Capstone Project

Machine Learning Engineer Nanodegree

Ricardo Araujo
7 de julho de 2018

Sistema de Recomendação de Filmes

Visão Geral

Neste projeto irei desenvolver um modelo de Sistema de Recomendação de filmes. Mas o que são Sistemas de Recomendação? Se você usa plataformas de streaming de filmes, como Netflix e Amazon Prime Video, já deve ter recebido uma notificação por e-mail de recomendação de algum filme ou série que você não assistiu. Para lhe fazer essa recomendação a plataforma utilizou algoritmos que juntos formam um Sistema de Recomendação. O mesmo foi desenvolvido para aprender os seus gostos pessoais e oferecer uma experiência personalizada, otimizando o uso na plataforma.

Mas como a plataforma aprende seus gostos pessoais? Toda vez que algum filme é classificado, marca de 1 a 5 estrelinhas ou dá um "joinha", você está ensinando o algoritmo sobre o que você gosta de assistir. Mesmo que nenhum filme seja classificado, um algoritmo mais complexo pode fazer o uso de dados implícitos, como o número de vezes que você assistiu ao mesmo filme e o número de cliques, e fazer uso desses dados na recomendação.

Existem outros grandes exemplos na indústria englobando outros domínios, como e-commerce e sites de notícias. Entre as empresas que utilizam sistemas de recomendação estão nomes como Amazon, Netshoes e Globo.com.

Para desenvolver este projeto irei utilizar os dados fornecidos pelo Grouplens, um laboratório de pesquisa do Departamento de Ciência da Computação e Engenharia da Universidade de Minnesota.

Descrição do problema

O modelo de sistema de recomendação que irei desenvolver tem como foco principal prever as classificações de usuários para filmes que ele não classificou e recomendar os que tiveram classificação mais alta. Podemos utilizar diversos algoritmos de filtragem colaborativa, que é muito utilizado para realizar recomendações. Irei apresentar dois tipos de recomendação, uma utilizando o algoritmo SVD (Singular Value Decomposition) e outra KNN (K-Nearest Neighbour). O SVD será usado para prever classificações de filmes que o usuário não classificou utilizando decomposição de matrizes e o KNN será usado para selecionar filmes similares.

Métricas

As métricas que irei utilizar nesse modelo são o Erro Quadrático Médio (sigla em inglês, *RMSE*) e o Erro Médio Absoluto (sigla em inglês, *MAE*) para medir a precisão. Como métricas de similaridade teremos o Coeficiente de Correlação de Pearson e a Similaridade Cosseno. Os dois últimos serão utilizados com o KNN para criar uma lista de K "*neighbors*" com alto coeficiente de correlação.

Erro Quadrático Médio e Erro Médio Absoluto

Em sistemas de recomendação baseados em dados explícitos (ou classificações), se usa dados de classificações de usuários para gerar um conjunto de classificações preditas, você então pode usar essas predições para "*rankear*" filmes. O erro que medimos com *RMSE* e *MAE* é baseado na diferença entre classificações que prevemos para um filme e na que o usuário explicitamente deu a um filme. Onde o retorno do valor predito é escalar, sendo as métricas ideais para esse propósito.

$$\boldsymbol{RMSE} = \sqrt{\frac{\mathbf{1}}{|\boldsymbol{R'}|}\sum_{\boldsymbol{r'\in R'}}(\boldsymbol{r_{ui}} - \boldsymbol{r'_{ui}})^2}$$

$$\boldsymbol{MAE} = \frac{\mathbf{1}}{|\boldsymbol{R'}|}\sum_{\boldsymbol{r'\in R'}}|\boldsymbol{r_{ui}} - \boldsymbol{r'_{ui}}|$$

Coeficiente de Correlação de Pearson

O coeficiente de correlação de Pearson será utilizado para medir a similaridade entre filmes. Por exemplo, digamos que as classificações entre os filmes "Piratas do Caribe" e "Peter Pan" tem uma correlação alta. Baseado nessa correlação podemos prever que o usuário John gostaria de "Peter Pan", dado o fato que ele gostou de "Piratas do Caribe". O Coeficiente de Correlação de Pearson é uma medida do quão linearmente relacionadas são duas distribuições.

Se considerarmos que um dado usuário μ_i classificou filmes com distribuição $R_i \sim (\mu_i, \sigma_i)$, então a similaridade entre os usuários μ_u e μ_v é o coeficiente de correlação entre as duas distribuições.

$$\boldsymbol{R_i, R_j : \rho_{ij}} = \frac{\sum(\boldsymbol{R_i} - \boldsymbol{\mu_i})(\boldsymbol{R_j} - \boldsymbol{\mu_j})}{\boldsymbol{\sigma_i} * \boldsymbol{\sigma_j}}$$

Similaridade Cosseno

Ao invés de utilizar o Coeficiente de Correlação de Pearson, podemos utilizar a Similaridade Cosseno para determinar a similaridade entre dois usuários. É a mesma fórmula do Coeficiente de Correlação de Pearson mas sem a normalização das médias das distribuições:

$$s_i = \frac{\sum(R_i)(R_j)}{\sigma_i * \sigma_j}$$

Análise Exploratória

Nesta seção inicio com as implementações de código utilizadas no projeto. Todo o código está implementado em Python e está presente no iPython notebook que acompanha este relatório.

Neste projeto irei utilizei três arquivos CSV, o ratings.csv, o movies.csv e o links.csv. O ratings.csv representa a classificação de um filme feita por um usuário e o movies.csv contém o filme, gênero e o ano. O arquivo links.csv contém o filme e dois códigos numéricos para acessar as APIs dos IMDb e TMDb, que contém informações sobre determinado filme.

Este *dataset* descreve classificações de 1 a 5 estrelas de usuários do MovieLens, um serviço de recomendação de filmes. Ele contém 100004 classificações de 9125 filmes. Os dados foram criados por 671 usuários entre 9 de janeiro de 1995 e 16 de outubro de 2016. Os usuários foram selecionados de forma aleatória e todos têm pelo menos classificado 20 filmes. O *dataset* se encontra nesta página: https://goo.gl/sm6ggD.

Vamos começar observando a estrutura do arquivo ratings.csv:
userId: ID do usuário.
movieId: ID do filme classificado pelo usuário.
rating: Nota do usuário para o filme. Em escala de 5 estrelas, com incrementos de meia (0.5) estrela.
timestamp: Horário da classificação em formato timestamp.

O dataset ratings contém 100004 entradas de dados, cada entrada de dados corresponde a uma nota fornecida por um usuário a determinado filme.
A coluna “timestamp” foi trocada por uma coluna “year”. O arquivo foi carregado no dataframe do pandas “df_ratings”.

Estrutura do arquivo movies.csv:
movieId: ID do respectivo filme.
title: Nome do filme.
genres: Gênero do filme

O dataset movies contém 9125 entradas de dados, cada entrada corresponde a um filme diferente. O atributo movieId do dataset movies corresponde ao

mesmo movieId do dataset ratings. Será usado para recuperar o nome do filme após o cálculo dos algoritmos SVD e Knn. A coluna gênero não será utilizada nos algoritmos, mas será útil para fazer visualização dos dados. A coluna “title” possui o nome do filme seguido pelo ano de lançamento, por exemplo “Toy Story (1993)”. Foi extraído o ano desta coluna e criada uma nova coluna chamada “year”. Será necessário para visualização explotarória. O arquivo foi carregado no dataframe do pandas “df_filmes”.

Estrutura do arquivo links.csv:
movieId: ID do respectivo filme.
imdbId: ID para buscar informações na API do IMDb.
tmdbId: ID para buscar informações na API do TMDb.

O dataset links contém 9125 entradas de dados, cada entrada corresponde a um um código do link de um filme diferente. Posteriormente irei utilizar uma API para facilitar a visualização da capa dos filmes recomendados e este dataset nos será útil, pois os códigos das colunas imdbId e tmdbId são para acessar informações sobre o filme correspondente a numeração do movieId. O arquivo foi carregado no dataframe do Pandas, “df_links”.

Limpeza dos dados

Os dados deste *dataset* no geral vieram bem estruturados e não foram necessárias muitas modificações. Todas as entradas de “movieId”, que são importantes para visualizar os filmes recomendados estão presentes nos 3 dataframes. No dataframe de filmes foram encontradas 4 entradas de dados faltando na coluna “year”. Decidi manter as linhas pois a remoção pode prejudicar na recomendação, pois algum filme recomendado poderia estar entre os removidos. No dataframe de links existem 13 entradas de dados faltando na coluna “tmdbId”, o que pode atrapalhar na visualização da capa dos filmes caso algum desses seja selecionado para recomendação, mas não é algo que vai atrapalhar na construção do modelo.

Visualização Exploratória

Na **Figura 1** temos o gráfico de distribuição das classificações que os usuários deram aos filmes:

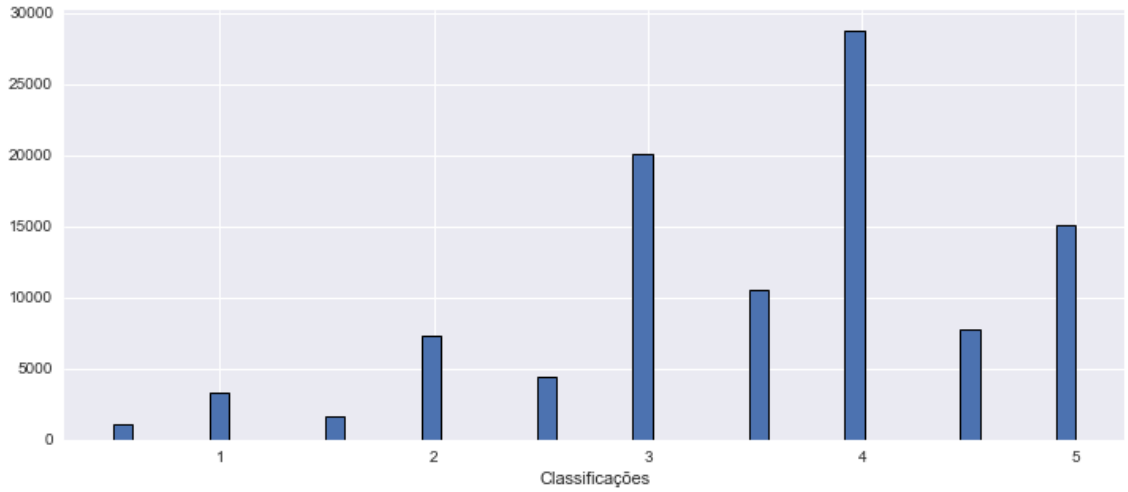

Figura 1

Podemos notar que a maioria das classificações se concentra entre 3 e 5, sendo que a moda é 4, e vemos que não temos muitas classificações entre 0.5 e 2.5. Sendo uma distribuição enviesada à esquerda. Isto significa que a maioria dos usuários tende a classificar apenas filmes que eles acharam mediano e filmes que gostaram. Outro ponto interessante é que há uma queda acentuada em classificações com números não inteiros. Temos uma grande quantidade de classificações com nota 4, mas poucas classificações com nota 3.5 e 4.5. Após analisar a plataforma do Movielens creio que a explicação mais plausível para isto é que seria mais prático selecionar, por exemplo, 3 estrelas ao invés de 3.5 ou 2.5, pois necessita de um esforço a mais do usuário. Isolando inteiros e não inteiros eles seguem a mesma tendência enviesada para a esquerda. Mas isto é apenas um detalhe, não relevante na construção do modelo.

```
df_ratings.describe()
```

	userId	movieId	rating
count	100004.000000	100004.000000	100004.000000
mean	347.011310	12548.664363	3.543608
std	195.163838	26369.198969	1.058064
min	1.000000	1.000000	0.500000
25%	182.000000	1028.000000	3.000000
50%	367.000000	2406.500000	4.000000
75%	520.000000	5418.000000	4.000000
max	671.000000	163949.000000	5.000000

Figura 2

Na **Figura 2** podemos notar que o percentil de 50% tem valor 4, confirmando a alta taxa de notas 4.

Quantidade de produções cinematográfica ao longo dos anos

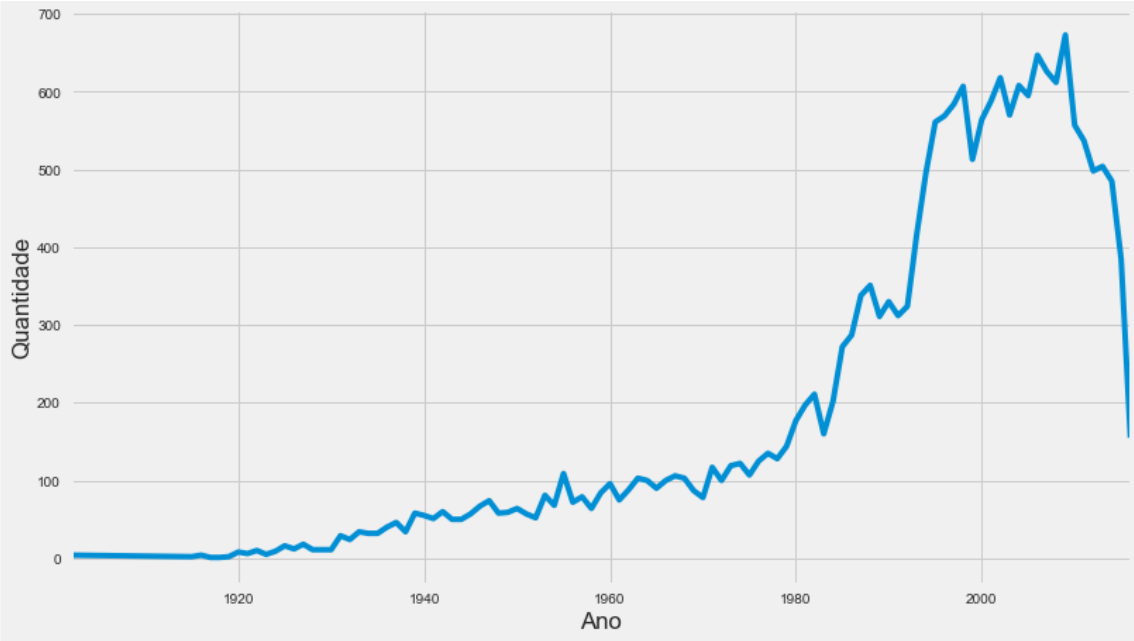

Figura 3

Na **Figura 3** vemos que as produções cinematográficas se mantêm em crescimento desde o seu surgimento. Houve um elevado crescimento a partir dos anos 80, década em que houveram muitos lançamentos de *blockbusters* e *sequels*. A queda acentuada no final do gráfico creio que seja devido aos filmes terem sido coletados até o mês de outubro de 2016, não contabilizando filmes que foram lançados até o final do ano. Os anos onde houveram mais produções foram no final da primeira década dos anos 2000. Como mostrado na **Figura 4** um trecho do Jupyter Notebook:

```
byYear.sort_values().tail(5)
```

```
year
2008    612
2002    618
2007    626
2006    647
2009    673
Name: year, dtype: int64
```

Figura 4. Quantidade de filmes lançados no ano (presentes no dataset)

Popularidade dos gêneros

Na **Figura 5** é relatado os gêneros que são mais populares entre os usuários, correspondente ao tamanho da letra.

Figura 5. Popularidade dos gêneros.

Não é de se surpreender que filmes de drama, comédia, suspense(Thriller), ação e romance são os mais populares. São os gêneros que tiveram mais crescimento ao longo dos anos, como será mostrado no próximo gráfico.

Gêneros mais populares ao longo dos anos

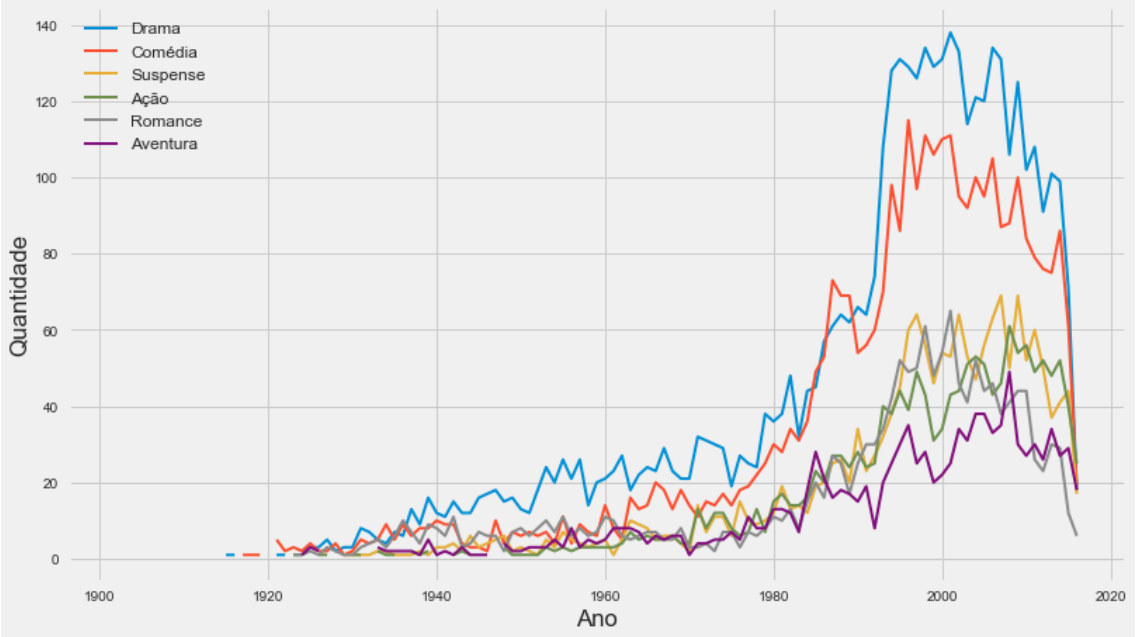

Figura 6. Gêneros mais populares ao longo dos anos.

Nesta representação gráfica da **Figura 6** temos os gêneros mais populares através dos anos. Podemos notar que segue a mesma tendência de crescimento nos anos 80 do primeiro gráfico. Os filmes de drama e comédia tiveram um crescimento bem mais elevado que os demais gêneros. O lançamento de filmes nestas categorias aumentou bastante a quantidade de lançamentos desde os anos 80 até o final da primeira década dos anos 2000.

Outros gêneros ao longo dos anos

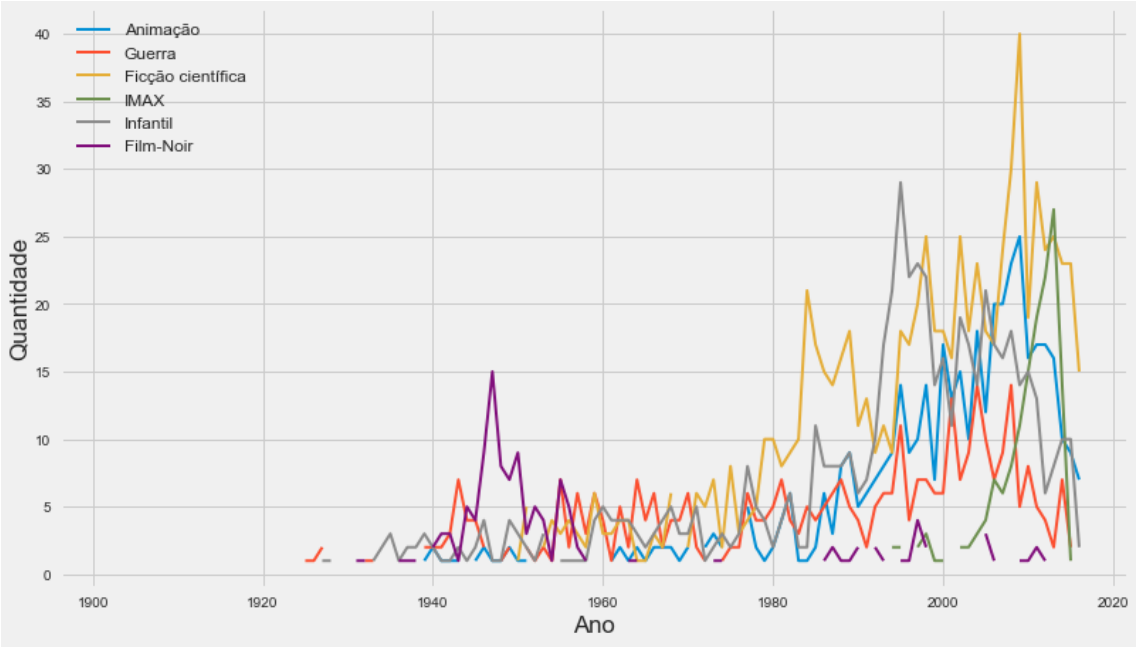

Figura 7

Neste gráfico da **Figura 7** plotei alguns gêneros que merecem a nossa atenção. Podemos notar que os filmes de Film-Noir tiveram seu ápice entre as décadas de 1940 e 1960 e são quase extintos nos dias de hoje. Filmes de animação tiveram um elevado crescimento nos anos 2000, pois nesta época as tecnologias de computação gráfica tiveram grande melhora de qualidade. Houve crescimento no lançamento dos filmes de guerra nos anos 60 e depois nos anos 2000. O primeiro na mesma época em que houve a guerra do Vietnam e o segundo na época da Guerra do Iraque. Na mesma época que houve um pico de filmes de ficção científica na metade dos anos 2000, houveram picos em filmes de animação e guerra. Podem estar relacionados devido a lançamento de filmes que possuem esses gêneros agregados.

Algoritmos

Para realizar as recomendações irei utilizar um método chamado filtragem colaborativa. Filtragem colaborativa é subdividida em dois tipos: Memory-based e Model-based. Os dois tipos serão testados no projeto e explicados nessa seção.

Filtragem Colaborativa

O método de filtragem colaborativa [1] é o mais utilizado na indústria atualmente e o mais bem-sucedido em sistemas de recomendação. O mesmo se baseia em coletar e analisar as informações, comportamentos, preferências ou atividades de usuários e prever o que eles gostariam, baseado na

similaridade ou comparação de atributos dos usuários. Uma maneira de quantificar o filme que um usuário gostaria ou não gostaria é usar as notas que ele deu a outros filmes. Esta nota pode ser numa escala de 1 a 5, mas também pode ser em formato binário como “gostei” ou “não gostei”.

Filtragem Colaborativa: Memory-based

Um sistema de recomendação é dito como memory-based quando o algoritmo computa as similaridades entre os itens (filmes) ou usuários sem a necessidade de criar um modelo. Também pode ser chamado de Neighbourhood-based, porque busca pelos melhores candidatos dentre os mais similares.
Este método é subdividido em User-based e Item-based. Em um algoritmo User-based definimos um usuário, encontramos usuários que são similares a este usuário baseado na similaridade de notas aos filmes, e recomendamos os filmes que esses usuários gostaram. Em um algoritmo Item-based definimos um item (filme), encontramos usuários que também gostaram deste item e buscamos outros itens que esses usuários ou usuários similares também gostaram.
Em ambos os métodos precisamos encontrar os K-Nearest Neighbour de um usuário ou item. No entanto, bancos de dados tendem a ter muito mais usuários do que filmes, por isso sabemos que a busca é muito mais custosa na dimensão de usuários do que de filmes. No cotidiano, usuários interagem mais com o banco de dados do que os filmes. Geralmente os filmes não mudam com tanta frequência que os atributos dos usuários. No domínio de sistemas de recomendação de filmes a abordagem Item-based é a mais recomendada por este motivo.

Em resumo:

- User-based: “Usuários similares a você também gostaram de...”
- Item-based: “Usuários que gostaram deste filme também gostaram de...”

Muitos algoritmos têm sido usados para medir a similaridade entre usuários e entre itens. Por exemplo, K-Nearest Neighbour [2] e Coeficiente de correlação de Pearson. Neste projeto o algoritmo KNN será utilizado para criar uma lista dos K neighbours com mais correlação, utilizando o Coeficiente de Correlação de Pearson ou Similaridade Cosseno, na abordagem Item-based. Está é uma abordagem menos custosa, se levarmos em conta a quantidade de usuários e a quantidade de filmes.

Filtragem Colaborativa: Model-based

Este tipo de Filtragem Colaborativa é baseado em decomposição de matrizes, que é muito utilizada como um método de aprendizagem não-supervisionada para decomposição de variáveis latentes, ou seja, atributos que não foram

observados diretamente, mas foram inferidos através de um modelo matemático. O objetivo da decomposição de matrizes é aprender as preferências latentes dos usuários e os atributos latentes dos itens (filmes) e prever as notas que cada usuário daria para os filmes que ele não classificou, através do produto escalar dos atributos latentes dos usuários e filmes. Digamos que temos uma matriz bastante esparsa, ou seja, com muitos elementos que valem 0, e com muitas dimensões. Cada coluna representa um filme e cada linha representa um usuário. Onde os elementos diferentes de 0 correspondem a nota que um usuário deu a determinado filme e os elementos iguais a 0 são as notas de filmes que o usuário não classificou. Fazendo a decomposição de matrizes podemos reestruturar a matriz usuário-item em uma estrutura low-rank (matrizes com menores dimensões), podendo representar a matriz principal pela multiplicação dessas duas matrizes low-rank onde seus elementos são preenchidos por variáveis latentes. Multiplicando essas matrizes low-rank obtemos valores muito aproximados da matriz original e os elementos com valores de 0 da matriz original são preenchidos. Assim fazendo uma previsão da nota que o usuário daria para determinado filme.

Um dos modelos de decomposição de matrizes mais utilizados em sistemas de recomendação é o *Singular Value Decomposition (SVD)* [3].

Sendo X uma matriz m × n (usuário-item):

$$X = U \times S \times V^T$$

- U é a matriz ortogonal $m \times r$. É associada aos usuários. São os *eigenvectors* de XX^T.
- S é uma matriz diagonal não negativa com números reais na diagonal, parecida com uma matriz identidade. Corresponde a raiz quadrada dos *eigenvalues* de $X^T X$ ou XX^T.
- V^T (V transposta) é uma matriz ortogonal $r \times m$. É associada aos itens. São os *eigenvectors* de $X^T X$.

A matriz X é decomposta em $U, S\ e\ V$. A matriz U representa os valores latentes correspondentes os usuários e a matriz V representa os valores latentes dos itens. Podemos prever os valores esparsos aproximados da matriz X fazendo o produto escalar da sua decomposição:

$$\begin{bmatrix} X_{11} & \cdots & X_n \\ \vdots & \ddots & \vdots \\ X_m & \cdots & X_{m\times n} \end{bmatrix} \approx \begin{bmatrix} U_{11} & \cdots & U_r \\ \vdots & \ddots & \vdots \\ U_m & \cdots & U_{m\times r} \end{bmatrix} \times \begin{bmatrix} S_{11} & 0 & \cdots & 0 \\ 0 & S_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & S_{r\times r} \end{bmatrix} \times \begin{bmatrix} V_{11} & \cdots & V_m \\ \vdots & \ddots & \vdots \\ V_r & \cdots & V_{r\times n} \end{bmatrix}^T$$

Adicionando Viés

Alguns usuários costumam classificar filmes medianos com notas boas, outros classificam filmes medianos como notas baixas. Ou seja, alguns usuários têm

tendência a dar notas altas e outros mais críticos a dar notas baixas. Baseado nisso apenas notas de filmes não podem determinar sozinhas quais filmes seriam recomendados, é aí que o viés (*biases*, ou *baselines* [4]) é adicionado a equação.
Uma *baseline* estimada para uma nota desconhecida r_{ui} seria descrita como:

$$b_{ui} = \mu + b_u + b_i$$

Onde b_{ui} é a *baseline* em relação a nota r_{ui}. μ diz respeito a média de notas dos usuários, b_i e b_u são os desvios que o usuário e o item apresentam da média.
Por exemplo, imagine que queremos estimar a baseline do filme Y pelo usuário X. Digamos que μ, a nota média entre os filmes, é 3.5. Mas o filme Y tende a ser classificado 0.5 acima da média e o usuário X tende a classificar com 0.3 abaixo da média. Para estimar a *baseline* deste filme calculamos 3.5 – 0.3 + 0.5.
Para encontrar um valor definitivo para as notas podemos calcular:

$$r_{ui} = b_{ui} + R_{ui}$$

Onde R_{ui} é o valor aproximado de uma nota de um usuário a um determinado item. É encontrado pelo algoritmo SVD.
A diferença $r_{ui} - R_{ui}$ corresponde ao erro e_{ui}, que pode ser minimizado utilizando Gradiente Descendente Estocástico ou Alternating Least Squares.

Modelo de Referência

Como modelo de referência [5] utilizei o de um usuário do Kaggle que utilizou dados do Netflix Prize. Foram feitar predições utilizando filtragem colaborativa com algoritmo SVD, implementado utilizando a biblioteca Surprise. O mesmo obteve uma média de RMSE de 0.98 e MAE de 0.78. Foram mantidos os parâmetros padrão do algoritmo SVD da biblioteca Surprise [6]. Foi um bom resultado para os dados do Netflix Prize, vamos compará-lo ao meu modelo final, que no caso utilizarei os do Movie Lens.

Criação do Modelo

Nesta fase do projeto farei uma Busca de Grade no algoritmo SVD e KNN para verificar os melhores parâmetros de acordo com o RMSE e MAE para serem utilizados nos modelos. Após a Busca de Grade farei a implementação dos algoritmos e vou verificar se as recomendações foram boas com estes parâmetros.

Calibragem dos Algoritmos

Na calibragem do algoritmo K-Nearest Neighbour foram levados em conta principalmente a forma como foram estimadas as *baselines* os as medidas de

similaridade. No teste de Busca de Grade obtive 0.95 de pontuação RMSE e 0.73 para o MAE utilizando ALS e Coeficiente de Correlação de Pearson.
A **Figura 8** ilustra o código utilizado na Busca de Grade:

```
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df_ratings[['userId', 'movieId','rating']], reader)

param_grid = {'bsl_options': {'method': ['als', 'sgd'],
                              'reg': [1, 2]},
              'k': [2, 3],
              'sim_options': {'name': ['pearson_baseline', 'cosine'],
                              'min_support': [1, 5],
                              'user_based': [False]}
              }

knn_gs = GridSearchCV(KNNBaseline, param_grid, measures=['rmse', 'mae'], cv=3)
knn_gs.fit(data)

# melhor pontuação para RMSE
print(knn_gs.best_score['rmse'])

# melhores parâmetros para o modelo
print(knn_gs.best_params['rmse'])
```

```
# melhor pontuação para MAE
print(knn_gs.best_score['mae'])
# melhores parâmetros de acordo com o MAE
print(knn_gs.best_params['mae'])
```

Figura 8

A **Figura 9** mostra o resultado obtido para a pontuação RMSE e MAE.

```
0.951820691987
{'bsl_options': {'method': 'als', 'reg': 1}, 'k': 3, 'sim_options': {'name': 'pearson_baseline', 'min_support': 5, 'user_base
d': False}}
0.72329153285
{'bsl_options': {'method': 'als', 'reg': 1}, 'k': 3, 'sim_options': {'name': 'pearson_baseline', 'min_support': 5, 'user_base
d': False}}
```

Figura 9

Na calibragem do algoritmo SVD foram levados em conta o número de repetições do Gradiente Descendente Estocástico (SGD), a *Learning Rate,* o parâmetro de regularização e se deve ser enviesado (utilizar *baselines*) ou não. Obtive como melhor pontuação RMSE, 0.88 e MAE, 0.68, com o SGD de 50 repetições, *Learning Rate* de 0.005, parâmetro de regularização 0.1 e com o algoritmo enviesado, ou seja, ajustando as tendências dos usuários, como explicado anteriormente.
A **Figura 10** ilustra o código utilizado na Busca de Grade e os resultados:

```
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df_ratings[['userId', 'movieId','rating']], reader)

param_grid = {'n_epochs': [20, 30, 40, 50], 'lr_all': [0.002, 0.005],
              'reg_all': [0.1, 0.4], 'biased':[True, False]}
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
gs.fit(data)

# melhor pontuação para RMSE
print(gs.best_score['rmse'])

# melhor pontuação para MAE
print(gs.best_score['mae'])

# melhores parâmetros de acordo com o RMSE
print(gs.best_params['rmse'])

# melhores parâmetros de acordo com o MAE
print(gs.best_params['mae'])
```

```
0.886766102109
0.683036656981
{'n_epochs': 50, 'lr_all': 0.005, 'reg_all': 0.1, 'biased': True}
{'n_epochs': 50, 'lr_all': 0.005, 'reg_all': 0.1, 'biased': True}
```

Figura 10

Implementação

Com os parâmetros escolhidos, agora iremos para a fase de implementação dos algoritmos. Após o treinamento vamos retornar os filmes mais recomendados.

Implementação: K-Nearest Neighbour

Como citado na seção Algoritmos, a Filtragem Colaborativa com KNN é do tipo Memory-based, que se baseia em similaridades. Nesta implementação o algoritmo KNN irá utilizar o “userId”, “movieId” e “rating” para encontrar similaridades entre filmes. Esta abordagem é utilizada definindo um filme, depois encontrando usuários que gostaram deste filme e posteriormente se recomenda outros filmes que estes usuários também gostaram. Os parâmetros utilizados serão os definidos pela Busca de Grade, pois foram os que obtiveram maior pontuação.

A **Figura 11** mostra o código do Jupyter notebook utilizado no treinamento do modelo.

```
# Carregando os dados de df_ratings na biblioteca Surprise
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df_ratings[['userId', 'movieId','rating']], reader)

# ajustando os parâmetros e definindo no algoritmo KNN
sim_options = {'name': 'pearson_baseline','min_support': 5,'user_based': False}
bsl_options = {'method': 'als',
               'n_epochs': 5,
               'reg': 1,
               }
algo = KNNBaseline(k=3, sim_options=sim_options, bsl_options=bsl_options)

# treinando o conjunto de dados
trainset = data.build_full_trainset()
algo.fit(trainset)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

Figura 11

Após treinar o modelo, criei algumas funções para tratar o retorno dos K Neighbour. A função “title_to_rid” foi criada para converter o nome do filme, conforme definido no dataframe “df_filmes”, para o “inner id” e “raw id”. O “raw id” é o mesmo “movieId” definido no “df_filmes”, o “inner id” é o id que a Surprise utiliza para fazer o tratamento dos dados e é criado pela biblioteca. Utilizei a função “get_neighbours” [7], que retorna os K neighbours mais similares, e tem como entrada o “inner id” do filme no qual quer encontrar os similares e tem como retorno também o “inner id”.
Assim criei a função “visualizar_sim” que retorna os filmes mais similares e o gênero dos mesmos, para validação, e tem como entrada o nome do filme e os K Neighbours. A implementação dessas funções está no arquivo iPython notebook, “RecSys-Projeto.ipynb”, que acompanha este relatório.
Na **Figura 12** é demostrada a chamada da função “visualizar_sim”, tendo como entrada o filme Toy Story (1995) e 10 Nearest Neighbours.

```
print('KNN com Pearson ---> TOP 10 filmes mais similares a Toy Story:')
visualizar_sim('Toy Story (1995)', 10)
```

Figura 12

Por enquanto vamos nos atentar apenas em visualizar os filmes que retornaram, deixarei para comentar sobre os gêneros desses filmes posteriormente.

A **Figura 13** mostra o retorno da função com os 10 filmes mais parecidos com Toy Story (1995).

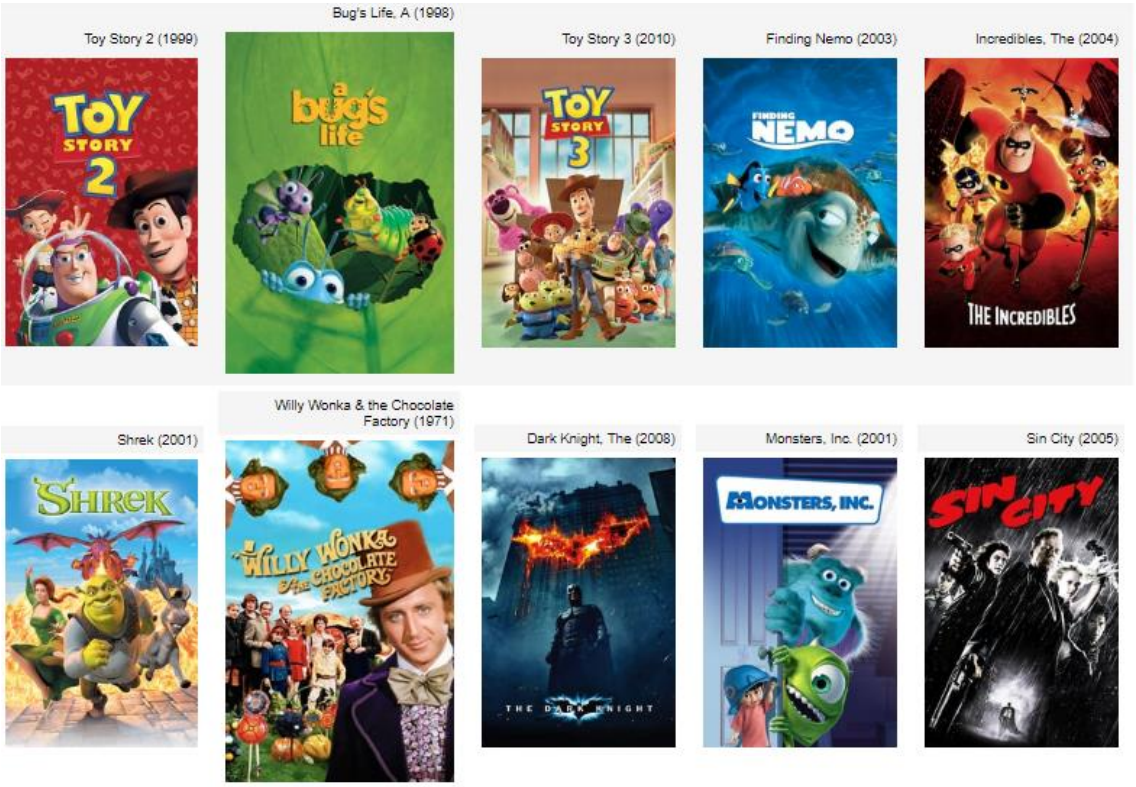

Figura 13

Podemos notar que a maioria dos filmes são de animação e de gênero infantil. Mas dois deles, *Dark Knight* e *Sin City*, são filmes que não têm similaridades com *Toy Story* e não deveriam ser recomendados a uma criança, pois podem conter nudez e violência. Após outro teste utilizando Similaridade Cosseno obtive um resultado pior do que o anterior, como mostrado na **Figura 14**.

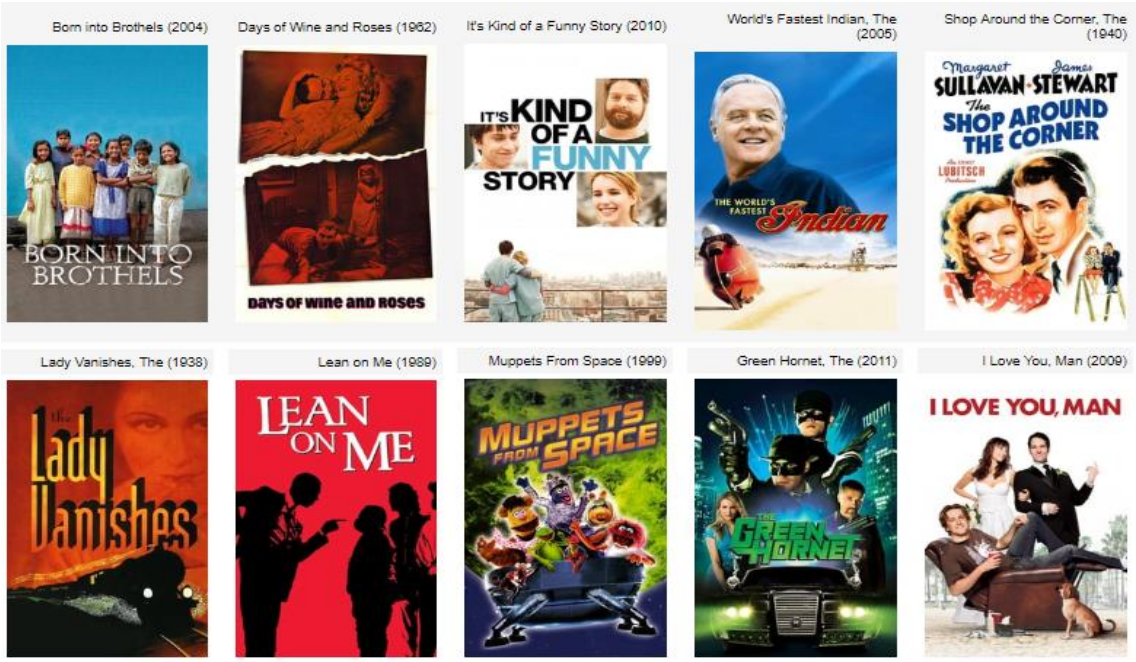

Figura 14

Estas recomendações nada têm de similaridade com *Toy Story*, demonstrando assim que Similaridade Cosseno não foi uma boa escolha para este conjunto de dados. Vamos reajustar o algoritmo utilizando Descida de Gradiente Estocástico (SGD) ao invés de Alternating Least Squared (ALS), com e Coeficiente de Correlação de Pearson. Após o reajuste obtive o resultado da **Figura 15**.

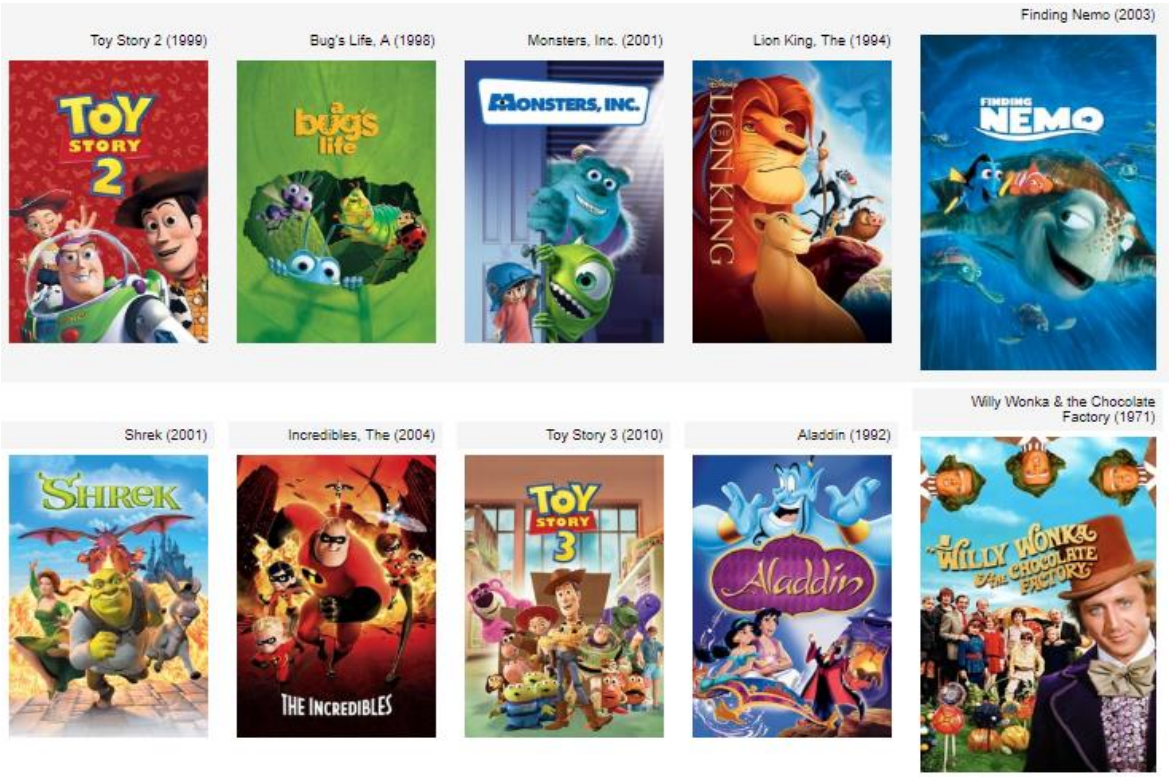

Figura 15

Acredito que este seria um bom resultado na recomendação. Para validar o último resultado, realizei um teste buscando um filme diferente, e neste caso foi “Lord of the Rings: The Fellowship of the Ring”. A **Figura 16** mostra o resultado obtido.

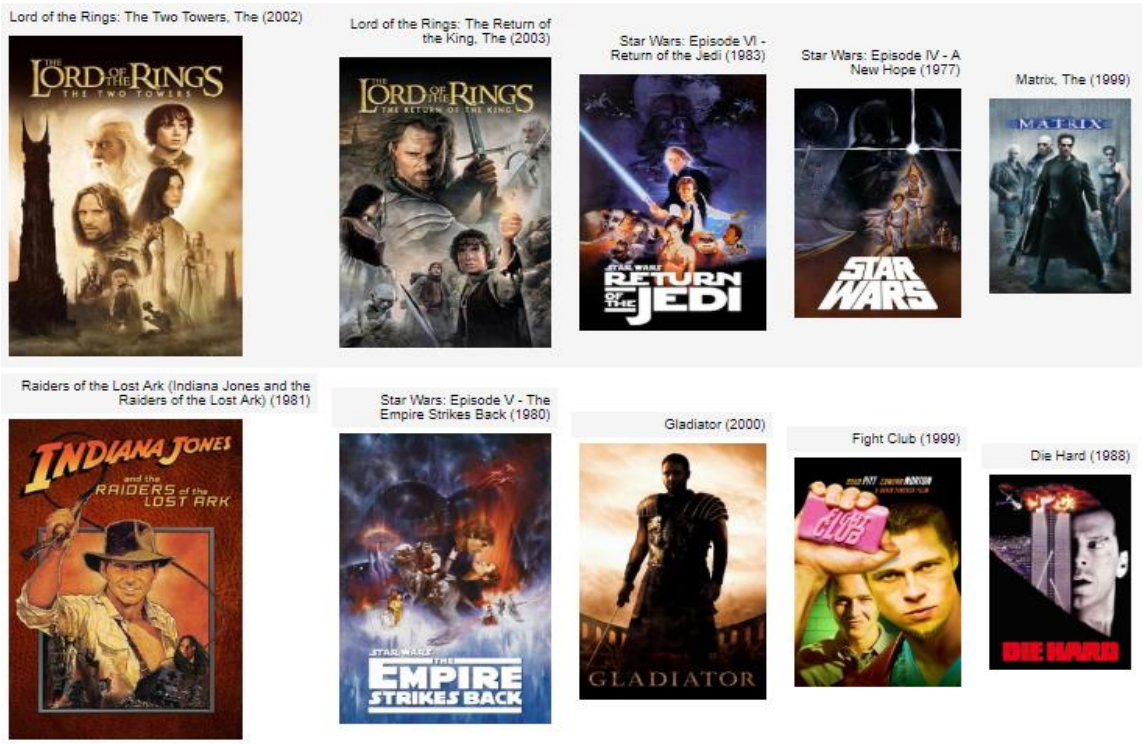

Figura 16

Foi um bom resultado, visto que os mais recomendados foram as sequências do filme. E os outros são filmes que possuem gêneros aventura, fantasia ou ação, os mesmos de “Lord of the Rings”. Obtive resultados de filmes com gêneros parecidos mesmo sem utilizar os gêneros como parâmetros no treinamento do algoritmo, o que demonstra que foi um bom ajuste para esses dados. Mesmo obtendo uma pontuação mais alta de RMSE com ALS, utilizando SGD obtive um melhor resultado na recomendação.

Implementação: Singular Value Decomposition (SVD)

Na implementação do SVD utilizei os melhores parâmetros para estes dados, de acordo com os que foram encontrados pela Busca de Grade. A **Figura 17** mostra a implementação desse algoritmo no iPython notebook, devidamente comentada.

```
# carregando o dataframe do pandas no Surprise
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df_ratings[['userId', 'movieId','rating']], reader)

# Definindo os parâmetros e treinando o algoritmo
trainset = data.build_full_trainset()
algo = SVD(n_epochs= 50, lr_all= 0.005, reg_all= 0.1, biased=True)
algo.fit(trainset)

# Prevendo as notas que não estão no conjunto de treinamento
testset = trainset.build_anti_testset()
# predictions retorna uma tupla que contém todas as notas estimadas que
# os usuários deram aos filmes
predictions = algo.test(testset)
```

Figura 17

Para retornar as predições defini a função “get_top_n”, encontrada no repositório do Github do Surprise. Para obter as melhores predições para determinado usuário criei a função “best_recommended” que tem como entrada o “userId” e retorna os 10 mais recomendados para o usuário, o gênero do filme que foi recomendado e a contagem total de cada gênero para análise na validação.
A **Figura 18** mostra o retorno dos mais recomendados para o usuário 4.

```
Top 10 de filmes mais recomendados para o usuário 4
Hustler White (1996)
Romance
Drop Dead Fred (1991)
Comedy|Fantasy
Frequency (2000)
Drama|Thriller
Anaconda (1997)
Action|Adventure|Thriller
Nowhere (1997)
Comedy|Drama
Daytrippers, The (1996)
Comedy|Drama|Mystery|Romance
Puppet Master II (1991)
Horror|Sci-Fi|Thriller
Kull the Conqueror (1997)
Action|Adventure
Loaded (1994)
Drama|Thriller
Soul Food (1997)
Drama
```

Figura 18

Avaliação

Validação: Singular Value Decomposition

Durante a fase de validação realizei alguns testes aplicando diferentes valores aos parâmetros do modelo e percebi que a busca de grade não foi satisfatória, pois os parâmetros que obtive com a busca de grade tiveram desempenho com

os dados de teste inferior aos que eu testei manualmente. No caso os parâmetros da busca de grade me trouxeram 0.88 para RMSE e 0.68 para MAE, com 30 repetições de SGD, parâmetro de regularização 0.02. O melhor resultado obtive com 1 repetição de SGD e parâmetro de regularização 0.8, me trazendo um score de 0.97 de RMSE e 0.77 de MAE. Segue a **Figura 19** relatando os resultados.

Repetições SGD	Learning Rate	Regularização	RMSE	MAE
1	0.005	0.8	0.9757	0.7705
1	0.005	0.65	0.9681	0.7645
5	0.005	0.8	0.9596	0.7540
5	0.005	0.65	0.9352	0.7299
20	0.005	0.02	0.9049	0.6971
30	0.005	0.02	0.8891	0.6840

Figura 19

Após encontrar uma melhor pontuação, reajustei os parâmetros para realizar a recomendação e foram retornados outros filmes, acredito que tenha sido uma recomendação mais apurada pelo fato de ter cerca de 0.09 a mais de RMSE do que o ajuste anterior. A reimplementação se encontra no iPython notebook e retornou o resultado da **Figura 20**.

```
Top 10 de filmes mais recomendados para o usuário 4
Van, The (1996)
Comedy|Drama
Mirage (1995)
Action|Thriller
Crash (1996)
Drama|Thriller
Twin Town (1997)
Comedy|Crime
Nowhere (1997)
Comedy|Drama
Keys to Tulsa (1997)
Crime
Amityville Curse, The (1990)
Horror
Soul Food (1997)
Drama
Fly, The (1958)
Horror|Mystery|Sci-Fi
Frequency (2000)
Drama|Thriller
```

Figura 20

Outra forma que usei para avaliar o modelo foi observando os gêneros dos filmes que foram mais sugeridos ao usuário 4 e os gêneros entre os filmes que ele mais gostou, ou seja, deu nota 5. Os gêneros mais bem classificados por ele são de filmes de comédia, drama e ação. Estes mesmos gêneros estão em

maior quantidade nos filmes mais recomendados a este usuário. O que mostra que as recomendações podem ter seguido um padrão relacionado aos gêneros. A implementação de código está no iPython notebook.

A pontuação que obtive foi muito próxima ao do modelo de referência, sendo que não utilizei os valores padrão do Surprise. O modelo de referência utilizou 20 para n_epochs, 0.005 para lr_all e 0.02 para reg_all. Mesmo não atingindo a pontuação de 0.98 para RMSE, obtive uma boa pontuação, 0.97. Mas acredito que o modelo de referência poderia ser melhor avaliado, comparando manualmente os gêneros que foram recomendados a um usuário em específicos, pois o resultado de um sistema de recomendação é um pouco difícil de mensurar. Vai depender muito se o usuário que recebeu recomendações vai realmente gostar dos filmes.

Validação: K-Nearest Neighbour

Como o algoritmo K-Nearest Neighbour nesta implementação não calcula predições para cada K, ou seja, a função “get_neighbors” do Surprise retorna apenas os vizinhos mais próximos e não a nota que este vizinho teria (até porque neste contexto estamos calculando apenas a similaridade entre itens e não a nota que determinado usuário deu para tal item), não é possível utilizar métricas como previsão e revocação para estimar o quão boa foi a recomendação. Então o método que utilizei foi analisando manualmente o conteúdo dos filmes. Como foi visto na fase de implementação, utilizei o filme Toy Story como base para obter os 10 mais similares. Na primeira recomendação houveram 2 filmes, que não se encaixavam no perfil de similaridade do Toy Story, que contém violência e conteúdo inadequado para menores, mas após reajustes consegui obter um resultado satisfatório retornando apenas conteúdo infantil e de animação.

Conclusão

Apesar de ter utilizado uma biblioteca, minha maior dificuldade foi entender a implementação do algoritmo SVD utilizando baselines para poder ajustar o algoritmo, exigiram muitas horas de estudo lendo *papers* e *websites* focados no assunto.
Consegui ter um bom resultado em ambas as recomendações e acredito que funcionaria bem como um modelo para um pequeno cliente, mas certos aspectos do modelo poderiam ser melhorados com mais tempo e mais conhecimento. O objetivo principal do algoritmo é personalizar a experiência do usuário, e por isso um *dataset* com dados implícitos ou explícitos do usuário seria de grande valor para uma recomendação mais personalizada. E dados como diretor do filme, elenco e país de produção também seriam muito úteis, pois durante a recomendação certos filtros poderiam ser aplicados. Isto aumentaria a complexidade do sistema e demandaria mais tempo.
Durante o desenvolvimento deste projeto tive contato com dois tipos de recomendação com Filtragem Colaborativa. Mas poderiamos combinar

Filtragem Colaborativa com Content-Based Recommendation [8] formando um sistema híbrido [9], superando assim as limitações de cada um dos tipos anteriores. Sistemas híbridos são mais complexos e mais caros para implementar, mas temos muitos casos de sucesso na indústria, como o Google News. Conforme dados de mais usuários e filmes fossem armazenados e *feedbacks* fossem enviados, uma análise exploratória e ajustes no modelo seriam necessários para que as recomendações permaneçam precisas. Este seria um processo cíclico de constante aperfeiçoamento de um sistema de recomendação.

Referências

[1]https://en.wikipedia.org/wiki/Collaborative_filtering
[2]http://cs229.stanford.edu/proj2006/HongTsamis-KNNForNetflix.pdf
[3]https://www.youtube.com/watch?v=P5mlg91as1c
[4]http://www.cs.rochester.edu/twiki/pub/Main/HarpSeminar/Factorization_Meets_the_Neighborhood-_a_Multifaceted_Collaborative_Filtering_Model.pdf
[5]https://goo.gl/e7SA2h
[6]http://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorization.SVD
[7]https://surprise.readthedocs.io/en/stable/algobase.html#surprise.prediction_algorithms.algo_base.AlgoBase.get_neighbors
[8]http://www.fxpal.com/publications/FXPAL-PR-06-383.pdf
[9]https://pdfs.semanticscholar.org/7176/d0f8a5bf35605bde5d9459dd67641410e1d8.pdf