Desenvolvimento de um plugin para navegador, utilizando técnicas de filtragem colaborativa, para realizar recomendações no repositório Merlot

Gustavo Magalhães, Luan Einhardt, Marcelo Fay

Bacharelado em Ciência da Computação – Universidade Federal de Pelotas (UFPel)
{gldmagalhaes, ldseinhardt, mlcfay}@inf.ufpel.edu.br

Abstract. This paper is going to present the development of a server and two plug-ins, one for Google Chrome and one for Mozilla Firefox, that will give the users of the Merlot repository recommendations based on the rating they give to the subjects they are studing using colaborative filtering. Using Apache manhout for colaborative filtering.

Keywords: recomendation system, colaborative filtering, plugin, Merlot

1 Introdução

Com o advento da internet e sua posterior popularização, além das inúmeras facilidades de navegação providas desde então, o conteúdo na web cresceu vertiginosamente. Devido este fato, o acesso a inúmeros produtos e serviços está disponível para os usuários. No entanto, os usuários demonstram dificuldade ao escolher entre tantas opções e, para predizer o que determinado usuário pode preferir, surgiram os sistemas de recomendação computacionais. O escopo deste artigo consiste em filtragem colaborativa, que é uma especialização do sistemas de recomendação convencional devido ao fato de envolver colaboração humana. Pode-se definir filtragem colaborativa, de forma geral, como um processo de filtragem em busca de informações ou padrões com utilização de técnicas que envolvam a colaboração de múltiplos agentes, pontos de vista, fontes de dados, etc [1].

A filtragem colaborativa é um método de realizar predições sobre os interesses de um usuário em particular através da coleta de preferências de um grupo finito de usuários – colaboradores – em um ambiente controlado. Partindo-se do pressuposto que se um usuário X compartilha da mesma opinião, em determinando assunto, que um usuário Y, podemos inferir que o usuário X é mais propenso a ter a opinião do usuário Y em um outro assunto qualquer do que ter a opinião de uma pessoa escolhida aleatoriamente. No repositório Merlot, especificado previamente pelo professor para a coleta dos dados, os usuários realizam a colaboração baseado na avaliação de itens.

A filtragem colaborativa não requer metadados dos itens, porém necessita de um histórico de avaliação dos mesmos para a recomendação. A hipótese base

do método de filtragem colaborativa consiste na questão de que se um usuário concordou no passado, é provável que ele concorde futuramente. Este artigo visa sobre a realização de recomendação geral e/ou específica para determinado usuário baseado em disciplinas da sua área, através de uma base de dados previamente obtida, na forma de um plugin para os navegadores Google Chrome e Mozilla Firefox. O recomendador foi concebido com o uso da linguagem Java e a biblioteca Apache Mahout para o mesmo, tendo a base de dados um pré-processamento necessário, a fim de fitrá-la por disciplinas, em linguagens PHP e consultas a banco de dados MySQL.

O artigo está disposto da seguinte forma: Na seção 2 é exposta a motivação da filtragem colaborativa, como ela é implementada, uma breve explanação da categoria utilizada neste artigo, além de algumas das dificuldades e desafios que a filtragem colaborativa enfrenta atualmente e como realizar a avaliação de um bom recomendador. Na seção 3 serão descritas as ferramentas que foram utilizadas e experimentos conduzidos, com alguns trechos de código a fim de um melhor entendimento por parte do leitor.

2 Filtragem Colaborativa

A motivação da filtragem colaborativa deriva do fato que as pessoas geralmente obterão as melhores recomendações de pessoas com gostos similares aos seus. Com base nesta motivação, são exploradas técnicas de combinação entre pessoas de gostos similares e feitas as recomendações baseadas na técnica utilizada. Geralmente, o fluxo de um sistema de filtragem colaborativa fica reduzido em:

1. Um usuário expressa sua preferência através da distribuição de um *rating* – classificação – em uma escala pré-determinada pelo desenvolvedor do ambiente. Esta classificação pode ser compreendida como uma representação aproximada do quanto o usuário gostou do item em questão.
2. O sistema de recomendação explora a base de dados em busca de pessoas com um gosto similar ao do usuário ativo – aquele que está sendo avaliado no momento, através da comparação de classificação para outros itens similares recomendados por ambos.
3. Uma vez encontrados os usuários com maior similaridade, o sistema de recomendação recomenda itens que estes usuários avaliaram mais positivamente mas ainda não foram avaliados pelo usuário ativo (presume-se que a falta de classificação acarreta em desconhecimento do item pelo usuário).

Existem duas categorias principais, sendo elas baseada em usuário (*user-based*) e baseada em item (*item-based*). No escopo deste trabalho, será abordado apenas a categoria baseada em usuário. Esta categoria consiste em dois passos:

– Busca por usuários que compartilhem os mesmos padrões de classificação do usuário ativo.
– Utilização destes padrões de classificação para predizer a classificação do usuário ativo para um item determinado.

Uma aplicação típica desta categoria é o algoritmo de k-vizinhos mais próximos – do inglês, *K-Nearest Neighbors* – disponível na biblioteca Apache Mahout, que foi utilizado nesse trabalho. Tal algoritmo faz uma busca na base de dados pelos k-vizinhos que melhor se assemelham ao usuário ativo e os retorna para a aplicação.

Para determinar o quão similar um usuário é com relação ao usuário ativo, existem diversas métricas, dentre elas a correlação de Pearson. Podemos definir a correlação de Pearson como sendo uma medida de correlação linear entre duas variáveis, retornando um valor no intervalo de [-1:1], onde o valor 1 seria a correlação positiva total – ambos os usuários apresentam os mesmos gostos em todos os itens – o valor 0 indica que não há nenhuma correlação entre estes usuários e, por fim, o valor -1 indica a correlação negativa total, ou seja, são usuários de gostos completamente opostos. Podemos descrever a correlação de Pearson pela equação 1.

$$\frac{\sum XallRat^2 * \sum YallRat^2}{\sqrt{Xrat * Yrat}} \tag{1}$$

onde $XallRat^2$ significa o quadrado de todas as classificações do usuário X, $YallRat^2$ significa o quadrado de todas as classificações do usuário Y e os termos $Xrat$ e $Yrat$ representam as classificações para todos os itens nos quais ambos os usuários avaliaram.

2.1 Dificuldades e desafios

1. Início em frio, o qual é aplicado tanto para usuários quanto para itens. Deriva do fato que se há um novo usuário no sistema e este ainda não realizou nenhuma classificação, é impossível prever seus gostos. Para itens, assim que um novo item é inserido no sistema, sem nenhum histórico de classificação, é complicado recomendá-lo para algum usuário.
2. Esparsidão da tabela usuário-item, visto que esta tabela é geralmente grande, no qual existem diversos itens sem qualificação alguma por usuários e usuários que não qualificam nada. Isto faz com os recomendadores tenham que se preocupar com performance.
3. Escalabilidade é outro fator importante, visto que, mesmo que a complexidade de um recomendador seja $O(n)$, este n ainda é bastante grande. Uma técnica que reduz este efeito e utilizada pelo Twitter é a clusterização de computadores, com grandes quantidades de memória [2].
4. *Shilling Attacks*, os quais baseiam-se no fato de um usuário de má índole, que tenta burlar o sistema de classificação, classificando muito positivamente seus itens e classificando muito negativamente os itens de seus competidores, por exemplo.
5. O usuário ovelha negra, aquele que apresenta inconsistência em suas classificações, não podendo ser classificado em qualquer grupo de usuários e assim não contribui de forma benéfica para a filtragem colaborativa.

6. A super-especialização também é outra dificuldade recorrente. Baseia-se no fato de que o usuário acaba gostando excessivamente de algum gênero de itens e apenas itens deste gênero são recomendados para o usuário, que entra num ciclo vicioso e, mesmo que classifique positivamente outro gênero de itens, o sistema de classificação vai demorar a incorporar esta mudança pelo simples fato de que este usuário possui um número tão elevado de classificações em um gênero que os outros gêneros passam a ser tratados quase como insignificantes.

2.2 Avaliação de Recomendadores

Porém, como a tarefa de recomendação é baseada na predição, é passível de erro. Portanto, o sistema não é perfeito. De alguma forma, devemos avaliar o quão acertada ou até mesmo incorreta é a predição do recomendador. Sem dúvidas, se pudéssemos contar com o usuário ativo avaliando o recomendador, descrevendo se o mesmo está correto ou incorreto sobre suas predições, teríamos o melhor avaliador possível para o algoritmo de recomendação utilizado. No entanto, este não é um caso real neste projeto e, portanto, precisamos obter uma estimativa de o quão preciso é o recomendador sem saber as preferências reais do usuário.

De acordo com [3], seus estudos revelam que os usuários, ao passar do tempo, vão avaliando de forma diferente o mesmo item, o que, de certa forma, vai de encontro a premissa da filtragem colaborativa, a qual assume que o usuário não avaliará diferentemente um item em instantes de tempo distintos. Com o intuito de aferirmos o recomendador, serão utilizadas duas medidas de erros, sendo elas o erro absoluto médio – do inglês, Mean Absolute Error - MAE, e a raiz do erro quadrático médio – do inglês, Root Mean Square Error - RMS.

O MAE pode ser definido como a média dos erros absolutos entre o valor estimado e o valor real de uma determinada gama de itens para o usuário ativo. A equação que descreve o MAE está explicitada na equação 2.

$$|MAE| = \frac{\sum_{i=1}^{N} |p_i - r_i|}{N} \tag{2}$$

Já a RMS pode ser definida como a raiz quadrada do quadrado das diferenças entre os valores estimados e valores reais de uma determinada gama de itens para o usuário ativo. A equação que descreve a RMSE está explicitada na equação 3.

$$|RMS| = \sqrt{\frac{\sum_{i=1}^{N} |p_i - r_i|^2}{N}} \tag{3}$$

Quanto mais próximo de 0 for o valor retornado, melhor, indicando que o erro é pequeno, significando que a preferência estimada difere muito pouco com relação a preferência real. Se o resultado for 0, sinal de que, para aquela situação, o recomendador foi perfeito. Na prática, não podemos definir um limiar para erro que seja aceitável, tudo dependerá do conjunto de dados e de cada usuário particularmente.

3 Material e métodos

Os experimentos foram conduzidos com base no repositório Merlot, o qual é um recurso computacional interdisciplinar para aprendizado e ensino online. Os itens explorados neste artigo serão os materiais e os membros, disponíveis para acesso. Para condução dos experimentos, foi-se utilizada a linguagem de programação Java com o incremento da biblioteca Apache Mahout, além de PHP e consultas MySQL.

3.1 Java e Apache Mahout

A biblioteca Apache Mahout disponibiliza diversos métodos de aprendizado de máquina, tais como classificação, clusterização e recomendação. Uma de suas vantagens é a escalabilidade que, em parte, soluciona uma das dificuldades elencadas na subseção 2.1, permitindo assim a execução de forma distribuida em uma rede. Na Apache Mahout, encontramos também as filtragens colaborativas tanto baseada em itens quanto baseada em usuários. A entrada de dados para o recomendador é feita através de um arquivo que respeite umas das seguintes configurações:

- <user_ID,item_ID>
- <user_ID,item_ID,*rating*>

Após efetuar a leitura dos dados, o próximo passo foi criar o recomendador. O recomendador padrão foi então sobrescrito para avaliar a similiridade entre usuários através da correlação de Pearson e a vizinhança estabelecida foi definida como 10. Logo, dado o modelo de dados, o algoritmo de similidade entre usuários e uma vizinhança pré-determinada, é retornado um recomendador genérico. Este trecho de código está explicitado no Código 1.1.

```
// Leitura do arquivo de ratings e geracao do modelo
FileDataModel model = new FileDataModel(new File(ratings));

// Construcao de um recomendador
RecommenderBuilder builder = new RecommenderBuilder() {
    @Override
    public Recommender buildRecommender(DataModel dm) throws
        TasteException {
        UserSimilarity sim = new PearsonCorrelationSimilarity(
            dm);
        UserNeighborhood neighborhood = new
            NearestNUserNeighborhood(10, sim, dm);
        return new GenericUserBasedRecommender(dm,
            neighborhood, sim);
    }
};
```

Código 1.1. Leitura da entrada e construção do recomendador

O trecho de código contigo no Código 1.2 visa sobre a parte de erros explicada anteriormente. Em ambos os testes, a percentagem do conjunto de treinamento foi de 0.8, cuja finalidade é utilizar este percentual das preferências de cada usuário para produzir as recomendações. O último atributo corresponde a percentagem da base de dados a ser considerada. Este *score*, que é o valor retornado como erro, é então aferido 50 vezes e obtido sua média, de forma a melhorarmos a precisão.

```
RecommenderEvaluator evaluatorMAE = new
    AverageAbsoluteDifferenceRecommenderEvaluator();
RecommenderEvaluator evaluatorRMS = new
    RMSRecommenderEvaluator();
double scoreMAE = 0, scoreRMS = 0;
int num = 50;
for(int i = 0; i < num; i++) {
    scoreMAE += evaluatorMAE.evaluate(builder, null, model,
        0.8, 1.0);
    scoreRMS += evaluatorRMS.evaluate(builder, null, model,
        0.8, 1.0);
}
scoreMAE /= num;
scoreRMS /= num;
```

Código 1.2. Avaliação do recomendador

O trecho de código que descreve a inicialização das variáveis e mensagens de log serão omitidas neste artigo, além de outras otimizações que serão omitidas na forma de código, como um booleano que determina se os cálculos de erros devem ou não ser executados, pois o intuito é calcular os erros apenas a primeira vez e, posteriormente, manter uma cache destes dados, haja visto que os erros são calculados para um conjunto de dados e não para cada recomendação.

Por fim, as recomendações são geradas para o usuário e disponibilizadas em uma lista no formato JSON para uso da aplicação, como demonstrado no Código 1.3.

```
//Gera as recomendacoes para o usuario
List<RecommendedItem> list = builder.buildRecommender(model).
    recommend(iduser, 10);
JSONArray objects = new JSONArray();
for(RecommendedItem rec : list){
    JSONObject object = new JSONObject();
    object.put("idobject", rec.getItemID());
    object.put("value", rec.getValue());
    objects.put(object);
}
result.put("objects", objects);
System.out.println(result);
```

Código 1.3. Saída do recomendador java

3.2 PHP, MySQL e Plugin

O plugin foi desenvolvido para rodar no Google Chrome e no Mozilla Firefox. Inicialmente, foi desenvolvido para o Google Chrome. Foi criado, então, o *manifest.json*, que basicamente é um arquivo de metadados que contém propriedades como o nome da extensão, descrição, versão e o que mais o autor achar pertinente, além de permissões e um javascript. Através do javascript, é realizado um *matching* e, toda vez que for aberta uma página de membro, serão inseridos 3 arquivos no site. O CSS, que é o responsável pela apresentação visual da página, o framework jQuery e o plugin de fato. As permissões são as mesmas da aba ativa, ou seja, as de membros, além de permitir acesso ao servidor local do recomendador, que foi colocado no *loopback*/*localhost*, porta 8080 – 127.0.0.1:8080/*.

Após o *matching*, carregam-se os arquivos da página e o javascript extrai o ID do usuário da URL. Então, são realizadas duas requisições HTTP para o servidor, através de Ajax, que já possui a base de dados instanciada. Há, também, no servidor, uma tabela que salva a cache de erros por disciplina. Nas requisições são informados o tipo de recomendação, se será geral ou específica, e a ID do usuário. Por fim, o servidor retorna, para cada tipo de recomendação, um erro ou dados de uma recomendação. Tais dados são os erros MAE e RMS, e uma lista de recomendações, contendo ID do objeto, título, descrição, link e o seu valor dado pelo recomendador. De posse destes dados, o CSS se encarrega de montar visualmente duas colunas de recomendação, uma para recomendação geral e outra para recomendação específica.

De modo simplificado, o servidor está esperando requisições. Quando o servidor receber uma requisição pedindo as recomendações gerais disponíveis para o usuário e para a área que o mesmo escolheu, será feita uma consulta na base de dados para saber se o usuário existe. Caso ele exista, será feita uma consulta para saber se este usuário já avaliou algum material. Dependendo do tipo de recomendação, haverá ou não um fitro baseado na disciplina. Se este usuário já fez classificação de algum material, podemos predizer do que ele pode gostar. Então, é gerado a partir de uma nova consulta, um arquivo contendo a classificação dos objetos, dependendo do tipo de recomendação. O código MySQL que gera o arquivo de classificação no caso de uma recomendação específica é explicitado abaixo, no Código 1.4.

```
1 SELECT DISTINCT
2     users_ID_User, ID_LO, Rating
3 FROM
4     lovaloration INNER JOIN loval_com ON ID_Rating =
          ID_Valoration
5     NATURAL JOIN locomments
6     NATURAL JOIN locat_dat
7 WHERE
8     ((users_ID_User != 0) AND (Rating != 0) AND (ID_Cat = {
          $ID_CATEGORY}))
9 ORDER BY (users_ID_User)
```

Código 1.4. Geração de arquivo de classificação

Este arquivo está no formato <user_ID,item_ID,*rating*>. Como a geração destes arquivos também é custosa computacionalmente, é feito uma cache. Além disto, tanto a cache de cálculo de erros como a cache do arquivo de classificação podem ser desativadas pela constante CACHE = false. Havendo arquivo de classificação, seja ele geral ou específico, é invocado o recomendador java através do comando exec() do PHP. A saída do recomendador java é uma lista JSON com as ID dos objetos e o valor de suas recomendações, respectivamente, como pode ser visto no Código 1.3. De posse desta lista, a saída da aplicação é montada fazendo uma consulta no banco de dados por informações de título, descrição e link de cada objeto. Para invocar a aplicação utilizando PHP, o Java deve estar no path de sua distribuição.

No Google Chrome, o plugin é gerado pelo próprio navegador e, para disponibilizar na Chrome Web Store é necessário uma conta de desenvolvedor, o que não possuímos. No Mozilla Firefox, o mesmo procedimento é realizado, porém o *manifest.json* é renomeado para *package.json* e se fez necessário também o Add-On SDK, disponibilizado pela própria Mozilla, para permitir gerar e testar o arquivo de distribuição com extensão .xpi. Para disponibilização, a loja de complementos do Mozilla Firefox é mais simples que a do Google Chrome, visto que não há exigência de conta de desenvolvedor e nem taxas a serem pagas.

4 Funcionamento

Inicialmente, gostaríamos de informar que os plugins desenvolvidos no decorrer desse trabalho não estão disponíveis para o público no presente momento. Entretanto, se estivessem disponíveis, poderiam ser instalados a partir de suas respectivas lojas.

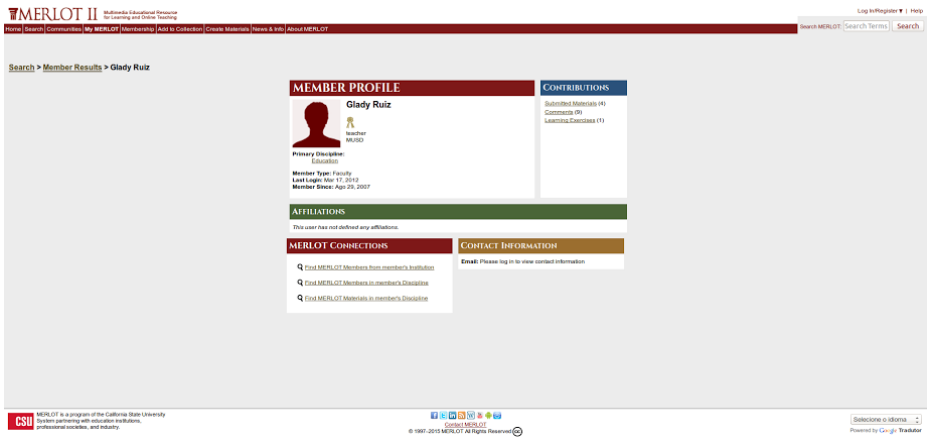

Figura 1. O perfil de um usuário no repositório Merlot

Na Figura 1 podemos ver o perfil de um usuário sem a utilização de nenhuma ferramenta externa além do que nos é provido pelo repositório Merlot, mostrando apenas as atividades que o usuário tomou parte e as informações de perfil que foram preenchidas durante o seu cadastro.

Após a instalação e habilitação de um dos plugins desenvolvidos nesse trabalho, não havendo necessidade de nenhuma outra interação do usuário com o mesmo, o usuário obtém acesso as informações geradas pela filtragem colaborativa, conforme pode ser visto na Figura 2, onde além de recomendarmos assuntos que podem ser de interesse do usuário, informamos o quão interessante pode ser, utilizando uma classificação de 1 a 5 estrelas, assim como taxas de erros geradas pelo recomendador.

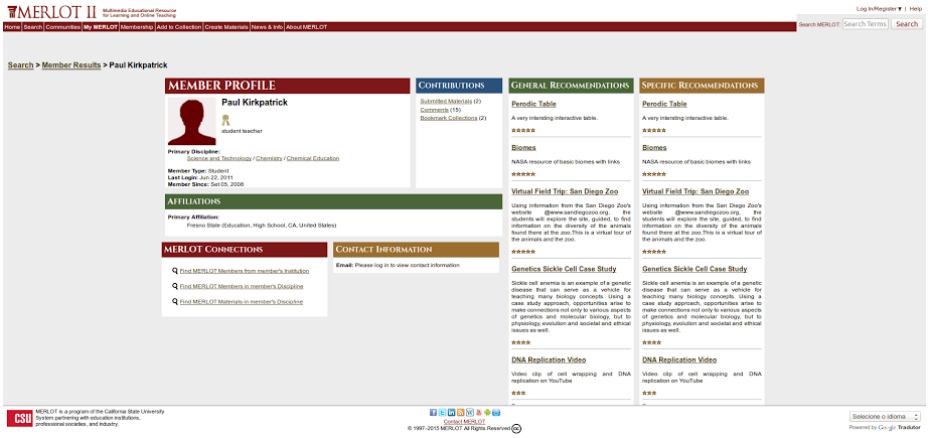

Figura 2. O perfil de um usuário no repositório Merlot, utilizando o plugin desenvolvido

Como também pode ser visto na Figura 2, o plugin adiciona esta nova informação de forma a não quebrar a formatação do site, para dar uma experiência mais transparente para o usuário.

5 Participação dos Membros do Grupo

As responsabilidades do desenvolvimento desse trabalho foram divididas entre os membros do grupo. A parte mais extensa desse trabalho foi a pesquisa, o que ocasionou que todos buscassem informações sobre quais seriam as plataformas adequadas e quais seriam as ferramentas necessárias para o desenvolvimento, tanto do servidor quanto dos plugins.

Todas as informações obtidas pelas pesquisas de cada usuário foram discutidas arduamente pelos envolvidos até reduzirmos a duas plataformas de desenvolvimento. Após as decisões quanto ao rumo do desenvolvimento dos plugins

terem sido tomadas, foram desenvolvidos no tempo livre dos integrantes os plugins e o servidor. Este artigo também foi escrito colaborativamente entre os membros do grupo.

Cabe ressaltar, também, que alguns trechos de implementação foram desenvolvidos individualmente em determinado momento por um ou outro membro do grupo, no entanto toda e qualquer modificação foi trazida a atenção dos demais integrantes, gerando uma discussão sadia com o intuito de termos uma implementação robusta.

6 Conclusões

No decorrer desse trabalho, foram utilizados conhecimentos de mineração de dados e de programação para desenvolver dois plugins, um para o Mozilla Firefox e outro para o Google Chrome, cujo finalidade foi oferecer um sistema de filtragem colaborativa para usuários do repositório Merlot.

O objetivo desse trabalho foi desenvolver os conhecimentos aprendidos sobre o sistema de filtragem colaborativa para, durante a disciplina de *Learning Analytics* (Analíticas de Aprendizagem), ministrada para o curso de Ciência da Computação da Universidade Federal de Pelotas, averiguar qual seria a dificuldade de implantar esse sistema em um ambiente real e efetuar sua implantação na forma de plugin que, posteriormente, poderá estar disponível para a comunidade.

Durante o desenvolvimento dos plugins, tornou-se claro que a filtragem colaborativa necessita de um grande número de usuários ativos para responder de forma adequada para cada usuário pertencente ao repositório, o que não se apresentou como um obstáculo neste trabalho, devido ao fato de possuírmos uma base de dados relativamente extensa.

A única dificuldade real que foi encontrada derivou-se do fato de não possuírmos pleno domínio das ferramentas utilizadas. Por consequente, tornou-se necessário ir em busca de conhecimento sobre o funcionamento da biblioteca Apache Mahout, através da leitura de sua API nos métodos que foram utilizados e, também, aprender como desenvolver plugins para o Google Chrome e para o Mozilla Firefox, sendo esta uma etapa anteriormente desconhecida pelos integrantes e que, ao final do trabalho, foi bastante gratificante. Tal aprendizado ocupou a maior parte do tempo de desenvolvimento desse projeto. Por fim, a implementação foi mais uma questão de aplicação do que foi aprendido, do que do desenvolvimento de algo novo.

Sendo assim, os integrantes perguntam-se qual seria a razão da filtragem colaborativa não estar integrada em diversas plataformas existentes hoje, pois comprovamos que é possivel implantá-la externamente sem muito custo computacional para os níveis de tais organizações, principalmente com a abrangência que possuem.

Finalmente, concluimos que é possivel, além da relativa simplicidade, o desenvolvimento e implantação de um plugin externo que funciona com a mesma

precisão e pode ser apresentado de forma a parecer para o usuário final como se fosse parte integrante do repositório Merlot.

7 Agradecimentos

Os integrantes do grupo gostariam de agradecer imensamente o colega Henrique Lemos dos Santos pela sua disponibilidade e auxílio antes e durante o desenvolvimento deste projeto. Mesmo sem fazer parte do grupo, Henrique contribuiu de forma positiva e tornou possível que este projeto saísse da cabeça dos integrantes para o mundo real.

Referências

1. Terveen, L., Hill, W.: Beyond Recommender Systems: Helping People Help Each Other. Addison-Wesley, 1–21 (2001)
2. Gupta et al.,: WTF: The who-to-follow system at Twitter. WWW '13 Proceedings of the 22nd international conference on World Wide Web, 505–514 (2013)
3. Hill, W. et al.,: Recommending and evaluating choices in a virtual community of use. Proceedings of the SIGCHI conference on Human factors in computing systems, page 194–201 (1995)