UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO MULTIDISCIPLINAR

JONES MARQUES AUGUSTO

Recsys.jl: Um framework para geração e avaliação de listas de recomendação

Prof. Filipe Braida do Carmo, M.Sc.
Orientador

Nova Iguaçu, Fevereiro de 2016

Recsys.jl: Um framework para geração e avaliação de listas de recomendação

Jones Marques Augusto

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Multidisciplinar da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

_____

Jones Marques Augusto

Aprovado por:

_____

Prof. Filipe Braida do Carmo, M.Sc.

_____

Prof. Carlos Eduardo Ribeiro de Mello, Ph.D.

_____

Prof. Leandro Guimaraes Marques Alvim, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Fevereiro de 2016

Agradecimentos

Agradeço em primeiro lugar a minha família, que sempre me apoiou e orientou a buscar algo melhor. Ensinando a sempre dar prioridades aos estudos e todas essas coisas que quando somos mais jovens não prestamos tanto atenção, mas que agora prestes e me formar vejo que valeram muito a pena.

Preciso dar um destaque especial a minha namorada Elaine Tady, por estar comigo desde o 2º período sempre me ajudando a resolver todos os problemas existentes, seja causado pela faculdade ou pela vida. Para mim foi ótimo saber que sempre tive alguém com quem contar.

Agradeço é claro também a todos os amigos que conheci nessa caminhada chamada graduação, grupos de estudos, grupos de trabalhos e etc. Sem eles o caminho seria muito mais difícil. Por falar em amigos, tenho que citar o grupo auto-intitulado Pilastra, muitas histórias e risadas que tornaram essa vida de estudante muito mais fácil, a eles só tenho a agradecer.

Agradeço ao meu orientador/professor Filipe Braida, em primeiro lugar por ter me aceitado como seu orientando e em segundo lugar por ter dado todo suporte que necessitei ao produzir esse trabalho. Também tenho que destacar os excelentes professores do curso de Ciência da Computação do IM-UFRRJ, sempre dispostos a ajudar, sem nenhum tipo de prepotência para lidar com "meros alunos", as aulas muitas vezes iam além de simples conteúdo no quadro.

RESUMO

Recsys.jl: Um framework para geração e avaliação de listas de recomendação

Jones Marques Augusto

Fevereiro/2016

Orientador: Filipe Braida do Carmo, M.Sc.

O acelerado crescimento e a grande variedade de informação disponível na *Web* somado à rápida introdução de novos serviços de *e-business* frequentemente deixa os usuários em uma situação desconfortável ao tomar decisões. A multiplicidade de escolhas em vez de produzir benefícios, diminui o seu bem-estar. Nesse contexto surgiram os sistemas de recomendação, responsáveis por filtrar a maior parte das informações e sugerir através de recomendações personalizadas, utilizando diversas técnicas para isso.

Um dos grandes problemas para quem começa a estudar a área de sistemas de recomendação é a necessidade de reimplementar os algoritmos clássicos da literatura para usá-los em suas pesquisas, esse fato acaba tomando muito tempo do pesquisador. Utilizar *frameworks* com esses algoritmos já implementados normalmente é a saída mais fácil.

Este trabalho tem como objetivo implementar um *framework* que seja capaz de receber e testar diferentes algoritmos de recomendação, assim como prover ao usuário algumas técnicas previamente implementadas. Este *framework* deve ser de fácil utilização e eficiente, pois será na grande maioria das vezes utilizado para fins educacionais.

ABSTRACT

Recsys.jl: Um framework para geração e avaliação de listas de recomendação

Jones Marques Augusto

Fevereiro/2016

Advisor: Filipe Braida do Carmo, M.Sc.

The rapid growth and the wide variety of information available in the web coupled with the rapid introduction of new e-business services often left users in an awkward situation when making decisions. The multiplicity of choices, instead of producing benefits diminish their well-being. In this context came the recommendation systems, responsible for filtering most of the information and suggest just what the user wants, using various techniques to do so.

A major problem for those who begins to study the recommender systems area is the need to redeploy the literary classics algorithms to use them in their research, this fact end up taking a long time researcher. Use frameworks with those algorithms already implemented is usually the easy way out.

This work aims to implement a framework that is able to receive and test different recommendation algorithms, as well as provide the user with some previously implemented techniques. This framework should be user-friendly and efficient as it is in most cases used for educational purposes.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

1.1 Motivação

Um sério problema acontece dentro da *Wold Wide Web* (WWW), o número incrivelmente grande de informação contido em milhões de páginas. Em janeiro de 2016, o número de páginas já chegava em 47 bilhões (WORLDWIDEWEBSIZE, 2016). Com essa quantidade de informação acaba sendo impossível filtrar e absorver um conteúdo que seja realmente útil.

Nesse contexto, na década de 90 (no auge da explosão da *internet*) surge a área de sistemas de recomendação com intuito de auxiliar seus usuários a lidar com a enorme quantidade de dados a que são expostos através recomendações personalizadas de conteúdos ou serviços. Diversas empresas tem seus principais diferencias competitivos nesse tipo de tecnologia, destacam-se a *Amazon*[1] com livros e *Netflix*[2] com filmes.

A recomendação é comum a qualquer pessoa e pode ser observada no dia a dia. Ao entrarmos em uma loja de roupas por exemplo, assim que um dos vendedores nota o seu interesse por alguma das peças, este já lhe oferece algumas outras que sejam parecidas ou se adequem ao seu gosto baseado na sua primeira escolha. Com

[1] `http://www.amazon.com/`
[2] `http://www.netflix.com/`

isso uma grande parte de produtos pode ser filtrada, sem a necessidade de que se olhe todo o resto da loja.

Assim também se comportam os sistemas de recomendação. Com base nos gostos de seus usuários, ou seja, nas preferências que os usuários tenham sobre alguns itens específicos o sistema de recomendação pode predizer quais os itens que o usuário ainda não tenha consumido, mas que provavelmente gostaria de consumir. Na literatura podemos encontrar diversas abordagens para essa recomendação, por exemplo os sistemas de recomendação baseados em filtragem colaborativa (Goldberg et al., 1992).

Os sistemas de recomendação baseados em filtragem colaborativa são de fácil entendimento por terem uma ligação direta com o que acontece no dia a dia. Suas recomendações são feitas utilizando as preferências explícitas dos usuários sobre os itens, identificando itens avaliados em comum entre os usuários, ou usuários comuns entre itens. Porém existem limitações nessa abordagem, pois normalmente a quantidade de notas dadas pelos usuários são poucas se comparadas a quantidade de usuários e itens dentro do sistema (Adomavicius and Tuzhilin, 2005).

Atualmente muitos *frameworks* tem se proposto a auxiliar usuários iniciantes na área de sistemas de recomendação, trazendo ferramentas ou mesmo algoritmos clássicos já implementados. Esses *frameworks* devem seguir algumas características específicas, como ser desenvolvido em uma linguagem de alto nível para que haja uma maior abstração das técnicas implementadas, assim como um melhor entediamento do código em si.

Porém ao fazer isso, não se pode perder performance. Muitas linguagens de programação ao criarem abstrações para facilitar seus usuários acabam sacrificando a performance de seus programas. Ao passo que sistemas de recomendação geralmente trata problemas que em sua grande maioria necessita de respostas em um curto espaço de tempo, a rapidez na execução também deve ser uma prioridade.

Este trabalho tem como objetivo propor um *framework* seguindo os preceitos citados anteriormente. Voltado para um contexto educacional, de modo a fazer um

usuário inciante avançar nos estudos da área de sistemas de recomendação.

1.2 Organização

Essa monografia está organizada da seguinte maneira:

Capítulo 2 traz uma visão geral da fundamentação teórica utilizada no trabalho. Abordando temas como sistemas de recomendação e suas classificações;

Capítulo 3 apresenta o *framework* proposto, sua arquitetura, ferramentas e algoritmos implementados.

Capítulo 4 mostra uma a metodologia utilizada para realização dos experimentos e seus resultados, de modo a validar sua eficiência e corretude.

Por fim, no capítulo 5 é apresentado a conclusão do trabalho seguido pelas referências bibliográficas deste trabalho.

Capítulo 2

Fundamentação Teórica

2.1 Sistemas de Recomendação

É facilmente observável a dificuldade encontrada pelas pessoas em tomar decisões, seja na compra de um produto ou em uma simples escolha de um filme para assistir. Esse problema foi agravado com a expansão da *Internet* e o inicio do comércio eletrônico, visto que as possibilidades de escolhas agora são praticamente infinitas.

O acelerado crescimento e a grande variedade de informação disponível na *Web* somado à rápida introdução de novos serviços de *e-business* frequentemente deixava os usuários em uma situação desconfortável ao tomar decisões. A multiplicidade de escolhas, em vez de produzir benefícios, diminuía o seu bem-estar. Na verdade a escolha, com sua implicação de liberdade, autonomia, e autodeterminação, pode se tornar excessiva, criando um senso de liberdade opressiva que pode levar ao descontentamento do usuário. (Ricci et al., 2011)

O ato de recomendar itens é algo comum ao ser humano e é uma das maneiras de solucionar o problema de escolha acima descrito, pessoas que precisam adquirir algo novo normalmente recorrem a alguém para o ajudar com sugestões, por exemplo, na compra de um livro. Sistemas de Recomendação são técnicas e ferramentas de *software* que oferecem sugestões de itens úteis a um usuário. As sugestões se referem

a diversos processos de decisão, como a compra de itens, a leitura de notícias *online* ou mesmo a visualização de vídeos. (Ricci et al., 2011)

Essa lógica é aplicada pelos *sites* que oferecem algum tipo de serviço para seus usuários, por exemplo o *Youtube*[1], famoso *site* de *streaming* de vídeos, o *site* tem em sua posse todo o histórico de movimentação dos seus usuários, e este pode sugerir um vídeo de comédia para um usuário que na grande maioria das vezes acessou a página para assistir vídeos de comédia ou semelhantes.

O primeiro sistema de recomendação proposto foi o *Tapestry* (Goldberg et al., 1992), que utilizava algoritmos de filtragem colaborativa (*collaborative filtering*), ou seja, suas sugestões baseiam-se nas similaridades de um usuário ativo para com os outros usuários. O sistema sugere apenas itens que os usuários mais similares tenham consumido. Tinha como objetivo a seleção de *e-mails*, através de regras (filtros) definidas pelo usuário.

Embora o estudo de sistemas de recomendação seja relativamente novo em comparação com pesquisas sobre outras ferramentas e técnicas de sistemas de informação clássicas, como banco de dados ou motores de buscas, e tenha emergido como uma área independente de pesquisa em meados dos anos 1990 (Ricci et al., 2011). Recentemente o interesse por sistemas de recomendação aumentou drasticamente e seu uso pode ser notado em diversos sistemas *online*, como *Amazon*[2], *Last.fm*[3], *Netflix*[4].

(Adomavicius and Tuzhilin, 2005) descreveu o problema de recomendação como: Dado os conjuntos U de todos os usuários e I de todos os possíveis itens a serem recomendados, como exemplo livros, músicas ou vídeos. Dado r a função de utilidade que quantifica a importância de um item i para o usuário u, i.e, $r : U \times I \rightarrow R$, onde $R \in \mathbb{N}$ ou $R \in \mathbb{R}$, é o conjunto de preferência do usuário pelo item. Então, para cada $u \in U$, é escolhido o item $i' \in I$ que maximiza a função utilidade r.

$$\forall u \in U,\ i'_u = \arg\max_{i \in I} r(u, i).$$

[1] http://www.youtube.com/
[2] http://www.amazon.com/
[3] http://www.lastfm.com/
[4] http://www.netflix.com/

O problema central dos sistemas de recomendação é o fato da função de utilidade r geralmente não ser definida em todo espaço $U \times I$, mas só em um subconjunto deste. Em sistemas de recomendação, a função utilidade é representada pelas avaliações de itens por usuário e é inicialmente definida apenas sobre os itens anteriormente avaliados pelos usuários.

A tabela 2.1 é um exemplo de uma matriz usuário-item para recomendação de filmes cuja função utilidade é definida pelas notas dadas pelos usuários aos filmes assistidos por eles, que assumem valores de 1 a 5. O símbolo $\emptyset$ significa que a relação usuário-item ainda não existe, ou seja, o usuário ainda não deu uma nota para o item. Portanto, o sistema deve ser capaz de estimar essas notas de forma a oferecer boas recomendações baseadas nas predições.

	Titanic	Poderoso Chefão	Matrix
Elaine	4	$\emptyset$	3
Gabriel	4	5	5
Lucas	$\emptyset$	5	$\emptyset$

Tabela 2.1: Fragmento de uma matriz de notas de um sistema de recomendação de filmes.

Existem duas abordagens possíveis para estimar a preferência de um usuário por um item que não está explicitado. A primeira é utilizando heurísticas para definir a função utilitária f. O segundo método é a utilização de uma função que minimiza algum critério de desempenho, como exemplo a raiz quadrada da média do erro (RMSE). (Adomavicius and Tuzhilin, 2005)

Uma vez que todas as relações usuário-item desconhecidas são estimadas utilizando alguma das duas abordagens citadas acima, uma forma de fazer a recomendação de um item para um usuário pode ser feita selecionando o item com o valor estimado de maior significado, por exemplo a maior nota, para ser recomendado para o usuário. Podendo também ser recomendado os N melhores itens para o usuário, esta técnica é denominada *TopN*. (Adomavicius and Tuzhilin, 2005)

Ricci et al. (2011) apresenta uma taxonomia de seis diferentes classes de abor-

dagens de recomendação:

- Baseada em Conteúdo: O sistema recomenda itens que são semelhantes aos que o usuário gostou no passado;
- Filtragem Colaborativa: O sistema recomenda a partir dos itens consumidos pelos outros usuários similares, ou seja, com perfis de preferências parecidos com o usuário;
- Demográfico: Este sistema recomenda itens com base no perfil demográfico do usuário;
- Baseada no conhecimento: O sistema recomenda itens com base no conhecimento de domínio sobre como o item satisfaz as preferências do usuário, ou seja, como o item é útil para o usuário;
- Baseada na comunidade: Este sistema recomenda itens com base nas preferências dos "amigos" do usuário;
- Abordagens híbridas: Estes sistemas combinam as abordagens mencionadas acima.

2.2 Abordagem de Recomendação

Nesta seção será explicada, em maior nível de detalhamento a abordagem mais utilizada de sistemas de recomendação (Filtragem Colaborativa), assim como seus algoritmos.

2.2.1 Filtragem Colaborativa (*Collaborative Filtering*)

A filtragem colaborativa utiliza as informações e preferências de pessoas parecidas para fazer suas recomendações. É a abordagem mais utilizada e se assemelha bastante ao que as pessoas fazem normalmente, buscar pessoas com gostos parecidos com o seu para lhe recomendar algo, ou mesmo quando um item se torna popular por troca de informações entre pessoas, o famoso "boca a boca".

A ideia central é saber o quão similares são seus usuários, para que se possa recomendar baseado nas escolhas dos usuários que estejam em um dado perfil específico. Mais formalmente falando, podemos encontrar um subconjunto V de usuários similares que tenham explicitado uma nota em um dado item i. Supondo um usuário u que pertença a esse perfil de usuários, podemos afirmar que a nota dele sobre o item i será dada pelas notas dadas pelo grupo V para o mesmo item i.

A filtragem colaborativa tem uma grande vantagem, ela diminui a chance de ocorrer somente recomendações de um mesmo estilo de item, por exemplo apenas um estilo de filme. Isso faz com que seus usuários recebam itens diversificados, aumentando assim as chances de uma predição interessante para o usuário, ou seja, aumentando a *serendipity*. (Ge et al., 2010)

Os sistemas de recomendação geralmente utilizam grandes bases de itens, tornando assim a matriz usuário-item muito esparsa, diminuindo a qualidade e o desempenho da recomendação, visto que o desempenho e a qualidade esta diretamente relacionada com a esparsidade da matriz. (Linden et al., 2003)

Com a esparsidade dos dados também aparece o problema chamado *Cold-starter*, que é a entrada de novos usuários ou itens no sistema, tornando difícil encontrar um conjunto de usuários similares, impossibilitando a recomendação de algum item para esse usuário. (Schein et al., 2002)

Uma abordagem muito utilizada para contornar os problemas descritos acima é utilizar alguma técnica de redução de dimensionalidade como a decomposição de valores singulares (SVD) para lidar com a esparsidade e com isso fazer boas recomendações. (Ricci et al., 2011)

De acordo com Adomavicius and Tuzhilin (2005) os algoritmos usados em filtragem colaborativa podem ser agrupados em duas classes gerais: Baseado em memória (*Memory-based*) e baseado em modelo (*Model-based*).

2.2.1.1 Baseado em memória

São heurísticas que fazem previsões de nota com base em toda a coleção de itens previamente avaliado pelos usuários. Para expressar o valor desconhecido de uma classificação $r_{c,s}$ para um usuário c e um item s é normalmente computado o agregado das notas dos outros usuários (geralmente os N mais similares) para o mesmo item s. (Adomavicius and Tuzhilin, 2005)

$$r_{c,s} = \operatorname*{agr}_{c \in \hat{C}} r_{c',s}$$

Onde $\hat{C}$ representa o conjunto dos N usuários mais similares ao usuário c e que tenham explicitado uma nota para o item s. Uma maneira simples de fazer a agregação das notas pode ser vista a seguir:

$$r_{c,s} = \frac{1}{N} \sum_{c' \in \hat{C}} r_{c',s}$$

Porém, a abordagem mais comumente utilizada é a da soma ponderada:

$$r_{c,s} = k \sum_{c' \in \hat{C}} sim(c, c') \times r_{c',s}$$

Onde k é geralmente dado por $k = 1 / \sum_{c' \in \hat{C}} |sim(c, c')|$ servindo como um fator de normalização. Diferentes aplicações podem utilizar suas próprias medidas de similaridade, desde que seus cálculos sejam normalizados usando o fator k.

A similaridade entre os usuário c e c', $sim(c, c')$, é basicamente uma medida de distâncias e é usada como uma ponderação, isto é, quanto mais similares c e c' são, maior é o peso que a $r_{c',s}$ terá na predição de $r_{c,s}$. Note que $sim(x, y)$ é um artefato heurístico, introduzido de modo a diferenciar os níveis de similaridades dos usuários (isto é, poder achar um conjunto de pares ou uma vizinhança mais próxima para cada usuário) e também para simplificar o processo de estimar uma avaliação.

Um problema surge quando é usado a soma ponderada, pois não é levado em

conta o fato de diferentes usuários poderem utilizar a escala de avaliações de forma diferente. Uma forma de solucionar esse problema é também fazer a normalização das notas. Neste caso, ao invés de utilizar o valor absoluto das avaliações, a soma ponderada é calculada utilizando os desvios em relação a média das notas do usuário correspondente. (Adomavicius and Tuzhilin, 2005)

$$r_{c,s} \;=\; \bar{r}_c \;+\; k \sum_{c' \in \hat{C}} sim(c,\, c') \;\times\; (r_{c',s} \;-\; \bar{r}_{c'})$$

Onde $\bar{r}_c$ é a média das avaliações do usuário c, dada pela equação:

$$\bar{r}_c \;=\; \frac{1}{|S_c|} \sum_{s \in S_c} r_{c,s},\; S_c \;=\; \{s \in S \mid r_{c,s} \;\neq\; \emptyset\}$$

Computar as similaridades dos usuários ou dos itens é um passo crítico dos algoritmos de filtragem colaborativa baseados em memória. Em uma abordagem baseada em itens, a ideia básica para calcular a similaridade entre o item i e o item j é trabalhar apenas com os usuários que tenham avaliado ambos os itens, para depois determinar a similaridade entre os dois itens. Em uma abordagem baseada em usuário, se calcula a similaridade entre os usuários u e v que tenham avaliado os mesmos itens. (Su and Khoshgoftaar, 2009)

O cálculo das similaridades entre os usuários em um sistema de recomendação colaborativo pode ser feito de diversas formas. As duas abordagens mais populares são as abordagens por correlação e a baseada em cosseno. Para apresentá-las, temos S_{xy} como o conjunto de todos os itens que ambos os usuários x e y explicitaram uma nota, isto é, $S_{yx} = \{s \in S \mid r_{x,\,s} \neq \emptyset \quad \& \quad r_{y,\,s} \neq \emptyset\}$. (Adomavicius and Tuzhilin, 2005)

A correlação de Pearson é uma abordagem para o cálculo de similaridades baseada em correlação:

$$sim(x,\, y) \;=\; \frac{\sum_{s \in S_{xy}} (r_{xs} \;-\; \bar{r}_x)\,(r_{ys} \;-\; \bar{r}_y)}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} \;-\; \bar{r}_x)^2 \sum_{s \in S_{xy}} (r_{ys} \;-\; \bar{r}_y)^2}}$$

Na abordagem baseada em cossenos, os dois usuários x e y são tratados como dois vetores m-dimensional, onde $m = |S_{xy}|$. Então a similaridade entre os usuários x e y pode ser calculada pelo cosseno do ângulo entre os dois vetores:

$$sim(x, y) = cos(\vec{x}, \vec{y}) = \frac{\vec{x} \bullet \vec{y}}{\|\vec{x}\|_2 \times \|\vec{y}\|_2} = \frac{\sum_{s \in r_{xy}} r_{x,s}\, r_{y,s}}{\sqrt{\sum_{s \in S_{xy}} r_{x,s}^2}\sqrt{\sum_{s \in S_{xy}} r_{y,s}^2}}$$

Onde $\vec{x} \bullet \vec{y}$ denota o produto interno entre os vetores $\vec{x}$ e $\vec{y}$.

Também pode ser usado um método de cálculo de similaridades baseado em probabilidade condicional. A similaridade entre um item x e um y vai ser dada pela probabilidade do usuário consumir o item x dado que já consumiu o item y, a expressão pode ser vista a seguir:

$$sim(x, y) = \frac{Freq(x\, y)}{Freq(y)}$$

Note que diferentes sistemas de recomendação podem usar diferentes abordagens para calcular as similaridades e estimar as notas dos usuários da forma mais eficiente possível. Uma estratégia comum é calcular todas as similaridades $sim(x, y)$ antecipadamente, não precisando recalcular-las enquanto não há uma mudança drástica na rede de pares. Então, quando o usuário solicita por uma recomendação, a nota pode ser eficientemente calculada sob demanda usando as similaridades pré-computadas. (Adomavicius and Tuzhilin, 2005)

2.2.1.2 Baseado em modelo

Algoritmos baseados em modelo usam as coleções das notas para aprender um modelo, e este é usado para prever as notas. A ideia geral é encontrar um modelo baseado nas interações usuário-item e assim representar as características latentes desses usuários e itens no sistema, como a classe de preferências do usuário ou a categoria de um item. (Ricci et al., 2011)

Um exemplo de abordagem probabilística para Filtragem Colaborativa proposto

por Breese et al. (1998), para calcular avaliações ainda desconhecidas:

$$r_{c,\,s} \;=\; E(r_{c,\,s}) \;=\; \sum_{i\,=\,0}^{n} i \times \Pr(r_{c,\,s} \;=\; i \mid r_{c,\,s'}, s' \;\in\; S_c)$$

Presume-se que os valores das avaliações são números inteiros entre 0 e n e a expressão Pr é a probabilidade do usuário c dar uma nota em particular ao item s, dada as avaliações do usuário sobre os itens anteriormente classificados. Breese et al. (1998) propõe duas alternativas de modelos probabilísticos para o cálculo da probabilidade: modelo por agrupamento (*clusters models*) e rede Bayesiana (*Bayesian networks*).

No primeiro modelo, os usuários de preferências parecidas são agrupados em classes C, com o número de classes e os parâmetros do modelo sendo aprendidos a partir dos dados. Assume-se que as avaliações dos usuários são condicionalmente independentes, dado que a estrutura do modelo é a de um modelo Bayesiano ingênuo (*Naive Bayes*).

$$\Pr(C \;=\; c,\, v_1, \ldots, v_n) \;=\; \Pr(C \;=\; c) \prod_{i\,=\,1}^{n} \Pr(v_i \,|\, C \;=\; c)$$

Esse modelo é conhecido como modelo de mistura multinomial e pode ser utilizado para calcular as probabilidades da expressão anteriormente apresentada. Onde $\Pr(C \;=\; c)$ são as probabilidades de um usuário pertencer a classe C e $\Pr(v_i \,|\, C \;=\; c)$ são as probabilidades condicionais de notas dada a classe C.

O segundo modelo representa cada item do domínio como um nó da rede Bayesiana, onde o estado de cada nó corresponde aos possíveis valores de avaliação de cada item, os itens que ainda não foram avaliados recebem o indicador "*no vote*".

É aplicado a seguir um algoritmo para treinar o modelo em cima dos dados, este algoritmo procura as dependências de cada item para montar uma rede. Ao final cada item terá um conjunto de itens pais e estes terão os melhores valores para uma predição de nota.

Cada conjunto de probabilidades condicionais é representada como uma árvore

de decisão que codifica as probabilidades condicionais para cada nó. (Breese et al., 1998)

A Figura 2.1 é um exemplo de uma árvore de decisão gerada para identificar a probabilidade de um indivíduo assistir ou não ao seriado "*Melrose Place*", com os pais "*Friend's*" e "*Beverly Hills, 90210*". Os gráficos em baixo da árvore indicam a probabilidade de um usuário ter assistido ou não um filme especifico, condicionado a ter assistido os seriados-pais.

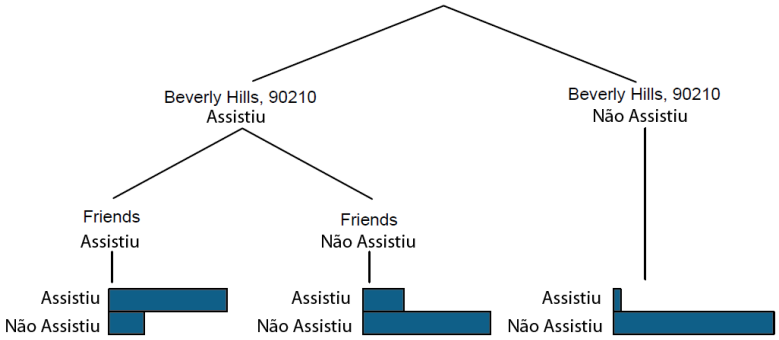

Figura 2.1: Árvore de decisão para audiência de séries (Breese et al., 1998).

2.3 Algoritmos

Será apresentado a seguir alguns exemplos de algoritmos da literatura sobre os temas anteriormente abordados. Para um melhor entendimento segue uma explicação das variáveis utilizadas:

- U - conjunto de todos os usuários.
- I - conjunto de todos os itens.
- $\vec{u}$ - vetor que representa os itens avaliados pelo usuário u.
- $\vec{i}$ - vetor que representa os usuários que avaliaram o item i.
- $top[u]$ - conjunto dos K usuários mais similares ao usuário u.

- $top[i]$ - conjunto dos K itens mais similares ao item i.

Algoritmo 1 básico para conhecer e armazenar os K usuários ou itens mais similares. O algoritmo calcula e compara as similaridades de todos para todos, utilizando um dos métodos de similaridades mostrado anteriormente, e armazena para cada usuário ou item os seus K mais similares. Esse método é muito utilizado principalmente nos algoritmos baseados em memória.

Algorithm 1 Algoritmo para armazenar os K itens ou usuários mais similares

1: **para** cada $u \in U$ **faça**
2: **para** cada $v \in U$ **faça**
3: $similaridades[u][v] = sim(\vec{u}, \vec{v})$
4: $similaridades[v][u] = similaridades[u][v]$
5: $similaridades[u][u] = -\infty$;
6: **fim para**
7: $top[u] = \max_k similaridades[u]$
8: **fim para**
9: **retorna** top

No algoritmo 2, o conjunto R é formado pela união de todos os itens avaliados pelos k usuários mais similares excluindo desse conjunto os itens que u já avaliou. Os itens a serem recomendados serão os N itens de maiores frequências no conjunto C (Karypis, 2001).

Já no algoritmo 3, o conjunto R é formado pela união dos k itens mais similares aos itens já avaliados pelo usuário alvo u excluindo desse conjunto os itens que u já avaliou. O conjunto C é formado pelo somatório das similaridades entre cada item do conjunto R com cada item que o usuário u avaliou. Os itens recomendados serão os de maiores somatórios no conjunto C (Karypis, 2001).

O algoritmo 4 recomenda os itens com as maiores notas previstas pelo algoritmo *userKNN* para um dado usuário. Primeiramente é calculado e armazenado os K usuários mais similares ao usuário u, como mostrado no algoritmo 4.

Algorithm 2 Algoritmo de recomendação baseado em usuário (Karypis, 2001).

1: **para** cada $u \in U$ **faça**
2: $R = \{\}$
3: **para** cada $v \in top[u]$ **faça**
4: $R = R \cup \vec{v}$
5: **fim para**
6: $C = R - \vec{u}$
7: $topN[u] = itensMaisFrequentes(C)[1...N]$
8: **fim para**
9: **retorna** $topN$

Algorithm 3 Algoritmo de recomendação $TopN$ baseado em item (Karypis, 2001).

1: **para** cada $u \in U$ **faça**
2: $R = (\bigcup_{i \in \vec{u}} top[i]) - \vec{u}$
3: **para** cada $r \in R$ **faça**
4: $C[r] = \sum_{i \in \vec{u}} similaridades[r][i]$
5: **fim para**
6: $topN[u] = ItensDeMaiorSomatorio(C)[1...N]$
7: **fim para**
8: **retorna** $topN$

Após o conjunto $top[u]$ ser definido é calculada uma nota para cada item ainda não avaliado pelo usuário. Desses são recomendados os que tiverem as maiores notas (Adomavicius and Tuzhilin, 2005).

Algorithm 4 Algoritmo userKNN para recomendação de itens

1: **para** cada $u \in U$ **faça**
2: **para** cada $i \in \{I - \vec{u}\}$ **faça**
3: $C[i] = media(\vec{i}) + k * [\sum_{v \in top[u]} similaridades[u, v] * (v_i - media(\vec{v}))]$
4: **fim para**
5: $topN[u] = ItensDeMaioresNotas(C)[1...N]$
6: **fim para**
7: **retorna** $topN$

Por ultimo, o algoritmo 5 recomenda os itens mais populares até então, é o único algoritmo baseado em modelo da lista. Pode-se utilizar uma estrutura para armazenar os valores correspondentes a quantidade de vezes que certo item foi consumido pelos usuários, de modo a otimizar a procura pelos mais populares.

Algorithm 5 Algoritmo MostPop

1: **para** cada $u \in U$ **faça**
2: $C = \emptyset$
3: **para** cada $i \in \{I - \vec{u}\}$ **faça**
4: **se** $i \notin map$ **então**
5: $map[i] = count(\vec{i})$
6: **fim se**
7: $C[i] = map[i]$
8: **fim para**
9: $topN[u] = ItensComMaioresValores(C)[1...N]$
10: **fim para**
11: **retorna** $topN$

Todos os algoritmos foram apresentados em sua forma clássica, ou seja, implementados como estão documentados na literatura. Isso é importante para que se tenha um maior controle dos resultados obtidos, afim de comparar seus resultados com outras implementações que venham a ser feitas ou mesmo comparar com resultados encontrados na literatura.

Capítulo 3

Proposta

3.1 Introdução

Desde a popularização da *Internet* o comércio eletrônico se expandiu oferecendo inúmeras vantagens e desvantagens se comparado com as lojas físicas. Umas das desvantagens a princípio é a não possibilidade de receber uma recomendação para um item como acontecia com a interação entre o vendedor e o cliente. Outra característica é a capacidade destas lojas virtuais possuírem uma quantidade ilimitada de produtos.

Nesse cenário surgiram os sistemas de recomendação para apoiar esses clientes, sugerindo produtos e serviços de acordo com o gosto de cada pessoa, tornando mais pessoal o consumo na *Internet*. Evitando assim que os clientes se sentissem perdidos com a quantidade de informação a que são expostos (Ricci et al., 2011).

No contexto da Filtragem Colaborativa, existe um conjunto de usuários U e um conjunto de itens I. Cada usuário u define sobre um conjunto de itens I' um valor r. Sendo esse valor r definido dentro do conjunto de preferências R, que expressa o interesse do usuário u sobre cada item do conjunto I'.

A Filtragem Colaborativa utiliza como base somente a relação usuário-item, que é representada por uma nota. A relação entre essas duas entidades é normalmente representada por uma matriz com características esparsas, pois pequenas quantida-

des de relações são definidas se comparado ao tamanho total da matriz. Devido a sua simplicidade e eficiência esse método é um dos mais usados atualmente (Su and Khoshgoftaar, 2009), necessitando somente do conjunto de preferências dos usuários sobre os itens.

Um dos grandes problemas para quem começa a estudar a área de sistemas de recomendação é a não disponibilidade dos algoritmos clássicos da literatura, forçando o pesquisador a implementá-los para serem usados em sua pesquisa. Há ainda problemas que utilizam esses algoritmos clássicos como entradas para seus próprios, como exemplo o problema de *could start*.

Nos últimos anos surgiram muitos *frameworks* com o intuito de apoiar pesquisadores e usuários em geral, lhes poupando tempo. Esses projetos trazem algumas soluções já implementadas de forma que o usuário apenas as aplique.

Utilizar *frameworks* ou APIs que contenham esses algoritmos já implementados é uma boa alternativa para fugir do problema acima descrito. Porém, em meio a tantos projetos disponíveis, o pesquisador deve escolher aquele que trará melhores resultados, seja em termos de eficiência ou mesmo em comodidade.

Muitos *frameworks* hoje em dia trabalham com previsão de notas e recomendação de itens, porém a maioria não é focado em pesquisa ou voltado para educação. Muitos ainda são escritos em linguagens de programação que não são apropriadas para tal projeto, visto que a aplicação necessita de desempenho e além disso deve trazer um certo conforto para o usuário.

Programadores e cientistas preferem linguagens dinâmicas de alto nível para o desenvolvimento de algoritmos e análise de dados, como exemplo MATLAB®, Octave e SciPi. Contudo, essas linguagens abdicam do desempenho a favor da abstração, por isso também não são totalmente indicadas para a solução do problema.

3.2 Trabalhos relacionados

Atualmente existem muitos projetos que se propõem a auxiliar os usuários nos problemas descritos anteriormente.

O MyMediaLite (Gantner et al., 2011) por exemplo, se propõe a ser uma biblioteca *open source* para algoritmos de sistemas de recomendação. Trabalha tanto com previsão de avaliação (por exemplo, uma escala de notas de 1 a 5) quanto com previsão de item somente por retorno positivo (cliques em um item ou compra de um produto, por exemplo). Pode ser utilizado nas linguagens de programação C#, Cloujure, F#, Python e Ruby.

Um dos projetos mais conhecidos atualmente é o Apache Mahout, mas especificamente o *framework* Mahout Taste (Ekstrand et al., 2011). Reunindo diversos algoritmos e técnicas para recomendação baseado em Filtragem Colaborativa, por exemplo, *User-BasedCF*, *Item-BasedCF*, SVD++ e Fatoração de matriz com ALS, dentre outros. Suas soluções são eficientemente escaláveis.

Essas duas ferramentas são voltadas para grandes bases de dados e aplicações em sistemas de produção, mesmo os algoritmos aplicados sendo considerados básicos. Por isso eles não são indicados quando o contexto é de pesquisa ou educação.

O *framework* LensKit (Ekstrand et al., 2011) sim poderia ser aplicado a pesquisa e educação, pois possui quatro algoritmos básicos implementados: *User-BasedCF*, *Item-BasedCF*, Fatoração de matriz e *Slope-One*, além de várias ferramentas para auxiliar a criação e validação de modelos. Foi desenvolvido utilizando a linguagem Java e Groovy.

Outro que pode ser citado é o LibRec (Guo et al., 2015), uma biblioteca Java com licença GPL (software livre). Traz inúmeros algoritmos implementados, com fácil configuração, se propõe a ser mais rápido se comparado a outras bibliotecas.

Porém uma desvantagem já mencionada anteriormente é a linguagem de programação usada. A linguagem Java dificulta mais do que ajuda os usuários nesse contexto. É altamente indicado usar uma linguagem técnica/científica para atacar

o problema.

3.3 Proposta

O objetivo é desenvolver um *framework* que seja capaz de receber e testar diferentes algoritmos de recomendação de listas utilizando a abordagem de Filtragem Colaborativa. O *framework* será de fácil utilização, com um cunho científico e educacional. A linguagem de programação e os padrões usados no desenvolvimento não devem ser uma barreira aos utilizadores/colaboradores do projeto. Com isso o *framework* deverá possuir as seguintes características:

- O *framework* precisa ser elaborado para facilitar o aprendizado da área e que seja de fácil utilização para um professor em classe de aula.
- Precisa automatizar o processo de avaliação do desempenho dos modelos implementados utilizando *datasets* clássicos da área e assim centralizando o foco do pesquisador no desenvolvimento e aprimoramento dos modelos.
- A criação de novos modelos deverá ser feito utilizando mínimo esforço do pesquisador.

Para um desenvolvimento que não esbarre nos problemas descritos anteriormente, será utilizada a linguagem de programação *Julia* (Julia). Além de ser uma linguagem de alto nível, dinâmica de alto desempenho para computação técnica, tem uma sintaxe simplificada e possui licença livre diferentemente do MATLAB®.

3.4 Linguagem de programação Julia

A área científica necessita de uma linguagem de alta performance e que seja de fácil manipulação, mas o que se vê na prática é o contrario, muitos programadores tem usado linguagens dinâmicas lentas, pois abdicam da velocidade para obter uma maior abstração no código.

A linguagem Julia foi desenvolvida para unir esses dois mundos, graças as técnicas modernas para criação de linguagens e compilação. Julia é uma linguagem dinâmica, apropriada para computação numérica e cientifica, com um desempenho comparável a linguagens estáticas tradicionalmente utilizadas.

Possui tipagem opcional e é multi-paradigma, combinando características de programação imperativa, funcional e orientada a objetos. Sua sintaxe se assemelha ao GNU Octave ou MATLAB® trazendo um conforto para aqueles programadores que já estão familiarizados com essas linguagens que dominam o cenário da computação cientifica.

As características que se destacam no Júlia em comparação com outras linguagens dinâmicas são:

- A biblioteca padrão foi escrita utilizando a própria linguagem Julia, incluindo operadores primitivos como operações aritméticas de inteiros.
- Uma grande variedade de tipos para construir objetos, que também podem ser utilizados para fazer declarações de tipos.
- A habilidade de definir o comportamento de funções com base na combinação de vários tipos de argumentos.
- Bom desempenho, aproximando-se de linguagens estáticas e compiladas como C.

A imagem 3.1 a seguir apresenta uma comparação de desempenho entre algumas das principais linguagens de programação atualmente. A linguagem de programação C é utilizada como base para comparação das demais, 1.0 representa o tempo de execução da linguagem C e todas as outras linguagens são medidas o quanto sua execução varia da linguagem C.

A comparação é feita após executar diferentes procedimentos e algoritmos, como ordenar um vetor utilizando o algoritmo *QuickSort*, por exemplo. Quanto menor for o resultado da comparação melhor.

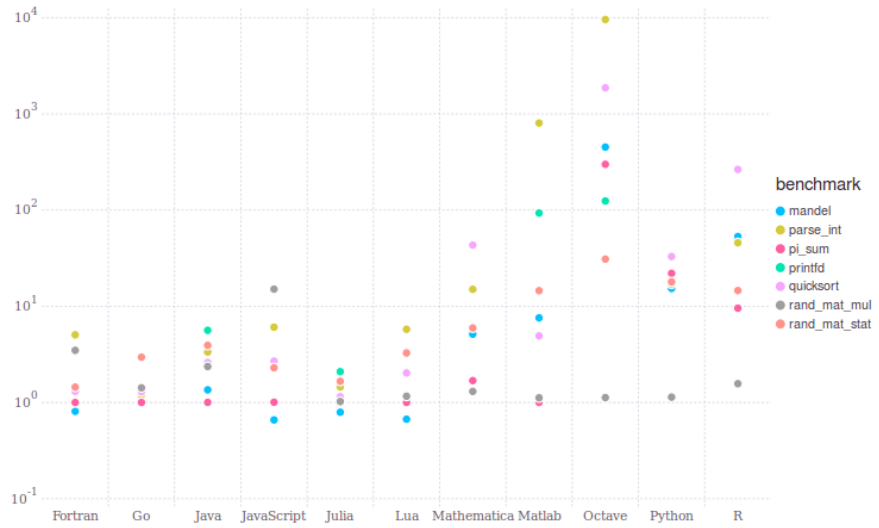

Figura 3.1: Comparação das linguagens de programação (Julia).

É fácil ver a melhor performance com relação ao tempo de execução da linguagem Julia se comparado com outras linguagens de alto nível de mesmo proposito. Ficando sempre muito próxima da linguagem C.

3.5 Arquitetura

Como já mencionado, o *framework* Recsys.jl será desenvolvido utilizando a linguagem Julia, já que possui propósito científico e educacional. A figura 3.2 apresenta uma visão dos principais componentes do *framework* para geração de listas de recomendação.

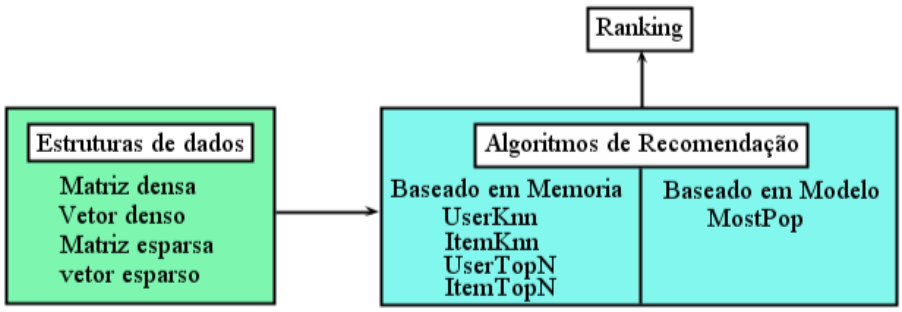

Figura 3.2: Visão geral dos componentes presentes no *framework* Recsys.jl.

A imagem 3.3 mostra um diagrama de classes representando a organização de

todas as classes envolvidas no projeto, assim como suas interações.

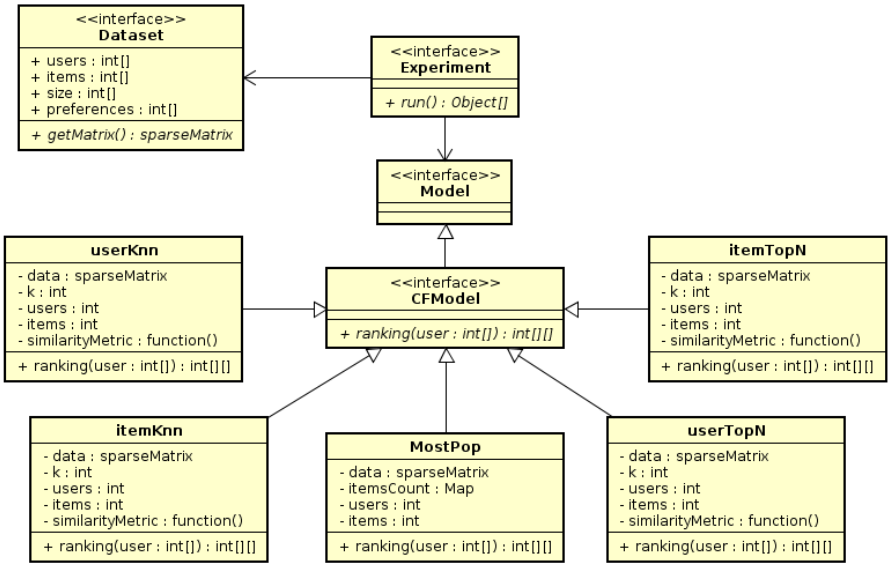

Figura 3.3: Diagrama de classes do *framework* Recsys.jl.

O modelo para recomendação em Filtragem Colaborativa é definido como uma classe que possui dois métodos obrigatórios: *new* e *ranking*. O primeiro carrega quais são os procedimentos de criação no modelo. E o segundo método, *rank*, recebe uma lista com os identificadores dos usuários que necessitam da recomendação, como pode ser visto na imagem 3.4.

```
type NewModel <: CFModel
    rank::Function
    function NewModel(data::Dataset)
        this = new()
        this.ranking = function(data::Array)
            return data
        end
        return this
    end
end
```

Figura 3.4: Exemplo de criação de um modelo de Filtragem Colaborativa.

O único parâmetro obrigatório no método de criação do modelo é um *Dataset*. Por sua vez, *Dataset* é uma classe criada com o papel de representar uma base de dados de Filtragem Colaborativa, ou seja, é constituído com as notas dos usuários sobre os itens consumidos. Podendo utilizar como entrada um arquivo CSV contendo essas informações, ou se nenhum parâmetro for passado, será usado como padrão a base MovieLens 100k (Miller et al., 2003).

Para instanciar um novo *Dataset*, deve ser passado o caminho do diretório do arquivo CSV, podendo explicitar, ou não, o número total de usuários, itens e relações, como visto na imagem 3.5.

```
dataset = Dataset("example/example.csv", 100, 50, 20)
experiment = Recsys.HoldOutTopN(0.9, dataset)
function how_to_create_model(data)
    return Recsys.userKNNTopN(data, 20)
end
result = experiment.run(how_to_create_model)
@show result
```

Figura 3.5: Exemplo de um experimento usando a técnica HoldOut.

Para facilitar a manipulação desse *Dataset* em inúmeros casos é instanciado uma estrutura chamada matriz esparsa (*SparseMatrix*), com intuito de abstrair a esparsidade da matriz de relações usuário-itens que forma o *Dataset*, problema esse já citado anteriormente. Supondo que a matriz esparsa seja indexada por linha e que cada linha represente um usuário, cada linha será formada por um vetor contendo apenas os itens que tenham sido avaliados pelo usuário, sem espaços vazios.

Uma classe para facilitar os procedimentos de avaliação dos modelos pode ser definida como visto na imagem 3.6, onde se cria um experimento usando a estratégia chamada *Hold-Out* (10% para testar e 90% para treinar o modelo) para que o *framework* realize os testes.

Além de definir um experimento por *HoldOut*, pode-se também utilizar a estratégia *K-Fold*, onde a base é dividida em k subconjuntos, onde $k - 1$ servem para

```
experiment = Recsys.HoldOutTopN(0.9)
function how_to_create_model(data)
    return Recsys.userKNNTopN(data, 20)
end
result = experiment.run(how_to_create_model)
@show result
```

Figura 3.6: Exemplo de um experimento usando a técnica HoldOut.

treinar o modelo e o último subconjunto serve para teste, como visto na imagem 3.7, na qual é apresentado um experimento executado utilizando apenas o *fold* 1 para testes.

```
experiment = Recsys.KFoldTopN(10)
function how_to_create_model(data)
    return Recsys.userKNNTopN(data, 20)
end
fold = 1
result = experiment.run(how_to_create_model, fold)
@show result
```

Figura 3.7: Exemplo de um experimento usando a técnica K-Fold.

Porém utilizando o método *K-Fold* também é possível executar todos os subconjuntos (*folds*) ao mesmo tempo, como pode ser visto a seguir.

```
experiment = Recsys.KFoldTopN(10)
function how_to_create_model(data)
    return Recsys.userKNNTopN(data, 20)
end
result = experiment.runAll(how_to_create_model)
@show result
```

Figura 3.8: Exemplo de um experimento usando a técnica K-Fold.

Como mostrado, o *framework Recsys.jl* atende a todos os requisitos para auxiliar seus usuários em seus projetos. Seja esse um usuário iniciante ou experiente.

Capítulo 4

Experimentação

4.1 Introdução

Na sessão anterior foi apresentada a proposta do presente trabalho de um *framework* para auxiliar pesquisadores e usuários em geral que tenham interesse na área de sistemas de recomendação. Para tal, esse *framework* deve seguir preceitos que resolvam ou diminuam os problemas enfrentados por quem começa a estudar a área.

Nesse capítulo será apresentado o comparativo das técnicas implementadas no *framework* proposto, essas técnicas também carregam configurações específicas que também serão testadas. O objetivo do experimento é avaliar cada uma das técnicas e verificar o seu desempenho e estabilidade, seguindo uma série de medidas bem definidas.

Todos os métodos e *frameworks* foram executados em um *notebook* de configurações: processador Intel Core I5-3320M (2.66GHz x 4), 8GBs de mémoria RAM e sistema operacional Ubuntu 15.04 x64.

4.2 Base de Dados

O conjunto de dados utilizado nos experimentos foi o *MovieLens* (Miller et al., 2003). Desenvolvida pelo *GroupLens*[1], esta base é baseada em um sistema de recomendação de filmes, o portal possui diversos tamanhos de bases, porém nesse experimento será usada a base com 100.000 avaliações. Nela existem 943 usuários que se relacionam com 1682 filmes por meio de notas de valor inteiro que variam entre 1 a 5.

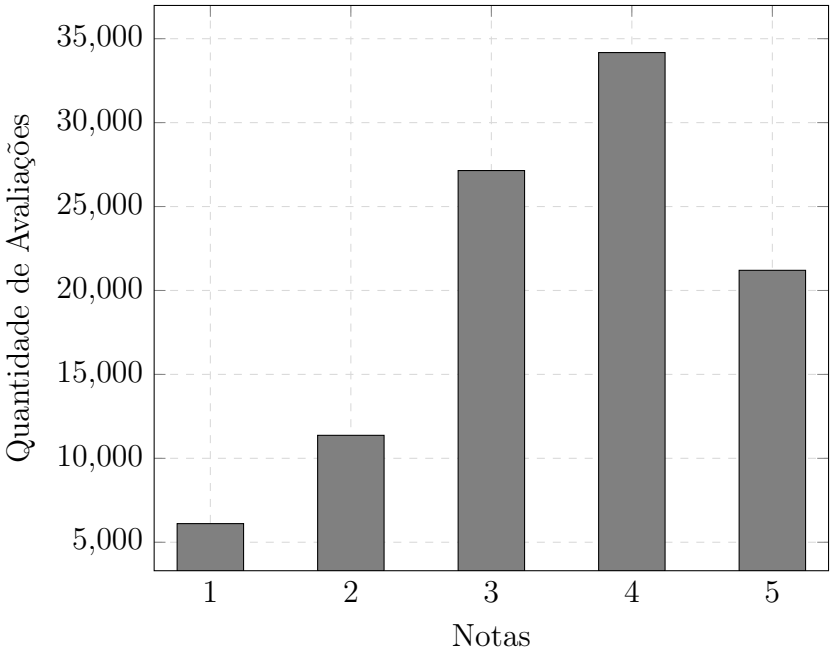

Figura 4.1: Quantidade de avaliações por nota da base *MovieLens*.

Como era de se supor, por se tratar de uma base de sistema de recomendação, ela possui poucas avaliações se comparada com o tamanho total das possibilidades de relações. A proporção de avaliações com relação ao total, ou seja, o índice de esparsidade é de 6.30%. A distribuição das notas possui o comportamento de uma normal com média 3.5299 e desvio padrão de 1.1257, como mostra a figura 3.8 (Carmo, 2013).

A quantidade de avaliações feitas por um usuário possui média de 106.04 avaliações e se comporta como uma distribuição exponencial, como pode se observar na

[1] `http://www.grouplens.org/`

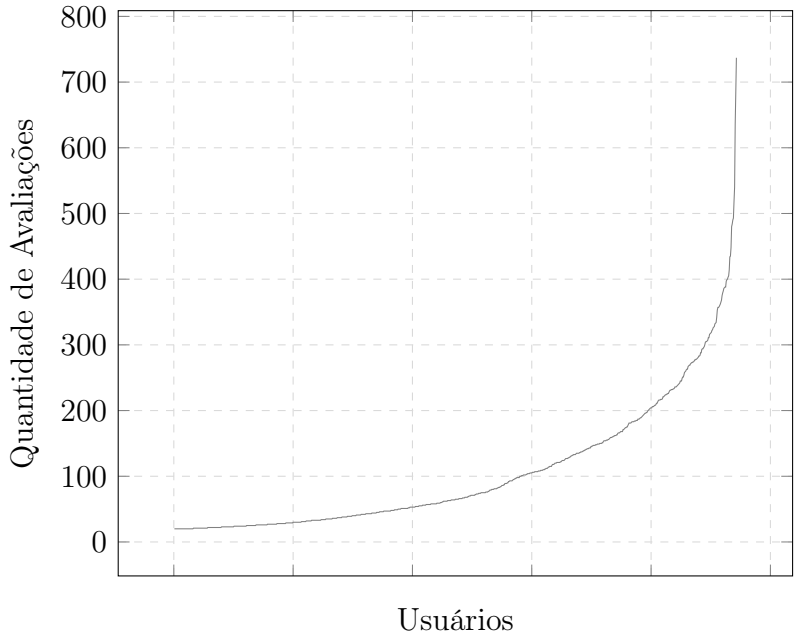

Figura 4.2: Quantidade de avaliações por cada usuário da base *MovieLens*.

Figura 4.1, sendo que a mínima é de 20 avaliações e a máxima de 737. Essa característica de possuir um valor alto para o mínimo de avaliações por usuário pode ser explicado pelas características do sistema *MovieLens*. Durante o ato de criação da conta, o usuário é obrigado a avaliar essa quantidade mínima de filmes (Carmo, 2013).

Uma distribuição parecida pode ser vista na Figura 4.2, que representa a quantidade de avaliações por filme, com uma média de 59.45 avaliações. Como não há uma obrigatoriedade sobre uma quantidade mínima de avaliações, a quantidade mínima é de 1 avaliação e a máxima é de 583 (Carmo, 2013).

4.3 Metodologia e Organização dos Experimentos

Como a proposta é desenvolver um *framework* para geração e avaliação de listas de recomendação, os experimentos serão voltados para testar a qualidade dessas recomendações feitas comparando-as com resultados de técnicas encontradas na literatura e em outros *frameworks*.

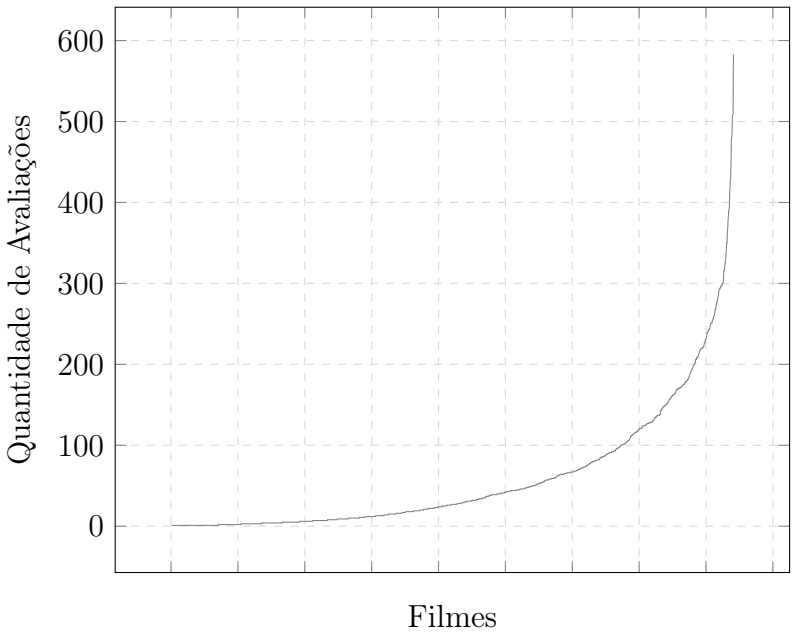

Figura 4.3: Quantidade de avaliações de cada filme da base *MovieLens*.

As medidas serão feitas calculando o *Recall*, *Precision*, *F1 Score*, *MAP* e *nDCG* das listas de itens recomendados, de modo a quantificar a qualidade da recomendação feita. Também será marcado o tempo de execução de cada método para que se tenha uma comparação de eficiência, marcação essa que só contará o tempo de treinamento mais o tempo de teste do método e será expressa na forma $MM : SS$.

- *Precision* - Também conhecida como valor preditivo positivo. É a fração dos itens recomendados corretamente com relação ao total de itens recomendados (Herlocker et al., 2004).

$$Precision = \frac{|itens\ relevantes \cap itens\ recomendados|}{|itens\ recomendados|}$$

- *Recall* - Também conhecida como sensibilidade. É a fração dos itens recomendados corretamente com relação ao total de itens que deveria ser recomendado (Herlocker et al., 2004).

$$Recall = \frac{|itens\ relevantes \cap itens\ recomendados|}{|itens\ relevantes|}$$

- *F1 Score* - Une as medidas anteriores por meio de uma média harmônica

(Herlocker et al., 2004).

$$F1Score = 2 * \frac{(Recall * Precision)}{(Recall + Precision)}$$

- *MAP - Mean Average Precision* Aplica o *Precision* e *Recall* em cada posição da lista de recomendação. É calculado uma média de todas as *Precision* com relação aos *Recall* calculados ao longo da lista de recomendação (Agichtein et al., 2006).

$$MAP = \sum_{i=1}^{n} \frac{AveP(i)}{n}$$

- *nDCG - Normalized Discounted Cumulative Gain* é uma medida de qualidade de ranqueamento, que quantifica o ganho da lista de recomendação se baseando na posição de cada item recomendado (Clarke et al., 2008).

$$nDCG = \frac{DCG}{IDCG} = \frac{rel_1 + \sum_{i=2}^{n} \frac{rel_i}{log_2 i}}{relSorted_1 + \sum_{i=2}^{n} \frac{relSorted_i}{log_2 i}}$$

Cada experimento apresenta a execução de um método ou mais métodos de recomendação, variando os parâmetros das configurações de cada algoritmo para um melhor ajuste, de modo a comparar seus resultados com as outras técnicas e *frameworks* já estabelecidos, *LibRec* e *MMLite*.

O *framework* possui dois métodos para validar o modelo submetido ao experimento, *HoldOut* e *KFold*, já citados anteriormente, mas que serão descritos com mais detalhes nas sessões 4.3.1 e 4.3.2.

4.3.1 *HoldOut*

Uma base de dados é separada em dois conjuntos sem interseção de tamanhos variados de acordo a necessidade do projeto, porém normalmente usa-se o padrão de 2/3 para o conjunto de treino e 1/3 para o conjunto de testes.

O conjunto de treino é o responsável por treinar o modelo que será avaliado. Para um sistema de recomendação, que é o foco do trabalho, o conjunto de testes

terá 90% das relações de preferências dos usuários sobre os itens, informações essas contidas na base de dados.

O conjunto de teste é utilizado ao final da fase de treino, apenas para qualificar o modelo. Aplicado a um sistema de recomendação, as relações entre usuários e itens do conjunto de teste servem de parâmetro para saber se foi realmente recomendado um item que o usuário consumiria, ou seja, um item que esteja relacionado ao usuário no conjunto de teste.

4.3.2 *K-Fold*

Já no método *K-Fold* a base de dados é divida em K conjuntos sem interseção de igual tamanho, sendo esses chamados de *folds*. A ideia é parecida com o *HoldOut*, porém aqui são $k-1$ conjuntos de treino e apenas 1 conjunto de teste.

Esse procedimento é executado K vezes, alternando circularmente os conjuntos a cada iteração, ou seja, a cada execução um conjunto diferente será utilizado para teste. Ao final das K iterações, pode ser calculada uma média dos resultados de cada iteração por exemplo.

4.3.3 Resultados

Nesta seção serão apresentados os resultados obtidos em cada experimento. E ao final, será feita uma análise conclusiva destes resultados.

4.3.3.1 Experimento *MostPop*

Nesse experimento é apresentado os resultados do algoritmo *MostPop* com o tamanho da lista de recomendação N igual a 5 e a 10. A tabela 4.1 apresenta os resultados das medidas *Precision*, *Recall* e *F1 Score*:

A seguir são apresentadas as medidas *MAP*, *nDCG* e o tempo de execução de cada *framework*:

MostPop

	$N = 5$			$N = 10$		
MostPop	*Recsys.jl*	*MMLite*	*LibRec*	*Recsys.jl*	*MMLite*	*LibRec*
Precision	0.217	0.212	0.211	0.193	0.192	0.190
Recall	0.071	0.071	0.070	0.115	0.115	0.116
F1 Score	0.107	0.106	0.105	0.144	0.143	0.144

Tabela 4.1: Tabela de resultados e comparações do método *MostPop*.

MostPop

	Recsys.jl	*MMLite*	*LibRec*
MAP	0.186	0.136	0.135
nDCG	0.417	0.479	0.477
Tempo	00:02	00:03	00:03

Tabela 4.2: Tabela de resultados e comparações do método *MostPop*.

4.3.3.2 Experimento *UserKnn* e *ItemKnn*

Nesse experimento é apresentado os resultados dos métodos *UserKnn* e *ItemKnn* também com o tamanho da lista de recomendação N igual a 5 e a 10. Para este experimento foi definido uma vizinhança de K igual a 80, ou seja, para fazer a recomendação será utilizado os 80 usuários mais similares de cada usuário que precise da recomendação. Para o cálculo das similaridades foi utilizado o método de similaridades por cosseno.

Segue abaixo as tabelas com os resultados das medidas *Precision*, *Recall* e *F1 Score*:

A seguir seguem as tabelas com os resultados das medidas *MAP*, *nDCG* e o tempo de execução dos métodos nos *frameworks*.

4.3.3.3 Experimento *UserTopN* e *ItemTopN*

Nesse experimento é apresentado os resultados dos algoritmos *UserTopN* e *ItemTopN* propostos por (Karypis, 2001) com tamanho da lista de recomendação N va-

UserKnn

	$N = 5$			$N = 10$		
framework	*Recsys.jl*	*MMLite*	*LibRec*	*Recsys.jl*	*MMLite*	*LibRec*
Precision	0.394	0.397	0.338	0.322	0.334	0.280
Recall	0.138	0.138	0.116	0.212	0.218	0.182
F1 Score	0.204	0.204	0.172	0.256	0.263	0.220

Tabela 4.3: Tabela de resultados e comparações do método *UserKNN*.

ItemKnn

	$N = 5$			$N = 10$		
framework	*Recsys.jl*	*MMLite*	*LibRec*	*Recsys.jl*	*MMLite*	*LibRec*
Precision	0.220	0.314	0.318	0.190	0.279	0.260
Recall	0.034	0.096	0.103	0.062	0.167	0.164
F1 Score	0.060	0.147	0.155	0.094	0.208	0.201

Tabela 4.4: Tabela de resultados e comparações do método *ItemKnn*.

UserKnn

framework	*Recsys.jl*	*MMLite*	*LibRec*
MAP	0.316	0.268	0.208
nDCG	0.450	0.610	0.554
Tempo	12:21	01:21	05:57

Tabela 4.5: Tabela de resultados e comparações do método *UserKnn*.

ItemKnn

framework	*Recsys.jl*	*MMLite*	*LibRec*
MAP	0.151	0.223	0.187
nDCG	0.379	0.563	0.536
Tempo	21:43	02:19	10:15

Tabela 4.6: Tabela de resultados e comparações do método *ItemKnn*.

riando entre 5 e 10 e assim como o experimento anterior será utilizado um tamanho de K igual a 80. Para o cálculo de similaridades foi utilizado o método de similaridade por cosseno, para o algoritmo *ItemTopN* foi utilizado também o método de

similaridade por probabilidade condicional (representado por *ItemTopN/CP*).

A tabela a seguir mostra o resultado das medidas *Precision*, *Recall* e *F1 Score*, também apresenta a comparação com os outros métodos implementados no *framework Recsys.jl*:

Comparativo

	Prec@5	*Prec@10*	*Rec@5*	*Rec@10*	*F1@5*	*F1@10*
UserKnn	0.394	0.322	0.138	0.212	0.204	0.256
ItemKnn	0.220	0.190	0.034	0.062	0.060	0.094
UserTopN	0.378	0.314	0.131	0.205	0.194	0.249
ItemTopN	0.363	0.300	0.134	0.205	0.196	0.243
ItemTopN/CP	0.328	0.268	0.105	0.163	0.159	0.203
MostPop	0.217	0.193	0.071	0.115	0.107	0.144

Tabela 4.7: Tabela de resultados e comparações do algoritmo *UserKNN*.

A seguir são apresentadas as medidas *MAP*, *nDCG* e o tempo de execução de cada método no *framework Recsys.jl*.

Comparativo

	MAP	*nDCG*	Tempo
UserKnn	0.343	0.534	12:21
ItemKnn	0.151	0.379	21:43
UserTopN	0.348	0.539	02:54
ItemTopN	0.321	0.525	08:20
ItemTopN/CP	0.269	0.509	07:00
MostPop	0.186	0.417	00:02

Tabela 4.8: Tabela de resultados e comparações do algoritmo *UserKNN*.

4.3.4 Análise dos Resultados

Todas as implementações executadas no *framework Recsys.jl* tiveram resultados equiparados aos outros *frameworks* com relação a qualidade da recomendação.

A única exceção foi o método *ItemKnn* que teve resultados bem menores que os esperados, erros na implementação seriam as causas mais prováveis.

Outro ponto que se destaca foram os tempos de execução cronometrados. A linguagem Julia tem tudo para ser superior em performance se comparada com Java e de ao menos se equiparar ao C# (linguagens do *LibRec* e *MyMediaLite* respectivamente) como foi mostrado na sessão 3.4. Mas não foi o que se viu nos resultados.

O tempo alto na maioria dos métodos executados pelo *Recsys.jl* deve-se as implementações dos algoritmos. Pois ainda necessitam ser otimizados utilizando funções e estruturas especificas da linguagem de modo a acelerar a execução.

Apesar do desempenho em relação ao tempo de execução ter ficado a desejar, a facilidade no desenvolvimento e utilização do *framework Recsys.jl* é superior aos demais *frameworks* aqui listados. Tendo em vista que esse era um dos objetivos a serem alcançados.

Capítulo 5

Conclusão

5.1 Considerações acerca do trabalho

Esse trabalho propôs um *framework* que auxiliasse usuários no estudo de sistemas de recomendação. Foram implementados diversos métodos baseados em filtragem colaborativa e algumas ferramentas, de modo a diminuir o esforço que um usuário iniciante enfrentaria ao começar na área.

Por ser voltado para um meio educacional, o *framework Recsys.jl* precisou ser elaborado de forma simples, de modo a diminuir os problemas que pudessem ocorrer na utilização. Toda sua arquitetura é organizada em módulos de forma a facilitar o entendimento do usuário e com uma sintaxe simples e de alto nível proveniente da linguagem Julia utilizada no desenvolvimento que também auxilia o usuário.

Dessa forma esse trabalho cumpriu com o que foi proposto inicialmente, se equiparando em muitos pontos aos principais projetos da área. Tendo conseguido unir eficiência e um bom entendimento, criando facilidades aos seus usuários.

5.2 Trabalhos futuros

Como já comentado anteriormente, o desenvolvimento dos métodos e das ferramentas deve ser refatorado utilizando técnicas otimizadas disponibilizadas pela

linguagem Julia, como também é um projeto recente está em constante atualização. Novas funções e melhorias são lançadas a cada nova versão e esse trabalho foi desenvolvido utilizando a versão 0.3.11 do Julia.

Por ser um projeto recente, o *Recsys.jl* ainda possui poucas técnicas disponibilizadas se comparado aos outros *frameworks* da área. Por isso em futuras atualizações será desenvolvido novas abordagens aumentando assim a gama de possibilidades de algoritmos que o usuário possa escolher.

Referências

Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26. ACM, 2006.

John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

Filipe Braida Carmo. *Transformando o problema de filtragem colaborativa em aprendizagem de máquinas supervisionado*. PhD thesis, Universidade Federal do Rio de Janeiro, 2013.

Charles LA Clarke, Maheedhar Kolla, Gordon V Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 659–666. ACM, 2008.

Michael D Ekstrand, Michael Ludwig, Jack Kolb, and John T Riedl. Lenskit: a modular recommender framework. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 349–350. ACM, 2011.

Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Mymedialite: A free recommender system library. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 305–308. ACM, 2011.

Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 257–260. ACM, 2010.

David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35 (12):61–70, 1992.

Guibing Guo, Jie Zhang, Zhu Sun, and Neil Yorke-Smith. Librec: A java library for recommender systems. In *Posters, Demos, Late-breaking Results and Workshop Proceedings of the 23rd International Conference on User Modeling, Adaptation and Personalization*, 2015.

Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

Julia. The julia language.

George Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254. ACM, 2001.

Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

Bradley N Miller, Istvan Albert, Shyong K Lam, Joseph A Konstan, and John Riedl. Movielens unplugged: experiences with an occasionally connected recommender system. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 263–266. ACM, 2003.

Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

WORLDWIDEWEBSIZE. The size of the world wide web. `http://www.worldwidewebsize.com/`, 2016.