PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Estudo comparativo de algoritmos de sistemas de recomendação de filmes

Pedro Chamberlain Matos

PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC
DEPARTAMENTO DE INFORMÁTICA

Programa de Graduação em Ciência da Computação

Rio de Janeiro, novembro de 2021

Pedro Chamberlain Matos

Estudo comparativo de algoritmos de sistemas de recomendação de filmes

Relatório de Projeto Final, apresentado ao **Programa de Graduação em Ciência da Computação** da PUC-Rio como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Marco Serpa Molinaro

Rio de Janeiro

Novembro de 2021.

Resumo

Matos, Pedro Chamberlain. Molinaro, Marco. Estudo comparativo de algoritmos de sistemas de recomendação de filmes. Rio de Janeiro, 2021. 46 p. Relatório Final de Projeto Final – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Recomendações personalizadas de séries e filmes são um aspecto importante dos serviços online de streaming. Esse projeto teve como principal objetivo a implementação e avaliação de algoritmos utilizados por sistemas de recomendação de filmes. Foram analisados quatro métodos de sistemas de recomendação, dois de filtragem colaborativa e dois de filtragem baseada em conteúdo. Além da análise do resultado desses métodos, foi elaborado um novo método híbrido que utilizou de dois métodos analisados anteriormente, um de filtragem colaborativa baseada na fatoração de matrizes pela decomposição em valores singulares (SVD) e a filtragem baseada em conteúdo utilizando um cálculo de similaridade entre filmes por suas informações textuais. O novo método foi implementado e analisado, apresentando resultados superiores a todas as abordagens anteriores. Os dados de avaliação se basearam em notas de filmes dadas por usuários da plataforma MovieLens.

Palavras-chave

Sistemas de Recomendação, Cinema, Filtragem Colaborativa, Machine Learning, Python

Abstract

Matos, Pedro Chamberlain. Molinaro, Marco. Comparative study of movie recommendation algorithms. Rio de Janeiro, 2021. 46 p. Graduation Project Report – Department of Informatics. Pontifícia Universidade Católica do Rio de Janeiro.

Personalized recommendations for series and movies are an important aspect of online streaming services. This project's objective was to implement and evaluate a series of algorithms used by movie recommendation systems. Four recommendation algorithms were analyzed, two by method of collaborative filtering and two by method of content-based filtering. In addition to the analysis of these, a new hybrid method was developed using two of the priorly analyzed algorithms: a collaborative filtering algorithm based on matrix factorization by singular value decomposition (SVD) and a content-based filtering algorithm using a similarity calculation of textual information between movies. The new method was implemented and analyzed, exhibiting better results than all previous methods. The evaluation data was based on movie ratings given by users of the MovieLens social platform.

Keywords

Recommender Systems, Films, Collaborative Filtering, Machine Learning, Python

Lista de Figuras

Lista de Tabelas

Sumário

1. Introdução

Desde o surgimento dos serviços de streaming de filmes em 2007 com o advento da Netflix, o acesso a uma quantia infindável de conteúdos digitais pelo público geral tem crescido de maneira exponencial. Antes disponíveis apenas pelo intermédio de mídias físicas como videocassetes e discos ópticos, a modalidade do streaming revolucionou a forma de distribuição de conteúdos multimídia pela Internet, tornando-os mais acessíveis e de mais fácil utilização [1].

Apesar da facilidade de acesso à informação online que nos é concedida por robustas ferramentas de busca elaboradas por empresas como a Google, é cada vez mais difícil nos decidirmos sobre o que consumir no nosso tempo livre. Apesar do nosso acesso à cultura nunca ter sido tão grande quanto é agora, receber recomendações certeiras sobre o que consumir tem se tornado uma tarefa árdua, cada vez mais difícil com o constante influxo de novidades no mundo do cinema, da televisão, da indústria musical etc.

Apesar de recomendações soltas em conversas com colegas ou por imensas campanhas de marketing ocasionalmente servirem nossos interesses, raras são as vezes em que essas sugestões acertam em cheio as afinidades particulares dos consumidores. Um estudo de 2016 feito pela Reelgood e Learndipity Data descobriu que usuários da Netflix demoram, em média, 18 minutos para escolher um filme do seu catálogo para assistir, isso quando um usuário não desiste por completo de assisti-lo. Em comparação, espectadores de TV à cabo conseguem selecionar um canal para assistir na metade desse tempo, em 9 minutos [2].

Para resolver essa questão, uma das soluções mais estudadas e utilizadas pelo mercado de streaming é a implementação de **sistemas de recomendações**, que utilizam um conjunto de ferramentas e algoritmos, geralmente implementados a partir de técnicas de Machine Learning, que são utilizados para recomendar itens aos seus respectivos usuários [3]. Aplicando essa ideia para o mundo do cinema, um sistema de recomendação ajuda seus usuários a encontrar mídias como filmes, estes sendo retornados numa lista ranqueada após uma análise de sugestões de outros usuários ou do conhecimento prévio de preferências do usuário principal. Segundo um estudo da Netflix, 80% do tempo que seus usuários passam dentro da plataforma deve-se aos seus sistemas de recomendação, demonstrando o quão fundamental a aplicação desses sistemas é para o melhor rendimento da plataforma de streaming [4][5].

1.1 Objetivo do Projeto

O objetivo deste trabalho é fazer um estudo comparativo entre a qualidade das recomendações feitas por sistemas de recomendação de filmes, utilizando uma série de métodos e algoritmos diferentes a partir de testes com uma base de dados de filmes avaliados por usuários. Foi elaborado também, com o intuito de melhorar os resultados anteriormente analisados, um novo modelo de sistema de recomendação híbrido, utilizando os algoritmos de filtragem colaborativa e de filtragem baseada em conteúdo que possuíram os melhores resultados nas avaliações anteriores.

Para a filtragem colaborativa, os algoritmos analisados foram quatro diferentes variações do algoritmo de K-vizinhos mais próximos (popularmente conhecida como KNN), que avalia a média das avaliações dos vizinhos de cada usuário para predizer suas avaliações, e o algoritmo de SVD, que elabora recomendações a partir de uma fatoração de matrizes da base de dados pela técnica de decomposição em valores singulares.

Para a filtragem baseada em conteúdo, os algoritmos analisados foram os de árvores de decisão e um algoritmo de cálculo de similaridade usando *Word Embedding,* usando a técnica de Word2Vec, a partir da informação textual associada aos filmes.

Os algoritmos foram implementados e testados a partir da linguagem de programação Python, uma das mais populares atualmente no que se trata o uso de técnicas de Machine Learning [6]. Outro motivo para a escolha da linguagem foi a diversidade de bibliotecas de Python que poderiam servir de auxílio ao estudo, como a bibliotecas do scikit-learn, que possui uma série de algoritmos de Machine Learning [7], e o Surprise, uma biblioteca para a construção e análise de sistemas de recomendação baseados em filtragem colaborativa que lidam com dados explícitos de classificação [8].

1.2 Cronograma

O cronograma elaborado para o projeto se desenrolou da seguinte maneira:

Na primeira etapa do projeto, o aluno realizou um estudo mais aprofundado sobre sistemas de recomendação, machine learning, e os métodos selecionados de filtragem colaborativa e filtragem baseada em conteúdo. O intuito original era elaborar um documento separado relatando cada método, mas a própria elaboração do primeiro relatório de Projeto Final I, com o material bibliográfico utilizado, foi suficiente.

Na segunda etapa do projeto, foi delimitado como os modelos seriam implementados em código Python. As bibliotecas de auxílio para o desenvolvimento também foram selecionadas. Os sistemas de recomendação baseados em filtragem colaborativa, que seriam implementados na etapa seguinte, acabaram por ser implementados antecipadamente. Nesse mesmo período, houve a escrita do Relatório de Projeto Final I.

Na terceira etapa do projeto, os modelos de filtragem baseada em conteúdo foram implementados em código Python de acordo com os planejamentos anteriormente concebidos.

Na quarta etapa do projeto, o desempenho e a precisão dos modelos implementados pelo projeto foram comparados entre si. Com base nesses testes, um novo sistema de recomendação foi implementado, os métodos mais bem sucedidos de filtragem colaborativa e de filtragem baseada em conteúdo. Após isso, o desempenho desse algoritmo foi testado e comparado aos anteriores.

Na quinta e última etapa do projeto, foi redigido e revisado o Relatório do Projeto Final II, registrando os resultados dos estudos comparativos.

Tabela 1 – Cronograma do Projeto

Etapas	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro
1ª	X	X						
2ª		X	X					
3ª				X	X			
4ª						X	X	
5ª							X	X

2. Situação Atual

Como sistemas de recomendação podem ser aplicados de maneiras vastamente diferentes, mas não existe necessariamente uma aplicação que funciona sempre da melhor maneira. A escolha do método depende de uma série de fatores relacionados aos tipos de informações disponíveis ao desenvolvedor em respeito aos respectivos itens e usuários de um sistema. Ao tratarmos de filmes, por exemplo, temos diversas informações disponíveis, como o título e ano de lançamento, seu orçamento, diretores e atores, assim como as próprias avaliações dos usuários de uma rede, podendo elas serem medidas a partir da coleta de avaliações. Essas avaliações podem ser explícitas (estrelas, marcações de “gostei” e “não gostei”) ou implícitas (tempo assistido por filme).

As abordagens mais popularmente conhecidas para a implementação de sistemas de recomendação são as de filtragem colaborativa e a baseada em conteúdo [3]. Exploraremos conceitos relevantes a elas a seguir.

2.1 Filtragem colaborativa

A filtragem colaborativa foi a primeira abordagem a ser estudada para a elaboração de sistemas de recomendação [9]. A ideia por trás dela parte do princípio de que: se dois usuários possuíram opiniões similares sobre os mesmos itens no passado, então eles também irão compartilhar desta mesma opinião no futuro para outros filmes. Para definir isso, ocorre uma coleta e análise das preferências do usuário principal comparada a outros usuários para, assim, determinar os padrões de semelhança entre eles, e utilizar os filmes que um usuário gostou para recomendar a outro usuário com gostos semelhantes.

Existem diferentes formas de implementar a filtragem colaborativa: Aquelas baseadas em memória, que fazem a seleção dos filmes a serem recomendados a partir do cálculo entre a similaridade das notas dos usuários ou itens, e a abordagem baseada em modelos de fatores latentes, que inferem as afinidades do usuário principal a partir da descoberta e análise de certas características implícitas que uma série de itens diferentes possuem em comum. Os métodos mais popularmente reconhecidos para cada uma dessas abordagens são, respectivamente, o de k-vizinhos mais próximos (KNN) e o de fatoração de matrizes, em particular a de decomposição em valores singulares (SVD – *singular value decomposition*) [3]. Para mais detalhes, confira a Seção 3.1 deste projeto.

Apesar do seu uso ser de extrema popularidade por gigantes do mercado, como a Amazon [10], os modelos de filtragem colaborativa geralmente enfrentam uma série de problemas que podem agravar a precisão de suas recomendações:

Existe, por exemplo, a questão do *cold start*, uma situação que ocorre frequentemente na adição de novos usuários, ou filmes, a um modelo. Quando um novo usuário é inserido pela primeira vez num sistema já existente, ele geralmente possui uma quantia precária de informações relacionadas aos outros usuários do sistema, dificultando a classificação dessa entidade pelo modelo. No nosso contexto, podemos pensar na ocasião em que um novo usuário numa rede social de filmes ainda não deu uma nota sequer para os filmes disponíveis.

Existe também a questão de escalabilidade, outra situação que acaba por ocorrer primariamente em sistemas com maiores bases de dados com milhares de usuários e filmes, que requisita um poder de computação mais robusto pela empresa da plataforma.

2.2 Filtragem baseada em conteúdo

Ao contrário da filtragem colaborativa, a filtragem baseada em conteúdo tem como partida a utilização exclusiva das informações dos itens, como por exemplo, no caso de filmes, o diretor, os atores, o ano de lançamento, gênero e até, possivelmente, imagens do filme. A ideia parte do princípio de que: se um usuário possuiu uma opinião sobre um item com certas características no passado, então ele terá uma opinião similar sobre um outro item com características similares no futuro.

A filtragem baseada em conteúdo prioriza itens que possuem características similares aos quais o usuário avaliou com uma boa nota, e tende a evitar outros itens que ele deu avaliações inferiores. Como essa abordagem consiste em comparar os atributos do perfil com dados conhecidos sobre os filmes (título, diretor, duração etc.), as classificações de filmes feitas por outros usuários não são utilizadas.

A filtragem baseada em conteúdo pode ser implementada de formas vastamente diferentes. Dentre elas, podemos destacar como exemplo a utilização de árvores de decisão. Nesta abordagem, a árvore de decisão constrói um modelo preditivo que mapeia a entrada (por exemplo, um filme) para um valor previsto pelas preferências do usuário.

Outra abordagem notável de filtragem baseada em conteúdo é a de *Word Embedding* (em inglês, incorporação de palavras), que aborda técnicas da área de processamento de linguagem natural. Esta subárea da computação tem como intuito a geração e compreensão computacional de linguagens naturais (ou seja, a língua comum entre seres humanos) a partir de técnicas de *machine learning*.

No caso do modelo de Word Embedding, palavras são coletadas de textos simples por uma mineração de dados, sendo posteriormente serializadas em valores numéricos reais em um espaço vetorial com tamanho pré-definido. Cada palavra é mapeada para um vetor, sendo estes valores apreendidos de uma maneira similar a uma rede neural, ou seja, somos capazes de utilizar dados do material textual coletado para prever certos comportamentos oriundos dele.

Aplicando esta abordagem para o nosso contexto de filmes, podemos recolher informações textuais relacionadas ao que nosso usuário principal gosta (ex.: sinopse) e formular um modelo que determina as suas preferências. Para identificarmos se um filme não-visto pode ser de interesse de um determinado usuário, podemos conferir as informações textuais dele e compará-las ao modelo elaborado. Se estas informações forem suficientemente similares, ele é recomendável.

Assim como a filtragem colaborativa, a filtragem baseada em conteúdo enfrenta seus próprios problemas. Como o modelo só faz recomendações com base nos interesses de um único usuário, ele acaba tendo uma capacidade limitada de expandir os seus interesses já existentes, podendo sofrer de recomendações demasiadamente tendenciosas. Existe, também, a possibilidade de enfrentarmos certos problemas relacionados ao detalhamento informativo dos filmes. Como as características selecionadas para representar filmes são definidas à mão, as recomendações podem acidentalmente ter um viés que tornam elas imprecisas. Usuários podem discordar sobre como um filme é caracterizado pela plataforma. Pensemos num exemplo onde uma plataforma descreve um filme como uma comédia, mas um usuário que gosta de comédias crê que se trata de um drama. Se ele der uma nota baixa para esse filme, o sistema então entenderá que ele não gosta de comédias.

2.3 Filtragem híbrida

A filtragem híbrida funciona a partir da combinação de duas ou mais técnicas de filtragem colaborativa e filtragem baseada em conteúdo. Essa junção é feita com o intuito de complementar as limitações de cada uma destas abordagens, evitando os problemas inerentes de cada sistema. Como citado anteriormente, os algoritmos de filtragem colaborativa possuem o problema de *cold start* de usuário, que dificulta a inserção de novos usuários e/ou filmes num sistema de recomendações. Se elaborarmos, no entanto, um algoritmo híbrido que combina a filtragem colaborativa com a baseada em conteúdo, nós poderíamos evitar ou diminuir o escopo do problema de *cold start,* visto que a filtragem baseada em conteúdo não possui esse problema.

As abordagens de filtragem híbrida tendem a ser as mais utilizadas dentro do mercado porque elas obtêm resultados melhores em comparação aos métodos utilizados individualmente. Entre as aplicações mais populares, temos o exemplo notável do concurso Netflix, um evento elaborado pela empresa de streaming em 2006 que tinha como desafio a criação de um sistema de recomendação que fosse, no mínimo, 10% melhor que o da empresa. O algoritmo vencedor da competição foi um algoritmo híbrido constituído por mais de 100 algoritmos diferentes. Ele nunca chegou a ser utilizado comercialmente pela empresa devido a sua possível aplicação elevar os custos a um panorama não-aproveitável, mas alguns dos algoritmos utilizados pela equipe acabaram por ser reaproveitados no sistema já existente [11].

3. Métodos de Sistemas de Recomendação

Conforme explicado anteriormente neste documento, os sistemas de recomendação são algoritmos criados com o intuito de elaborar recomendações de itens relevantes para seus utilitários, sejam esses itens jogos eletrônicos, livros, produtos ou, como especificado neste trabalho, filmes.

Neste projeto final, foram analisadas cinco implementações diferentes de um sistema de recomendação, duas deles baseadas em filtragem colaborativa, duas baseadas em filtragem baseada em conteúdo, e uma baseada num método híbrido.

Consideramos métodos que utilizam avaliações dos usuários, ou seja, um valor dentro de uma escala numérica entre 0.5 e 5.0 que indica o nível de quanto um usuário gostou ou não de um determinado filme.

3.1 Sistemas Baseados em Filtragem Colaborativa

Os sistemas baseados em filtragem colaborativa são aqueles que utilizam uma série de usuários para recomendar filmes para usuários semelhantes. Eles podem ser divididos em duas categorias: os que são baseados em vizinhança e aqueles que são baseados em modelos.

Para explorar os métodos de vizinhança e de modelos, foram selecionados respectivamente os algoritmos de k-vizinhos mais próximos e o de SVD. Nesta seção, descreveremos o funcionamento de cada uma dessas aplicações.

3.1.1 K-vizinhos mais próximos

No método de k-vizinhos mais próximos, as avaliações previstas para um usuário são inferidas a partir de uma análise das avaliações concentradas dentro de uma vizinhança. Essas vizinhanças podem ser definidas de duas maneiras, uma vizinhança baseada em usuários ou uma vizinhança baseada em itens.

3.1.1.1 K-vizinhos mais próximos baseado em usuários

No modelo baseado em usuários, nós utilizamos uma matriz de avaliações $M,\ n \times m$, com n usuários e m filmes. O valor de um item da matriz M determina a nota dada por um usuário para um determinado filme.

Quando um usuário u ainda não assistiu a um filme j, o valor dessa avaliação será nulo. Podemos, porém, com base em avaliações já feitas pelo usuário u, "prever" a avaliação do filme j, que ele ainda não assistiu, usando as avaliações de usuários que já assistiram o filme e têm um gosto parecido com o usuário u, ou seja, aqueles usuários que são "vizinhos" dele.

Para identificarmos quais são os usuários vizinhos de cada usuário, ou seja, aqueles que possuem um gosto mais "próximo", nós podemos fazer um cálculo

da distância das notas já existentes utilizando todas combinações de usuários u e v da base de dados.

Para fazermos esse cálculo, podemos utilizar da distância euclidiana entre as linhas da matriz de avaliações M correspondente a dois usuários u e v, onde levamos em conta apenas os itens avaliados por ambos os usuários, denotados pelo conjunto I_{uv}.

$$d(u,v) = \sqrt{\sum_{i \in I_{uv}} (\boldsymbol{r_{u,i}} - \boldsymbol{r_{v,i}})^2}$$

Após feito o cálculo dessas distâncias para todas as combinações de usuários u e v numa base de dados, podemos selecionar um número k que determina a quantia de usuários com gosto mais próximo do usuário n, ou seja, seus "vizinhos de gosto". Esse valor será utilizado para prevermos a avaliação do usuário n sobre um determinado filme m a partir de uma média ponderada das notas dadas por esses k vizinhos. Para esse cálculo, podemos utilizar a fórmula de similaridade de cosseno ou a correlação de Pearson.

$$sim(u,u') = \cos(\theta) = \frac{\boldsymbol{r_u} \cdot \boldsymbol{r_{u'}}}{\|\boldsymbol{r_u}\| \cdot \|\boldsymbol{r_{u'}}\|} = \sum_i \frac{\boldsymbol{r_{ui}} \cdot \boldsymbol{r_{u'i}}}{\sqrt{\sum_i \boldsymbol{r_{ui}^2}} \sqrt{\sum_i \boldsymbol{r_{u'i}^2}}}$$

$$Sim(u,v) = Pearson(u,v) = \frac{\sum_{k \in I_{uv}} (\boldsymbol{r_{uk}} - \mu_u) \cdot (\boldsymbol{r_{vk}} - \mu_{\boldsymbol{v}})}{\sqrt{\sum_{k \in I_{uv}} (\boldsymbol{r_{uk}} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_{uv}} (\boldsymbol{r_{vk}} - \mu_v)^2}}$$

3.1.1.2 K-vizinhos mais próximos baseado em itens

No modelo baseado em itens, nós utilizamos o mesmo esquema da matriz utilizada pela implementação anterior, porém o cálculo de proximidade não é feito entre os n usuários da matriz, mas sim entre os m filmes avaliados.

3.1.2 SVD

Nos métodos de filtragem colaborativa baseados em modelos, ao invés de compararmos individualmente as notas de cada combinação de usuários para diretamente inferirmos essas previsões, uma modelagem resumida dos dados é construída antes das previsões serem realizadas. Podemos utilizar uma série de métodos para inferir correlações entre filmes que não poderiam ser descobertas "por olho" na nossa matriz de avaliações.

O método baseado em modelo selecionado para esse projeto final tem como base a técnica de fatoração de matrizes pela decomposição em valores singulares, ou, apenas, *Singular Value Decomposition* (SVD). O método trata-se de uma

técnica de fatoração matricial que reduz o número de características de um conjunto de dados, reduzindo a dimensão espacial dela de n-dimensão para k-dimensão (onde $k < n$). Essa fatoração permite a descoberta de fatores latentes que os filmes podem compartilhar de forma implícita.

Assim como o método de k-vizinhos mais próximos, ela utiliza uma estrutura matricial onde cada linha representa um usuário u, e cada coluna representa um filme i e os elementos desta matriz são as classificações que são dadas aos itens pelos usuários. A diferença principal é que não utilizaremos essa matriz "pura".

Para encontrarmos as correlações entre os filmes, nós decompomos a matriz de interação usuário-item num produto de duas matrizes retangulares de menor dimensionalidade. Na nossa matriz R inicial, fazemos uma fatoração que retorna essas duas novas matrizes: uma U de dimensões $m\ x\ k$, que pode ser vista como a matriz de usuário, e a V de dimensões $n\ x\ k$, que podemos ver como a matriz de itens. A partir dessa fatoração, temos que: $R \cong U \times V^T$. Na Figura 1, extraída de [3], podemos ver um exemplo dessa fatoração, onde 1 significa que o usuário gostou do filme, 0 significa que o usuário não gostou, e -1 indica as entradas vazias, filmes que o usuário ainda não assistiu. Note que os usuários 1 a 3 possuem, coincidentemente, um interesse maior por filmes do gênero de "história", enquanto os usuários 5 a 7 quase nunca assistem filmes desse gênero, preferindo os de romance. O usuário 4, contrariando os grupos anteriores, possui interesse em ambos os gêneros.

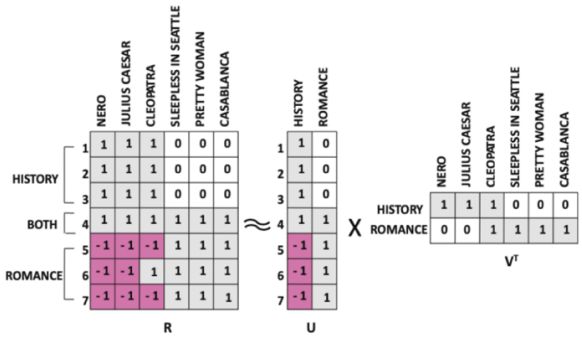

Figura 1 – Exemplo de Fatoração de Matriz

Cada coluna das matrizes U e V representa um vetor latente, onde cada linha representa um fator latente. Na matriz U, de usuário, cada uma de suas linhas representa um fator relacionado a um usuário i, tendo k coordenadas correspondentes à afinidade deste com os k conceitos latentes da matriz R. Já na matriz V, de item, cada uma das linhas representa um item j com cada um desses k conceitos.

O método de SVD permite, a partir da decomposição da matriz R, a descoberta de um conjunto de duas matrizes U e V que possibilitam a elaboração de valores utilizáveis para o cálculo da "predição" das notas dos filmes ainda não-avaliados da matriz R original.

A partir do produto interno de $\langle u_i, v_j \rangle$, onde u_i é um item da matriz U correspondente ao usuário i, e v_j é um item da matriz V correspondente ao filme j, temos como retorno desse produto o valor de $R_{i,j}$, ou seja, a nota prevista pelo modelo do usuário i para o filme j, como descrito na fórmula abaixo:

$$R_{i,j} \cong \langle u_i, v_j \rangle$$

A partir dessa fórmula, podemos predizer a nota R_{ij} de um usuário i para um determinado filme j calculando o produto escalar entre u_i, as afinidades do usuário i a certos conceitos presentes na base de dados, e v_j, as afinidades desses conceitos com o filme j.

3.2 Sistemas Baseados em Filtragem por Conteúdo

Como os sistemas baseados em conteúdo possuem uma base de dados com uma grande variedade de informações não-estruturados sobre filmes diferentes (ex.: sinopses, gêneros, atores etc.), é necessário converter esses dados em valores padronizados e coletar o que é interessante para a predição de notas. Esse passo é conhecido como o de **pré-processamento e extração de *features*** e é o primeiro passo de qualquer sistema de recomendação baseado em conteúdo.

Como um modelo baseado em conteúdo é específico para um determinado usuário, esse modelo precisa ser construído para prever os interesses desse usuário para filmes da base de dados com base nas suas avaliações. Na segunda etapa do sistema, fazemos a **aprendizagem do perfil de usuário com base no conteúdo**, onde verificamos o histórico de avaliações do usuário e criamos um modelo que será referido como o de perfil do usuário, relacionando conceitualmente os interesses do usuário aos atributos de cada filme.

Por fim, o modelo aprendido da etapa anterior será utilizado para fazer recomendações sobre os filmes. Essa última etapa é a de **filtragem e recomendação**.

Para explorar os métodos de filtragem baseado em conteúdo, foram selecionados os métodos que utilizam árvores de decisão e *word embeddings*. Nesta seção, descreveremos o funcionamento de cada uma dessas aplicações.

3.2.1 Árvores de Decisão

No método de árvores de decisão, uma árvore de decisão é construída para um usuário e é utilizada como seu perfil, detalhando suas afinidades com filmes já assistidos. As características de cada um dos filmes e como estes foram avaliados pelo usuário são utilizados como *features* para construir um modelo que explica as preferências do usuário.

Para fazermos a previsão da nota de um filme, inserimos as *features* relacionadas a ele e verificamos como ela se relaciona com o perfil do usuário. De acordo com o valor da *feature*, o filme desce um nó específico da árvore.

Na Figura 2, extraída de [12], podemos ver um exemplo de como esse perfil de usuário pode ser construído.

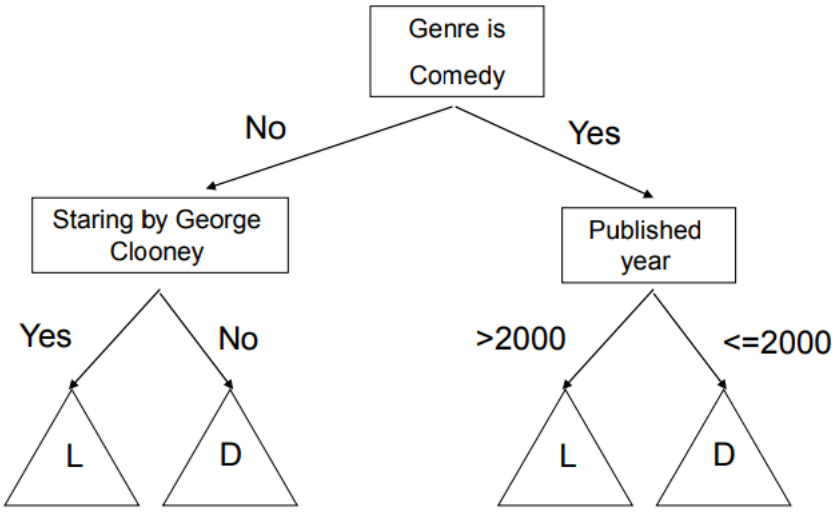

Figura 2 – Exemplo de Árvore de Decisão para um perfil de usuário

A partir da etapa de pré-processamento e a extração de *features*, como no exemplo acima, podemos utilizar as *features* que, por exemplo, identificam o gênero narrativo, ator, e o ano de lançamento dos filmes, para prever se esse usuário gostará ou não de um filme.

A estrutura de árvores de decisão geralmente é utilizada como uma maneira de classificar itens para determinadas classes. No exemplo acima, se o filme não é classificado como uma comédia e não é estrelado pelo ator George Clooney, o usuário provavelmente não gostará do filme, o que faria ele ser classificado como "D" (*dislike*, em português: não gostar). Caso o filme seja classificado como uma comédia e tenha sido lançado depois do ano 2000, ele provavelmente gostaria dele, inferindo-se, portanto, que o filme provavelmente seria de interesse do usuário, sendo classificado como "L" (*like*, em português: gostar).

Como os usuários na base de dados escolhida para este projeto utilizam avaliações explícitas numa escala numérica entre 1.0 e 5.0, foram criadas cinco classes, cada uma simbolizando uma dessas avaliações, determinando então a possível nota do usuário para um filme i. Essa decisão foi feita porque, apesar do banco de dados possuir notas decimais, o uso de todas provocaria a criação de dez classes distintas. A solução para esse problema foi que as avaliações pelo usuário que possuíssem um valor decimal $(0.5, 1.5, \dots, 4.5)$ seriam arredondadas para cima e respectivamente classificadas pelo número inteiro.

3.2.2 Word Embedding

O método de *Word Embedding* descrito a seguir utiliza informações textuais da sinopse dos filmes assistidos e avaliados por um usuário para construir um perfil de usuário.

Como o conceito de Word Embedding não é tão trivial quanto o método de árvores de decisão, é importante entender a fundo do que se tratam *Word Embeddings*, como é feita a técnica de Word2Vec, e como esses conceitos são implementados para prever avaliações de um usuário a partir de informações textuais.

Word Embeddings (em português, incorporações de palavras) são uma classe de técnicas na área de processamento de linguagem natural em que palavras individuais de um corpo de texto são representadas computacionalmente a partir de vetores de valor real em um espaço vetorial pré-definido. Essa representação permite com que palavras semanticamente similares possam ser facilmente comparadas e identificadas a partir de uma representação numérica. Cada palavra é mapeada e representada por um vetor de valor real, muitas vezes com dezenas ou centenas de dimensões.

Para fins de estudo, a utilização de um modelo de *Word Embedding* pré-treinado como o da Google é o ideal, dado que ele já coletou uma série de dados textuais extensa pela Internet. Um novo modelo de *Word Embedding* até pode ser criado a partir de um novo vocabulário com um corpo de texto próprio, mas essa atividade demandaria um trabalho de mineração de dados bastante caro para atingir uma qualidade similar aos modelos da Google, Wikipédia etc. [13][14].

Na Figura 3, extraída de [15], podemos ver como a palavra "*king*" (em português, rei) é representada vetorialmente a partir de um *Word Embedding* pré-treinado da Wikipédia:

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961
, -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 ,
-0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,
-1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```

Figura 3 – Representação vetorial da palavra "king" por um modelo de Word Embedding

Ao introduzirmos outras palavras nesse modelo de *Word Embedding*, podemos ver pela Figura 4, uma representação visual extraída de [15] (onde o vermelho significa que o valor do vetor está perto de 2, branco se estiver perto de 0, e azul se estiver perto de -2), que palavras como "*boy*" e "*man*" (em português, respectivamente "menino" e "garoto") e *"king"* e *"queen"* (em português, "rainha") possuem uma similaridade semântica.

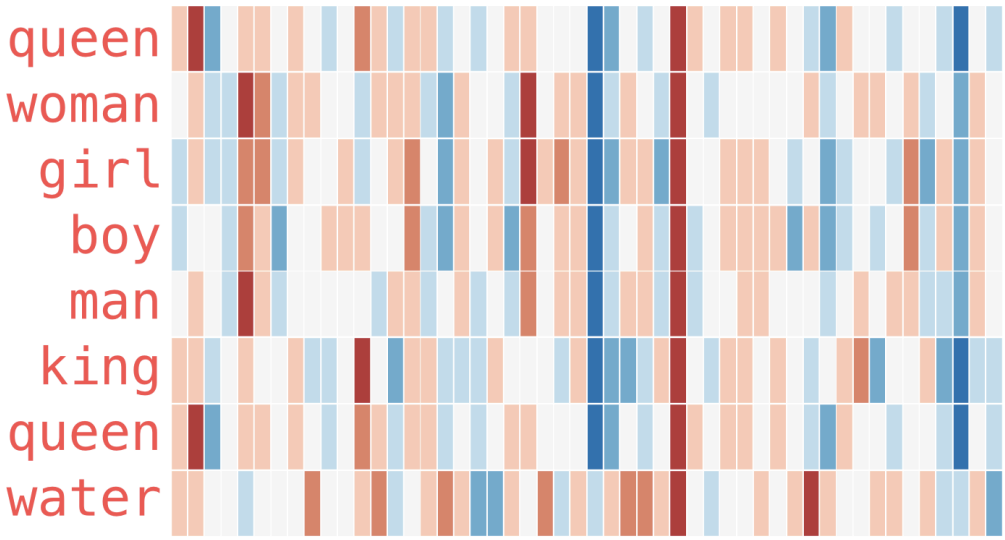

Figura 4 – Representação visual de palavras similares por um modelo de Word Embedding

Existem várias técnicas para construir *Word Embeddings*. Uma das mais populares é a de Word2Vec, que será explicada a seguir.

No contexto deste trabalho, quando um usuário gosta de um filme, nós gostaríamos de reunir uma lista de recomendações com filmes que possuem sinopses similares ao apreciado pelo usuário principal. Ao lidarmos com o conceito de *Word Embeddings*, uma das formas para efetuar esse processo é o de calcular a similaridade da sinopse do primeiro filme com os filmes restantes no banco de dados. Quando todas essas comparações terminam, recebemos uma lista de filmes em ordem decrescente, escalados de mais a menos similar ao filme gostado.

Para implementarmos o método descrito anteriormente, podemos nos utilizar do método de Word2Vec médio [16], um *Word Embedding* que representará por inteiro a sinopse de um filme.

A fórmula de Word2Vec médio recebe de entrada a sinopse de um filme, ou seja, uma sequência de N palavras. Cada uma das N palavras são

respectivamente transformadas em *Word Embeddings* pelo método de Word2Vec. Após todas as palavras da sinopse terem sido transformadas em vetores numéricos, fazemos um somatório entre todos eles. A soma final é, então, dividida pela quantia de N palavras presentes na sinopse, retornando, então, um único vetor numérico. Esse vetor retornado representa a sinopse do filme em apenas um único *Word Embedding*.

Para que todo o banco de filmes de uma plataforma possa ser comparado, a fórmula de Word2Vec médio é rodada por todos os filmes do banco de dados. A similaridade entre os vetores (*Word Embeddings*) dos filmes pode ser feita pela fórmula de similaridade de cosseno entre vetores.

Feito o cálculo de similaridade do primeiro filme com todos os outros restantes no banco de dados, conseguimos retornar uma lista em ordem decrescente, listando-os do mais ao menos similar.

3.3 Sistemas Híbridos

Para fazermos uso do melhor poder algorítmico dos sistemas apresentados nesse resumo até agora, nós podemos confeccionar sistemas de recomendação híbridos que juntam dois algoritmos de tipos diferentes. Esses sistemas podem explorar melhor as diferentes fontes de dados disponíveis e elaborar inferências mais robustas com base nelas

Sistemas de recomendação híbridos podem ser divididos em vários tipos, entre eles: *Weighted* (ponderado)*, Switching* (trocador) e *Cascade* (cascata) [3].

No sistema híbrido ponderado, as avaliações de vários sistemas de recomendação são combinadas em uma única pontuação através do cálculo ponderado entre as pontuações de componentes distintos do conjunto de dados.

No sistema híbrido trocador, o sistema alterna entre vários sistemas de recomendação, dependendo de suas necessidades. O sistema pode, por exemplo, selecionar, com base nos dados fornecidos, um algoritmo que fornecerá a recomendação mais precisa.

No sistema híbrido de cascata, um sistema de recomendação retorna sua lista ranqueada de recomendações e o outro aperfeiçoa essas recomendações, verificando os vieses do sistema de recomendação anterior.

Para este projeto, foi definido que o algoritmo que se sobressaísse comparado aos outros dos seu mesmo tipo (ou seja, um de filtragem colaborativa, outro baseado em conteúdo) seriam combinados para formular um sistema híbrido com formato de cascata, sendo utilizados em conjunto para refinar as recomendações feitas ao usuário.

Como veremos na Seção 4.6, os algoritmos que mais se destacaram nos testes feitos no projeto foram os de SVD e o de *Word Embeddings* usando a técnica de Word2Vec. Por conta desses resultados, foram criados dois sistemas híbridos de cascata. Um sistema que inicia retornando uma lista ranqueada pelo algoritmo SVD, que é posteriormente refinada pelo algoritmo de Word2Vec, e outro sistema que faz o mesmo processo de criação e refinamento, porém começando com o algoritmo de Word2Vec, e terminando com o primeiro de SVD.

3.4 Avaliação dos Sistemas de Recomendação

Antes de prosseguirmos com detalhes sobre a implementação dos algoritmos do projeto, é importante que se tenha um bom entendimento de como a qualidade e eficiência dos sistemas de recomendação é calculada.

Nas subseções a seguir, será explicado como o banco de dados é utilizado para construir cenários de teste para os algoritmos implementados, e quais métricas foram utilizadas para avaliar os resultados destes testes no escopo do projeto.

3.4.1 Metodologia de Avaliação

Para fazermos a avaliação dos sistemas de recomendação descritos nesse projeto, nós precisaremos utilizar um *dataset* contendo um histórico de múltiplas avaliações por uma ampla quantia de usuários. Esse *dataset* será utilizado para fazer a predição de notas e construir recomendações de filmes para seus respectivos usuários. A acurácia e qualidade dessas predições serão então validadas por uma série de métricas que determinarão se o sistema de recomendações é preciso.

Para elaborarmos predições para notas de um usuário, precisamos montar um perfil de gosto para ele. Esse perfil pode ser recolhido do *dataset* a partir de um recorte com todas as avaliações já feitas por ele. Para validarmos as predições feitas com base nesse perfil, temos que aplicar um corte nele, onde teremos acesso dois tipos de grupos de dados: os dados de treino e de teste do usuário.

Os dados de treino baseiam-se em filmes que o usuário já assistiu acoplados às notas que ele os deu no passado. Essas notas serão utilizadas pelo sistema de recomendações para construir esse perfil de gosto e elaborar predições de notas.

Os dados de teste são outros filmes que o usuário já avaliou no passado, mas que não estão presentes nos dados de treino. Por conta dessa ausência, o sistema de recomendação não possui acesso as notas dadas pelo usuário para esses filmes na hora de formular seu perfil de usuário. Como veremos nos

resultados dos testes desse projeto na Seção 4.6, quanto maior for a quantia dos dados de treino em comparação aos dados de teste, melhores serão os resultados.

A partir desses dados, podemos fazer testes onde o sistema de recomendação tentará, com base no perfil de gosto elaborado pelos dados de treino, predizer as notas do usuário para filmes presentes nos dados de teste. A partir das predições calculadas por esses testes, podemos verificar integralmente a similaridade entre as predições com o valor real das notas com base em certas métricas de avaliação. Dependendo do resultado dessa análise, poderemos averiguar se o sistema em questão está fazendo predições confiáveis, em comum acordo com o gosto do usuário, ou não.

Podemos, portanto, resumir o protocolo de teste de qualidade dos modelos de sistemas de recomendação da seguinte maneira:

1. O *dataset* é particionado em dados de treino e teste aleatoriamente. Para que o resultado dos testes fosse mais extenso possível, foi decidido que seriam utilizadas nove partições diferentes. O percentual da primeira partição tem percentual de treino 90%, enquanto o percentual de teste é de 10%. Para as posteriores partições, o valor da divisão seria incrementado por 10% até chegarmos na última partição, onde o treinamento teríamos apenas 10% treinamento e 90% de dados de teste.
2. O perfil de usuário seria criado a partir do pré-processamento de cada um dos algoritmos (ex.: a fatoração da matriz usando SVD).
3. Para cada usuário, seriam utilizados os dados de treino para ser gerada uma lista ordenada de filmes recomendados que não estão no conjunto de treino desse usuário, com as recomendações mais promissoras primeiro na lista.
4. A qualidade das 20 primeiras recomendações da lista será feita por meio das métricas de avaliação, que serão descritas na seção a seguir. Esse número de recomendações foi decidido tendo em conta as limitações do poder de processamento do computador do aluno. Caso o número fosse maior, o procedimento de avaliação demandaria muito mais tempo do que o necessário.
5. Com isso feito para cada um dos usuários, tiraríamos a média das métricas de qualidade retornadas por todas as listas de recomendação.

3.4.2 Métricas de Avaliação

Geralmente, sistemas de recomendação como os descritos acima criam uma lista ranqueada de itens para um usuário, sendo os k primeiros os mais recomendados para o usuário, esse k podendo variar dependendo do sistema, item ou usuário utilizado. No contexto desse projeto, como delimitado anteriormente, $k = 20$.

Para a validação das listas ranqueadas retornadas por sistemas de recomendação descritos anteriormente, foram selecionadas uma série de métricas de avaliação de ranqueamento baseados em utilidade.

O objetivo geral das métricas baseadas em utilidade é encontrar uma quantificação nítida de quão útil o cliente pode achar a classificação do sistema de recomendação. Definiremos aqui um conjunto I_u, uma variável que representa os filmes que já foram avaliados por um usuário u, mas que serão separados do perfil do usuário para serem utilizados como os dados de teste para os modelos de recomendação. Nosso objetivo é medir o quão bem a ordem dos itens recomendados pelos sistemas reflete o conjunto I_u.

3.4.2.1 R-Score

Em muitos modelos de recomendação, o usuário costuma levar em conta apenas um conjunto pequeno dos primeiros itens que o são recomendados [8]. No nosso contexto, podemos esperar que os usuários observem apenas alguns filmes do topo da nossa lista de recomendações. A métrica de R-Score supõe que a qualidade das recomendações diminui de maneira exponencial conforme descemos a lista ranqueada para cada usuário u.

Seja $r_{u,i}$ a nota real que o usuário u dá para um filme i, v_i a posição que o filme i aparece na lista de recomendações (e.g. $v_i = 1$ para o primeiro filme recomendado), C_u a média de todas notas do conjunto de treino do usuário u, que delimita o ponto neutro de interesse do usuário, e a um parâmetro de "meia vida", que controla o declínio exponencial do valor das posições na lista ranqueada:

$$\sum_{i \,\epsilon\, I_u} \frac{\max\{r_{u,i} - C_u, 0\}}{2^{\frac{v_i - 1}{a-1}}}$$

A ideia da fórmula é que, quanto mais distante do início da lista o filme aparecer, menos relevante seu valor é. Quando o filme i está na posição $v_i = a$ na lista de recomendação, dividimos pela metade o cálculo $\max\{r_{u,i} - C_u, 0\}$ do

filme i para que ele perca exatamente metade do seu valor. Se o filme aparecer depois de a na lista, seu valor será menor ainda [3].

3.4.2.2 Soma das notas reais

A fórmula da soma das notas reais é uma versão simplificada da fórmula de R-Score, onde o parâmetro de "meia vida" $a = \infty$ e a média das notas no conjunto de treino do usuário $C_u = 0$.

$$\sum_{i \in I_u} r_{u,i}$$

3.4.2.3 Erro de predição

Além da qualidade das listas ranqueadas, precisamos verificar também se o perfil de usuário sendo elaborado pelos modelos está calculando previsões de notas que são precisas. Para conferirmos isso, podemos fazer usar a fórmula de erro de predição.

O cálculo do erro da predição do modelo é feito pela subtração de cada uma das notas previstas dos filmes i pelo usuário u, representadas por $p_{u,i}$, pelas notas reais $r_{u,i}$ que o usuário u já havia dado para os mesmos filmes.

$$\sum_{i \in I_u} | \, p_{u,i} - \, r_{u,i} \, |$$

4. Implementação e Avaliação dos Sistemas de Recomendação

O trabalho realizado consistiu no estudo, implementação e comparação de resultados da avaliação de sistemas de recomendação de filmes. Foram elaborados uma série de scripts a partir de notebooks Jupyter pela linguagem Python.

Os detalhes da implementação serão abordados nas próximas subseções.

4.1 Seleção do Dataset

É impossível testar um sistema de recomendações sem uma base de dados, e para que essa análise seja próspera, a base precisa ser suficientemente robusta. Para progredir com o projeto, era necessário buscar um dataset para testarmos as implementações e suas recomendações.

À primeira vista, o dataset fornecido pela MovieLens aparentava possuir dados suficientes para testarmos todos métodos a serem analisados [17][18]. Foi conferido, porém, que esse dataset guardava dados relacionados à avaliação dos usuários referentes aos filmes os quais eles já haviam assistido, mas não haviam valores relacionados ao conteúdo sobre os filmes (data de lançamento, sinopse, atores etc.). Por conta dessa falta informacional do dataset, seria possível apenas a validação dos sistemas de recomendação baseados em filtragem colaborativa, visto que neles não se utiliza nada além das notas feitas por outros usuários. Foi encontrado, porém, uma versão incrementada do dataset da MovieLens no Kaggle, em domínio público, que possuía metadados sobre os filmes avaliados [19]. Este dataset foi, portanto, a base selecionada para a análise dos métodos implementados.

As colunas do *dataset* que foram predominantemente utilizadas para os estudos do projeto foram as seguintes:

- *Film ID*: Um valor numérico que corresponde aos IDs dos filmes registrados na plataforma MovieLens. Nesse *dataset,* foram encontrados 45569 IDs de filmes distintos.
- *User ID*: Um valor numérico que corresponde aos IDs dos usuários registrados na plataforma MovieLens. Nesse *dataset,* foram encontrados 671 IDs de usuários distintos.
- *Original Title*: Um valor textual que corresponde ao nome de um filme.
- *Original Language*: Um valor textual que corresponde à língua falada, ou primariamente utilizada, no filme.
- *Genres*: Uma lista com os gêneros pertencentes a um filme. Todo filme do *dataset* precisa ter, no mínimo, um gênero.

- *Overview*: Um valor textual que corresponde a sinopse do filme.
- *Release Date*: Um valor textual num formato DD/MM/AAAA que corresponde à data de lançamento do filme.
- *Rating*: Um valor flutuante que corresponde a nota explícita dada por um usuário para indicar o quanto ele gostou ou não de um determinado filme. A nota precisa ser especificada dentro de uma escala numérica entre 0.5 e 5.0, sendo ela subdividida por cinco décimos. Quanto maior o número, mais o usuário gostou do filme.

4.2 Seleção de bibliotecas Python

Com uma base de dados escolhida para testar os algoritmos, foram selecionados pacotes disponíveis na linguagem de programação Python que poderiam auxiliar na implementação e análise dos algoritmos de recomendação implementados.

Os pacotes selecionados foram:

1. Gensim: uma biblioteca projetada para representação semântica vetorial de textos simples, que seria ideal para a implementação do algoritmo de filtragem de conteúdo baseado em cálculos de similaridade textual entre filmes usando *Word Embedding* [20].
2. Scikit-learn: uma biblioteca que fornece ferramentas eficientes para a construção de modelos de *machine learning,* sendo ela ideal para a implementação do algoritmo de filtragem de conteúdo que utilizará árvores de decisão ;
3. Surprise: uma biblioteca destinada para a implementação e análise de sistemas de recomendação que utilizam dados de classificação explícitos, sendo ela ideal para o desenvolvimento dos algoritmos de filtragem colaborativa;

4.3 Tratamento dos Dados

O primeiro passo da implementação consistiu na verificação da disponibilidade dos valores que poderiam ser utilizados nos modelos de sistemas de recomendação, e a correção de quaisquer problemas associados a eles.

Num geral, os problemas que necessitaram de correção estavam atrelados aos sistemas de recomendação baseados em filtragem por conteúdo.

Referente ao modelo que utilizava árvores de decisão, para que pudéssemos usar valores como gênero, língua original e data de lançamento do filme com o algoritmo da biblioteca scikit-learn, tivemos que fazer uma série de modificações no *dataset*.

A partir de todos os tipos de gêneros narrativos possíveis para os filmes, criamos uma série de features de valor booleano para delimitar se as obras faziam, ou não, parte deles (e.g. *isAction*, *isAdventure*, *isAnimation* etc.) Fizemos o mesmo tratamento com os valores de língua original dos filmes, mas nesse caso, por existirem dezenas de línguas diferentes no banco de dados, utilizamos apenas as cinco mais populares, como observado na seção a seguir (e.g. *inEnglish*, *inFrench*, *inItalian*, *inJapanese*, *inGerman*).

Originalmente, nós iríamos utilizar o valor de bilheteria dos filmes como uma das features, mas foi observado que apenas 16% dos filmes do banco de dados possuíam essa informação, o que impossibilitou o uso dessa coluna como feature no modelo.

Também no modelo de árvores de decisão, foi necessário alterar o valor de data de lançamento dos filmes. Ao invés de utilizarmos o valor inteiro da data de lançamento do filme como originalmente prescrito no banco de dados (DD/MM/AAAA), esse valor foi convertido para representar numericamente apenas o ano de lançamento do filme, fazendo com que o usuário pudesse ser identificado como apreciador, ou não, de filmes lançados antes, ou depois, de um certo ano.

Para o modelo de *Word Embeddings,* foi necessário fazer um rigoroso pré-processamento das sinopses de cada filme. A sinopse de cada filme passou pelas seguintes alterações:

- Remoção de caracteres não-pertencentes à codificação ASCII
- Conversão de todos os caracteres para caixa baixa
- Remoção de palavras de parada, ou seja, as palavras que são irrelevantes para a análise textual das sinopses (ex.: artigos, preposições, conjunções etc.) [21]

- Remoção de código HTML
- Remoção de sinais de pontuação (ex.: vírgulas, pontos de exclamação etc.)

4.4 Análise Exploratória

Com o intuito de conhecer melhor o banco de dados extraído do Kaggle, foi feita uma breve análise dele.

Foi descoberto que ele possui 671 usuários e 45569 filmes diferentes. Foi descoberto, a partir da curva de quantia de filmes avaliados pelos usuários da MovieLens, que ela apresentava uma distribuição de cauda longa, ou seja, a quantidade de usuários que assistem poucos filmes é muito maior em comparação com aqueles que assistem muitos filmes, como se pode ver na Figura 5.

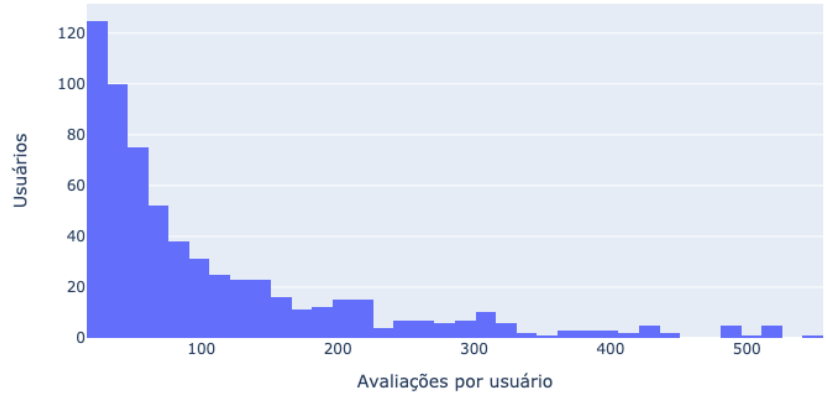

Figura 5 – MovieLens: Quantia de avaliações feitas por usuário

Foi descoberto que, em sua maioria, as avaliações feitas por usuários têm uma tendência a serem positivas. Como se pode ver na Figura 6, todas as variações de nota superiores a 3.0 são muito mais presentes do que qualquer uma das notas iguais ou inferiores a 2.5, tendo como exceção um leve distanciamento entre a quantia das notas 4.5 e 2.0:

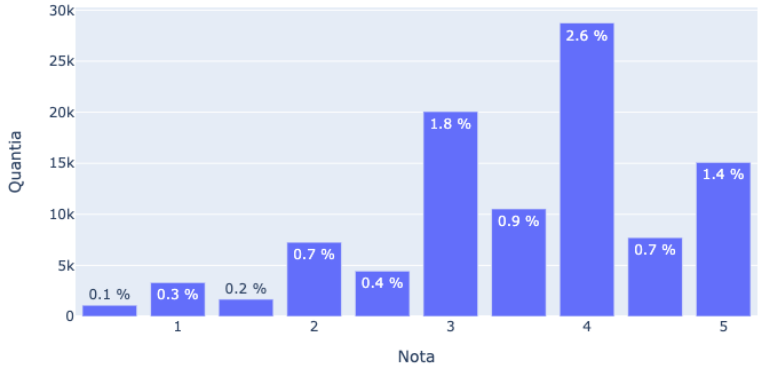

Figura 6 – MovieLens: Variação das notas dadas pelos usuários

Como se pode ver na Figura 7, foi identificado que o banco de dados possuía uma predominância esmagadora de filmes que utilizavam o inglês como sua língua principal, o que poderia tornar os dados enviesados ao cinema norte-americano. Apesar do cinema hollywoodiano possuir um alcance bastante superior aos seus adversários desde a metade do século XX [23], seria interessante a concepção de um banco de dados que possuísse uma quantia mais igualitária entre as diferentes indústrias do cinema mundial.

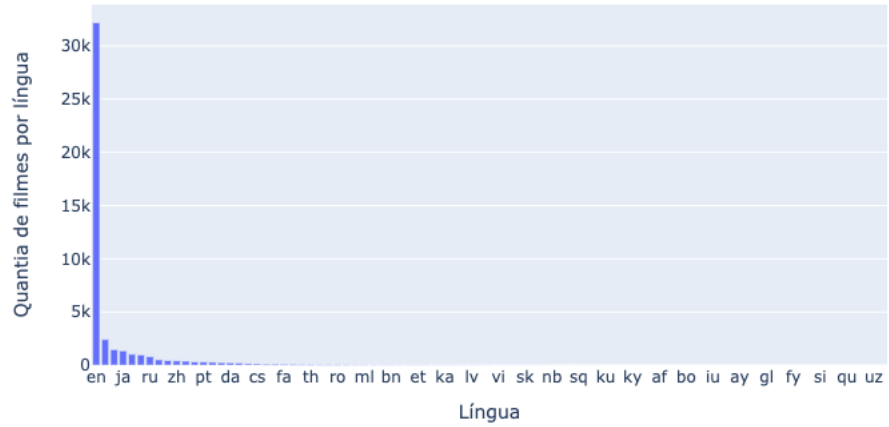

Figura 7 – MovieLens: Quantia de filmes por língua original

Foi identificado, também, pela Figura 8, que mais da metade dos filmes presentes dentro do banco de dados foram lançados após o início do século XXI, o que poderia ser explicado por um viés de recência dos usuários:

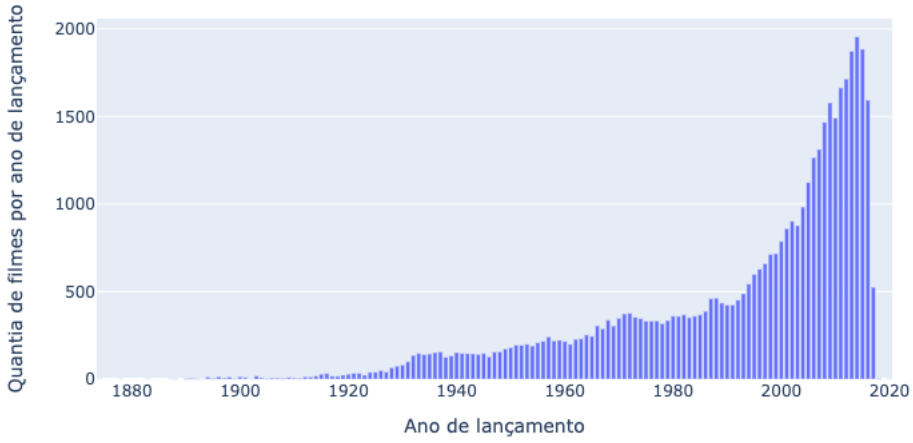

Figura 8 – MovieLens: Quantia de filmes por ano de lançamento

4.5 Algoritmos Comparados nos Testes

Foi gerada uma matriz R com as avaliações explícitas para os filmes já assistidos pelos usuários. Para gerarmos uma lista de recomendações para um usuário u, é necessário prever os valores $r_{u,j}$ dos filmes j faltantes nessa matriz R. As listas de recomendações para cada usuário u serão, então, compostas pelos filmes que geraram os maiores valores $r_{u,j}$.

Para a predição dos valores $r_{u,j}$, foram utilizados os seguintes algoritmos:

- Filtragem Colaborativa, fornecidos pela biblioteca Surprise:
  - KNNBasic
  - KNNBaseline
  - KNNWithMeans
  - KNNWithZScore
  - SVD
- Filtragem baseada em conteúdo:
  - DecisionTreeClassifier, fornecido pela biblioteca scikit-learn
  - *Word Embeddings* usando Word2Vec
- Filtragem híbrida
  - Cascata com SVD de entrada e refinamento por Word2Vec
  - Cascata com Word2Vec de entrada e refinamento por SVD

Uma breve explicação para cada um desses algoritmos será feita a seguir.

Os algoritmos de KNNBasic, KKNWithMeans, KNNWithZScore e KNNBaseline são baseados na inferência por vizinhança de k usuários (veja a Subseção 3.1.1). Para o valor de k, o número de vizinhos a ser levado em conta para a agregação foi definido pelo padrão do pacote Surprise: $1 \leq k \leq 40$.

Os algoritmos do pacote Surprise utilizam a medida de diferença quadrática média (mais conhecida como MSD, *Mean Squared Difference*) para calcular similaridade entre os vizinhos v de um usuário u [8]. Sendo $I_{u,v}$ o conjunto de filmes que avaliados tanto pelo usuário u quanto pelo seu vizinho v, e $r_{v,i}$ a avaliação do vizinho v de u para o filme i:

$$MSD(u,v) = \frac{1}{|I_{u,v}|} \cdot \sum_{i \in I_{u,v}} (r_{u,i} - r_{v,i})^2$$

$$sim(u,v) = \frac{1}{msd(u,v) + 1}$$

O algoritmo **KNNBasic** é o algoritmo padrão para K-vizinhos mais próximos da biblioteca Surprise. Ele utiliza a função de predição mais simples, onde as avaliações $r_{u,i}$ são calculadas em função da similaridade com os vizinhos do usuário u e das avaliações dadas por eles. Sendo $N_i^k(u)$ o conjunto dos k vizinhos de u em relação ao filme i, temos que:

$$r_{u,i} = \frac{\sum_{v \in N_i^k(u)} \text{sim}(u,v) \cdot r_{v,i}}{\sum_{v \in N_i^k(u)} \text{sim}(u,v)}$$

O algoritmo **KNNBaseline** utiliza um estimador *baseline* que modela as avaliações de acordo com três variáveis: a avaliação média de todos os filmes i, definida por μ; a variação do desvio das notas pelos usuários u, ou seja, se os usuários geralmente dão notas mais positivas ou negativas, que é definida por b_u; a variação do desvio das notas recebidas pelos filmes i, que, assim como b_u, confere se os filmes recebem avaliações mais positivas ou negativas, sendo definida como b_i. Mais detalhes sobre a arquitetura deste estimador podem ser encontrados em [22].

Sendo $b_{u,i}$ a soma dos parâmetros do estimador *baseline*, as avaliações $r_{u,i}$ são calculadas pela seguinte fórmula:

$$b_{u,i} = \mu + b_u + b_i$$

$$r_{u,i} = b_{u,i} + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u,v) \cdot (r_{v,i} - b_{v,i})}{\sum_{v \in N_i^k(u)} \text{sim}(u,v)}$$

No modelo de **KNNWithMeans**, o algoritmo utiliza a média das notas feitas pelo usuário u e seus vizinhos. Sendo μ_u a média das avaliações do usuário u e μ_u a média das avaliações do vizinho v do usuário u:

$$r_{u,i} = \mu_u + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u,v) \cdot (r_{v,i} - \mu_v)}{\sum_{v \in N_i^k(u)} \text{sim}(u,v)}$$

O algoritmo **KNNWithZScore** também utiliza dos valores μ_u e μ_v do algoritmo de KNNWithMeans, mas adicionando na fórmula os valores de desvio padrão das médias de μ_u e μ_v, sendo elas, respectivamente, σ_u e σ_v.

$$r_{u,i} = \mu_u + \sigma_u + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u,v) \cdot \frac{(r_{v,i} - \mu_v)}{\sigma_v}}{\sum_{v \in N_i^k(u)} \text{sim}(u,v)}$$

Além dos quatro algoritmos de filtragem colaborativa explicados anteriormente, também foi utilizado o **SVD**, que foi explicado detalhadamente na Subseção 3.1.2.

O modelo de árvores de decisão utiliza o algoritmo **DecisionTreeClassifier** da biblioteca sci-kit learn. Trata-se de um algoritmo não-parametrizado de aprendizagem de árvores de decisão que produz árvores de classificação.

A partir das *features* dos dados de treinamento e um vetor de classes definindo as possíveis notas dos filmes (especificadas anteriormente na Subseção 3.2.1), uma árvore de decisão formula um perfil para o usuário u a partir de uma divisão recursiva das *features* dos filmes, de forma que os filmes com as mesmas classes de avaliação são agrupados em conjunto.

A partir da criação dessa árvore, os filmes i dos dados de treino são testados, um por um, cada um sendo definido como pertencente a uma "classe-nota". A nota $r_{u,i}$ pode ser inferida por essa classe.

O modelo de ***Word Embeddings*** **usando Word2Vec** foi elaborado pelo aluno, tomando como inspiração os algoritmos de filtragem colaborativa baseados em vizinhança. O processo a seguir supõe que o pré-processamento das sinopses dos filmes e a posterior criação de Word Embeddings para cada um deles, pelo método de Word2Vec médio, já foi feito (atividades respectivamente explicadas nas Seções 5.2 e 4.2.2.2).

Para cada filme j não assistido pelo usuário u, calculamos a distância média entre o *Word Embedding* do filme j com o de cada filme i, pertencente ao perfil de treino do mesmo usuário, pela fórmula de similaridade por cosseno. Nesse processo, particionamos uma lista dos k filmes i assistidos pelo usuário u mais próximos do filme j. Para esse projeto final, definimos arbitrariamente um valor de $k = 10$. A fórmula de similaridade por cosseno é feita da seguinte maneira:

$$\cos(\theta) = \frac{Word2Vec\ médio(i) \cdot Word2Vec\ médio(j)}{\|Word2Vec\ médio(i)\| \, \| Word2Vec\ médio(j)\|}$$

Para calcularmos a nota prevista do filme j pelo usuário u, ou seja, $r_{u,j}$, fazemos uma média ponderada das notas reais, ou seja, $r_{u,i}$, dos filmes i

pertencentes ao conjunto dos 20 filmes mais próximos de j, o peso sendo a distância de cada um dos filmes i ao filme j.

Sendo $FP_{u,j}$ a lista dos 20 filmes i assistidos pelo usuário u mais próximos do filme j não-visto, e fazemos a previsão:

$$r_{u,j} = \frac{\sum_{r_{u,i} \, \epsilon \, FP_{u,j}} r_{u,i} \cdot \mathrm{dist}(i,j)}{\sum_{i \, \epsilon \, FP_{u,j}} \mathrm{dist}(i,j)}$$

Os **algoritmos híbridos de cascata**, como explicados anteriormente na Seção 3.3, funcionam a partir da geração de recomendações, feitas por um primeiro algoritmo de entrada que eram posteriormente refinadas por um segundo algoritmo.

Em ambos os casos, o primeiro algoritmo gerava uma lista ranqueada de 20 recomendações de filmes para um usuário u. O segundo algoritmo, então, utilizando-se dos mesmos dados de treino do primeiro algoritmo, recolhia essa lista ranqueada de recomendações e recalculava as notas previstas para o usuário *u* dos filmes presentes nela. A partir dessas novas predições, o segundo algoritmo construía, consequentemente, um novo ranqueamento para a mesma lista.

4.6 Resultados dos Testes

Os resultados dos testes para cada uma das métricas descritas na Subseção 3.4.2 estão registrados nas tabelas citadas a seguir.

Foi conferida a média e o desvio padrão de cada uma das métricas com o intuito de verificar, respectivamente, o valor mais comum e o seu grau de variação dentro do escopo de teste para cada algoritmo.

Para cada dataset de treino, os 3 melhores resultados foram coloridos com variantes de verde. Inversamente, os 3 piores resultados foram coloridos com variantes de vermelho.

4.6.1 R-Score

O primeiro teste realizado levou em consideração o valor de R-Score, descrito na Subseção 3.4.2.1.

Tabela 2 – Resultado dos testes: Média de R-Score por algoritmo

Média do R-Score	90% Treino	80% Treino	70% Treino	60% Treino	50% Treino	40% Treino	30% Treino	20% Treino	10% Treino
KNN Basic	3.45	3.58	3.70	3.76	3.74	3.64	3.43	3.05	2.32
KNN Baseline	3.55	3.62	3.68	3.74	3.70	3.60	3.40	3.03	2.31
KNN With Means	3.43	3.53	3.58	3.63	3.60	3.50	3.32	2.97	2.27
KNN With Z Score	3.46	3.57	3.62	3.67	3.65	3.53	3.35	2.99	2.28
SVD	4.00	3.98	3.95	3.96	3.84	3.71	3.46	3.09	2.32
Árvores de Decisão	1.55	1.57	1.54	1.60	1.64	1.69	1.68	1.57	1.27
Word2Vec	1.64	1.58	1.59	1.61	1.65	1.73	1.69	1.59	1.33
SVD + Word2Vec	5.00	4.87	4.75	4.61	4.50	4.32	4.22	4.02	3.75
Word2Vec + SVD	2.10	1.86	1.56	1.57	1.61	1.62	1.58	1.47	1.14

Tabela 3 – Resultado dos testes: Desvio padrão de R-Score por algoritmo

Desvio padrão do R-Score	90% Treino	80% Treino	70% Treino	60% Treino	50% Treino	40% Treino	30% Treino	20% Treino	10% Treino
KNNBasic	2.28	2.52	2.62	2.67	2.72	2.78	2.78	2.79	3.03
KNNBaseline	2.24	2.50	2.62	2.65	2.70	2.74	2.78	2.79	3.01
KNNWithMeans	2.22	2.46	2.58	2.61	2.67	2.71	2.77	2.78	2.98
KNNWithZ Score	2.23	2.49	2.59	2.61	2.67	2.71	2.77	2.79	3.02
SVD	2.26	2.53	2.65	2.72	2.75	2.85	2.89	2.94	3.24
Árvores de Decisão	1.00	1.16	1.28	1.40	1.40	1.50	1.60	1.92	2.53
Word2Vec	1.04	1.15	1.26	1.36	1.39	1.44	1.58	1.80	2.48
SVD + Word2Vec	1.48	1.74	1.83	1.94	2.07	2.05	2.11	2.15	2.18
Word2Vec + SVD	1.03	1.12	1.18	1.25	1.30	1.37	1.57	1.58	2.24

Na Tabela 2 e 3, podemos observar que o algoritmo híbrido de SVD + Word2Vec obteve a melhor média em todos datasets de treino. Os algoritmos de filtragem colaborativa ficaram conjuntamente em segundo lugar, todos com um

resultado relativamente similar. Entre eles, o SVD e o KNNBasic obtiveram as melhores médias, porém com os piores valores de desvio padrão do conjunto geral de algoritmos.

Apesar dos algoritmos de Árvores de Decisão, Word2Vec, e o algoritmo híbrido de cascata Word2Vec + SVD terem obtido os menores valores de desvio padrão na maior parte dos datasets de treino, eles também foram os algoritmos que obtiveram os piores resultados na média do R-Score. Se desconsiderarmos eles, o algoritmo híbrido com SVD de entrada obteve consistentemente o menor desvio padrão de todos, como podemos observar no gráfico abaixo.

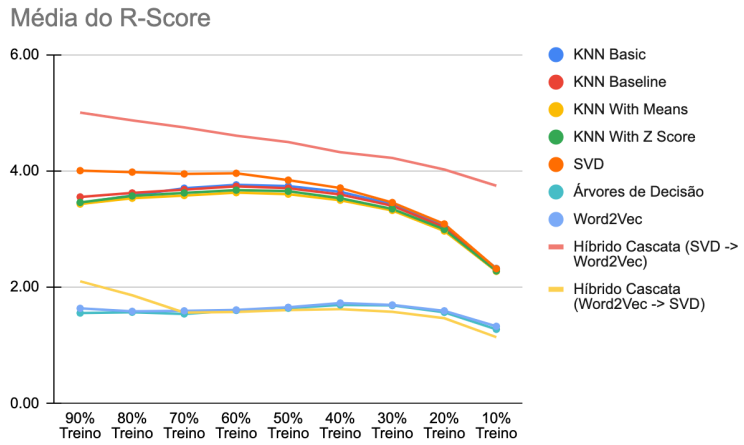

Figura 9 – Gráfico com resultados dos testes: Media do R-Score por algoritmo

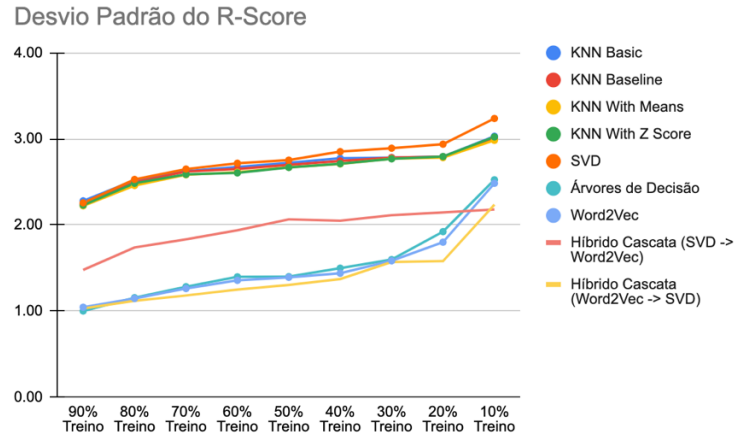

Figura 10 – Gráfico com resultados dos testes: Desvio padrão do R-Score por algoritmo

Conferindo a Figura 9 e 10, podemos verificar mais claramente que o algoritmo híbrido de cascata SVD + Word2Vec manteve consistentemente a melhor média da métrica, assim como um desvio padrão estável. Os algoritmos de filtragem colaborativa obtiveram resultados similares, o SVD tendo nos datasets iniciais um valor consideravelmente superior aos demais, até se adequar.

4.6.2 Soma das notas reais

O segundo teste realizado utilizou como métrica o valor da soma das notas reais, descrita na Subseção 3.4.2.2.

Tabela 4 – Resultado dos testes: Média da soma das notas reais por algoritmo

Média da soma das notas reais	90% Treino	80% Treino	70% Treino	60% Treino	50% Treino	40% Treino	30% Treino	20% Treino	10% Treino
KNNBasic	72.55	72.65	71.86	70.65	67.94	64.13	58.17	49.36	34.41
KNNBaseline	72.97	72.70	71.75	70.55	67.90	64.00	58.14	49.28	34.36
KNNWithMeans	72.03	72.08	71.36	70.06	67.48	63.59	57.80	49.09	34.27
KNNWithZScore	72.21	72.40	71.56	70.16	67.63	63.70	57.94	49.17	34.31
SVD	74.44	73.68	72.36	70.78	67.96	64.10	58.14	49.20	34.34
Árvores de Decisão	65.81	64.34	62.42	59.73	55.72	51.35	45.24	36.18	22.83
Word2Vec	65.52	64.01	62.23	59.28	55.44	51.32	45.09	36.30	22.85
SVD + Word2Vec	75.38	73.63	72.22	71.13	69.37	67.13	66.05	62.63	56.98
Word2Vec + SVD	67.60	59.02	56.14	52.62	52.21	47.83	41.90	33.62	20.24

Tabela 5 – Resultado dos testes: Desvio padrão da soma das notas reais por algoritmo

Desvio padrão da soma das notas reais	90% Treino	80% Treino	70% Treino	60% Treino	50% Treino	40% Treino	30% Treino	20% Treino	10% Treino
KNNBasic	9.88	11.03	12.50	14.11	16.32	19.16	22.27	25.38	26.02
KNNBaseline	9.83	11.10	12.41	14.12	16.33	19.07	22.22	25.31	25.94
KNNWithMeans	10.00	11.07	12.65	14.20	16.36	19.01	22.07	25.19	25.85
KNNWithZScore	10.08	11.24	12.54	14.15	16.32	18.97	22.12	25.24	25.87
SVD	10.08	11.22	12.48	14.18	16.40	19.12	22.25	25.21	25.92
Árvores de Decisão	13.13	14.09	15.53	17.00	18.90	20.37	21.60	21.81	18.02
Word2Vec	12.86	13.93	15.26	16.84	18.57	20.13	21.43	22.00	18.01
SVD + Word2Vec	15.23	16.04	16.70	16.72	16.83	17.20	16.94	17.41	16.32
Word2Vec + SVD	15.04	16.72	18.49	20.04	21.45	22.98	23.69	23.52	19.21

Verificando a Tabela 4 e 5, podemos notar que o algoritmo híbrido de SVD + Word2Vec obteve novamente as melhores médias pelo percurso dos testes, tendo como exceção os casos dos datasets com 80% e 70% de treino em que o algoritmo isolado de SVD o superou, ainda que por uma diferença de centésimos. Os algoritmos de filtragem colaborativa também repetiram os resultados em segundo lugar, com a exceção de que eles chegaram a obter os piores resultados de desvio padrão nos datasets com treino abaixo de 30%.

Os algoritmos de Árvores de Decisão, Word2Vec, e o algoritmo híbrido de cascata Word2Vec + SVD obtiveram, também novamente, as piores médias. Diferente da primeira vez, o desvio padrão se posicionou entre os piores.

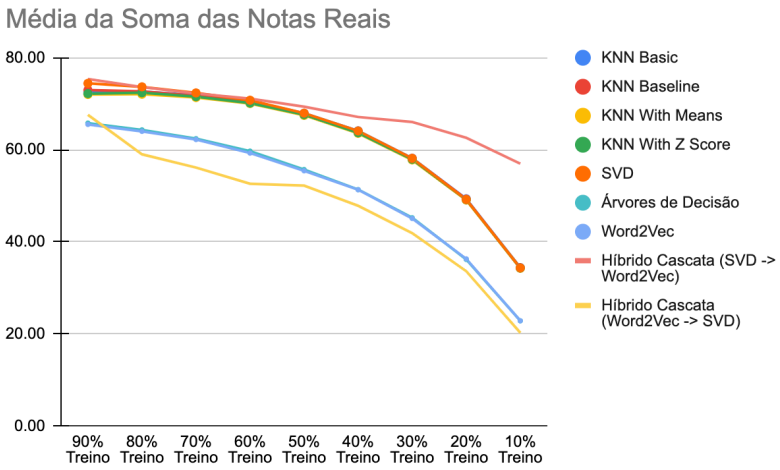

Figura 11 – Gráfico com resultados dos testes: Média da soma das notas reais por algoritmo

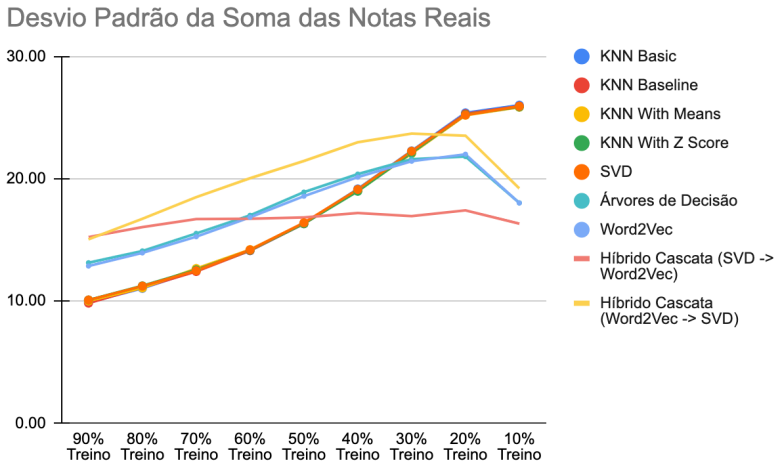

Figura 12 – Gráfico com resultados dos testes: Desvio padrão da soma das notas reais por algoritmo

É interessante notar, a partir da Figura 11 e 12, que, apesar do algoritmo híbrido de cascata SVD + Word2Vec ter iniciado com o maior desvio padrão de todos os métodos no dataset com 90% de treino, conforme a porcentagem de treino diminuiu, o valor se manteve estável ao ponto de ele obter, a partir do dataset com 40% de treino, o menor resultado entre os demais métodos.

4.6.3 Erro de predição

No terceiro e último teste, a métrica utilizada foi o valor do erro da predição da nota, descrita na Subseção 3.4.2.3.

Tabela 6 – Resultado dos testes: Média do erro de predição por algoritmo

Média do erro de predição	90% Treino	80% Treino	70% Treino	60% Treino	50% Treino	40% Treino	30% Treino	20% Treino	10% Treino
KNNBasic	6.47	9.22	10.97	12.13	13.00	13.72	14.52	15.39	16.92
KNNBaseline	6.06	8.60	10.23	11.35	12.18	12.88	13.68	14.58	16.15
KNNWithMeans	6.18	8.84	10.51	11.69	12.53	13.19	14.06	15.02	16.72
KNNWithZScore	6.11	8.69	10.30	11.51	12.31	13.06	13.75	14.46	16.24
SVD	6.08	8.48	10.03	11.08	11.84	12.40	12.83	13.26	13.95
Árvores de Decisão	6.75	10.75	13.42	15.14	17.11	17.97	19.17	19.66	20.40
Word2Vec	5.26	8.17	10.23	11.85	13.14	15.36	18.92	27.81	37.02
SVD + Word2Vec	3.28	3.84	4.14	4.37	4.63	4.78	4.94	5.10	5.32
Word2Vec + SVD	4.66	7.76	10.15	11.47	11.47	17.20	19.02	25.97	13.66

Tabela 7 – Resultado dos testes: Desvio padrão do erro de predição por algoritmo

Desvio padrão do erro de predição	90% Treino	80% Treino	70% Treino	60% Treino	50% Treino	40% Treino	30% Treino	20% Treino	10% Treino
KNNBasic	5.23	5.42	5.20	5.17	5.28	5.55	5.84	6.06	6.17
KNNBaseline	4.90	4.87	4.62	4.52	4.57	4.86	4.98	5.26	5.38
KNNWithMeans	5.04	5.17	4.93	5.00	5.09	5.27	5.47	5.65	5.83
KNNWithZScore	5.02	5.15	4.86	4.86	4.97	5.43	5.46	5.77	6.07
SVD	4.91	4.75	4.49	4.44	4.36	4.56	4.41	4.59	4.81
Árvores de Decisão	5.85	7.48	7.52	7.82	7.86	7.93	7.93	8.19	8.48
Word2Vec	4.29	5.50	9.45	10.88	12.40	14.12	20.48	40.53	82.31
SVD + Word2Vec	1.56	1.85	2.09	2.29	2.40	2.40	2.51	2.50	2.64
Word2Vec + SVD	4.78	7.32	9.22	9.91	11.05	16.20	21.33	30.98	37.72

Observando as Tabela 6 e 7, encontramos mais uma vez que o algoritmo híbrido de SVD + Word2Vec obteve as melhores médias e os menores valores de desvio padrão pelo percurso de todos datasets.

Os algoritmos de filtragem colaborativa não obtiveram resultados tão positivos quanto nos testes anteriores. O algoritmo de KNNBasic, por exemplo, chegou a ter uma das piores médias, excedendo as de Word2Vec no intervalo em que o dataset tinha 90% a 60% de treino, quase se equiparando às médias do algoritmo de Árvores de Decisão, que obteve os piores resultados na média da métrica até o dataset com 40% de treino.

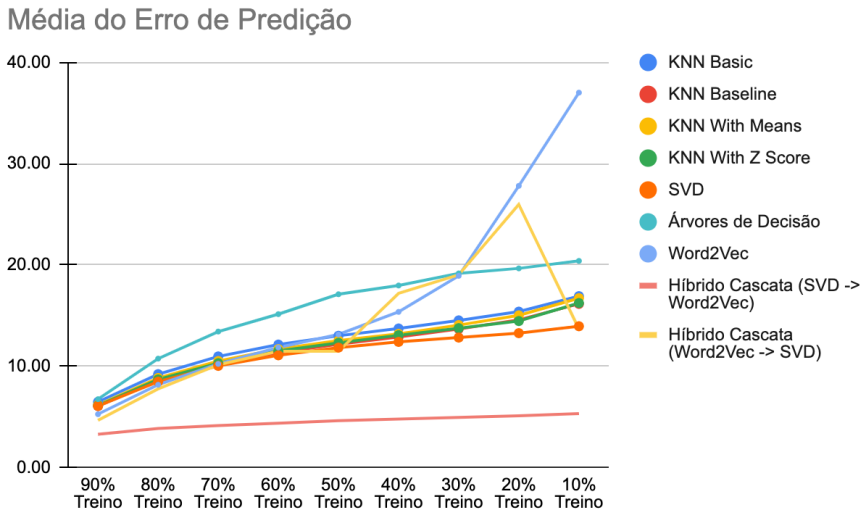

Figura 13 – Gráfico com resultado dos testes: Média do erro de predição por algoritmo

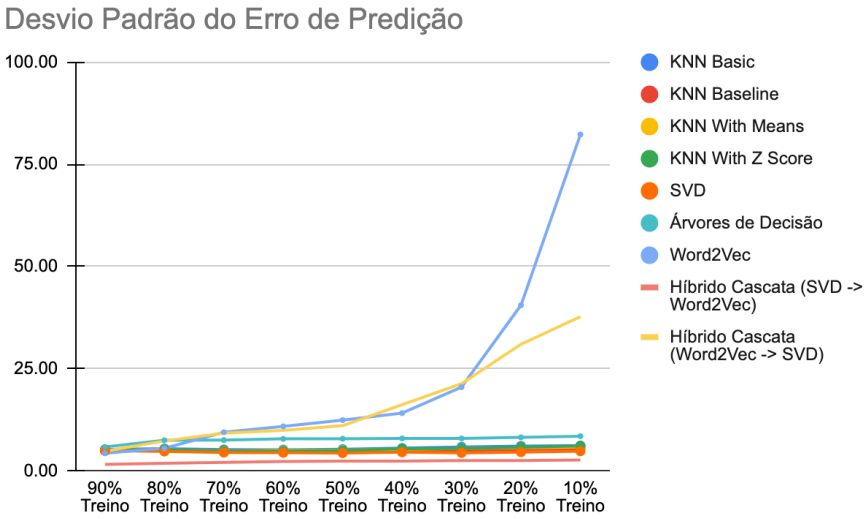

Figura 14 – Gráfico com resultado dos testes: Desvio padrão do erro de predição por algoritmo

Analisando a Figura 13 e 14, podemos observar que o algoritmo híbrido de cascata SVD + Word2Vec não só manteve um erro de predição estável e drasticamente inferior comparado aos demais, como também obteve o menor desvio padrão de todos os datasets de teste. Podemos notar, também, que tanto a média quanto o desvio padrão do erro de previsão dos modelos de filtragem baseada em conteúdo, em particular o de Word2Vec, subiram consideravelmente conforme a perda de dados de treino.

4.6.4 Discussão dos Resultados

A partir de uma breve análise dos resultados das tabelas e gráficos acima, podemos concluir que os sistemas de recomendação que obtiveram os melhores resultados foram: o sistema híbrido de cascata com SVD de entrada e Word2Vec de refinamento, e o algoritmo de SVD isolado.

O sucesso do algoritmo de SVD nos testes do projeto em comparação aos outros sistemas de recomendação de filtragem colaborativa era esperado pela sua imensa popularidade dentro do estado da arte dos sistemas de recomendação comerciais [3][4]. Quanto aos sistemas de recomendação baseados em vizinhança, apesar dos resultados não terem demonstrado uma diferenciação grande entre os quatro algoritmos disponíveis, eles obtiveram resultados positivos quando comparados aos sistemas de filtragem baseada em conteúdo.

Os sistemas de filtragem baseada em conteúdo não registraram dados tão positivos quando comparados aos sistemas de filtragem colaborativa. O algoritmo de Word2Vec, por exemplo, chegou a obter a menor média de erro de previsão comparado aos demais sistemas de recomendação individuais.

Apesar do algoritmo de Word2Vec não ter obtido bons resultados quando utilizado de forma isolada, ele serviu como uma boa ferramenta de refinamento quando acoplado ao algoritmo de SVD no sistema híbrido de cascata, chegando a superar consistentemente a qualidade do algoritmo de SVD quando utilizado isoladamente.

Vale destacar, também, que o refinamento dos resultados de SVD pelo algoritmo de Word2Vec fizeram com que o algoritmo se tornasse bastante estável mesmo quando o tamanho do perfil de treinamento era diminuído, como foi demonstrado nas anteriores subseções. Isso demonstra que o sistema híbrido com SVD de entrada, ainda que não forneça sempre os melhores resultados, pode ser mais confiável pela sua estabilidade.

Já no sistema híbrido com Word2Vec de entrada, onde o algoritmo de SVD foi utilizado para refinar as recomendações do Word2Vec, o sistema híbrido acabou retornando resultados muito piores. Era esperado que, ao reutilizarmos recomendações de um algoritmo que não obteve isoladamente resultados dos mais promissores, o refinamento dessas recomendações poderia torná-las ainda mais distante do gosto do usuário.

O mal sucedimento do sistema de árvores de decisão também era esperado, resultados similares podendo ser encontrado em experimentos relacionados em [12].

5. Considerações Finais

A produção deste projeto foi de grande proveito para o aluno. O projeto não só apresentou uma ótima oportunidade para conhecer a fundo a área de Machine Learning, como também auxiliou na reutilização e recapitulação de ensinamentos previamente feitos em matérias durante todo o percurso feito até agora na faculdade. Vale ressaltar, também, que o tema escolhido era de bastante interesse pelo aluno, e isso o ajudou consideravelmente, mantendo uma contínua motivação para cumprir o seu desenvolvimento.

Uma dificuldade que se fez presente durante toda a realização do projeto do foi a imensa dificuldade em realizá-lo de maneira remota por conta da pandemia causada pela Covid-19. Os esforços da faculdade sanaram alguns problemas, mas o período ainda contou como um grande desafio. Outra questão que dificultou o rendimento do TCC nesse período foi a conciliação horária entre a produção do projeto final com uma grande carga horária de aulas dentro da faculdade, sem contar as 30 horas de trabalho semanais no estágio.

Uma sugestão para futuros projetos é o da elaboração de um novo *dataset* de filmes. Ao analisar as informações dos filmes do banco de dados da MovieLens durante o projeto, foi encontrada uma predominância excessiva de filmes de origem norte-americana comparado a outros países, assim como uma quantia superior de filmes do século XXI comparado ao século anterior. Apesar dessa informação refletir, de certa maneira, o viés de recência dos usuários e o estado atual da indústria cinematográfica mundial, seria interessante a elaboração de um banco de dados que possuísse uma representação maior de filmes fora da indústria norte-americana, talvez por uma base de dados específica para usuários brasileiros, como o Filmow [24].

6. Referências Bibliográficas

1. REDAÇÃO VINDI. **Streaming no Brasil: panorama sobre o crescimento do setor** Disponível em: https://blog.vindi.com.br/streaming-no-brasil/ Acesso em 7 abr.2021
2. MAGLIO, Tony. **Netflix Users Spend 18 Minutes Picking Something to Watch, Study Finds** Disponível em: https://www.thewrap.com/netflix-users-browse-for-programming-twice-as-long-as-cable-viewers-study-says/ Acesso em 14 mai.2021
3. AGGARWAL, Charu C. **Recommender Systems: The Textbook** New York: Springer, 2016.
4. GOMEZ-URIBE, Carlos; HUNT, Neil. **The Netflix Recommender System: Algorithms, Business Value, and Innovation** ACM Digital Library, 2015. Disponível em: https://dl.acm.org/doi/10.1145/2843948 Acesso em 17 mai.2021.
5. CHONG, David. **Deep Dive into Netflix's Recommender System** Disponível em: https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48 Acesso em: 7 abr.2021
6. BEKLEMYSHEVA, Angela. **Why Use Python for AI and Machine Learning?** Disponível em: https://steelkiwi.com/blog/python-for-ai-and-machine-learning/ Acesso em 7 abr.2021.
7. BLONDEL, Mathieu; BRUCHER, Matthieu; COURNAPEAU, David; Dubourg, Vincent; DUCHESNAY, Édouard; GRAMFORT, Alexandre; GRISEL, Olivier; MICHEL, Vincent; PASSOS, Alexandre; PEDREGOSA, Fabian; PERROT, Matthieu; PRETTENHOFER, Peter; THIRION, Bertrand; VANDERPLAS, Jake; VAROQUAUX, Gaël; WEISS, Ron. **Sciki-learn: Machine Learning in Python** Disponível em: https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html Acesso em 10 abr. 2021
8. HUG, Nicolas. **Surprise: A Python library for recommender systems** Disponível em: https://joss.theoj.org/papers/10.21105/joss.02174 Acesso em 10 abr.2021.
9. HARDESTY, Larry. **The history of Amazon's recommendation algorithm** Disponível em: https://www.amazon.science/the-history-of-amazons-recommendation-algorithm Acesso em 7 abr.2021
10. KANTOR, Paul B.; RICCI, Francesco; ROKACH, Lior; SHAPIRA, Bracha. **Recommender Systems Handbook** New York: Springer, 2011.

11. NETFLIX. **Netflix Prize** Disponível em: https://www.netflixprize.com/ Acesso em 17 mai.2021
12. GERSHMAN, Amir; LÜKE, Karl-Heinz; MEISELS, Amnon; ROKACH, Lior; SCHCLAR, Alon; STURM, Arnon. **A Decision Tree Based Recommender System** Disponível em: https://cs.emis.de/LNI/Proceedings/Proceedings165/170.pdf Acesso em: 28 jun.2021
13. RUDER, Sebastian. **On word embeddings – Part 1** Disponível em: https://ruder.io/word-embeddings-1/index.html Acesso em 15 jun.2021
14. GOOGLE. **word2vec**. Disponível em: https://code.google.com/archive/p/word2vec/ Acesso em 12 de set.2021
15. ALAMMAR, Jay. **The Illustrated Word2Vec** Disponível em: http://jalammar.github.io/illustrated-word2vec/ Acesso em 15 jun.2021
16. SUBRAMANIAN, Dhilip. **Content-Based Recommendation System using Word Embeddings** Disponível em: https://medium.com/towards-artificial-intelligence/content-based-recommendation-system-using-word-embeddings-c1c15de1ef95 Acesso em 15 jun.2021
17. KONSTAN, Joseph A.; HARPER, F. Maxwell. **The MovieLens Datasets: History and Context** Disponível em: https://dl.acm.org/doi/10.1145/2827872 2015 Acesso em 9 abr.2021
18. ZHANG, Aston; LIPTON, Zack; LI, Mu; SMOLA, Alex. **Dive into Deep Learning: The MovieLens Dataset** Disponível em: https://d2l.ai/chapter_recommender-systems/movielens.html Acesso em: 10 abr.2021
19. BANIK, Rounak. **The Movies Dataset** Disponível em: https://www.kaggle.com/rounakbanik/the-movies-dataset Acesso em: 10 abr.2021
20. REHUREK, Radim; SOJKA, Petr. **Software Framework for Topic Modelling with Large Corpora** Disponível em: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.695.4595 Acesso em: 15 jun.2021
21. MARCEL, Frank. **Stop Words – Como funcionam palavras de parada?** Disponível em: https://www.agenciamestre.com/seo/stop-words-como-funcionam-palavras-de-parada/ Acesso em 1 out.2021
22. KOREN, Yehuda. **Factor in the Neighbors: Scalable and Accurate Collaborative Filtering** Disponível em:

https://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-koren.pdf Acesso em 7 out.2021

23. SCHMITZ, Andy. **Understanding Media and Culture: An Introduction to Mass Communication** Disponível em: https://saylordotorg.github.io/text_understanding-media-and-culture-an-introduction-to-mass-communication/s11-01-the-history-of-movies.html Acesso em 23 set.2021
24. **Filmow** Disponível em: https://filmow.com/ Acesso em 12 out.2021