UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA

PAULO FRANCISCO DA SILVA FREITAS
VINICIUS GOMES STEINDL AYRES

NeuralCF.jl: Modelos Neurais para Filtragem Colaborativa

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

Nova Iguaçu, Janeiro de 2020

NeuralCF.jl: Modelos Neurais para Filtragem Colaborativa

Paulo Francisco da Silva Freitas

Vinicius Gomes Steindl Ayres

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto Multidiciplinar 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:

_____

Paulo Francisco da Silva Freitas

_____

Vinicius Gomes Steindl Ayres

Aprovado por:

_____

Prof. Filipe Braida do Carmo, D.Sc.

_____

Prof. Bruno José Dembogurski, D.Sc.

_____

Prof. Fellipe Ribeiro Duarte, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Janeiro de 2020

Agradecimentos

Paulo Francisco da Silva Freitas

Primeiramente agradeço à minha mãe Denise, que é a minha referência como pessoa e sempre me apoiou em tudo na minha vida e é meu maior orgulho poder dizer que sou filho dela. Obrigado mãe. Ao meu pai Paulo, que apesar de não estar mais aqui, sempre me deu forças e acreditou em mim. Obrigado, Pai. à minha tia Sonia e ao meus avós Thereza e Damásio. Agradeço também a toda minha família que acompanhou minha jornada até aqui.

Agradeço ao meu amigo Vinicius Ayres, pois sem ele esse projeto não seria possível. Ao meu orientador Filipe Braida, que admiro e respeito desde quando tive aula com ele em Computação 1 pelo suporte e por acreditar em nós. Ao corpo de docentes do curso de ciência da computação da Universidade Federal Rural do Rio de Janeiro, que são excelentes profissionais.

Aos meus amigos que conheci durante a graduação Matheus, Ryan, Nelson, Esdras, Angelo, Michael, Julia, Vitor e Bruno que passaram comigo por vários momentos difíceis durante o curso e conseguimos supera-los, sempre de bom humor. Espero leva-los comigo para a vida toda.

Aos meus amigos que tenho desde a época da escola e levo para a vida toda Matheus, Antonio, Gabriel, Borges e Israel, que sempre estiveram comigo desde o momento em que eu escolhi esse curso até hoje, sempre me apoiando e me dando forças de diferentes maneiras.

E a por último, à todos que passaram por mim de uma maneira positiva durante

este período da minha vida, obrigado.

Vinicius Gomes Steindl Ayres

Quero agradecer, em primeiro lugar a Deus, que desde o início tem estado comigo e me colocado na direção certa em minha jornada, sempre me abençoando de uma forma que não tenho como dimensionar.

Agradeço aos meus pais, Delcio e Alessandra, que se dedicaram ao máximo por mim e pelo meu irmão, sempre me apoiaram em minhas escolhas e acreditaram em mim em todos os momentos. À minha vó Luiza, que apesar de todas as dificuldades, sempre incentivou a ir até o fim. A toda minha família, que direta ou indiretamente me possibilitaram chegar até aqui.

Gostaria de agradecer ao meu amigo Paulo Francisco, por todo empenho e dedicação que tornaram esse trabalho possível. À todo o corpo docente do curso de ciência da computação da Universidade Federal Rural do Rio de Janeiro, em especial ao meu orientador Filipe Braida, por toda a paciência e fé que teve em nós durante esse projeto final.

Agradeço a todos com que eu convivi durante a graduação, aprendi algo com cada pessoa e experiência. Quero agradecer principalmente aos meus amigos: Ângelo, Daniel, Gabriel, Gabriela, Leonardo, Lucas e Thales, que tornaram essa etapa mais fácil e carregaram junto comigo o peso da graduação.

Ingrid, minha melhor amiga e meu amor. Obrigado pela paciência de ouvir minhas reclamações e crises nesse período. Você sempre me deu forças e me fez querer buscar ser melhor. Obrigado por estar ao meu lado, confiar em mim mais do que eu mesmo e nunca duvidar ao me dizer que esse dia chegaria.

Dedico esse trabalho a minha família, aos meus amigos, e a todas pessoas que puderam contribuir para que eu chegasse aqui. Obrigado!

RESUMO

NeuralCF.jl: Modelos Neurais para Filtragem Colaborativa

Paulo Francisco da Silva Freitas e Vinicius Gomes Steindl Ayres

Janeiro/2020

Orientador: Filipe Braida do Carmo, D.Sc.

Sistemas de Recomendação são utilizados para filtrar informação a partir de uma grande quantidade de elementos. Há diversas técnicas e algoritmos para serem utilizados, adequando-se a situações diferentes, de acordo com o contexto em que é aplicado. A Filtragem Colaborativa é uma das técnicas mais utilizadas atualmente, que faz uma recomendação baseada em itens de usuários com o perfil semelhante ao seu. As Redes Neurais também vem se popularizando cada vez mais na conjuntura atual, porém não muito na área de Sistemas de Recomendação. Entretanto, alguns projetos tem utilizado Redes Neurais para realizar tarefas de recomendação, e vem encontrando sucesso. Esse projeto mostra o processo de criação de um pacote para o *framework Persa*, que reúne diversas técnicas de filtragem colaborativa, focando na utilização de modelos neurais, possibilitando o fácil acesso e utilização do público.

ABSTRACT

NeuralCF.jl: Modelos Neurais para Filtragem Colaborativa

Paulo Francisco da Silva Freitas and Vinicius Gomes Steindl Ayres

Janeiro/2020

Advisor: Filipe Braida do Carmo, D.Sc.

Recommender Systems are used to provide filtered information of a large amount of elements. There are different techniques and algorithms that can be utilized in a recommendation, and their choice is made knowing the domain of the problem and the area to be applied. The Collaborative Filtering is one of the most popular techniques ate the moment, it uses users with similar profiles to recommend itens that a certain user would like to recieve. The Deep Neural Networks are becoming more popular day after day in a various areas of Computer Science, but not a lot in Recommender Systems. Recently, some projects have started to use Deep Neural Networks to perform recommendation task, and have found success. This project will present the creation proccess of a package. for the framework Persa, that bring together certain collaborative filtering techniques, enabling them to be easy to access and utilized by the community.

Lista de Figuras

Lista de Códigos

Lista de Abreviaturas e Siglas

SR	Sistema de Recomendação
NeuMF	Neural Matrix Factorization
MF	Matrix Factorization
MLP	Multi Layer Perceptron
DL	Deep Learning
RN	Redes Neurais
FC	Filtragem Colaborativa

Sumário

Capítulo 1

Introdução

A cada dia que passa a tecnologia vem se modernizando e se integrando cada vez mais na rotina da sociedade. Conforme a tecnologia, em especial a computação, se integra no cotidiano das pessoas, o volume de informação produzida se torna cada vez maior, cada vez mais esses dados são acessados.

Com tamanho aumento na quantidade de informação gerada, grande parte dela acaba sendo irrelevante, e devido a isso, filtrar esses dados se tornou um trabalho rotineiro e cansativo para a sociedade. Para lidar com tal problema, surgem os Sistemas de Recomendação (SR), tornando esse processo mais eficiente e preciso. Grandes empresas como *Amazon*[1], *YouTube*[2] e *Netflix*[3] vem investindo nessa área cada vez mais, trazendo a informação que tenha maior chance de agradar o usuário, em busca de satisfazer suas necessidades e incentivar ao uso contínuo dos seus serviços.

Um Sistema de Recomendação é definido como um conjunto de técnicas e ferramentas que fornecem sugestões de itens que são relevantes ao usuário (RICCI; ROKACH; SHAPIRA, 2015). Dentre as abordagens existentes para a realização da recomendação, uma das mais populares é a Filtragem Colaborativa (FC), tornando esse processo mais eficiente e preciso. Esta que consiste em realizar a recomendação baseada em itens que usuários com perfis semelhantes já avaliaram positivamente

[1] https://amazon.com
[2] https//youtube.com
[3] https://netflix.com

(ZHANG; CHUA, 2017).

Desde o prêmio *Netflix*, os SR vem se tornando cada vez mais populares, pois um conjunto de dados de larga escala com 100 milhões de avaliações foi disponibilizado, o que atraiu diversos interessados para o campo (RICCI; ROKACH; SHAPIRA, 2015). Isso contribuiu para o surgimento de diversos *frameworks*, que colaboraram para o desenvolvimento da área, pois deu acessibilidade a comunidade de maneira simples.

Há diversos algoritmos de Filtragem Colaborativa na literatura, porém muitas vezes não há informações aprofundadas sobre as implementações, o que dificulta a utilização desses algoritmos pela comunidade.

O objetivo desse trabalho é ajudar na popularização de modelos neurais na área de SR, pois as redes neurais vem se destacando em diversas áreas da computação, porém não há muitos trabalhos que falam sobre a utilização delas em sistemas de recomendação.

O *NeuralCF.jl* é um pacote feito para o Persa[4], um *framework* para a linguagem Julia[5]. Ele irá disponibilizar diversos algoritmos que utilizem modelos neurais voltados para a Filtragem Colaborativa, com o intuito de ajudar e incentivar a comunidade de entusiastas dessa área a aprofundarem seus estudos e realizarem implementações. O pacote será aberto ao público, permitindo que interessados contribuam com suas ideias e implementações, a fim de criar um ambiente em que possa ser compartilhado e discutido diversas ideias sobre a área.

1.1 Objetivo

Os principais objetivos do pacote são:

- Estender o *framework* Persa, aproveitando de suas estruturas voltadas para a área para a implementação de diversos modelos, tais como: GMF, MLP e NeuMF (HE, 2015);

[4]https://github.com/JuliaRecsys/Persa.jl
[5]https://julialang.org

- Mostrar a aplicação de modelos neurais em Sistemas de Recomendação para o público;
- Disponibilizar modelos neurais de previsão que utilizem redes neurais disponíveis e de fácil utilização;
- Aproveitar a linguagem específica de domínio (DSL) que o persa possui para garantir seu funcionamento e extensão de forma semântica, amigável e previsível ao usuário;
- Disponibilizar modelos voltados para Filtragem Colaborativa para a comunidade de maneira aberta e fácil de usar;

1.2 Organização do Trabalho

Esse trabalho aborda o desenvolvimento do trabalho presente e está organizado como a seguir:

- Capítulo 2: será abordada a fundamentação teórica necessária para o trabalho, fazendo uma introdução aos SR e a Filtragem Colaborativa.
- Capítulo 3: serão abordados os modelos implementados no trabalho, explicando seu funcionamento e a ideia por trás deles;
- Capítulo 4: Serão apresentados os experimentos e resultados no desenvolvimento do pacote, fazendo uma comparação com implementações já existentes.
- Capítulo 5: Conclusão e trabalhos futuros.

Capítulo 2

Sistemas de Recomendação

Para o melhor compreensão de técnicas e tecnologias usadas no desenvolvimento do trabalho, este capítulo apresentará os conceitos teóricos mais relevantes para tal. Os seguintes assuntos serão tratados nas seções a seguir: Sistemas de Recomendação, seus impactos e utilizações no cotidiano, seus problemas, suas diferentes abordagens e técnicas de implementação.

2.1 Introdução

Com a popularização da internet, a quantidade de dados acessíveis cresceu em grande escala. Empresas como Netflix[1] e Amazon[2], por exemplo, possuem em seu catálogo mais de $17,000$ (BENNETT; LANNING, 2007) filmes e $410,000$ títulos (EKSTRAND; RIEDL; KONSTAN, 2011) para seleção, respectivamente. Diante desse mar de dados, encontrar conteúdo relevante tornou-se um processo exaustivo. Com o intuito de auxiliar os usuários a lidar com a enorme quantidade de dados, surgem os Sistemas de Recomendação (SR) como ferramenta para oferecer recomendações personalizadas de seus conteúdos ou serviços

Os SR são técnicas e ferramentas de software que sugerem itens mais relevantes ao usuário, baseado em seu perfil. Eles vem se provando cada vez mais importantes

[1] https://www.netflix.com/
[2] https://www.amazon.com/

para lidar com o excesso de informação que temos nos dias de hoje, fazendo sugestões ao usuário de acordo com seu perfil. Conforme o usuário vai interagindo com o sistema, ele vai se tornando cada vez mais preciso, adaptando aos gostos (RICCI; ROKACH; SHAPIRA, 2015). Dessa forma, uma empresa pode alavancar suas vendas, e melhorar a experiência do usuário ao encontrar mais itens de seu interesse.

O problema central dos SR é prever a nota de um usuário para um item, isto é, analisar o quão relevante aquele item ou produto é para aquele cliente. Para resolver tal problema, são usados diversos métodos de avaliação, tentando sempre se aproximar o máximo ao gosto do usuário, para dessa forma, tornar a recomendação a mais precisa o possível, e melhorando cada vez mais de acordo com o seu *feedback*.

Segundo Adomavicius e Tuzhilin (2005), o problema de recomendação pode ser definido mais formalmente como: Seja U o conjunto de todos os usuários e I o conjunto de todos os itens como filmes, livros, jogos que podem ser recomendados. Tanto a quantidade de usuários quanto a de itens pode ser enorme, alcançando milhões de dados, o que mostra que é necessário uma maneira eficiente para poder trabalhar com esses dados. Seja f a função utilidade que mede a importância do item i para um usuário u, *e.g.* $f : U \times I \rightarrow P$, onde P é um conjunto ordenado. O objetivo é encontrar para cada usuário $u \in U$ um item $i' \in I$ que maximiza a função utilidade:

$$\forall u \in U,\ i'_u = arg\ \max_{i \in I}\ f(u, i). \tag{2.1}$$

Em Sistemas de Recomendação, a função utilidade geralmente é representada por uma nota que indica o quão relevante um item é para um usuário. No exemplo da Figura 2.1 a função utilidade é representada por notas que variam de 1 a 5, e o conjunto vazio indica que o usuário não avaliou o item.

Nesses sistemas,o conjunto de itens que um usuário avalia costuma ser apenas uma pequena fração da base de itens por um todo. A matriz de relacionamento usuário-item costuma ser altamente esparsa, com menos de 1% de seus dados preenchidos(POZZA et al., 2011). Isso é um problema frequente na Filtragem Colaborativa.

	Star Wars	Toy Story	Senhor dos Anéis
Vinicíus	5	4	4
Paulo	3	5	Ø
Filipe	5	3	4
Matheus	Ø	3	2

Figura 2.1: Fragmento de uma matriz de notas de um Sistema de Recomendação de filmes que utiliza o conjunto de preferência como [1,5]

Há duas abordagens mais comuns de se resolver esse problema de esparsidade. A primeira é fazendo uso de heurísticas para definir a função utilidade, a função de utilidade pode ser definida pelo sistema ou até mesmo uma função arbitrária definida pelo usuário. O segundo método é estimando uma função utilidade que otimiza algum critério de desempenho, como exemplo a raiz quadrada da média do erro (RMSE). (ADOMAVICIUS; TUZHILIN, 2005)

Além disso,há diversas formas de se estimar a nota de um item não avaliado por um usuário dentro da área de SR. Existem diversos algoritmos que seguem abordagens diferentes. Em Burke (2007), foi proposta uma classificação em cinco abordagens diferentes, a seguir serão mostradas as três mais importantes para este trabalho:

- **Baseada em conteúdo**: utiliza carateristas do item para selecionar o melhor candidato;
- **Filtragem colaborativa**: busca itens bem avaliados de usuários com gostos parecidos para realização da recomendação;
- **Abordagens híbridas**: combina as abordagens acima.

Esse trabalho é focado na abordagem de Filtragem colaborativa, e a seguir será aprofundado alguns conceitos a respeito dessa estratégia.

2.2 Filtragem Colaborativa

Uma das técnicas mais utilizadas atualmente em SR é a Filtragem Colaborativa (FC), que como dito anteriormente, faz uma recomendação baseada em itens de usuários com o perfil semelhante ao seu. Nas próximas seções, esse método será mais explorado.

Um hábito muito comum das pessoas é pedir a opinião de uma outras antes de tentar consumir algo, seja um item, uma comida, um filme ou um jogo. Nessa premissa se baseia a ideia principal da FC. Essa abordagem busca identificar um conjunto de usuários U que possui um perfil semelhante ao usuário que receberá a recomendação, e recomendar itens que foram bem avaliados por esse conjunto.

Os usuários em um SR possuem objetivos e perfis diversificados. Para as experiências do usuário serem mais atraentes e personalizadas, os SR buscam explorar informações diversas sobre quem está utilizando o serviço. Os SR que utilizam a FC como um método para gerar suas recomendações se baseando em padrões e avaliações de outros usuários com o perfil semelhante, sem ser necessário o uso de informações do usuário que está recebendo a recomendação.

A FC possui diferentes benefícios em relação a abordagem baseada em conteúdo, principalmente o fato de não depender de informações externas de itens e usuários, pois a qualidade da recomendação está relacionada à qualidade das informações, já que a recomendação é construída a partir das informações disponíveis. Além disso, a FC apresenta maior diversidade e dinamicidade para as recomendações para um usuário, considerando que sua recomendação depende de um comportamento de itens e usuários, diferente da recomendação por conteúdo, que a dependência de dados externos causam a super especialização (RICCI; ROKACH; SHAPIRA, 2015).

Porém, há diversos empecilhos em tal abordagem. Bases de um Sistema de Recomendação possuem um vasto número de itens I e usuários U, o que acaba tornando a matriz $U \times I$ de um SR esparsa, o que é um problema, pois a o desempenho dos algoritmos é diretamente relacionado a esparsidade da base. (GIPSON, 2018).

Um problema comum da FC, é o *Cold Start*. Esse problema acontece quando

um novo item ou um novo usuário surge, pois a FC necessita de um conjunto de avaliações para que seja possível a realização de uma predição, assim, se somente as avaliações de usuários forem utilizadas para realizar a predição, não será possível a realização, pois não haverá uma comparação com nenhum outro usuário ou outros itens no sistema, tornando impossível que esse item seja recomendado até que ele receba uma avaliação, ou no caso de um usuário, até ele avaliar um item (RICCI; ROKACH; SHAPIRA, 2015).

Há diversas abordagens que buscam reduzir os efeitos do *Cold Start* e da esparsidade. Para resolver o problema da esparsidade, maior parte delas usa técnicas de redução de dimensões como a decomposição de valores singulares (SVD) ou a análise das principais componentes (PCA) para tratar a esparsidade enquanto provê boas recomendações (BILLSUS; PAZZANI, 1998). Uma das soluções conhecidas para o *Cold Start* é a filtragem baseada em conteúdo, pois caso um item novo seja inserido na base, ele pode ser recomendado para um usuário que tenha gostado de itens semelhantes a ele.

Na literatura os algoritmos de filtragem colaborativa são divididas em duas principais classes: baseados em memória e baseados em modelo (ADOMAVICIUS; TUZHILIN, 2005), que serão mais detalhadas nas próximas subseções.

2.2.1 Algoritmos Baseados em Memória

Os algoritmos que se baseiam em memória fazem a recomendação baseada nos itens já avaliados pelos usuários, ou seja, a avaliação de um item $u_{c,s}$ para um usuário c e um item s é calculada baseada nas recomendações para usuários similares para o mesmo item (ADOMAVICIUS; TUZHILIN, 2005). Em outros casos, a agregação pode ser um média aritmética entre as notas, porém no geral, é mais comum o uso da média ponderada das avaliações.

A similaridade entre dois usuários é, fundamentalmente, a distância entre eles e é usada como peso na realização da média ponderada, então, avaliações dos usuários mais próximos tem um peso maior do que a de usuários que são mais distantes entre si. Um fator importante sobre a média ponderada é que tal técnica não leva em

consideração o fato que diferentes usuários podem utilizar a escala de avaliação de maneiras diferentes. Por exemplo, escalas de 1 a 5 para a avaliação de filmes, pode ocorrer de usuários diferentes que gostaram do filme darem notas diferentes para ele, pois cada um tem maneiras diferentes de medir a escala de notas. A media ponderada ajustada tem colabora para a resolução desse problema, pois desta maneira, ao invés de usar o valor absoluto das avaliações, o desvio padrão da média de avaliações do usuário é utilizado.

Os usuários podem ser separados em níveis de similaridade utilizando estas técnicas, podendo definir, por exemplo, um conjunto de n usuários mais próximos de cada usuário alvo, deixando mais simples o processo de recomendação.

Tal abordagem é semelhante a uma recomendação boca-a-boca. Onde pessoas que estão interessadas em um determinado produto ou serviço (jogo, filme, aparelho, livros, etc.) obtenham recomendações de pessoas que sejam fontes confiáveis e que já tenham consumido o mesmo. Com tal opiniões, o interessado pode ponderar com relação à confiança dela pela a origem da recomendação, e sua expectativa sobre o serviço ou produto.

Muitas estratégias podem ser utilizadas no calculo da similaridade entre dois usuários ou itens nos SR que utilizam FC. Na maior parte dos casos, a semelhança entre dois usuários é baseada em itens que foram avaliados pelos dois. As aproximações mais populares são correlação (*correlation*) e a baseada em cosseno (*cossine-based*) (ADOMAVICIUS; TUZHILIN, 2005).

A similaridade baseada em correlação, entre dois usuários u e v ou entre dois itens i e j, é medida com o calculo da correlação de Pearson ou outras similaridades baseadas em correlações. A correlação de Pearson mede a correlação estatística (*Pearson's r*) entre duas variáveis linearmente dependentes (RESNICK; BERGSTROM; RIEDL, 1994).

A fórmula a seguir da a correlação de Pearson entre os usuários u e v:

$$w_{u,v} = \frac{\sum_{i \in I}(r_{u,i} - \overline{r_u})(r_{v,i} - \overline{r_v})}{\sqrt{\sum_{i \in I}(r_{u,i} - \overline{r_i})^2 (r_{v,i} - \overline{r_v})^2}} \tag{2.2}$$

onde o somatório de $i \in I$ é referente aos itens que ambos os usuários u e v avaliaram e r_u é a avaliação média dos itens co-avaliados do usuário u.

Para os itens, a fórmula é alterada da seguinte maneira:

$$w_{i,j} = \frac{\sum_{u \in U}(r_{u,i} - \overline{r_i})(r_{v,j} - \overline{r_j})}{\sqrt{\sum_{u \in U}(r_{u,i} - \overline{r_i})^2 (r_{u,j} - \overline{r_j})^2}} \tag{2.3}$$

tal que o conjunto $u \in U$ representa os usuários que avaliaram tanto o item i quanto o item j, $r_{u,i}$ é avaliação do usuário u a um item i e $\overline{r_i}$ é a media de avaliações do item i por outros usuários. como apresentado na Figura 2.2.

Algoritmos que utilizam a correlação de Pearson como base compõe uma parte representativa dos algoritmos de FC e são amplamente usados em pesquisas nessa área (SU; KHOSHGOFTAAR, 2009).

Já na similaridade baseada no cosseno, dois usuários, u e v, são considerados vetores (x_u,x_v) em um espaço m-dimensional. Então, a similaridade entre os dois vetores é dada pelo cosseno do ângulo formado entre eles (ADOMAVICIUS; TUZHILIN, 2005).

$$cos(x_u, x_v) = \frac{x'_u x_v}{||x_u||||x_v||} \tag{2.4}$$

Tal valor tem a possibilidade de ser utilizado para expressar a similaridade entre dois usuários, ou dois itens, tal que o espaço onde é formado pelas notas co-avaliadas entre dois objetos x_u e x_v, ou seja, $S_{u,v} = \{\ s \in S \mid r_{u,s} \neq \oslash\ \&\ r_{v,s} \neq \oslash\}$, tal que $S_{u,v}$ é o conjunto de intercepção entre os vetores e $sim(u, v)$ varia de [-1,1]. Como exemplo, os vetores $\vec{A} = \{x_1, x_2\}$ e $\vec{B} = \{x_2, y_2\}$, o vetor de similaridade do cosseno será:

$$cos(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}|| * ||\vec{B}||} = \frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2}\sqrt{x_2^2 + y_2^2}} \tag{2.5}$$

Em tal aproximação, a situação na qual usuários diferentes usam a escala de avaliação de maneira diferente não é tratada. Para esse valor ser ajustado, a média

de notas dos usuários é subtraída dos pares de itens co-avaliados. A similaridade baseada no cosseno ajustada possuí a mesma fórmula que a correlação de Pearson. A correlação de Pearson executa a similaridade baseada no cosseno como uma forma de normalização as notas do usuário de acordo com seu comportamento (ADOMAVICIUS; TUZHILIN, 2005).

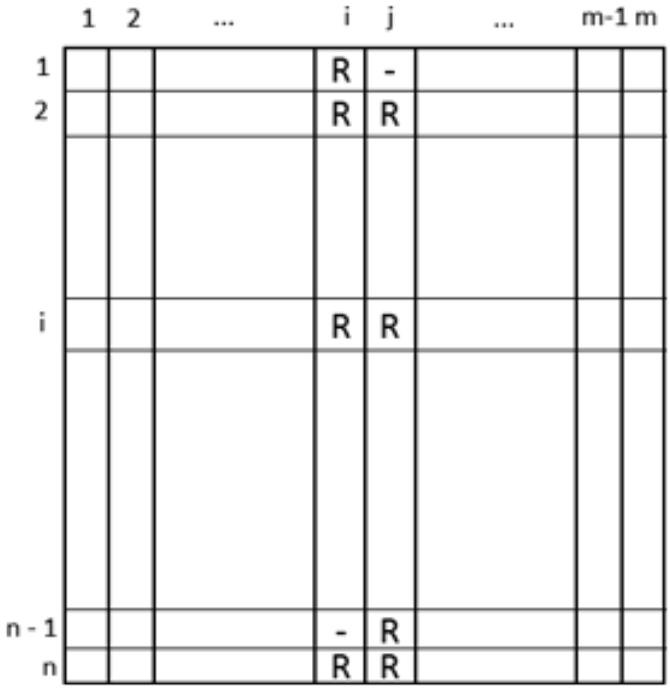

Figura 2.2: Similaridade entre os itens ($w_{i,j}$), que é calculada baseada nos itens avaliados pelos usuários 2, l e n.

2.2.2 Algoritmos Baseados em Modelo

Os algoritmos baseados em memória utilizam diretamente notas armazenadas no sistema para a realização de predições de uma nota de um item i para um usuário u, já os algoritmos baseados em modelo fazem uso de um modelo matemático para realizar tal previsão. Nesta categoria, os algoritmos realizam uma aproximação probabilística enquanto supervisionam o processo calculando o valor esperado de uma nota, baseado nas notas reais deste usuário para outros itens (ADOMAVICIUS; TUZHILIN, 2005).

Para a construção de modelos que buscam reconhecer padrões complexos baseados em dados de treino para a realização de predições inteligentes para os dados reais do sistemas de recomendação, são utilizadas técnicas de aprendizado de máquina e mineração (SU; KHOSHGOFTAAR, 2009). Normalmente, os algoritmos de classificação são utilizados como modelos de FC caso as notas não forem categorizadas, ou, SVD e modelos de regressão para notas numéricas.

Um modelo probabilístico é proposto para calcular as notas não explicitadas pelo usuários em Breese (1998).

$$r'_{u,i} = E(u,i) = \sum_{r=0}^{n} r \times Pr(r'_{u,i} = r | r'_{u,i'}, s' \in S_c) \tag{2.6}$$

2.2.3 Fatoração de Matrizes

Como dito em Science et al. (2015), os métodos mais bem sucedidos para modelos de fatores latentes são modelos baseados em *Fatoração de Matrizes* (FM) ou *Matrix Factorization*. Na sua forma básica, a FM caracteriza ambos os itens e usuários em vetores de fatores inferidos pelo padrão de avaliação dos itens. Essa correspondência entre fatores de usuários e fatores de itens leva à recomendação. Tais métodos vem se popularizando, graças a combinação de uma boa escalabilidade uma predição precisa. Elas oferecem bastante flexibilidade para modelagem de varias aplicações na realidade.

Um modelo de fatoração de matrizes mapeia item e usuário como um vetor de espaço latente de dimensão f, onde interações item-usuário são modeladas como produtos internos nesse espaço. Cada item i está associado a um vetor q_i, e cada usuário u está associado a um vetor p_u. Para dado elemento de i, elementos de q_i podem possuir estes fatores positivos ou negativos. Para um usuário u os elementos de p_u medem a extensão do interesse que esse usuário tem em um alto valor para esses fatores, tanto positivos quanto negativos. O produto resultante $q_i^T p_u$, mostra a interação entre o usuário u e o item i, o interesse do usuário nas características de tal item. Isso aproxima a nota do usuário u ao item i, que é denotada por r_{ui},

levando a uma estimativa dada por:

$$u_i = q_i^T p_u \tag{2.7}$$

A aproximação tomada pela FM é considerada a mais precisa aproximação para reduzir o problema de altos níveis de esparsidade em dados de sistemas de recomendação.

2.2.4 Redes Neurais

O cérebro tem a capacidade de organizar seus neurônios para realizar tipos de processamento de maneira mais rápida e eficiente do que um computador convencional (HAYKIN, 2001).

Em Selli e Jr. (2007), Redes Neurais (RN) são definidas como um mapeamento não linear de um vetor de entrada para um vetor de saída, que é feito através de camadas compostas por funções de ativação ou neurônios, onde as coordenadas de entradas são calculadas a partir da soma de seus valores com seus pesos e *bias*, a fim de produzir uma saída.

Em McCulloch (1943), é proposto um modelo matemático do fluxo de sinais em uma rede de neurônios. Este foi o primeiro modelo de neurônio artificial que ficou conhecido. Em 1959 Bernard Widrow e Marcian Hoff propuseram modelos de neurônios que foram chamados de Adalaine, um modelo desenvolvido para reconhecer padrões binários, podendo prever o próximo bit em uma leitura de linha telefônica, baseada nas leituras anteriores(YE; SUN; YANG, 2013).O modelo de um neurônio é apresentado na Figura 2.3

Para HAYKIN (2001) a Figura 2.3 representa um modelo não linear de um neurônio, e este possui três elementos básicos, um conjunto de *sinápses*, onde cada uma possui um peso. Uma entrada x_n na sinápse i conectada ao neurônio k é multiplicada pelo seu peso sináptico w_{ni}.Um somador, que soma os todos os sinais

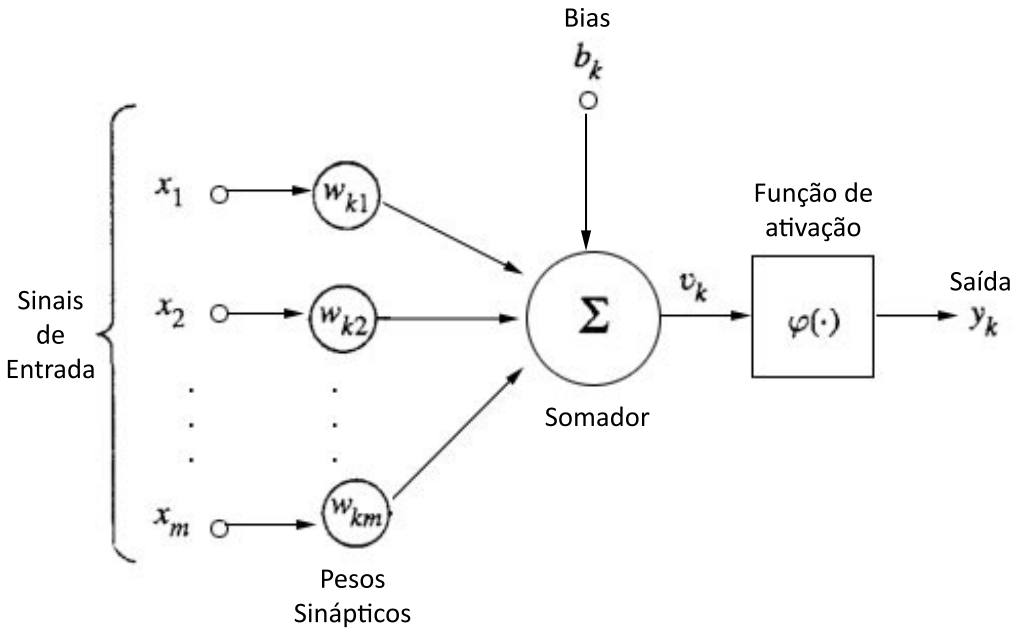

Figura 2.3: Modelo não linear de um neurônio
Fonte: baseado em HAYKIN (2001)

de entrada, ponderado pelos seus pesos e uma função de ativação que restringe a amplitude do sinal de saída do neurônio.Além disso também há o valor de *bias*, b_k, que aumenta ou diminui a entrada da função de ativação.

Como a capacidade computacional de um neurônio é limitada, usa-se um conjunto de neurônios artificiais ligados entre si, formando uma rede, essa rede é consegue resolver problemas de complexidade mais alta do que um neurônio individual conseguiria. A estrutura mais utilizada de RN é a estrutura de duas camadas, camada escondida e de saída, pois pode se aproximar de funções continuas e possui um alto poder computacional (BRAIDA, 2013).

De acordo com Zheng (2015) é provado que RN são capazes de aproximar qualquer função contínua, e recentemente, vem se encontrando efetivas em diversas áreas, como processamento de texto e reconhecimento de fala. Entretanto, há relativamente

poucos trabalhos empregando RN em sistemas de recomendação, em contraste à Fatoração de Matrizes que existe uma vasta quantidade de literatura sobre esses métodos.Porém, alguns trabalhos como Oord, Dieleman e Schrauwen (2013) e Wang e Wang (2015) conseguiram aplicar Redes Neurais para tarefas de recomendação e mostraram resultados promissores (ZHANG; CHUA, 2017).

Capítulo 3

NeuralCF.jl

Este capitulo falará sobre o desenvolvimento do trabalho, apresentando sua motivação, algoritmos implementados e a proposta de implementação que foi dada. Além disso, será apresentado o processo de desenvolvimento do trabalho, até a sua conclusão.

3.1 Motivação

Em Zhang et al. (2018), é dito que os SR tem um grande impacto na industria atual, pois são fatores importantes para melhorar a experiência do usuário. Eles contribuem no aumento do número de vendas e serviços prestados por diversos sites e aplicações *mobile*, entre outros. Cerca 80% dos filmes assistidos no Netflix[1] vem de recomendação, 60% dos cliques em vídeos do *YouTube*[2] são de recomendações da página principal(CHHABRA, 2017).

Outro fator que vem impulsionando a área de SR gradativamente é o aumento do poder computacional das CPUS, e mais recentemente das GPUS. O uso de redes neurais ficou mais relevante e possibilitou o desenvolvimento de redes mais profundas, dando origem ao *Deep Learning* (NASCIMENTO,).

[1] https://www.netflix.com/
[2] https://www.youtube.com/

Recentemente, o Deep Learning (DL) vem revolucionando o campo dos Sistemas de Recomendação, trazendo diversas possibilidades para melhorar a performance do recomendador. Diversas empresas estão adotando o DL para aperfeiçoar seus métodos de recomendação, e além disso, estudos demonstram que algoritmos de previsão baseados nessas técnicas vem ultrapassando os obstáculos de modelos convencionais, realizando indicações de maior qualidade. (ZHANG et al., 2018)

Entretanto, muitos destes trabalhos não possuem implementações disponíveis, o que dificulta o uso para grande parte da comunidade, dado que existem poucos *frameworks* que garantem tal acessibilidade ao público. Um deles é o Persa[3], que foi criado para o estudo de Filtragem Colaborativa na linguagem *Julia*[4], e oferece ferramentas que facilitam a construção de novos algoritmos que usam linguagens de computação cientifica, ajudando no seu desenvolvimento (XAVIER, 2018).

Em Xavier (2018), é mostrado diversos *frameworks* que surgem ao longo do tempo, implementando os mais clássicos algoritmos de Sistemas de Recomendação. Um dos exemplos citados é o *NumPy*[5], que é muito utilizado pela comunidade para estudo cientifico. Porém, *Python*[6] é uma linguagem de alto nível, o que faz com que partes que necessitam de desempenho tenham que ser reescritas em linguagens de nível mais baixo, como C++ e C.

Uma solução para isso seria a utilização de *Domanin Specific Languages* (DSL) que são linguagens criadas para atender as necessidades de um domínio especifico de aplicações (FOWLER, 2010). Como por exemplo, uma linguagem que possua matrizes de avaliações, onde especificando parâmetros como, usuário, item e contexto seja possível obter a nota do usuário para o contexto, ao invés do dado na matriz (XAVIER, 2018).

Há inúmeras vantagens ao se utilizar uma DSL, como alto nível de abstração, aumento de produtividade. legibilidade, concisão e organização do código. No entanto, também há desvantagens no seu uso, como o aumento de indireções devido a

[3]https://github.com/JuliaRecsys/Persa.jl
[4]https://julialang.org/
[5]http://www.numpy.org/
[6]https://www.python.org/

novos níveis de abstração, falta de suporte de ferramentas para seu desenvolvimento e manutenção, e a curva de aprendizada para uma linguagem aplicada somente no domínio do problema (FOWLER, 2010).

3.2 Proposta

A proposta deste trabalho é criar um pacote que enriquece o ecossistema de módulos do Persa. O objetivo do *NeuralCF* é reunir um conjunto algoritmos de Filtragem Colaborativa que fazem uso de modelos neurais e possibilitem o fácil acesso e utilização do público, colaborando para estudos e o desenvolvimento cientifico para a comunidade. Nesta sessão serão apresentados os algoritmos implementados no pacote de maneira mais detalhada.

3.2.1 Persa

Persa é um *framework* escrito na linguagem Julia criado para auxiliar no estudo de Sistemas de Recomendação, trazendo ferramentas que colaboram na construção de algoritmos. Aumenta a produtividade do desenvolvedor,pois não precisa se preocupar com as tarefas básicas, que já são implementadas pelo pacote.

O Persa estende a linguagem Julia, construindo uma linguagem de domínio específico (DSL). Isso influência diretamente na velocidade do desenvolvimento de novos SR, visto que o *framework* tira do desenvolvedor a preocupação com coisas básicas como o acesso aos dados.

Existem alguns outros projetos disponíveis para a comunidade que já estendem o Persa e o tornam ainda mais rico. O *DatasetsCF* possui alguns *datasets* já disponíveis para uso que se integram com o Persa. Outro projeto é o *EvaluationCF* que implementa diversas técnicas como o *Mean Absolute Error* (MAE) para avaliar a qualidade e ajudar a treinar os SR desenvolvidos.

3.2.2 Modelos Neurais

Dentro das diversas técnicas que os SR podem utilizar, esse projeto tem o foco em implementar as que fazem uso de Redes Neurais. A seguir, serão apresentados os algoritmos implementados inicialmente no pacote e algumas de suas características. Esses algoritmos são descritos em Zhang e Chua (2017), e foram adaptados para serem utilizados em Julia, aproveitando a DSL disponível pelo Persa. Logo abaixo, falaremos com mais detalhes sobre eles.

Dentro do campo de SR, o modelo de fatoração de matrizes é o mais popular algoritmo para Filtragem Colaborativa, e é bastante estudado na literatura. O *Generalize Matrix Factorization* (GMF) é uma variação da fatoração de matrizes mais generalizada, implementada no *framework Neural Collaborative Filtering* (NCF), que permite mudanças na fatoração de matrizes padrão. (ZHANG; CHUA, 2017) . Em Zhang e Chua (2017), os itens e usuários são representados em dois vetores diferentes, Entretanto, essa aproximação não leva em consideração interações entre as características latentes dos usuários e dos itens, o que não é bom para a Filtragem Colaborativa. Para lidar com este problema, é proposto a adição de camadas escondidas no vetor concatenado, usando o *Multi-Layer Perceptron* (MLP) para aprender sobre a interação entre essas características latentes. (ZHANG; CHUA, 2017) Essa estrutura pode ser vista na Figura 3.1.

Geralmente, algoritmos de fatoração de matrizes consideram que existam "números mágicos"que explicam a relação usuário x item. Em Zhang e Chua (2017), é proposta uma função que busca maximizar e encontrar esses números. Ele faz com que as variáveis latentes sejam modeladas na camada da rede neural e o procedimento de aprendizado aprenda esses valores. A cada camada percorrida pelo modelo,a as variáveis são concatenadas, em busca de encontrar a melhor combinação.

O *Neural Matrix Factorization* (NeuMF) é uma junção do MLP e do GMF, onde os dois dividem a mesma camada de *embedding*, e combinam os resultados das suas funções de interação. Um dos problemas desse método é o limite de performance do modelo em conjunto, por exemplo, eles teriam que usar uma camada de *embedding*

do mesmo tamanho, onde os tamanhos ótimos para cada método são diferentes. Isso pode prejudicar na otimização dos métodos, porém nas próximas seções será apresentada uma solução para esse problema. Uma sugestão do autor é a utilização de pré-treinos para os modelos de GMF e MLP. (ZHANG; CHUA, 2017)

Collaborative Filtering to Supervised Learning (COFILS) é uma metodologia em três etapas: O pré-processamento, a extração das variáveis latentes e a regressão/classificação. A primeira etapa é encarregada da representação dos dados, corrigindo-a para a melhor forma de representação do problema, devido aos valores desconhecidos. Na segunda etapa são utilizadas diversas técnicas de decomposição de valores singulares (SVD) para realizar a extração das variáveis latentes. Por ultimo, é aplicado um algoritmo de aprendizado supervisionado.(BRAIDA, 2013) Este será implementado em trabalhos futuros.

Inicialmente, são implementados os três algoritmos apresentados acima, mas a ideia é que o pacote seja aberto ao público, permitindo que a comunidade colabore com implementações futuras, dê feedbacks e utilize o pacote para seus projetos. Nas subseções seguintes, entraremos em mais detalhes sobre os algoritmos citados.

3.2.2.1 GMF

A fatoração de matrizes, como dito anteriormente, é a técnica mais popular de Filtragem Colaborativa e em Zhang e Chua (2017), é mostrado que o *framework* NCF pode interpreta-lo como um caso especial. Devido à codificação do usuário-item ID da camada de entrada, o vetor de embedding pode ser visto como o vetor latente de usuário-item. Seja o vetor latente de usuário p_u ser $P^T V_u^U$ e o vetor latente de item q_i ser $Q^T v_i^I$, define-se a função de mapeamento da primeira camada neural de Filtragem Colaborativa como

$$_1(p_u, q_i) = p_u \odot q_i \tag{3.1}$$

onde $\odot$ é o produto de cada elemento dos vetores, então o vetor da camada de saída é representado por:

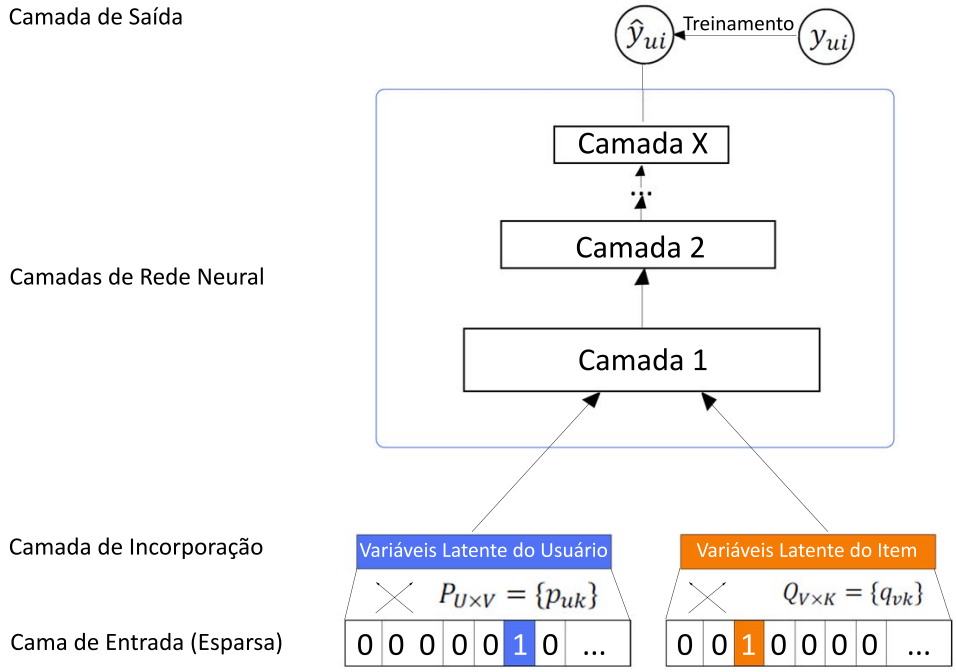

Figura 3.1: Framework do NCF.

$$\hat{y} = a_{out}(h^T(p_u \odot q_i)) \tag{3.2}$$

onde a_{out} e h denotam a função de ativação e o peso das arestas da camada de saída, respectivamente. Então usando uma função identidade para a_{out} e um vetor uniforme de 1 para h, pode-se recuperar o modelo de fatoração de matrizes.

Dentro do NCF, a fatoração de matrizes pode ser generalizada e estendida, possibilitando variações dos parâmetros como h e a_{out} para modificar o modelo. No NCF foi usado a função sigmoide

$$\sigma(x) = 1/(1 + e^{-x}) \tag{3.3}$$

como a_{out}, enquanto h aprende de dados com *log loss*, que é uma função de classificação que faz com que a precisão do classificador seja maximizada. Essa variação da fatoração de matrizes é chamada de GMF.

Além do que foi dito na subseção 3.2.2 sobre o MLP, pode-se adicionar ao modelo um alto grau de flexibilidade e não linearidade para aprender interações entre os vetores latentes de usuários e itens, p_u e q_i respectivamente, diferentemente do GMF, que usa somente o produto entre cada elemento do vetor, o MLP no *framework* NFC é definido como:

$$z_1 = \phi_1(p_u, q_i) = \begin{bmatrix} p_u \\ qi \end{bmatrix},$$

$$\phi_2(z_1) = a_2(W_2^T z_1 + b_2),$$

$$...$$

$$\phi_L(z_{L-1}) = a_L(W_L^T z_{L-1} + b_L),$$

$$\hat{y}_{ui} = \sigma(h^T \phi_L(z_{L-1})),$$

sendo W_n o peso da matriz, b_n o vetor de *bias* e a_n a função de ativação para a n-ésima camada do perceptron. Existem inúmeras funções de ativação disponíveis na literatura que podem ser usadas no MLP, como a função sigmoide, tangente hiperbólica e a função retificadora (ReLU). Cada uma delas tem um tipo de efeito no modelo. Sigmoide, por exemplo, restringe cada neurônio a pertencer no intervalo (0,1), o que pode causar saturação, quando os neurônios param de aprender seus resultados são próximos de 0 ou 1, limitando a performance do modelo. Já a função tangente é uma escolha melhor, e usada por muitos (CHUA, 2015), porém, ela só ameniza os problemas da função sigmoide, pois é uma versão redimensionada dela. Então, é escolhido usar a ReLU no NFC, pois é mais plausível biologicamente e é provada ser não saturada (GLOROT; BORDES, 2011), encorajando ativações esparsas, sendo adequada para dados esparsos e fazendo o modelo ser ter menos chances de sofrer sobreajuste (*overfitting*). Resultados mostram que ReLU obteve resultados melhores que os outros métodos (ZHANG; CHUA, 2017).

Para a estrutura da rede, a forma mais comum é seguir o modelo pirâmide, onde a camada da base é a maior, e cada camada seguinte é menor e com menos neurônios. A premissa é de que usando um menor de unidades ocultas, para as camadas mais altas, elas podem aprender mais características abstratas dos dados(HE, 2015).

3.2.2.2 NeuMF

Em Zhang e Chua (2017) é proposto um modelo que une o GMF com o MLP, o NeuMF. O NeuMF, como dito em 3.2.4, é uma fusão do GMF e do MLP onde ambos dividem a mesma camada de embedding, combinando a saída das iterações das suas funções.

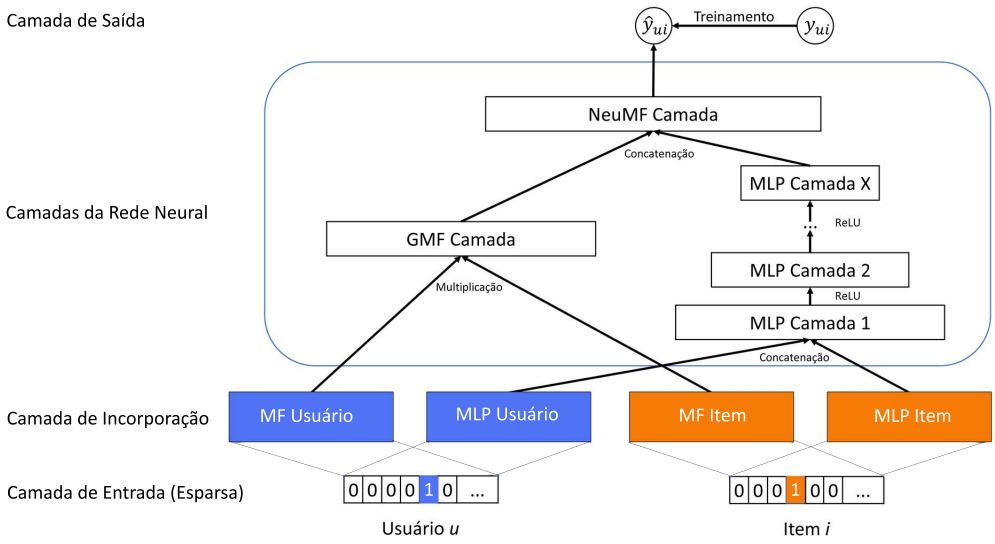

Figura 3.2: Modelo do NeuMF

O modelo que combina o GMF com uma camada do MLP é formulado como

$$\hat{y}_{ui} = \sigma(h^T a(p_u \odot q_i + W \begin{bmatrix} p_u \\ qi \end{bmatrix}, +b)). \tag{3.4}$$

Como foi descrito anteriormente, existem problemas com o compartilhamento da camada de *embedding*, que pode causar uma limitação de performance, e como sugerido, uma das maneiras de contornar esse problema é deixando o GMF e o MLP aprenderem de forma separada, e depois combinar os dois modelos com a

concatenação das ultimas camadas ocultas. A formulação do NeuMF é dada como:

$$\phi^{GMF} = p_u^G \odot q_i^G,$$

$$\phi^{MLP} = a_L(W_L^T(a_{L-1}(...a2(W_2^T \begin{bmatrix} p_u^M \\ q_i^M \end{bmatrix} + b_2)...)) + b_L),$$

$$\hat{y_{ui}} = \sigma(h^T) \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix},$$

onde p_u^G e p_u^M são respectivamente o *embedding* do usuário para o GMF e o MLP, enquanto q_i^G e q_i^M são os mesmos para os itens. Para a função de ativação do MLP, é usado o ReLU, pois como dito, obtém melhores resultados.

O NeuMF possui uma função objetivo não convexa, o que faz com que os métodos de otimização baseados em gradientes só encontrem soluções no ótimo local. A inicialização é uma parte importante para a convergência e performance em modelos de *deep learning* (ERHAN; COURVILLE; VINCENT, 2010). Como o NeuMF é uma agregação do MLP e do GMF, é proposto que para a sua inicialização se use modelos pré-treinados de ambos.

$$\mathbf{h} \leftarrow \begin{bmatrix} \alpha \mathbf{h}^{GMF} \\ (1-\alpha)\mathbf{h}^{MLP} \end{bmatrix}, \tag{3.5}$$

sendo $\mathbf{h}^{GMF}$ e $\mathbf{h}^{MLP}$ sendo $\mathbf{h}$ o vetor pré-treinado dos modelos GMF e MLP, respectivamente, α é um hiper parâmetro que determina o trade-off entre os modelos após o pré-treino. Primeiramente, ambos os modelos são treinados com inicializações aleatórias até a convergência. Então, os parâmetros do modelo são usados como seus respectivos parâmetros correspondentes no NeuMF.

Para treinar o GMF e o MLP do zero, é adotado o *Adaptive Moment Estimation* (Adam) (KINGMA; BA, 2015), que é uma adaptação da taxa de aprendizado para cada parâmetro, onde são realizados pequenas atualizações parar parâmetros frequentes e maiores atualizações para os menos frequentes.

O método Adam gera a convergência mais rapidamente para ambos os modelos, em comparação com o *Stochastic Gradient Descent* (SGD). Após inserir os parâmetros do pré-treino no NeuMF, o SGD é usado para realizar as otimizações ao invés de continuar com o Adam. Isso acontence porque no Adam é necessário salvar informações para atualizar os parâmetros corretamente, e na inicialização do NeuMF com os modelos pré-treinados, essa informação não é salva (ZHANG; CHUA, 2017).

Capítulo 4

Projeto

Com o objetivo de de facilitar o estudo de Sistemas de Recomendação e reduzir o tempo de desenvolvimento de modelos preditivos, foi criado um modulo com modelos neurais facilmente adaptáveis e extensíveis. Esse *framework* é baseado no ecossistema do Persa, por isso para tirar o máximo de proveito da biblioteca, alguns padrões de arquitetura foram seguidos.

Nas próximas seções mostraremos os modelos que foram utilizados, como foram modelados, além de algumas decisões de projeto tomadas e alguns exemplos de uso dos mesmos.

4.1 Persa

Como já foi dito anteriormente, o Persa contém ferramentas que auxiliam no desenvolvimento de técnicas de Filtragem Colaborativa. A seguir iremos mostrar alguns facilitadores que a DSL nos entrega. Um deles é a representação das possíveis avaliações de um usuário para um item no domínio do SR, como pode ser visto no Código 4.1.

```
julia> Persa.Preference([1, 2, 3, 4, 5])
Ratings Preference: [1, 2, 3, 4, 5]
```

Código 4.1: Definição das possíveis notas do SR no Persa

Além disso, ele também possui uma estrutura que permite representar a avaliação de um usuário sobre um item, como pode ser visto no Código 4.2.

```
julia> Persa.UserPreference(1, 1, Persa.Rating(1, preference))
(user: 1, item: 1, rating: 1)
```

Código 4.2: Definição da avaliação de um usuário sobre um item do SR no Persa

O maior destaque do Persa em relação ao acesso a dados é a sua estrutura chamada de *Dataset*. Esse objeto é composto por um vetor de avaliações de usuários, e possíveis notas, onde cada elemento pode ser representado como mostrado no Código 4.2. A biblioteca já define uma indexação de usuários e itens, o que torna toda a sintaxe mais semântica e de fácil compreensão. No Código 4.3, é apresentado uma definição de um *dataset* para SR no Persa.

```
julia> dataset = Persa.Dataset(data, 4, 3, preference)
Collaborative Filtering Dataset
- # users: 4
- # items: 3
- # ratings: 9
- Ratings Preference: [1, 2, 3, 4, 5]
```

Código 4.3: Definição de um dataset para o SR no Persa

O Código 4.4 mostra como é representado uma indexação usuário x item no Persa.

```
julia> dataset[1,1]
Rating: 4
```

Código 4.4: Indexação do dataset por usuário x item para o SR no Persa

Outro ponto forte do *framework* é a sua interface de modelos preditivos que interpreta as previsões como um acesso a uma matriz usuário x item. Dessa forma,

o acesso aos dados se torna padronizado. Para criar um novo modelo preditivo é necessário estender a estrutura chamada por *Model* na biblioteca. O Código 4.5 mostra a criação de um modelo preditivo.

```
mutable struct RandomModel <: Persa.Model
    preference::Persa.Preference
    users::Int
    items::Int
end
```

Código 4.5: Definição de um modelo previsão de notas aleatórias no Persa

Além disso, também é possível definir como o modelo irá ser treinado e prever os dados, como visto no Código 4.6.

```
Persa.predict(model::RandomModel, user::Int, item::Int) =
    rand(model.preference.possibles)
```

Código 4.6: Definição da previsão de notas um modelo preditivo aleatório no Persa

Sua arquitetura possui métodos já implementados que permitem entender o modelo como uma matriz de usuário por item, como se fosse um objeto do tipo *Dataset*. Um exemplo disso pode ser visto no Código 4.7, onde um modelo preditivo faz a previsão uma avaliação. Caso essa nota esteja fora do intervalo de valores aceitos no domínio do *dataset*, as ferramentas do Persa já se encarregam de corrigi-la e retornar a previsão correta.

Outra abstração importante do *framework* é sua estrutura de *Rating*, que ao tentar acessar uma avaliação de um usuário para um item que ainda não foi expressa, o objeto *dataset* retorna uma estrutura chamada *MissingRating* que funciona como um objeto nulo nesse ecossistema. Ao utilizar a interface de modelos preditivos para o acesso de dados, esse comportamento se altera um pouco, o sistema passa a retornar uma estrutura do tipo *PredictRating* que contém o valor previsto, e o valor corrigido.

```
julia> model = RandomModel(ds)
RandomModel(Ratings Preference: [1, 2, 3, 4, 5], 4, 3)


julia> model[1,1]
Rating: 2 (2)
```

Código 4.7: Previsão de notas de um modelo preditivo aleatório no Persa

4.2 NeuralCF.jl

Hoje existem diversos *frameworks* de machine learning disponíveis para a comunidade, dentre eles: *Tensorflow*, *Pytorch*, *Keras* e o *Flux*. Este ultimo escrito totalmente em Julia, e os demais escritos em Python.

O *Keras* é um framework que permite a criação de modelos neurais personalizados, onde é possível decidir detalhes como: o número de camadas ocultas, a função de ativação, e até mesmo como os pesos de suas camadas serão inicializados. Além disso ele possui disponível em sua biblioteca diversas métricas de avaliação e algoritmos que fazem a correção dos pesos nos nós da rede como o *Stochastic Gradient Descent* (SGD) e o ADAM. A biblioteca possui diversas funções de alto nível, comparado as demais bibliotecas da área, que atendem nosso problema e facilita o desenvolvimento deste projeto. O *Keras* já é um *framework* bem consolidado na área, com uma comunidade ativa e uma documentação rica. Por esses motivos, foi escolhido o Keras para auxiliar no desenvolvimento do projeto.

Toda a interface da nossa biblioteca foi implementada utilizando a linguagem Julia, com o objetivo de estender e utilizar a DSL do Persa. No entanto, o projeto também se beneficia das funções disponibilizadas pelo *Keras*, e implementa seus algoritmos de recomendação na linguagem Python para poder usar essa estrutura. Dessa forma, o nosso *framework* cria uma ponte entre essas duas linguagens, a Figura 4.1 mostra um esquema simplificado do funcionamento dessa arquitetura.

Para usufruir de abstrações de projetos já disponíveis na comunidade, como é o caso do Persa e da biblioteca de redes neurais *Keras* escrita em *Python*, alguns

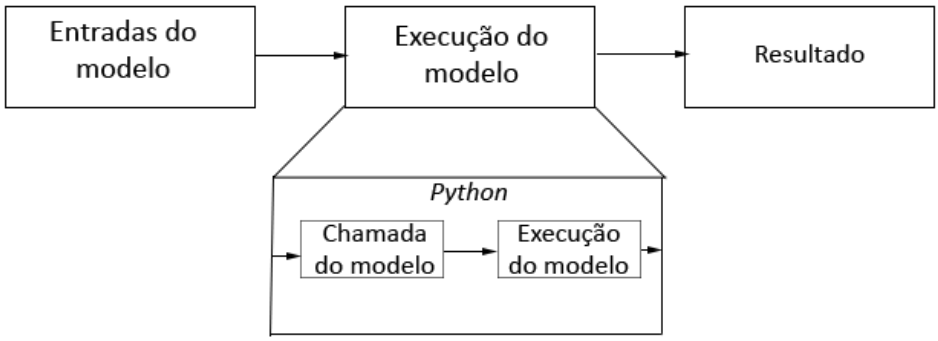

Figura 4.1: Estrutura de execução do processo realizado pelo pacote

padrões tiveram que ser seguidos. A seguir mostraremos como essas decisões impactaram na nossa modelagem da arquitetura dos nossos modelos preditivos e em sua implementação.

4.2.1 Modelos preditivos

Nas seções anteriores foram mostrados alguns dos facilitadores que o Persa entrega para seus usuários. Sua extensibilidade nos permite criar nosso modelo baseado nessa interface e assim utilizar todas essas ferramentas que o *framework* nos proporciona.

Este trabalho cria uma estrutura abstrata chamada *NeuralModel* que possui um relacionamento com a estrutura *Model* do Persa. Dessa forma nossos modelos se tornam compatíveis com todas as funções já implementadas pelo *framework*, como visto no Código 4.8

```
abstract type NeuralModel{T} <: Persa.Model{T} end

mutable struct NMFRegressor{T} <: NeuralModel{T}
    object::PyObject
    preference::Persa.Preference{T}
    users::Int
    items::Int
end
```

Código 4.8: Estrutura dos modelos neurais no *NeuralCF.jl*

Além disso, a linguagem Júlia nos permite fazer sobrecarga de métodos. Com isso podemos adaptar todas as funções que o Persa já implementa que não atendem nosso domínio e ainda manter o ecossistema do *framework* funcionando perfeitamente.

4.2.2 Implementação dos algoritmos preditivos

Uma das limitações em utilizar o *Keras* é o fato dele ser escrito em Python. Felizmente, a comunidade da linguagem Julia criou uma forma de romper esse obstáculo. Para isso foi utilizado o pacote PyCall que nos permite criar trechos de código em Python como mostra o Código 4.9 e chama-los diretamente através da linguagem Julia através do prefixo *py*, como visto pode ser visto no Código 4.10.

```
class MLPRegressor(Model):
  def __init__(self, users, items, outputs,
  factors = 10, nhiddens = [128, 128], dropout = 0.1):
        user_input = Input(shape=(1,), dtype='int32')
        item_input = Input(shape=(1,), dtype='int32')

        MLP_Embedding_User = Embedding(input_dim = users + 1,
         output_dim = factors, input_length=1)
        MLP_Embedding_Item = Embedding(input_dim = items + 1,
         output_dim = factors, input_length=1)
                              .
                              .
                              .
        last_hidden = hidden_layers(mlp_vector)
        baseline_output = Dense(1)(last_hidden)

        super(MLPRegressor, self)
         .__init__(inputs=[user_input, item_input],
         outputs=baseline_output)
```

Código 4.9: Definição de modelo neural usando o Keras

```
function MLPRegressor(
 dataset::Persa.Dataset, features::Int;
 nhiddens::Array{Int, 1} = [128, 128],
 dropout::Real = 0.5)

  preference = length(dataset.preference.possibles)
  model = py"MLPRegressor"(dataset.users,
   dataset.items,
   preference,
   factors = features,
   nhiddens = nhiddens,
   dropout = dropout)

  return MLPRegressor(model,
   dataset.preference,
   Persa.users(dataset),
   Persa.items(dataset))
end
```

Código 4.10: Exemplo de chamada do PyCall

4.3 Experimentos

Nesta seção serão apresentados a metodologia, e em seguida alguns detalhes da nossa base de dados utilizada. Será detalhada a organização dos experimentos, como foram executados e o seu resultado.

4.3.1 Metodologia

A metodologia da avaliação experimental da proposta será composta em duas etapas: a busca por hiperparâmetros do modelo neural para encontrar os melhores parâmetros, e a etapa de regressão/classificação. Nesta segunda etapa será feita uma

comparação entre alguns algoritmos de Filtragem Colaborativa baseados em modelos disponíveis no pacote *ModelBasedCF* que também faz parte do ecossistema do Persa.

4.3.2 Base de Dados

O conjunto conjunto de dados que foi utilizado para a realização dos experimentos deste projeto foi o *MovieLens*(MILLER et al., 2003).O *GroupLens*[1] desenvolveu um Sistema de Recomendação de filmes, que foi de onde veio essa base. Nela existem 100.000 avaliações, 943 usuários e 1682 filmes, seu conjunto de preferência é um número inteiro, que varia entre um a cinco. Fora as avaliações, também existem informações referente aos usuários, como sexo e idade, além das informações sobre os filmes, como titulo e data de lançamento. Entretanto, essas informações não foram utilizadas nos experimentos do projeto, pois sua utilização descaracterizaria a proposta dos algoritmos de filtragem colaborativa.

Ela possui poucas avaliações quando comparado com todas as possibilidades. O seu índice de esparcidade é de 6.30%, a distribuição de notas possui um comportamento de uma normal com média de 3.52 e desvio padrão de 1.12, mostrado na Figura 4.2.

Um usuário possui em média 106.04 avaliações, onde a avaliação mínima é de 20 avaliações e a máxima de 737. A característica de possuir um valor alto para o mínimo de avaliações por usuário é por causa das características do *MovieLens*, pois no momento de criação da conta, o usuário é obrigado a fazer essa quantidade mínima de avaliações.

Para os filmes, a média de quantidade de avaliações por filme é de 59.45, mas diferente dos usuários, não há restrição de quantidade mínima de avaliações, e a quantidade máxima de avaliações é de 583.

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

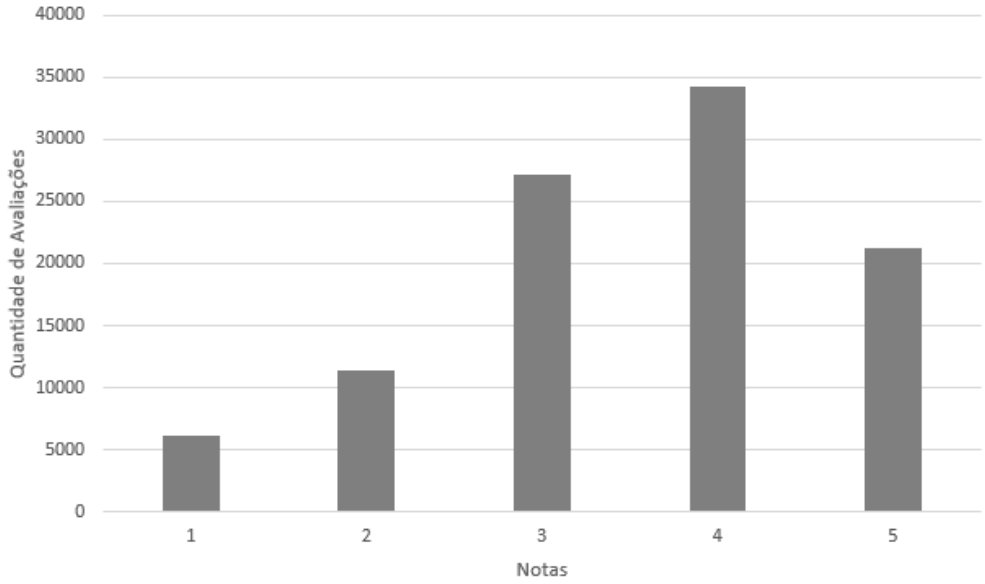

Figura 4.2: Quantidade de avaliações com relação a nota da base *MovieLens*

4.3.3 Avaliação

Para comparação dos resultados serão utilizadas três métricas que costumam ser utilizadas para avaliar a performance de sistemas de recomendação, são elas o *Mean Absolute Error* (MAE), o *Root Mean Squared Error* (RMSE) e o calculo de cobertura do modelo preditivo. A primeira delas é a métrica mais utilizada (GOLDBERG, 2001; HERLOCKER et al., 2004), é utilizada para calcular a média da diferença absoluta entre o valor real e as previsões. Sua formulação pode ser vista abaixo, onde n é o numero total de avaliações, p_{uv} é a previsão da nota de um usuário para um item e r_{uv} é a nota real.

$$MAE = \frac{\sum_{(u,v)} |p_{uv} - r_{uv}|}{n} \tag{4.1}$$

O cálculo do RMSE é bem semelhante ao MAE, sua principal diferença é que ele pune os erros maiores quando comparado com os menores. Essa métrica se tornou popular por causa do *Netflix Prize*. Sua equação pode ser vista abaixo, onde n é o numero total de avaliações, p_{uv} é a previsão da nota de um usuário para um item e r_{uv} é a nota real.

$$RMSE = \sqrt{\frac{\sum_{(u,v)}(p_{uv} - r_{uv})^2}{n}} \tag{4.2}$$

A cobertura de um SR é uma medida do domínio de itens sobre os quais o sistema pode fazer recomendações (HERLOCKER et al., 2004). Na literatura, o termo cobertura foi associado principalmente a dois conceitos: (1) a porcentagem dos itens para os quais o sistema pode gerar uma recomendação e (2) a porcentagem dos itens que podem ser recomendados para um usuário (ADOMAVICIUS; TUZHILIN, 2005; HERLOCKER et al., 2004). A fórmula para o cálculo pode ser vista abaixo, onde l é o conjunto de tuplas (usuário, item) na base e l_p é o conjunto de tuplas (usuário, item) que podem ser previstas.

$$Cobertura = \frac{l}{l_p} \tag{4.3}$$

4.3.4 Resultados

Nesta seção serão apresentados os resultados da busca por hiperparâmetros através de uma tabela com os melhores parâmetros obtidos para os modelos MLP, GMF e para o NeuMF. No final da seção será mostrada uma comparação entre os resultados obtidos pelos algoritmos implementados neste trabalho com algoritmos mais populares na literaturas.

4.3.4.1 Busca dos hiperparâmetros

Com o objetivo de obter bons parâmetros para a etapa de comparação entre alguns algoritmos mais consolidados na literatura, foi realizada uma etapa de busca dos hiperparâmetros. Foram variados os parâmetros: taxa de aprendizado, número de variáveis latentes do modelo, *dropout* das camadas ocultas e o algoritmo de correção dos pesos. A configuração das redes neurais foi mantida fixa em duas camadas internas contendo 128 neurônios cada uma nos NeuMF e MLP.

Na avaliação desses resultados foram usadas as métricas MAE, RMSE e cobertura.

Para validar os resultados foi utilizado o método de validação cruzada *Holdout* para encontrar uma média dos valores das métricas e seu desvio padrão. Os resultados para os modelos MLP, GMF e NeuMF podem ser vistos respectivamente, nas tabelas 4.1, 4.2 e 4.3, onde TA é a taxa de aprendizado, VL é o numero de variáveis latentes, D é o dropout e ACP é o nome do algoritmo de correção de custos.

TA	VL	D	ACP	MAE	σ(MAE)	RMSE	σ(RMSE)	Cobertura
0.01	2	0.1	Adam	0.72487	0.00421	0.99057	0.01309	1.0
0.1	8	0.1	SGD	0.72506	0.00073	0.96361	0.00032	1.0
0.01	4	0.1	Adam	0.73142	0.00350	1.00159	0.01414	1.0
0.1	4	0.5	SGD	0.73323	0.00783	0.97526	0.00398	1.0
1	4	0.1	Adam	0.73351	0.00711	0.964526	0.00231	1.0

Tabela 4.1: Melhores parâmetros obtidos na busca dos hiperparâmetros do *MLP*

TA	VL	ACP	MAE	σ(MAE)	RMSE	σ(RMSE)	Cobertura
0.01	2	Adam	0.74670	0.00513	0.97541	0.01302	1.0
0.01	4	Adam	0.74885	0.00378	0.97327	0.00676	1.0
0.01	8	Adam	0.75875	0.00433	0.98652	0.00498	1.0
0.01	16	Adam	0.78819	0.00682	1.02112	0.00899	1.0
1	4	Adam	0.83381	0.00864	1.13020	0.00727	1.0

Tabela 4.2: Melhores parâmetros obtidos na busca dos hiperparâmetros do *GMF*

TA	VL	D	ACP	MAE	σ(MAE)	RMSE	σ(RMSE)	Cobertura
1	2	0.1	Adam	0.76311	0.00318	0.98311	0.00423	1.0
1	4	0.5	Adam	0.76577	0.00324	0.98286	0.00612	1.0
1	2	0.5	Adam	0.76580	0.00602	0.98363	0.00669	1.0
0.1	8	0.1	SGD	0.76628	0.00405	0.98387	0.00760	1.0
0.1	4	0.1	SGD	0.76794	0.00674	0.98348	0.0069	1.0

Tabela 4.3: Melhores parâmetros obtidos na busca dos do *NeuMF*

4.3.4.2 Avaliação dos experimentos

Para comparação dos resultados foram utilizados algoritmos que se popularizaram após o prêmio Netflix[2], são eles o RSVD e IRSVD. Nos testes foi mantido em 0.001 a taxa de aprendizado e o numero de variáveis latentes desses algoritmos foi fixado em 20. Segundo trabalhos anteriores esses parâmetros apresentam bons resultados para esses modelos. (RICCI; ROKACH; SHAPIRA, 2015; BRAIDA, 2013).

[2]https://www.netflix.com

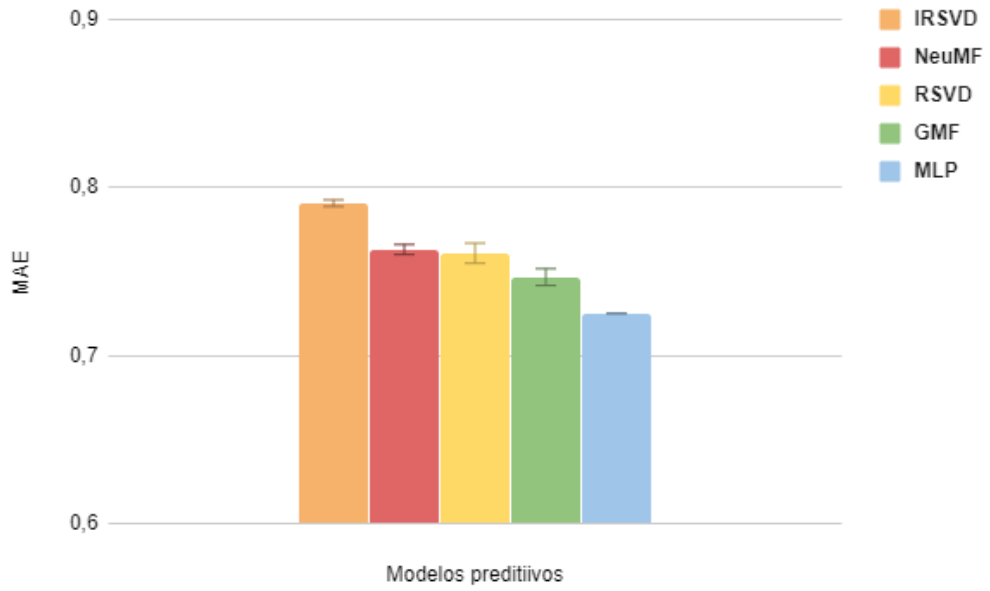

Figura 4.3: Gráfico de resultados para o MAE

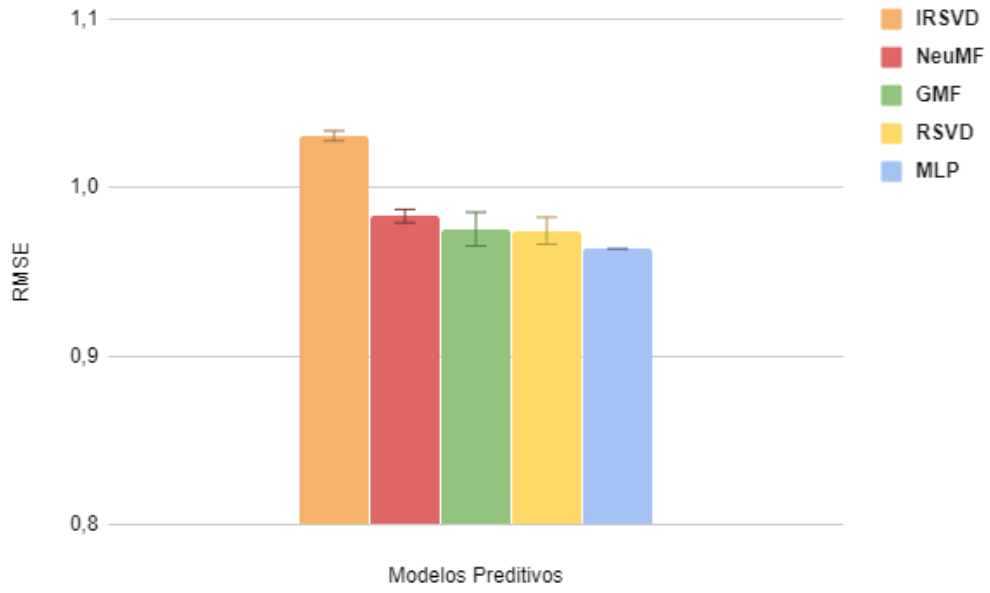

Figura 4.4: Gráfico de resultados para o RMSE

A forma como o Persa foi modelado, permitindo um grande potencial de extensão do código e a linguagem Julia ser de altíssimo nível, possibilitou o nosso *framework* ter uma alta legibilidade das técnicas desenvolvidas com a ferramenta, sem a necessidade

de implementar funções auxiliares que tornariam a estrutura projeto relativamente mais complexa, dificultando o entendimento do mesmo e tornando mais abstruso o desenvolvimento de novas técnicas.

As figuras 4.3 e 4.4 demostram que separados os algoritmos GMF e MLP conseguem apresentar uma boa performance. Entretanto, quando unidos em um único algoritmo, como é o caso do *NeuMF* o modelo perde um pouco do seu desempenho. Em um cenário geral os algoritmos apresentaram bons resultados comparados com o RSVD e o IRSVD.

Nos experimentos foi utilizada uma configuração de rede neural relativamente simples, com 2 camadas internas e cada um possuindo 128 nós. Existem diversas composições de redes mais complexas disponíveis na literaturas que tem potencial para melhorar a performance do modelo quando utilizado. No entanto, o objetivo destes experimentos não é mostrar uma configuração ótima, e sim uma arranjo competitivo com outros modelos.

Capítulo 5

Conclusões

Através dos resultados desse trabalho, foi demonstrado que a utilização de Redes Neurais em Sistemas de Recomendação é viável e tem potencial para entregar bons resultados.

5.1 Considerações finais

Foi desenvolvido um pacote para o *framework* Persa, com o intuito de facilitar e popularizar o estudo de modelos neurais na área de Filtragem Colaborativa, fazendo que o usuário tenha fácil acesso e possa usar um ambiente semanticamente amigável no momento da programação.

Para ajudar no desenvolvimento do projeto, foram utilizados os recursos disponibilizados pelo *Persa*, e foram adicionadas informações relevantes ao contexto, aproveitando das diversas vantagens da sua DSL.

A escolha da linguagem Julia também foi em prol do aproveitamento da DSL de suporte direto a linguagem, além da sua performance e da comunidade voltada para o estudo científico.

Como foi dito no início deste trabalho, a arquitetura das GPUS vem evoluindo, possibilitando otimizações cálculos baseados em Redes Neurais. O Keras, que foi utilizado na construção dos algoritmos preditivos, possui uma extensão que permite

que seus cálculos rodem em cima de uma GPU. Dessa forma, é possível processar mais dados, e se beneficiar do paralelismo e poder computacional desse tipo de hardware.

O pacote desenvolvido será disponibilizado como código aberto, possibilitando que estudiosos e entusiastas da área possam utilizar e colaborar para o desenvolvimento do projeto, visando a popularização e evolução desta área dentro da comunidade.

5.2 Limitações e trabalhos futuros

Um dos problemas encontrados durante o desenvolvimento do projeto foi na parte da integração com o projeto já desenvolvido, pois todo o projeto estava implementado em *Python*, e ouve a necessidade de se implementar os modelos na linguagem Julia. Além disso, existe o fato de Julia ser uma linguagem relativamente nova, e não havia muito conteúdo sobre a área para auxiliar na confecção do projeto.

Para o desenvolvimento futuro do pacote, se tem em mente a disponibilização do código em ambiente aberto, para que como dito anteriormente, a comunidade possa trabalhar em conjunto para o desenvolvimento da área. Uma das primeiras adições em mente para o projeto será a adição de um modelo que utilize a metodologia *COFILS*, que foi mencionada neste projeto.

Referências

ADOMAVICIUS, G.; TUZHILIN, A. Toward the Next Generation of Recommender Systems : A Survey of the State-of-the-Art and Possible Extensions. v. 17, n. 6, p. 734–749, 2005.

BENNETT, J.; LANNING, S. The Netflix Prize. p. 3–6, 2007.

BILLSUS, D.; PAZZANI, M. J. Learning collaborative information filters. In: *Proceedings of the Fifteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998. (ICML '98), p. 46–54. ISBN 1-55860-556-8. Disponível em: <http://dl.acm.org/citation.cfm?id=645527.657311>.

BRAIDA, F. Transformando o Problema de Filtragem Colaborativa em Aprendizado de Máquina Supervisionado. 2013.

BREESE, J. S. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. 1998.

BURKE, R. Hybrid Web Recommender Systems. p. 377–408, 2007.

CHHABRA, S. *Netflix says 80 percent of watched content is based on algorithmic recommendations*. 2017. Disponível em: <https://mobilesyrup.com/2017/08/22/80-percent-netflix-shows-discovered-recommendation/>.

CHUA, T.-s. Learning Image and User Features for Recommendation in Social Networks. p. 4274–4282, 2015.

EKSTRAND, M. D.; RIEDL, J. T.; KONSTAN, J. A. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, v. 4, n. 2, p. 81–173, 2011. ISSN 1551-3955. Disponível em: <http://dx.doi.org/10.1561/1100000009>.

ERHAN, D.; COURVILLE, A.; VINCENT, P. Why Does Unsupervised Pre-training Help Deep Learning ? v. 11, p. 625–660, 2010.

FOWLER, M. *Domain Specific Languages*. 1st. ed. [S.l.]: Addison-Wesley Professional, 2010. ISBN 0321712943, 9780321712943.

GIPSON, A. House Bill 950, Mississippi Legislature 2018 Regular Session. *Mississippi Legislature*, n. February, p. 2, 2018. ISSN 00224790 (ISSN).

GLOROT, X.; BORDES, A. Deep Sparse Rectifier Neural Networks. v. 15, p. 315–323, 2011.

GOLDBERG, K. Eigentaste : A Constant Time Collaborative. p. 133–151, 2001.

HAYKIN, S. *Redes Neurais - 2ed.* Bookman, 2001. ISBN 9788573077186. Disponível em: <https://books.google.com.br/books?id=lBp0X5qfyjUC>.

HE, K. Deep Residual Learning for Image Recognition. 2015.

HERLOCKER, J. L. et al. Evaluating Collaborative Filtering Recommender Systems. v. 22, n. 1, p. 5–53, 2004.

KINGMA, D. P.; BA, J. L. Adam: A Method for Stochastic Optimization. p. 1–15, 2015.

MCCULLOCH, W. A Logical Calculus Of The Ideas Immanent in Nervous Activity . v. 5, p. 115–133, 1943.

MILLER, B. et al. Movielens unplugged: Experiences with an occasionally connected recommender system. *International Conference on Intelligent User Interfaces, Proceedings IUI*, 05 2003.

NASCIMENTO, V. D. do. Filtragem Colaborativa Como Serviço Utilizando Processamento Na GPU.

OORD, A. van den; DIELEMAN, S.; SCHRAUWEN, B. Deep content-based music recommendation. In: BURGES, C. J. C. et al. (Ed.). *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013. p. 2643–2651. Disponível em: <http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf>.

POZZA, A. et al. Magnetic Resonance Enterography for Crohn's Disease: What the Surgeon Can Take Home. *Journal of Gastrointestinal Surgery*, v. 15, n. 10, p. 1689–1698, 2011. ISSN 1091255X.

RESNICK, P.; BERGSTROM, P.; RIEDL, J. GroupLens : An Open Architecture for Collaborative Filtering of Netnews. p. 175–186, 1994.

RICCI, F.; ROKACH, L.; SHAPIRA, B. *Recommender Systems Handbook*. [s.n.], 2015. ISSN 16130073. ISBN 978-1-4899-7636-9. Disponível em: <http://link.springer.com/10.1007/978-1-4899-7637-6>

SCIENCE, P. C. et al. Matrix Factorization Model in Collaborative Filtering Algorithms: A Survey. v. 49, p. 136–146, 2015.

SELLI, M. F.; JR., P. S. Online identification of horizontal two-phase flow regimes through gabor transform and neural network processing. *Heat Transfer Engineering*, Taylor Francis, v. 28, n. 6, p. 541–548, 2007. Disponível em: <https://doi.org/10.1080/01457630701193955>.

SU, X.; KHOSHGOFTAAR, T. M. A Survey of Collaborative Filtering Techniques. v. 2009, n. Section 3, 2009.

WANG, H.; WANG, N. Collaborative Deep Learning for Recommender Systems. 2015.

XAVIER, P. ContextCF . jl : Um Framework para CARS em Filtragem Colaborativa ContextCF . jl : Um Framework para CARS em Filtragem Colaborativa. 2018.

YE, F.; SUN, Y.; YANG, Q. *An Adaptive Linear Element Based Method for Identification of Linear*. IFAC, 2013. v. 46. 309–314 p. ISSN 1474-6670. ISBN 9783902823458. Disponível em: <http://dx.doi.org/10.3182/20130902-3-CN-3020.00074>.

ZHANG, H.; CHUA, T.-s. Neural Collaborative Filtering. p. 173–182, 2017.

ZHANG, S. et al. Deep Learning based Recommender System : A Survey and New Perspectives. v. 1, n. 1, p. 1–35, 2018.

ZHENG, Y. A User's Guide to CARSKit. 2015. ISSN 1548-825X.