: Set of the hidden items that were seen by the users

hits.per.user <- function(R,u) {
    length(intersect(R[R[,1]==u,2],Hid[Hid[,1]==u,2]))
}

####################################

# EVALUATION of the recommendations #

####################################

# Output: average by user of precision, recall and f1 (it was not
    used for the presented results); number of hits,
    recommendations and hidden items

rec.eval<-function(Recs){
    hits<-0
```

```
    recall.per.user<-NULL
    precision.per.user<-NULL
    f1.per.user<-NULL
    for(u in as.vector(unique(Recs[,1]))) {
        hits_tmp<-0
        hits_tmp<-hits.per.user(Recs,u)
        hits<-hits+hits_tmp
        r<-0
        r<-hits_tmp/length(Hid[Hid[,1]==u,1])
        p<-0
        p<-hits_tmp/length(Recs[Recs[,1]==u,1])
        recall.per.user <- c(recall.per.user,r)
        precision.per.user <- c(precision.per.user,p)
        if((!(r == 0)) | (!(p == 0))) {f1.per.user <- c(f1.per.user,((2*r*p)/(r+p)))}
        else {f1.per.user <- c(f1.per.user,0)}
    }
    OUT<-NULL
    OUT$hits<-hits
    OUT$nrecs <- length(Recs[,1])
    OUT$nhidden <- length(Hid[,1])
    recall<-sum(recall.per.user)/length(recall.per.user) # average by test user of recall
    OUT$recall<-recall
    precision<-sum(precision.per.user)/length(precision.per.user) # average by test user of precision
    OUT$precision<-precision
    f1<-sum(f1.per.user)/length(f1.per.user) # average by test user of f1
    OUT$f1<-f1
    OUT
}

##############################

# BATCH EVALUATION for FCBI #

##############################

# Output: set of the done recommendations, average time by test
    user of recommendation and measures evaluated

cf.reeval1 <- function(S,N) {
    res<- topNrec.batch1(S,N)
    Recs<-res$recs
    stats<-rec.eval(Recs)
    OUT<-NULL
    OUT$recs <- Recs
    OUT$stats <- stats
    OUT$Av_Time_Rec<-res$Av_Time_Rec
    OUT
}

##############################

# BATCH EVALUATION for FCBIE #

##############################

# Output: set of the done recommendations, average time by test
    user of recommendation and measures evaluated

cf.reeval2 <- function(S,N) {
    res<-topNrec.batch2(S,N)
    Recs<-res$recs
    stats<-rec.eval(Recs)
    OUT<-NULL
    OUT$recs <- Recs
    OUT$stats <- stats
```

```
    OUT$Av_Time_Rec<-res$Av_Time_Rec
    OUT
}

#################################

# BATCH EVALUATION for FCBIEInc #

#################################

# Determination of the top N recommendations for all the users in
    the Test Set to FCBIEInc

# Output: set of the done recommendations, measures evaluated and
    average time by test user of recommendation and update

cf.reeval3.s <- function(S,N,Int) {
    Recs<-data.frame(0)
    urecs<-NULL
    uids<-NULL
    count<-1
    aux_time_recA3<-matrix(data = NA, nrow = length(Test.Ids), ncol = 3, byrow = FALSE)
    aux_Update<-NULL
    recs<-NULL
    AUX<-NULL
    items<-items.in
    pos<-pos.in
    for (user1 in Test.Ids){
        begin_Rec<-Sys.time()
        recs<-topNrec.batch3(user1,S,N,items,pos)
        end_Rec<-Sys.time()
        aux_time_recA3[count,]<-c(user1, end_Rec-begin_Rec, ncol(S))
        count<-count+1
        uids<-c(uids,recs[,1])
        urecs<-c(urecs,recs[,2])
        begin_Update<-Sys.time()
        aux<-update.i.i.s(S,user1,Int,items,pos)
        S<-aux$S
        Int<-aux$Int
        items<-aux$items
        pos<-aux$pos
        end_Update<-Sys.time()
        aux_Update<-c(aux_Update,end_Update-begin_Update)
    }
    AUX<-matrix(c(uids,urecs),length(uids),2)
    Recs<-as.data.frame(AUX)
    stats<-rec.eval(Recs)
    OUT<-NULL
    OUT$recs <- Recs
    OUT$stats <- stats
    OUT$Av_Time_Rec<-sum(as.double(aux_time_recA3[,2]))/length(Test.Ids) # average time of recommendation
    OUT$Av_Time_Update<-sum(aux_Update)/length(Test.Ids) # average time of update
    OUT
}

#################################

# BATCH EVALUATION for FCBUEInc #

#################################

# Determination of the top N recommendations for all the users in
    the Test Set to FCBUEInc

# Output: set of the done recommendations, measures evaluated and
```

```
    average time by test user of recommendation and update

cf.reeval4.2.s <- function(S,N,Int.u,u.i) {
    res<-topNrec.batch4.2.s(S,N,Int.u,u.i)
    Recs<-res$recs
    stats<-rec.eval(Recs)
    OUT<-NULL
    OUT$recs <- Recs
    OUT$stats <- stats
    OUT$time.update<-res$Av_Time_Update # average update
    OUT$time.rec<-res$Av_Time_Rec # average time of recommendation
    OUT
}

########################################

# UPDATE of item-based similarity matrix #

########################################

# Output: updated similarity matrix S, items, pos and Int

# The update is done using only the known session of the active
    user

update.i.i.s<-function(S,user1,Int,items,pos){
    OUT<-NULL
    citem<-as.character(Test[Test[,1]==user1,2]) # items in the known session
    for(i in citem)
    {
        if(is.element(i,items)==FALSE)
        {
            pos<-c(pos,length(pos)+1) # update of pos: added 1
            names(pos)[length(pos)]<-i # update of pos: added the new item
            items[length(items)+1]<-i # update of items: added the new item
            Int<-rbind(Int,spam(0,ncol=ncol(Int),nrow=1)) # added a row to Int
            Int<-cbind(Int,spam(0,nrow=nrow(Int),ncol=1)) # added a column to Int
            S<-rbind(S,spam(0,ncol=ncol(S),nrow=1)) # added a row to S
            S<-cbind(S,spam(0,nrow=nrow(S),ncol=1)) # added a column to S
        }
    }
    Int[pos[citem],pos[citem]]<-as.matrix(Int[pos[citem],pos[citem]])+1 # update of Int
# update of S
    b<-as.vector(sqrt(diag(Int)))
    a<-as.matrix(as.vector(sqrt(diag(as.matrix((Int[pos[citem],pos[citem]])))))))
    den<-a%*%b # it determines the denominator of the similarity measure and the result is a square shaped matrix
    S[pos[citem],]<-as.vector(as.matrix(Int[pos[citem],]))/as.matrix(den)
    S[,pos[citem]]<-as.matrix(t(S[pos[citem],])) # it is used the transposed vector calculated previously
    OUT$items<-items
    OUT$pos<-pos
    OUT$Int<-Int
    OUT$S<-S
    OUT
}

########################################

# UPDATE of user-based similarity matrix #

########################################

# Output: updated similarity matrix S, items, pos.users, Users,
    Int.u and u.i

# The update is done using only the known session of the active
```

```
    user

update.u.u2.s<-function(S,user1,Int.u,u.i,pos.users,Users,items,pos){
    OUT<-NULL
    citem<-as.character(Test[Test[,1]==user1,2]) # items in the known session
    pos.users<-c(pos.users,length(pos.users)+1) # update of pos.users: added 1
    names(pos.users)[length(pos.users)]<-user1 # update of pos.users: added the active user
    Users[length(Users)+1]<-user1
    Int.u<-rbind(Int.u,spam(0,ncol=ncol(Int.u),nrow=1)) # added a row to Int.u
    Int.u<-cbind(Int.u,spam(0,nrow=nrow(Int.u),ncol=1)) # added a column to Int.u
    S<-rbind(S,spam(0,ncol=ncol(S),nrow=1)) # added a row to S
    S<-cbind(S,spam(0,nrow=nrow(S),ncol=1)) # added a column to S
    for(i in citem)
    {
        if(is.element(i,items)==FALSE)
        {
            pos<-c(pos,length(pos)+1) # update of pos: added 1
            names(pos)[length(pos)]<-i # update of pos: added the new item
            items[length(items)+1]<-i # update of items: added the new item
        }
    }
# update of u.i, it is needed to have always the knowledge of the
# items that were seen by the previous users
    u.i[[user1]]<-as.vector(citem)
# update of Int.u
    Int.u[pos.users[user1],]<-sapply(Users,function(w){length(intersect(u.i[[w]],citem))})
    Int.u[,pos.users[user1]]<-as.vector(as.matrix(Int.u[pos.users[user1],]))
# update of S
    b<-as.vector(sqrt(diag(Int.u)))
    a<-sqrt(as.matrix(Int.u[pos.users[user1],pos.users[user1]]))
    den<-a*b # it determines the denominator of the similarity measure and the result is a vector
    aux<-as.matrix(Int.u[pos.users[user1],])/den
    aux[is.nan(aux)]<-0 # aux is NA when den is 0, in this case the similarity is 0
    S[pos.users[user1],]<-aux
    S[,pos.users[user1]]<-t(S[pos.users[user1],])
    OUT$S<-S
    OUT$items<-items
    OUT$pos<-pos
    OUT$Users<-Users
    OUT$pos.users<-pos.users
    OUT$Int.u<-Int.u
    OUT$u.i<-u.i
    OUT
}

########

# FCBI #

########

# Reads data, setup train, test and hidden files

# Call the batch evaluation for FCBI

# Prints the results

cf.expA1 <- function() {
    begin_S<-Sys.time()
    S<-simmatrix1() # calculation of the similarity matrix
    end_S<-Sys.time()
    aux_t<-end_S-begin_S
    cat("\n")
    cat("...N=")
    n<-Nrep[1]
```

```
    cat(n,"- ")
    res<-cf.reeval1(S,n)
    stats<-res$stats
    results<-data.frame(N=n, HITS=stats$hits, N.RECS=stats$nrecs, N.HIDDEN=stats$nhidden, RECALL=stats$recall,
                        PRECISION=stats$precision, F1=stats$f1,TIME_CONSTR_S=aux_t,AVERAGE.TIME=res$Av_Time_Rec)
    for(N in Nrep[-1]) {
        cat(N,"-")
        res<-cf.reeval1(S,N)
        stats<-res$stats
        results <- rbind(results, c(N,stats$hits,stats$nrecs,stats$nhidden,stats$recall,stats$precision,stats$f1,
                            aux_t,res$Av_Time_Rec))
    }
    cat("\n")
    results
}

########

# FCBIE #

########

# Reads data, setup train, test and hidden files

# Call the batch evaluation for FCBIE

# Prints the results

cf.exp_spamA2 <- function() {
    begin_S<-Sys.time()
    S.s<-simmatrix2_spam() # calculation of the similarity matrix using package "spam"
    end_S<-Sys.time()
    aux_t<-end_S-begin_S
    cat("\n")
    cat("...N=")
    n<-Nrep[1]
    cat(n,"- ")
    res<-cf.reeval2(S.s,n)
    stats<-res$stats
    results<-data.frame(N=n, HITS=stats$hits, N.RECS=stats$nrecs, N.HIDDEN=stats$nhidden, RECALL=stats$recall,
                            PRECISION=stats$precision, F1=stats$f1,TIME_CONSTR_S=aux_t,AVERAGE.TIME=res$Av_Time_Rec)
    for(N in Nrep[-1]) {
        cat(N,"-")
        res<-cf.reeval2(S.s,N)
        stats<-res$stats
        results[nrow(results)+1,] <- c(N,stats$hits,stats$nrecs,stats$nhidden,stats$recall,stats$precision,stats$f1,
                                            aux_t,res$Av_Time_Rec)
    }
    cat("\n")
    results
}

############

# FCBIEInc #

############

# Reads data, setup train, test and hidden files

# Call the batch evaluation for FCBIEInc

# Prints the results

cf.exp_spamA3 <- function() {
```

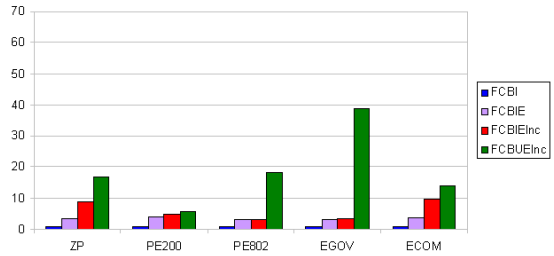
```
    begin_S<-Sys.time()
    Si.s<-simmatrix2_spam() # calculation of the initial similarity matrix using package "spam"
    end_S<-Sys.time()
    aux_t<-end_S-begin_S
    begin_cache<-Sys.time()
    Int<-cache_spam() # necessary matrix for the update of the similarity matrix
    end_cache<-Sys.time()
    aux_cache<-end_cache-begin_cache
    cat("\n")
    cat("...N=")
    n<-Nrep[1]
    cat(n,"- ")
    res<-cf.reeval3.s(Si.s,n,Int)
    stats<-res$stats
    results<-data.frame(N=n, HITS=stats$hits, N.RECS=stats$nrecs, N.HIDDEN=stats$nhidden, RECALL=stats$recall,
                        PRECISION=stats$precision, F1=stats$f1,TIME_CONSTR_S=aux_t,TIME_CACHE=aux_cache,
                        AVERAGE.TIME=res$Av_Time_Rec,AVERAGE.TIME.UPDATE=res$Av_Time_Update)
    for(N in Nrep[-1]) {
        cat(N,"-")
        res<-cf.reeval3.s(Si.s,N,Int)
        stats<-res$stats
        results[nrow(results)+1,] <- c(N,stats$hits,stats$nrecs,stats$nhidden,stats$recall,stats$precision,stats$f1,
                                       aux_t,aux_cache,res$Av_Time_Rec,res$Av_Time_Update)
    }
    cat("\n")
    results
}

############

# FCBUEInc #

############

# Reads data, setup train, test and hidden files

# Call the batch evaluation for FCBUEInc

# Prints the results

cf.exp_spamA4.2 <- function() {
    begin_S<-Sys.time()
    S_inicial.s<-simmatrix4.2_spam() # calculation of the initial similarity matrix using package "spam"
    end_S<-Sys.time()
    aux_t<-end_S-begin_S
    begin_cache<-Sys.time()
    cache<-cache.u_spam() # necessary data for the update of the similarity matrix
    end_cache<-Sys.time()
    aux_cache<-end_cache-begin_cache
    Int.u<-cache$Int.u
    u.i<-cache$u.i
    cat("\n")
    cat("...N=")
    n<-Nrep[1]
    cat(n,"- ")
    res<-cf.reeval4.2.s(S_inicial.s,n,Int.u,u.i)
    stats<-res$stats
    results<-data.frame(N=n, HITS=stats$hits, N.RECS=stats$nrecs, N.HIDDEN=stats$nhidden, RECALL=stats$recall,
                        PRECISION=stats$precision, F1=stats$f1,TIME_CONSTR_S=aux_t,TIME_CACHE=aux_cache,
                        AVERAGE.TIME.UPDATE=res$time.update,AVERAGE.TIME=res$time.rec)
    for(N in Nrep[-1]) {
        cat(N,"-")
        res<-cf.reeval4.2.s(S_inicial.s,N,Int.u,u.i)
        stats<-res$stats
        results[nrow(results)+1,] <- c(N,stats$hits,stats$nrecs,stats$nhidden,stats$recall,stats$precision,stats$f1,
```

```
                                        aux_t,aux_cache,res$time.update,res$time.rec)
    }
    cat("\n")
    results
}
```

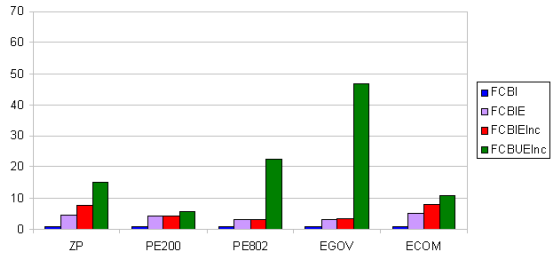
Apêndice B

Gráficos

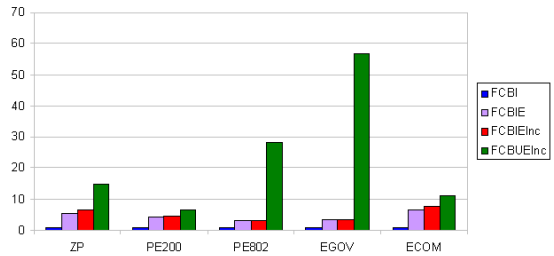
B.1 Tempo (relativo) de recomendação - *Neib*



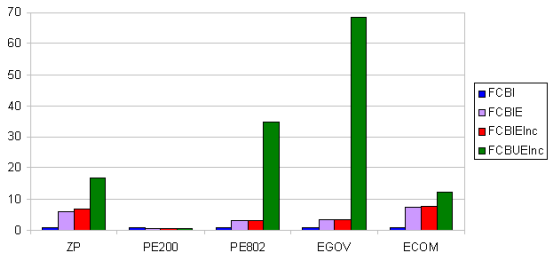
(a) Split=0,2



(b) Split=0,4

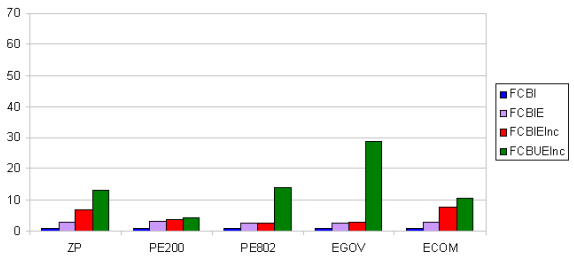


(c) Split=0,6

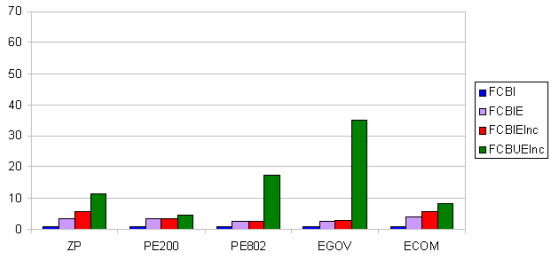


(d) Split=0,8

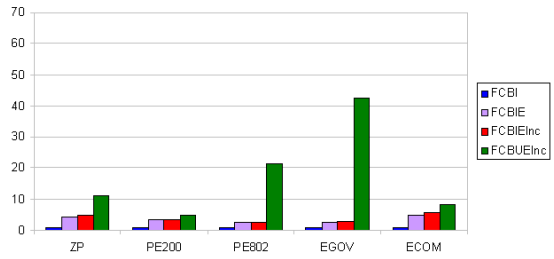
Figura B.1: *Tempo (relativo) de recomendação usando o tempo de FCBI como referência para Neib=3*



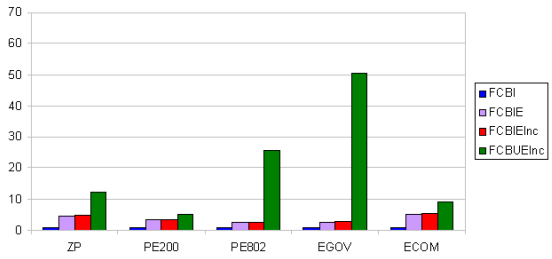
(a) Split=0,2



(b) Split=0,4



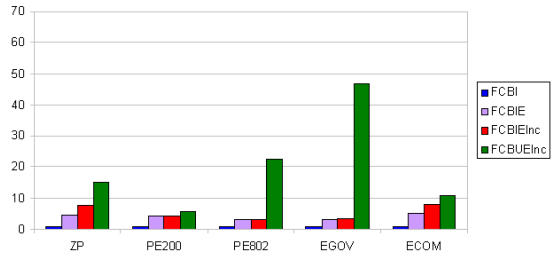
(c) Split=0,6



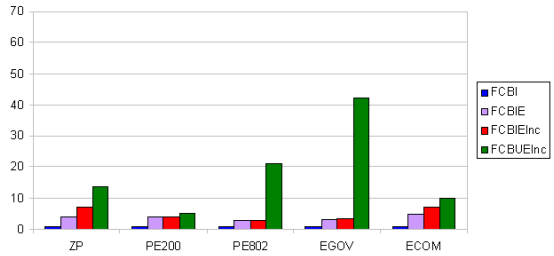
(d) Split=0,8

Figura B.2: *Tempo (relativo) de recomendação usando o tempo de FCBI como referência para Neib=10*

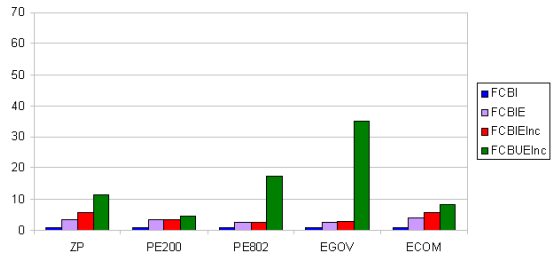
B.2 Tempo (relativo) de recomendação - *Split*



(a) Neib=3

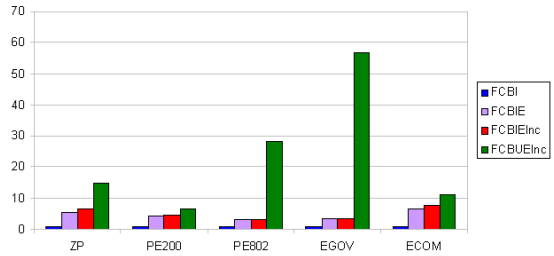


(b) Neib=5

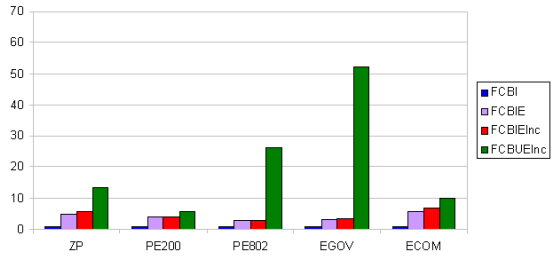


(c) Neib=10

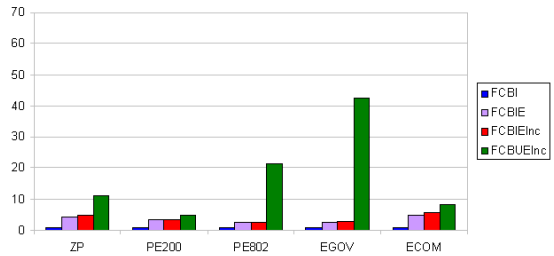
Figura B.3: *Tempo (relativo) de recomendação usando o tempo de FCBI como referência para Split=0,4*



(a) Neib=3



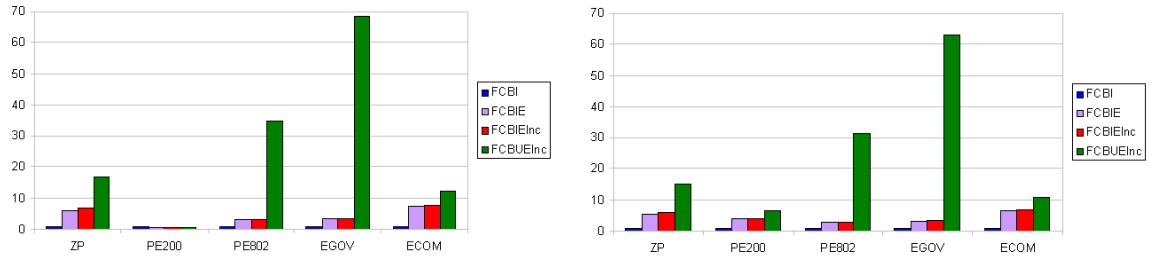
(b) Neib=5



(c) Neib=10

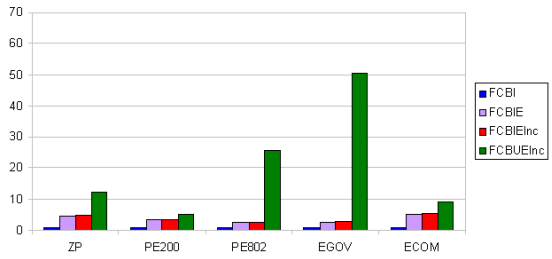
Figura B.4: *Tempo (relativo) de recomendação usando o tempo de FCBI como referência para Split=0,6*

B.3 Relação entre o número de utilizadores e o número de itens - Tempo de recomendação



(a) Neib=3

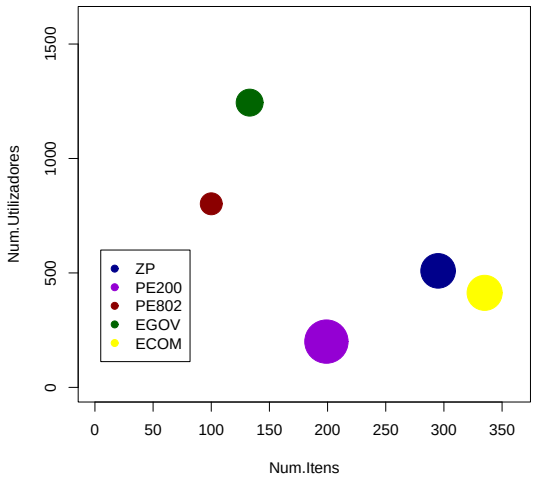
(b) Neib=5



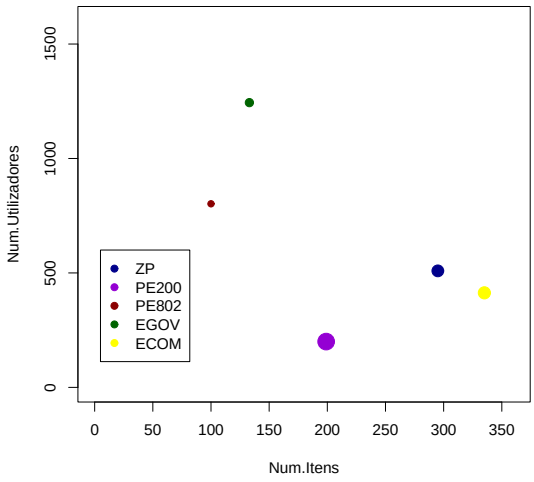
(c) Neib=10

Figura B.5: *Tempo (relativo) de recomendação usando o tempo de FCBI como referência para Split=0,8*

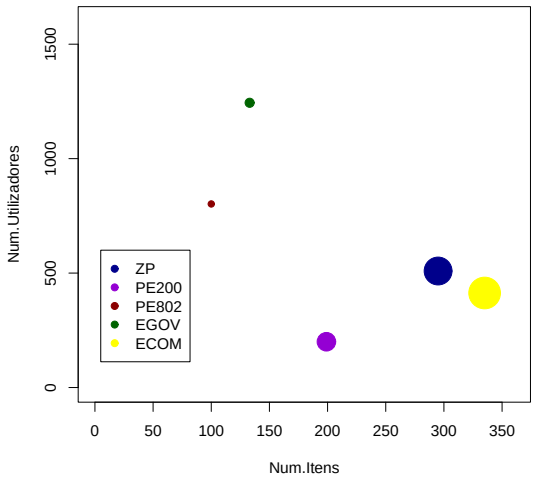
B.4 Relação entre o número de utilizadores e o número de itens - Tempo de actualização



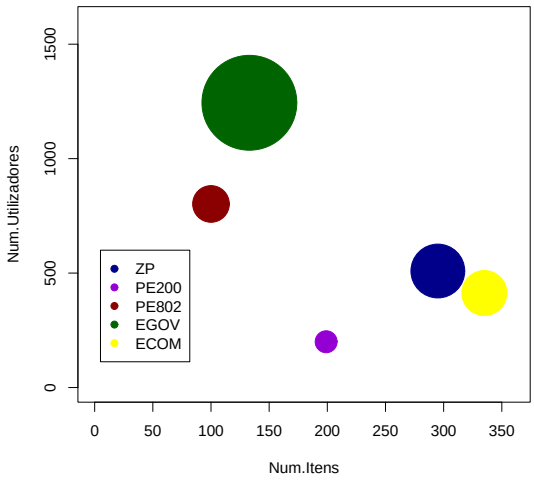
(a) FCBI tempo em centésimos de segundo



(b) FCBIE tempo em décimos de segundo



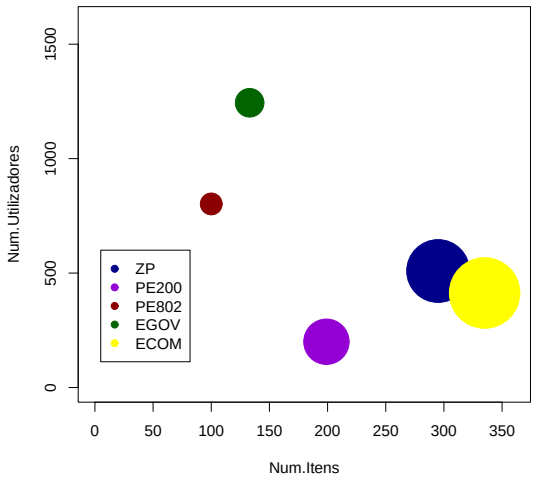
(c) FCBIEInctempo em décimos de segundo



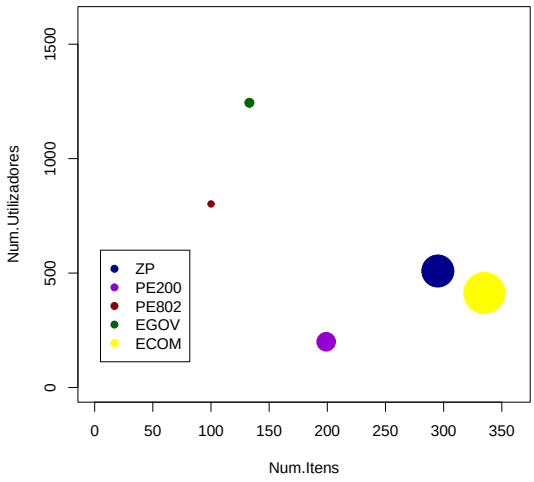
(d) FCBUEInc tempo em décimos de segundo

Figura B.6: *Relação entre o número de utilizadores e o número de itens para Split=0,2, o raio da bola é proporcional ao tempo médio real de recomendação*

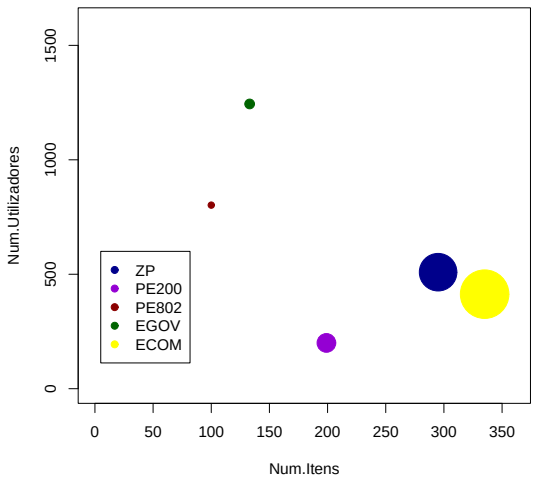
B.5 Recall obtido para as combinações possíveis de *Split* e *Neib*



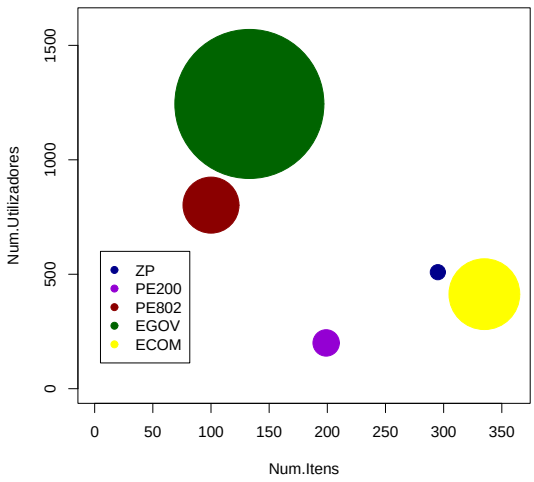
(a) FCBI tempo em centésimos de segundo



(b) FCBIE tempo em décimos de segundo

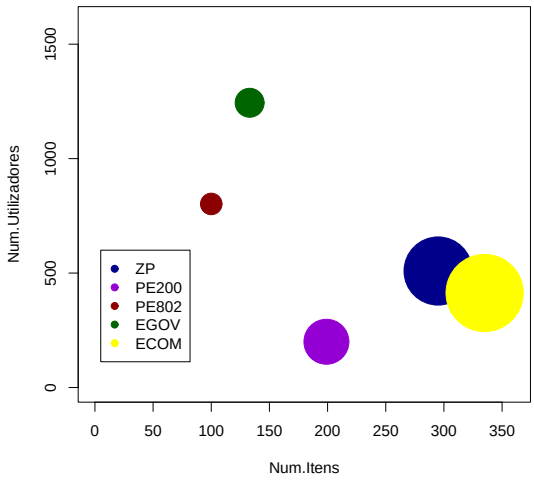


(c) FCBIEInc tempo em décimos de segundo

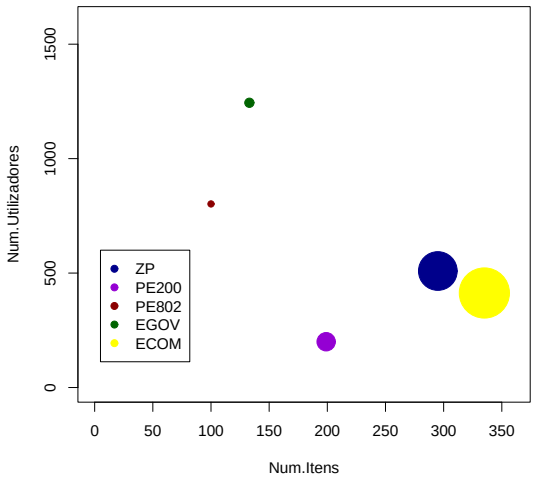


(d) FCBUEInc tempo em décimos de segundo

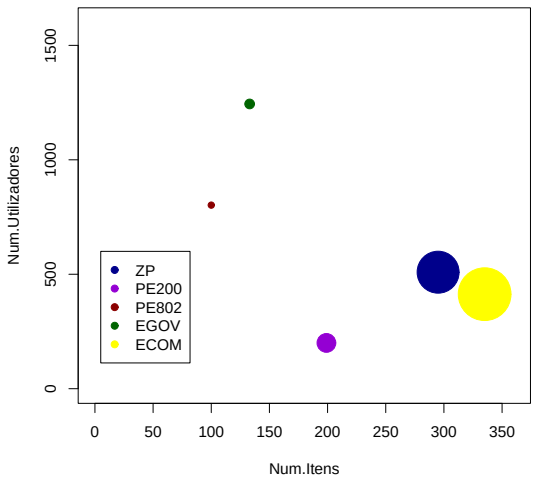
Figura B.7: *Relação entre o número de utilizadores e o número de itens para Split=0,6, o raio da bola é proporcional ao tempo médio real de recomendação*



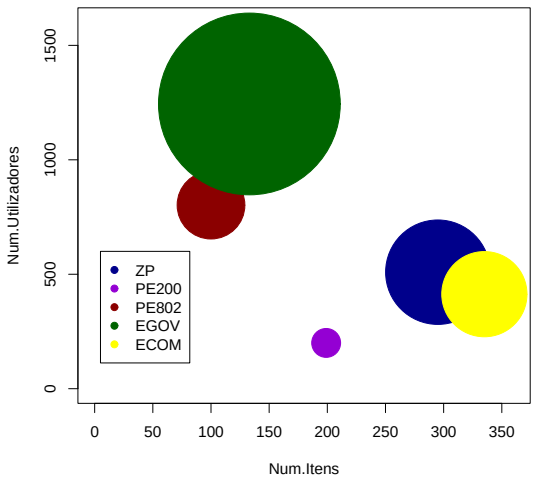
(a) FCBI tempo em centésimos de segundo



(b) FCBIE tempo em décimos de segundo

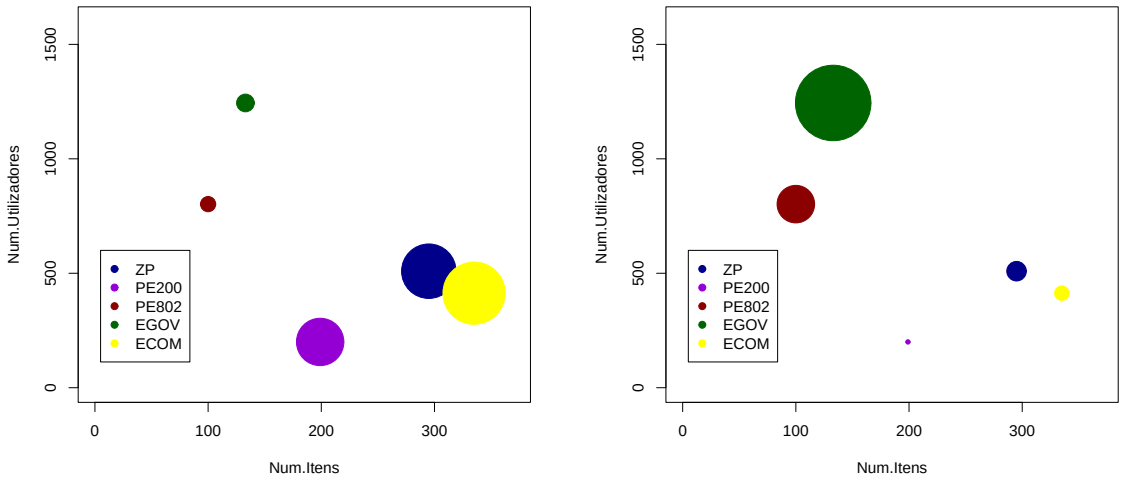


(c) FCBIEInc tempo em décimos de segundo



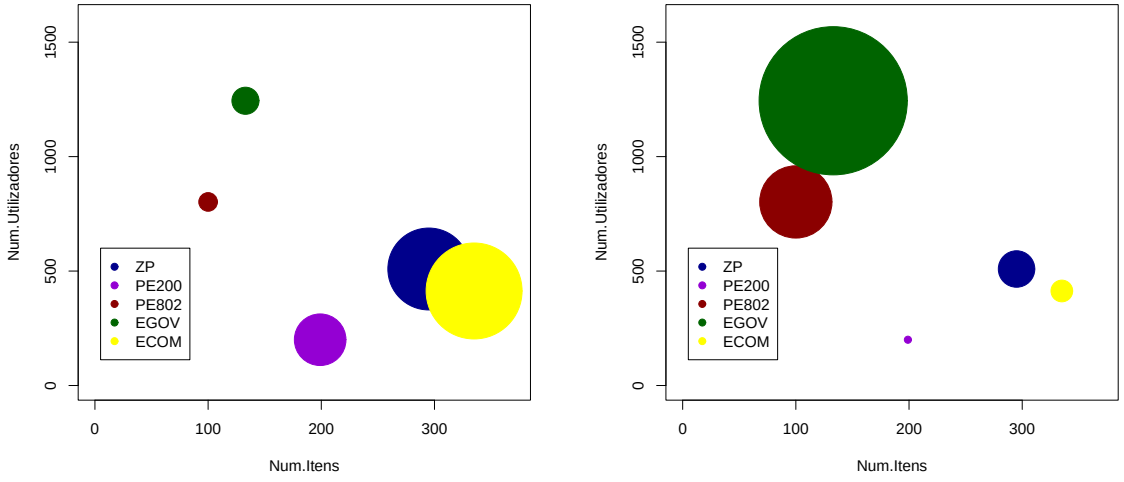
(d) FCBUEInc tempo em décimos de segundo

Figura B.8: *Relação entre o número de utilizadores e o número de itens para Split=0,8, o raio da bola é proporcional ao tempo médio real de recomendação*



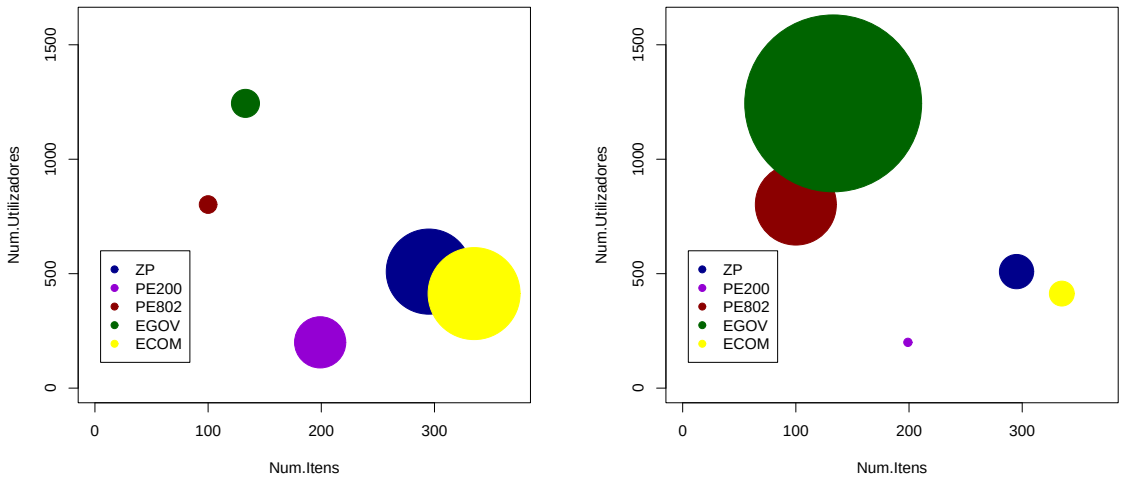
(a) FCBIEInc tempo em décimos de segundo (b) FCBUEInc tempo em décimos de segundo

Figura B.9: *Relação entre o número de utilizadores e o número de itens para Split=0,2, o raio da bola é proporcional ao tempo médio real de actualização*



(a) FCBIEInc tempo em décimos de segundo (b) FCBUEInc tempo em décimos de segundo

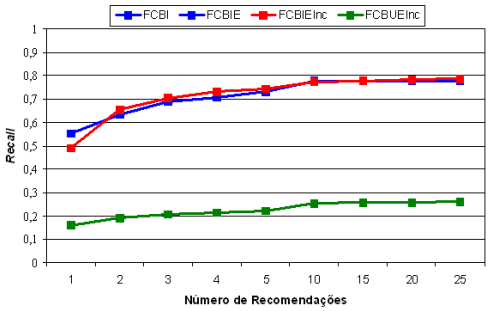
Figura B.10: *Relação entre o número de utilizadores e o número de itens para Split=0,6, o raio da bola é proporcional ao tempo médio real de actualização*



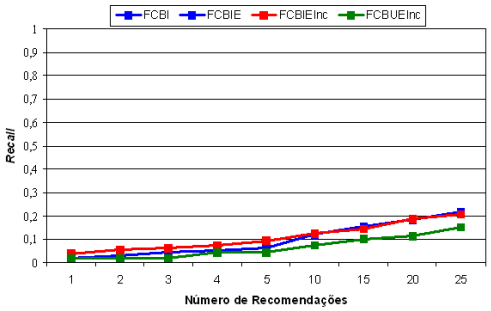
(a) FCBIEInc tempo em décimos de segundo

(b) FCBUEInc tempo em décimos de segundo

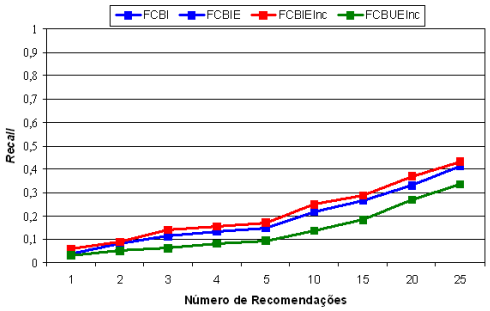
Figura B.11: *Relação entre o número de utilizadores e o número de itens para Split=0,8, o raio da bola é proporcional ao tempo médio real de actualização*



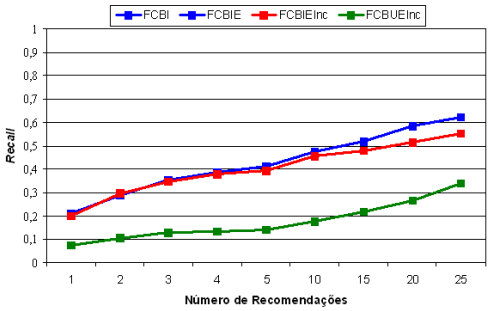
(a) ZP



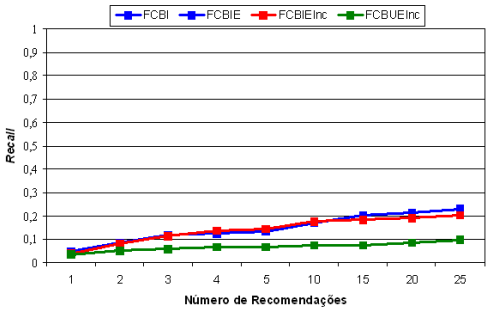
(b) PE200



(c) PE802

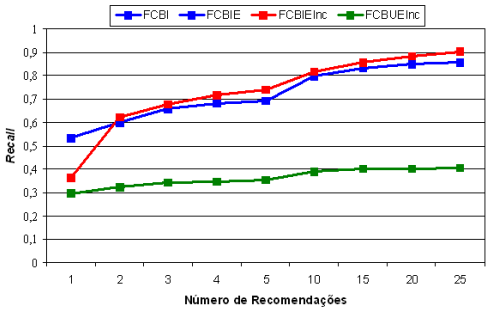


(d) EGOV

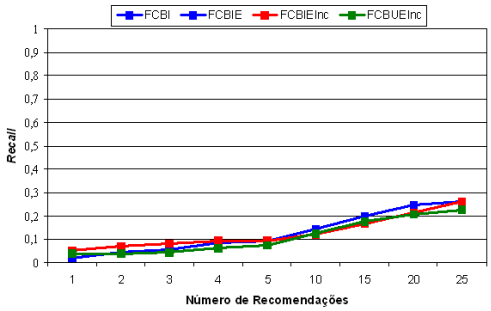


(e) ECOM

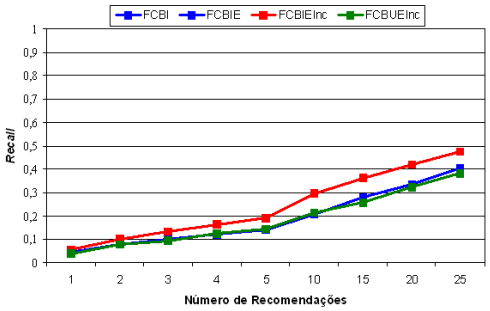
Figura B.12: *Recall obtido para Split=0,2 e Neib=3*



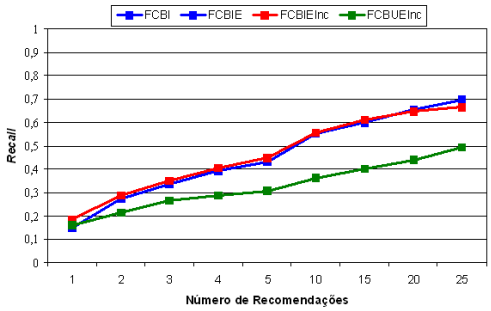
(a) ZP



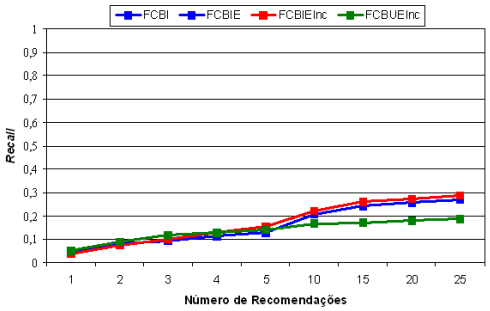
(b) PE200



(c) PE802

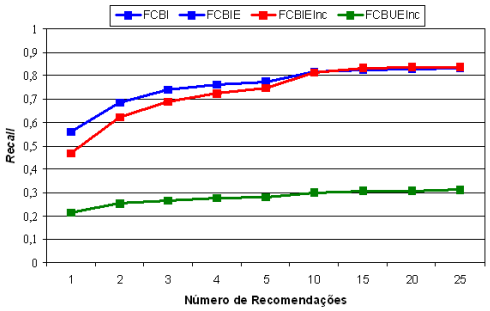


(d) EGOV

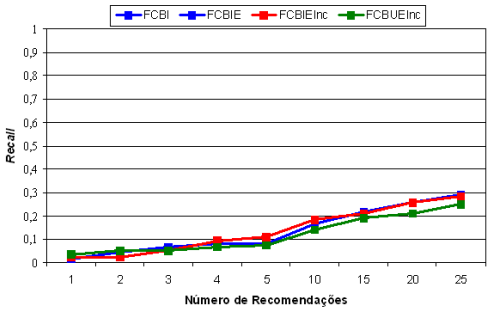


(e) ECOM

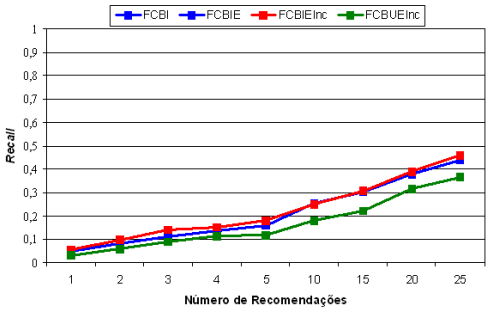
Figura B.13: *Recall obtido para Split=0,2 e Neib=10*



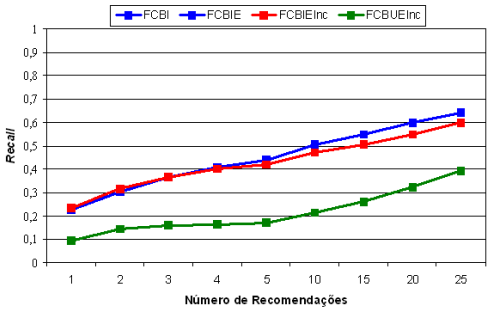
(a) ZP



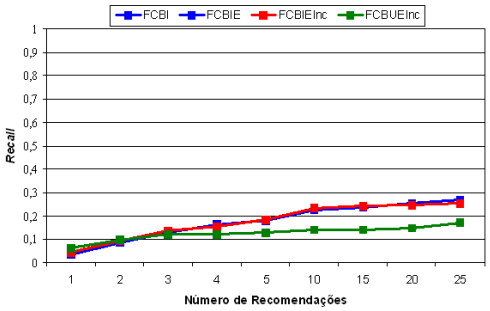
(b) PE200



(c) PE802

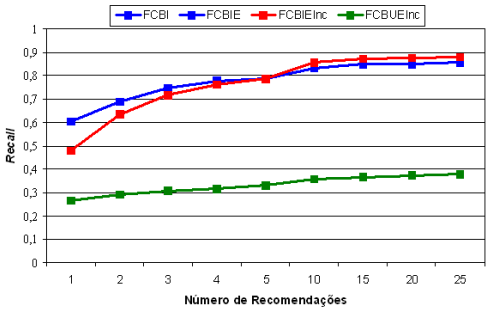


(d) EGOV

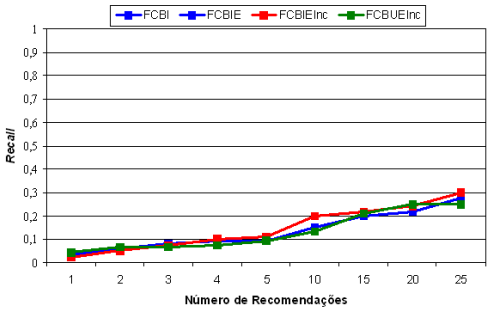


(e) ECOM

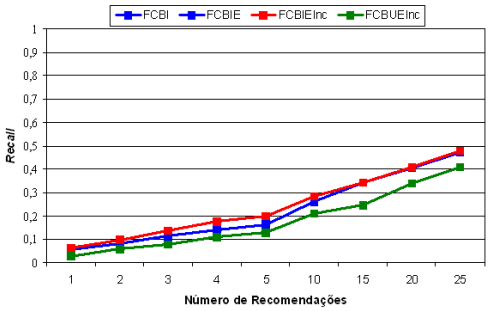
Figura B.14: *Recall obtido para Split=0,4 e Neib=3*



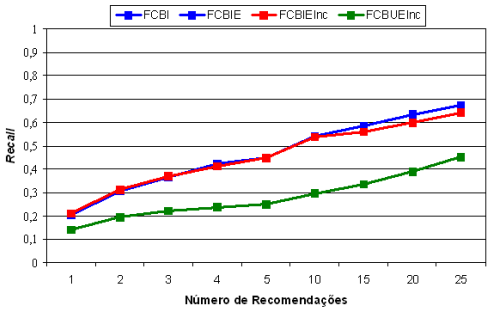
(a) ZP



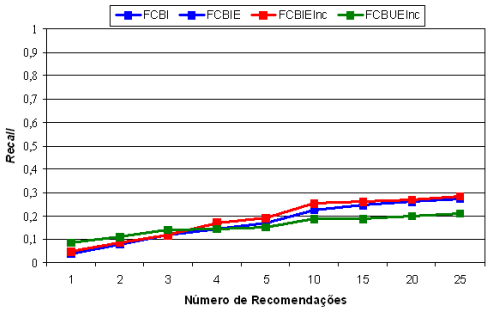
(b) PE200



(c) PE802

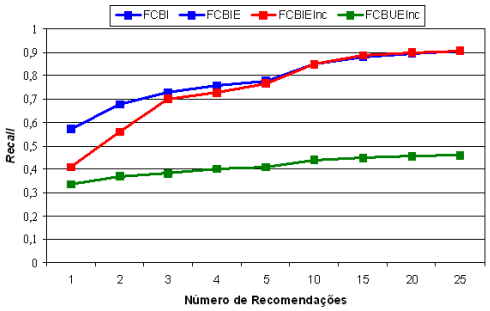


(d) EGOV

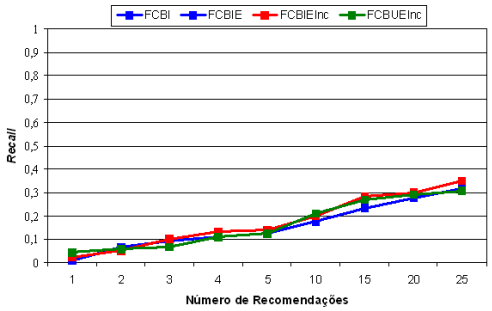


(e) ECOM

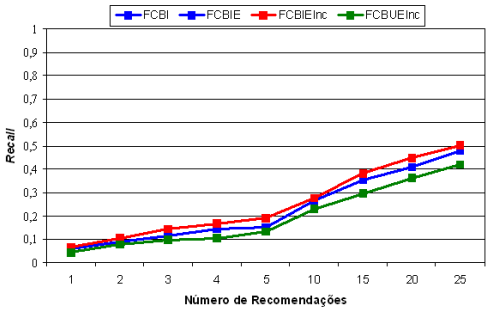
Figura B.15: *Recall obtido para Split=0,4 e Neib=5*



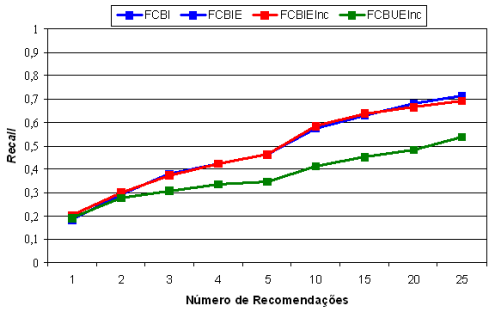
(a) ZP



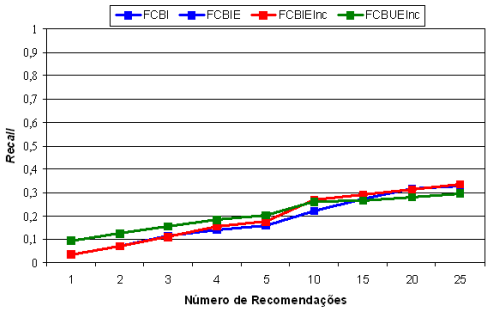
(b) PE200



(c) PE802

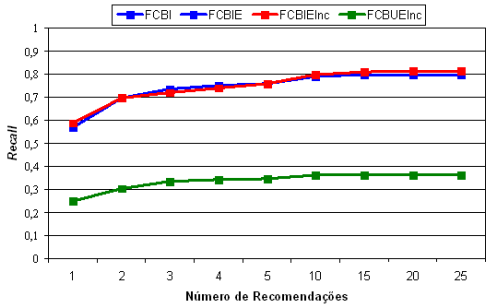


(d) EGOV

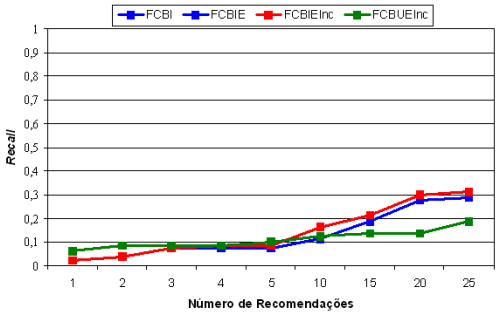


(e) ECOM

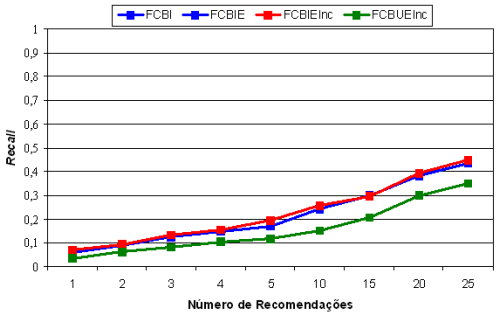
Figura B.16: *Recall obtido para Split=0,4 e Neib=10*



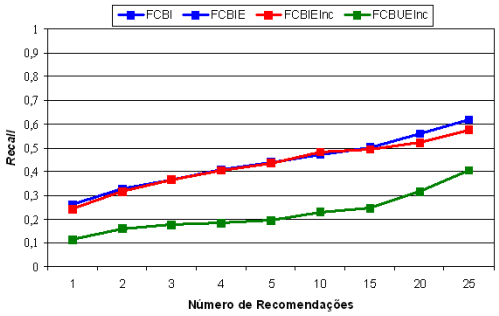
(a) ZP



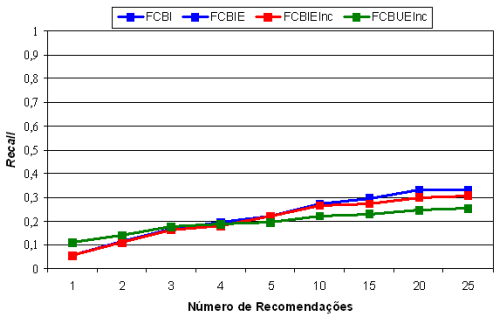
(b) PE200



(c) PE802

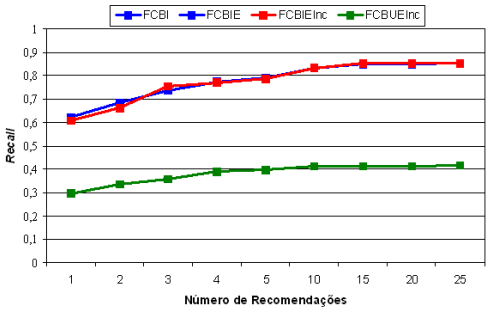


(d) EGOV

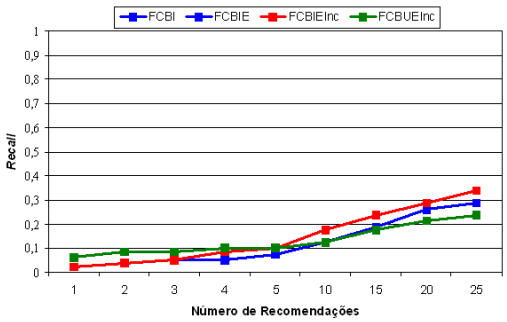


(e) ECOM

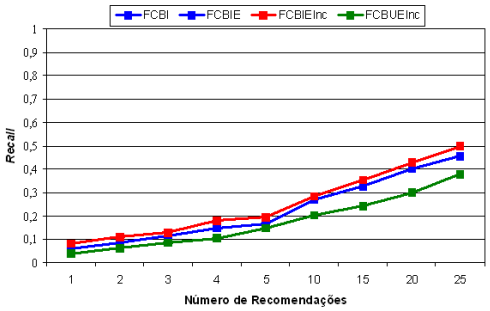
Figura B.17: *Recall obtido para Split=0,6 e Neib=3*



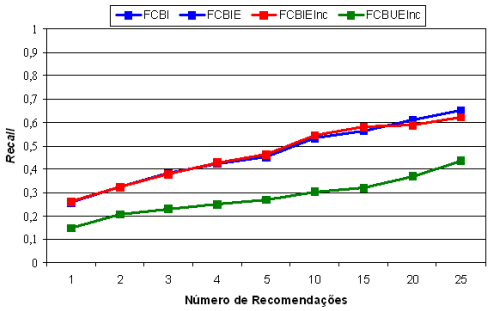
(a) ZP



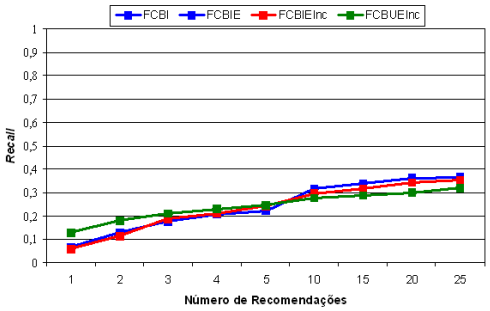
(b) PE200



(c) PE802

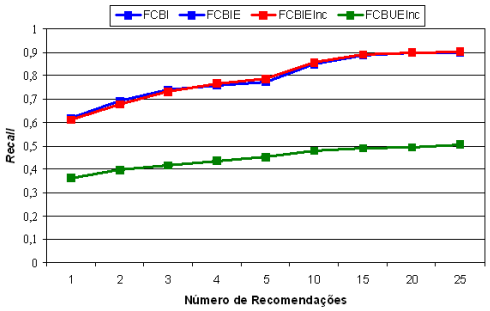


(d) EGOV

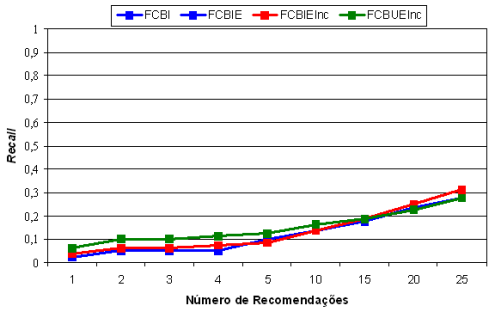


(e) ECOM

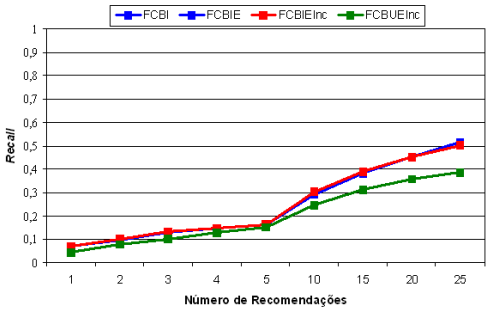
Figura B.18: *Recall obtido para Split=0,6 e Neib=5*



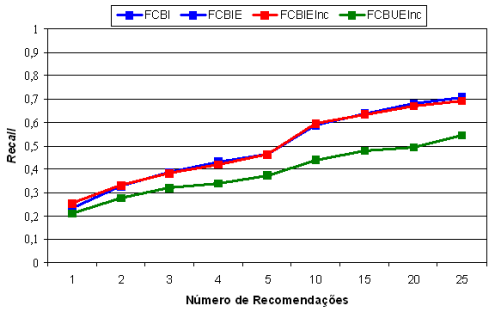
(a) ZP



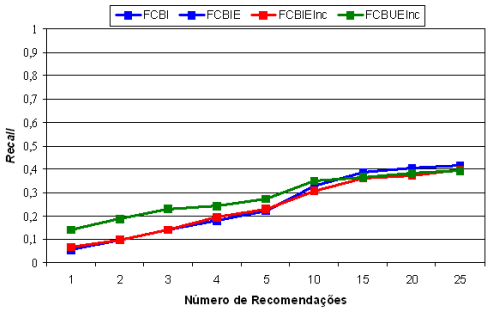
(b) PE200



(c) PE802

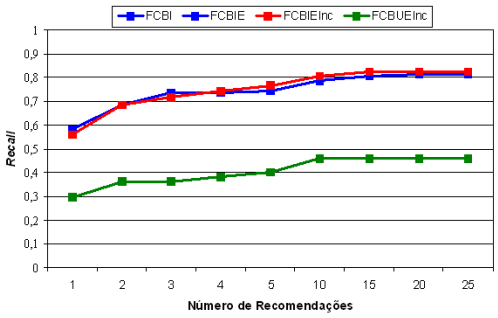


(d) EGOV

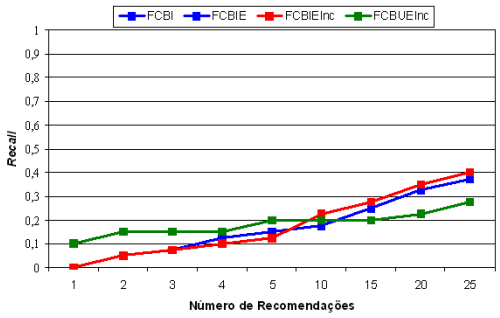


(e) ECOM

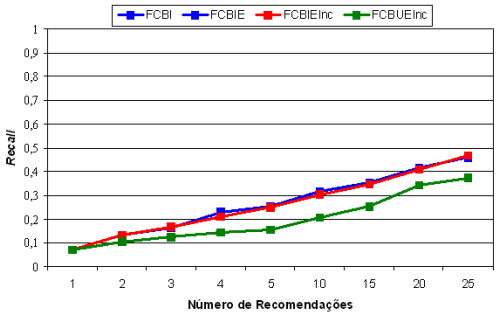
Figura B.19: *Recall obtido para Split=0,6 e Neib=10*



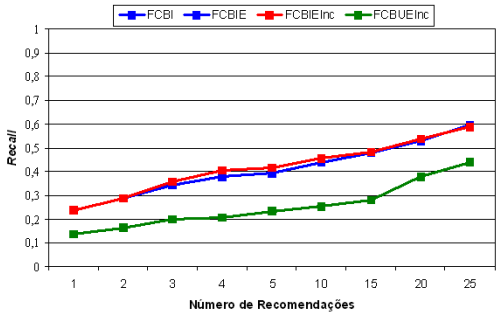
(a) ZP



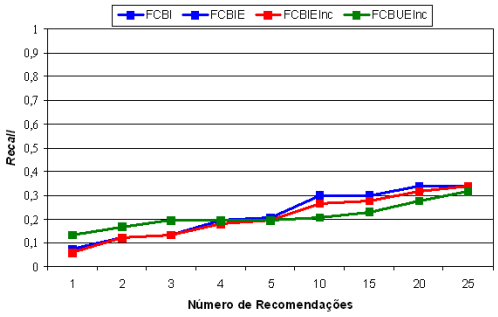
(b) PE200



(c) PE802

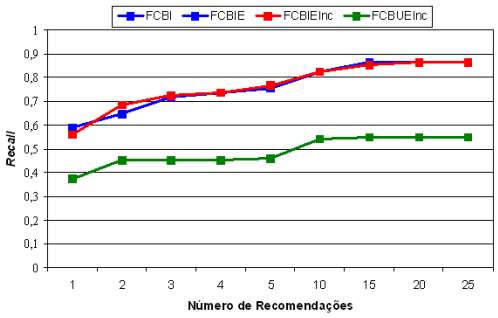


(d) EGOV

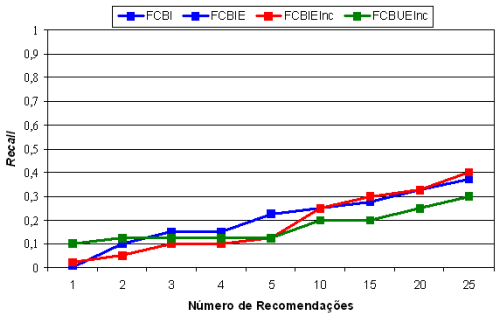


(e) ECOM

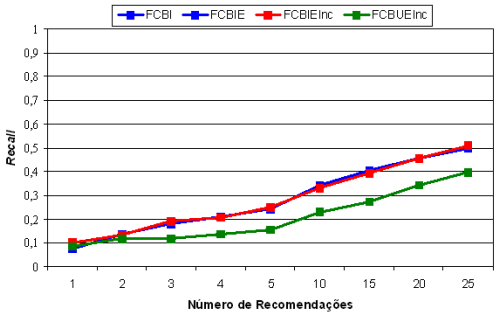
Figura B.20: *Recall obtido para Split=0,8 e Neib=3*



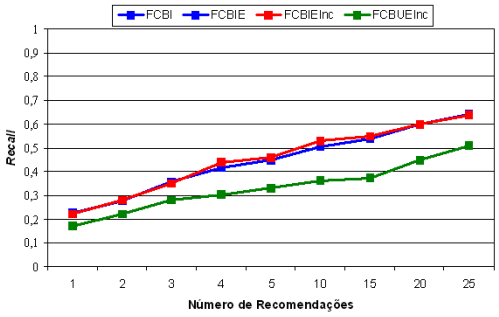
(a) ZP



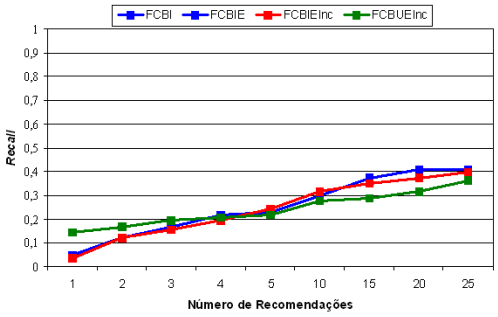
(b) PE200



(c) PE802

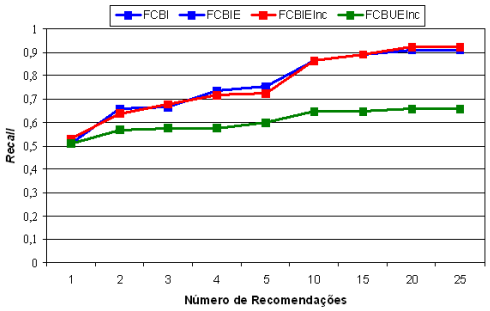


(d) EGOV

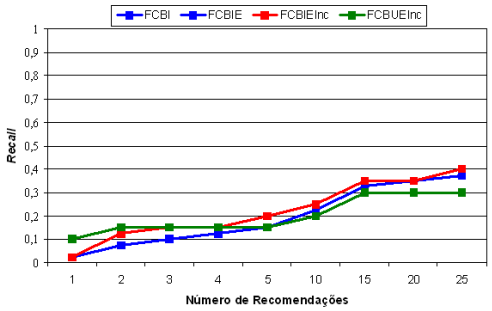


(e) ECOM

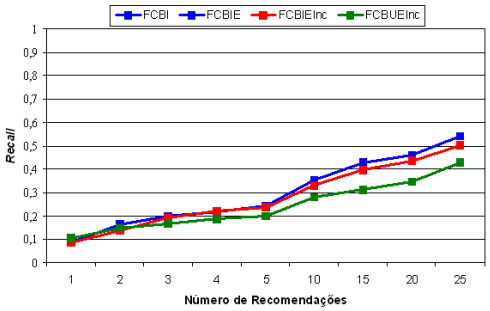
Figura B.21: *Recall obtido para Split=0,8 e Neib=5*



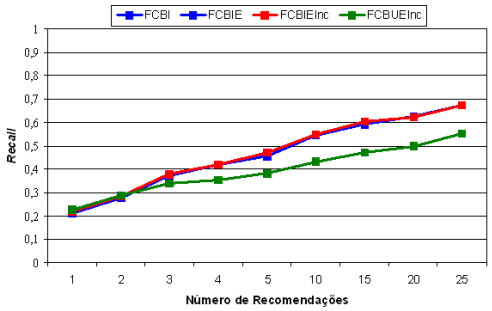
(a) ZP



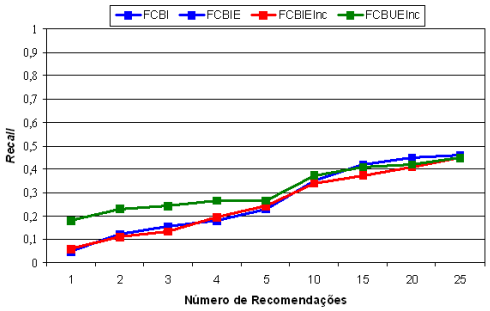
(b) PE200



(c) PE802



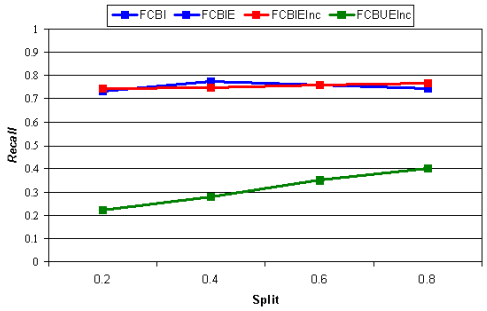
(d) EGOV



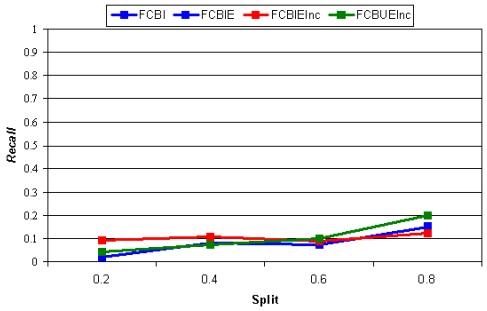
(e) ECOM

Figura B.22: *Recall obtido para Split=0,8 e Neib=10*

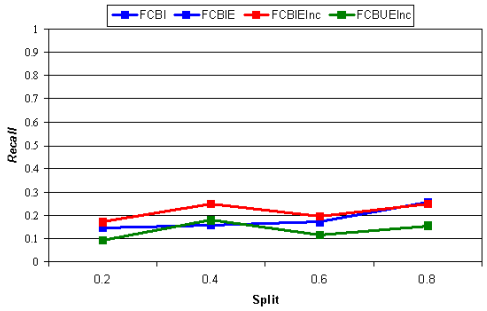
B.6 Recall obtido considerando que se está a fazer uma recomendação de 5 itens - variando *Neib*



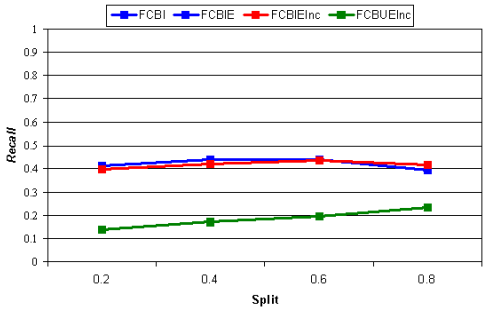
(a) ZP



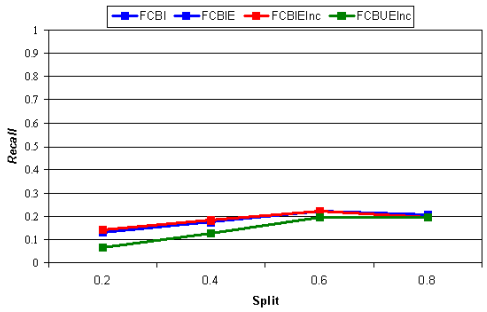
(b) PE200



(c) PE802

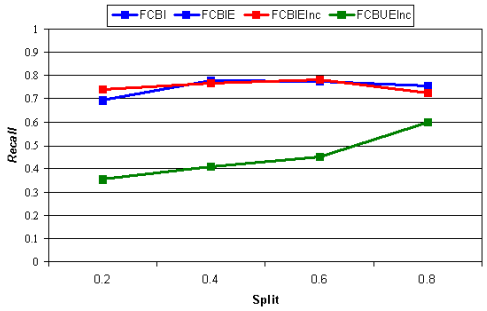


(d) EGOV

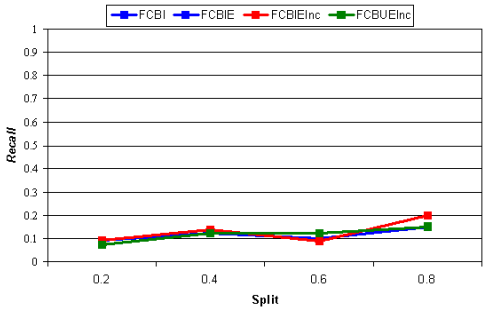


(e) ECOM

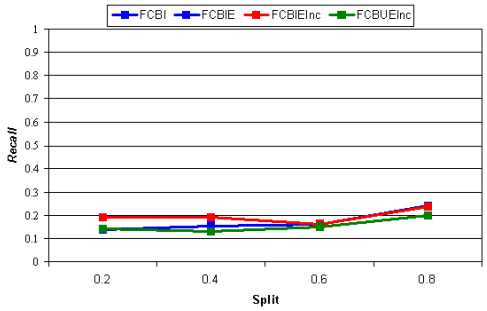
Figura B.23: *Recall obtido para Neib=3 considerando que se está a fazer uma recomendação de 5 itens por utilizador de teste*



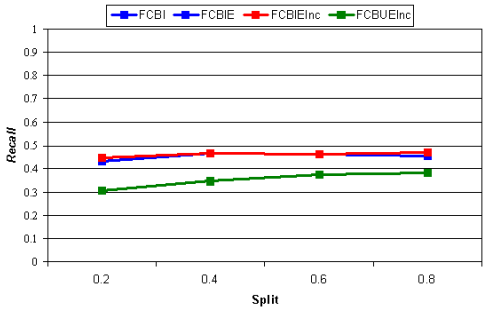
(a) ZP



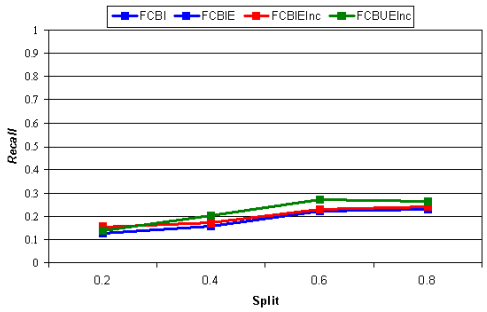
(b) PE200



(c) PE802



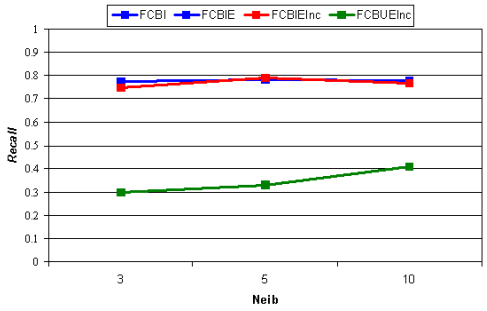
(d) EGOV



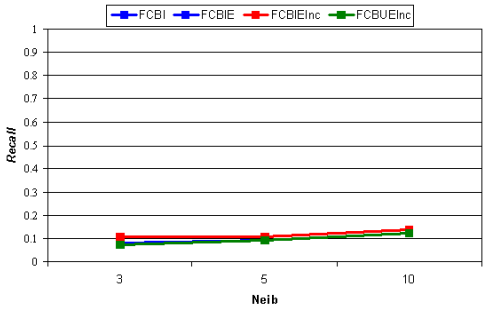
(e) ECOM

Figura B.24: *Recall obtido para Neib=10 ao recomendar-se 5 itens por utilizador de teste*

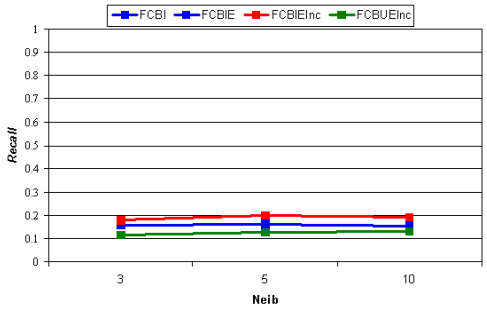
B.7 Recall obtido considerando que se está a fazer uma recomendação de 5 itens - variando *Split*



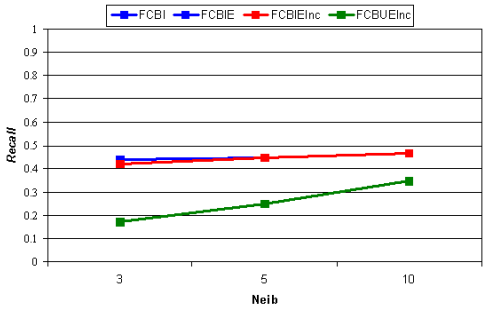
(a) ZP



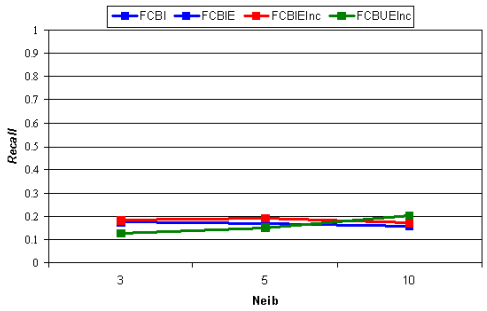
(b) PE200



(c) PE802

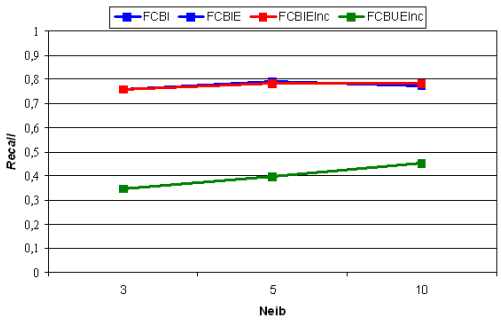


(d) EGOV

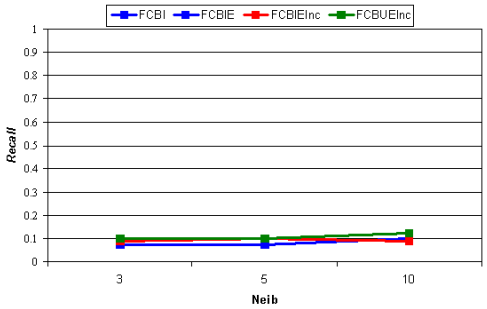


(e) ECOM

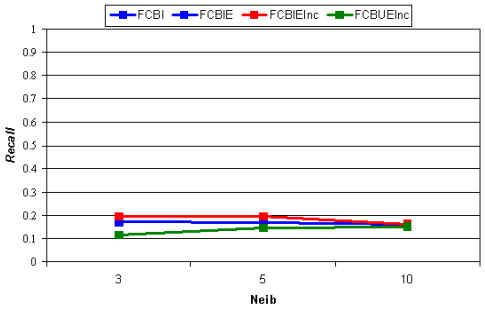
Figura B.25: *Recall obtido para Split=0,4 considerando que se está a fazer uma recomendação de 5 itens por utilizador de teste*



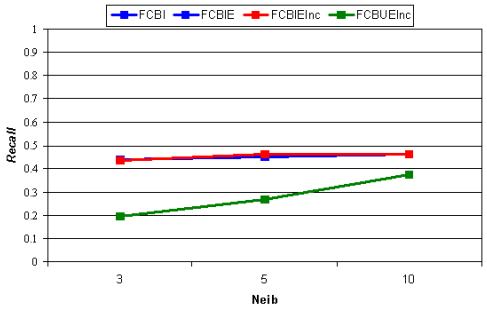
(a) ZP



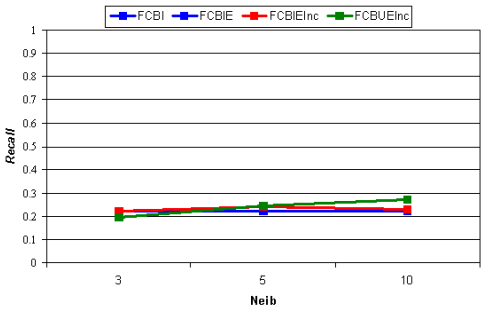
(b) PE200



(c) PE802

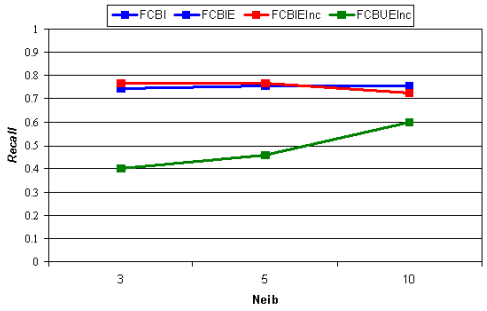


(d) EGOV

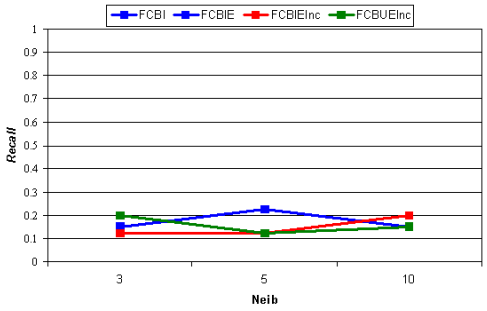


(e) ECOM

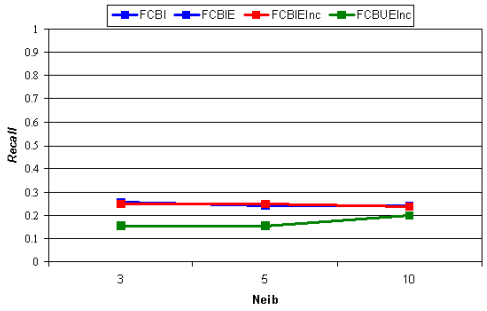
Figura B.26: *Recall obtido para Split=0,6 considerando que se está a fazer uma recomendação de 5 itens por utilizador de teste*



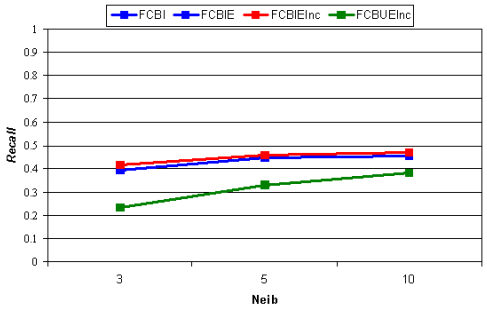
(a) ZP



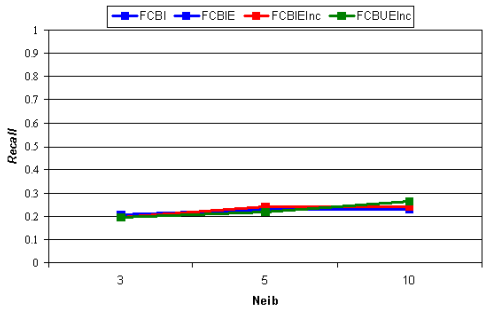
(b) PE200



(c) PE802



(d) EGOV



(e) ECOM

Figura B.27: *Recall obtido para Split=0,8 considerando que se está a fazer uma recomendação de 5 itens por utilizador de teste*

Bibliografia

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, pages 81–92, 2003.

[3] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. pages 6–12. AAAI Press, 1995.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In L. Popa, editor, *PODS*, pages 1–16. ACM, 2002.

[5] M. Balabanovic and Y. Shoham. Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40:66–72, 1997.

[6] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In J. W. Shavlik, editor, *ICML*, pages 46–54. Morgan Kaufmann, 1998.

[7] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In G. F. Cooper and S. Moral, editors, *UAI*, pages 43–52. Morgan Kaufmann, 1998.

[8] R. Burke. Integrating knowledge-based and collaborative-filtering recommender systems. In *In AAAI Workshop on AI in Electronic Commerce*, pages 69–72. AAAI, 1999.

[9] R. Burke. Knowledge-based recommender systems. In *Encyclopedia of Library and Information Systems*, page 2000. Marcel Dekker, 2000.

[10] R. D. Burke. Hybrid recommender systems: Survey and experiments. *User Model. User-Adapt. Interact.*, 12(4):331–370, 2002.

[11] H. Chen and M. Chau. Web mining: Machine learning for web applications. *Annual Review of Information Science and Technology*, 38:289–329, 2004.

[12] R. Cooley, B. Mobasher, and J. Srivastava. Web mining: Information and pattern discovery on the world wide web. In *ICTAI*, pages 558–567, 1997.

[13] P. Domingos and G. Hulten. Mining high-speed data streams. In *KDD*, pages 71–80, 2000.

[14] M. A. Domingues, A. M. Jorge, C. Soares, J. P. Leal, and P. Machado. A data warehouse for web intelligence. In *Proceedings of the Workshop on Business Intelligence 2007, in the 13th Portuguese Conference on Artificial Intelligence (EPIA 2007)*, Lecture Notes in Computer Science. to be published by Springer, 2007.

[15] M. A. Domingues, A. M. Jorge, C. Soares, J. P. Leal, and P. Machado. A platform to support web site adaptation and monitoring of its effects: A case study. In *In Proceedings of the 6th Workshop on Intelligent Techniques for Web Personalization and Recommender Systems (ITWP 2008)*, pages 29–36, 2008.

[16] O. Etzioni. The world-wide web: Quagmire or gold mine? *Commun. ACM*, 39(11):65–68, 1996.

[17] F. J. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. R. Santos. Incremental rule learning and border examples selection from numerical data streams. *J. UCS*, 11(8):1426–1439, 2005.

[18] J. Gama and P. P. Rodrigues. Learning from data streams. *to appear in Encyclopedia of Data Warehousing and Mining - 2nd Edition*, 2008], url = http://www.liaad.up.pt/pub/2008/GR08a.

[19] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35:61–70, 1992.

[20] W. C. Hill, L. Stead, M. Rosenstein, and G. W. Furnas. Recommending and evaluating choices in a virtual community of use. In *CHI*, pages 194–201, 1995.

[21] Z. Huang, H. Chen, and D. D. Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):116–142, 2004.

[22] A. M. Jorge, J. P. Leal, C. Soares, and J. Borges. Web site adaptation and automation: The site-o-matic project book. Internal Report, LIAAD, 2008.

[23] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *VLDB*, pages 180–191. Morgan Kaufmann, 2004.

[24] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, 1997.

[25] B. Krulwich. Learning document category descriptions through the extraction of semantically significant phrases. In *In Proceedings of the IJCAI'95 Workshop on Data Engineering for Inductive Learning*, 1995.

[26] W. S. Lee. Collaborative learning for recommender systems. In *In Proc. 18th International Conf. on Machine Learning*, pages 314–321. Morgan Kaufmann, 2001.

[27] H. Lieberman. Letizia: an agent that assists web browsing. pages 924–929, 1995.

[28] G. Linden, B. Smith, and J. York. Industry report: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Distributed Systems Online*, 4(1), 2003.

[29] C. Miranda and A. M. Jorge. Experiments with incremental collaborative filtering for implicit binary ratings. In *Proceedings of the 1st Workshop on Web and Text Intelligence (WTI 08)*, pages 35–42, Salvador, Bahia, Brazil, October 2008.

[30] C. Miranda and A. M. Jorge. Incremental collaborative filtering for binary ratings. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, 1:389–392, 2008.

[31] D. Mladeni. Personal webwatcher: design and implementation.

[32] B. Mobasher and S. S. Anand, editors. *Intelligent Techniques for Web Personalization, IJCAI 2003 Workshop, ITWP 2003, Acapulco, Mexico, August 11, 2003, Revised Selected Papers*, volume 3169 of *Lecture Notes in Computer Science*. Springer, 2005.

[33] O. Nasraoui, J. Cerwinske, C. Rojas, and F. A. González. Collaborative filtering in dynamic streaming environments. In P. S. Yu, V. J. Tsotras, E. A. Fox, and B. Liu, editors, *CIKM*, pages 794–795. ACM, 2006.

[34] O. Nasraoui and C. Petenes. Combining web usage mining and fuzzy inference for website personalization. In *In Proc. of WebKDD*, pages 37–46, 2003.

[35] O. Nasraoui, C. Rojas, and C. Cardona. A framework for mining evolving trends in web data streams using dynamic learning and retrospective validation. *Computer Networks*, 50(10):1488–1512, 2006.

[36] M. Papagelis. Crawling the Algorithmic Foundations Of Recommendation Technologies. Master's thesis, University of Crete, School of Sciences and Engineering, Computer Science Department, Greece, March 2005.

[37] M. Papagelis and D. Plexousakis. Qualitative analysis of userbased and item-based prediction algorithms for recommendation systems, cia 2004. *Journal of Engineering Applications of Artificial Intelligence*, 18:781–789, 2005.

[38] M. Papagelis, I. Rousidis, D. Plexousakis, and E. Theoharopoulos. Incremental collaborative filtering for highly-scalable recommendation algorithms. In M.-S. Hacid, N. V. Murray, Z. W. Ras, and S. Tsumoto, editors, *ISMIS*, volume 3488 of *Lecture Notes in Computer Science*, pages 553–561. Springer, 2005.

[39] M. Pazzani, D. Billsus, S. Michalski, and J. Wnek. Learning and revising user profiles: The identification of interesting web sites. In *Machine Learning*, pages 313–331, 1997.

[40] M. J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13:393–408, 1999.

[41] A. Popescul, L. H. Ungar, D. M. Pennock, and S. Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In J. S. Breese and D. Koller, editors, *UAI*, pages 437–444. Morgan Kaufmann, 2001.

[42] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007.

[43] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *IUI*, pages 127–134, 2002.

[44] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW*, pages 175–186, 1994.

[45] P. Resnick and H. R. Varian. Recommender systems - introduction to the special section. *Commun. ACM*, 40(3):56–58, 1997.

[46] F. Ruas, W. M. Jr., P. Araújo, and F. Ribeiro. Modeling web site personalization strategies. *J. Braz. Comp. Soc.*, 8(2):44–54, 2002.

[47] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *ACM Conf. on Electronic Commerce*, pages 158–167, 2000.

[48] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[49] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Application of dimensionality reduction in recommender systems - a case study. In *In ACM WebKDD Workshop*, 2000.

[50] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR*, pages 253–260. ACM, 2002.

[51] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *Human Factors in Computing Systems, CHI 95 Proceedings*, pages 210–217, 1995.

[52] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1(2):12–23, 2000.

[53] L. H. Ungar, D. P. Foster, E. Andre, S. Wars, F. S. Wars, D. S. Wars, and J. H. Whispers. Clustering methods for collaborative filtering. AAAI Press, 1998.

[54] K. Yu, A. Schwaighofer, V. Tresp, X. Xu, and H.-P. Kriegel. Probabilistic memory-based collaborative filtering. *IEEE Trans. Knowl. Data Eng.*, 16(1):56–69, 2004.

[55] K. Yu, X. Xu, J. Tao, M. Ester, and H.-P. Kriegel. Instance selection techniques for memory-based collaborative filtering. In R. L. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, editors, *SDM*. SIAM, 2002.

[56] C. Zeng, C.-X. Xing, and L.-Z. Zhou. Similarity measure and instance selection for collaborative filtering. In *WWW*, pages 652–658, 2003.