Danmarks Tekniske Universitet

DTU

Computational tools for data science: Movie Recommendation System

Authors

Andreas Olborg - s232086
Erik Wahlstrøm - s232047
Nicolai Nossum - s231823
Victor Norris - s232046

Table 1: Production per student.

Student	**What they did**
s232086	TD-IDF code and report
s232047	TD-IDF code and report
s231823	Word embeddings and report
s232046	Word embeddings and report

November 28, 2023

Contents

1 Introduction

In digital entertainment, movie recommendation systems have become integral to guiding viewers through many cinematic options. However, a common limitation of these systems is their confinement within single streaming platforms, leaving audiences with only a small view of available content. This project was initiated to address this gap by developing a cross platform recommendation system, offering a comprehensive solution to the modern cinema lover. In addition to the classical movie recommendation system we wanted to create a system capable of interpreting user-inputted text—be it detailed descriptions or simple buzzwords—and translating it into personalized movie recommendations through the application of word embedding techniques.
To realize this vision, we have employed a combination of data science methodologies, both from within our curriculum and from other sources. The technique we used from the course is Term Frequency-Inverse Document Frequency (TF-IDF) technology, our system evaluates the significance of words in movie descriptions, enabling it to match users' textual inputs with a curated selection of films.

2 Dataset and Prepossessing

2.1 Dataset and Parameters

The backbone of our movie recommendation system is a robust dataset sourced from Kaggle, specifically "The Movies Dataset" by Rounak Banik[1]. This collection of metadata offers a wide array of information on over 45,000 movies from the Movie Database (TMDb). It includes details such as cast, crew, plot summaries, genres, ratings, keywords and more, providing a rich foundation for analysis and recommendation.

2.2 Data Cleaning and Preprocessing

We chose to implement two files for our project, `movies_metadata` and `keywords`. The `movies_metadata` dataset is a collection of movie details, including aspects like genre and overview. These details are crucial for helping our recommendation system suggest movies that you might enjoy. The `keywords` dataset includes a list of words that capture the main themes of each movie, such as 'Adventure' or 'Romance', which aids in fine-tuning the movie suggestions to match your preferences. The first step was to clean up the data. We handled duplicates and made sure that the movie IDs were in the correct format, which is important for further analysis. Then, we looked for missing values in the `overview` and `keywords` attribute, and where data was missing added we an empty string.

Once the data was in order, we merged the `movies_metadata` with `keywords` on `id`. We also extracted the genres information and converted it from a JSON string to a list. This was done because genres are listed as dictionaries, and we are only interested in the string values. Our goal was to consolidate all this information into one place so that we could work

with it more efficiently.

3 TF-IDF

3.1 TF-IDF Vectorization and Cosine Similarity

We chose the TF-IDF Vectorizer for its efficacy in transforming textual data into numerical format, crucial for our movie dataset which is rich in textual content such as plots and genres. This method effectively distinguishes each movie by the uniqueness of its content, thereby enabling a more nuanced and meaningful comparison between films, which is essential for a content-based recommendation system. In the provided code, the TF-IDF Vectorizer is employed to transform the 'combined' column — a concatenation of the movie overviews, keywords, and genres — into a numerical representation that captures the importance of words within each movie's description. This numerical representation, known as the TF-IDF matrix, is instrumental in computing the similarity between different movies based on their textual content. By calculating the cosine similarity between these TF-IDF vectors, the system can identify which movies have similar content and thereby suggest films with similar themes or narratives to the user.

In practical terms, the TF-IDF Vectorizer downplays the role of common words that appear frequently across all movie descriptions (thus offering little to no unique information about any given movie) and amplifies the significance of distinctive words that can help differentiate between movies. This approach allows the system to interpret and respond to detailed descriptions or simple buzzwords with a personalized set of movie recommendations, enhancing the classical movie recommendation experience with the power of word embedding techniques.

3.2 Recommendation Algorithm

The adaptation of the cosine similarity-based recommendation algorithm was motivated by its appropriateness for sparse datasets, such as our TF-IDF vectors, and its capability to evaluate content similarity regardless of movie size. This approach ensures that recommendations are determined based on content relevance, which is fundamental for our system designed to align movie suggestions with user interests in specific themes and narratives. In our project, we developed the Recommendation Algorithm manually, without relying on external libraries. The function `cosine_similarity_manual` computes the cosine similarity by calculating the dot product of TF-IDF vectors, as they are already L2-normalized. This manual calculation of cosine similarity, combined with the ability to filter recommendations based on genres, refines the recommendation process. Our `get_recommendations_with_genre_filter` function first locates the index of the movie that matches the user-provided title, then computes similarity scores, and finally filters these based on the specified genre if needed. By directly manipulating the sparse TF-IDF matrix and integrating genre filters, our implementation enhances the traditional recommendation

system, ensuring dynamic, efficient, and contextually relevant suggestions.

3.3 Genre-Based Filtering in Recommendation Algorithm

In our enhanced recommendation system, we introduced genre-based filtering to complement the content similarity analysis enabled by TF-IDF and cosine similarity. This integration allows the system to refine recommendations not only based on content relevance but also according to specific genre preferences.

Implementation: The code enriches the recommendation process by considering the genre alignment along with content similarity. This dual-layered approach filters the contextually similar movies (obtained via TF-IDF and cosine similarity) further by their genre, ensuring that the recommendations are both topically relevant and genre-specific.

The reason behind incorporating genre-based filtering arises from the understanding that content similarity alone may not suffice to meet diverse user preferences. For example, a user interested in 'cars' might seek family-friendly movies rather than horror films featuring cars. By leveraging the rich genre data available in our dataset, we are able to fine-tune the recommendations, offering a more personalized and relevant selection of movies. This approach enhances user satisfaction by aligning recommendations closely with individual preferences in terms of both content and genre.

4 Self chosen method

4.1 Word embeddings

Word embeddings[2] are a form of word representation that gives words with similar meaning a similar representation. They enable machines to understand words in a more human-like context, and even allow arithmetics with the words. In the context of movie recommendations, using word embeddings allows a machine learning model to understand movie descriptions or reviews more in context. This depth of understanding helps create more accurate and relatable recommendations, as the system can capture subtleties and context in language that go beyond only matching keywords. In this project, we used word embeddings to convert text data(the overview and keywords of movies) into numerical values that can be processed by our neural network. The model then analyzes these vectors to predict the genre of a movie, based on their description(overview).

The use of word embeddings is significant because it goes beyond simple analysis and allows the model to understand context and semantics better, leading to more accurate and relevant recommendations. We ended up with a function where you can write a short summary of a movie. The model then predicts the genre(s) of your summary, and recommends a given amount of movies based on genre and popularity.

4.2 How we did it

For encoding, we used a MultiLabelBinarizer[3] to encode our movie genres into binary format. This allows the model to handle movies with multiple genres. This is important since all movies in our dataset had between 1 and 3 different genres. We also used a Tokenizer[4] from Keras, which converts the combined text into sequences of tokens, keeping a maximum of 2000 words and marking out-of-vocabulary words with a special token (<OOV>). These sequences are then padded to a uniform length for efficient neural network training.

The architecture of our neural network begins with an Embedding layer, which converts tokenized words into dense vectors of a fixed size 64. A LSTM layer(Long Short-Term Memory layer)[5] with return sequences is then used to capture the sequential dependencies in the text. This will help the model understand the context from overview even better. A GlobalAveragePooling1D layer follows, reducing the dimensionality of the output from the LSTM layer. The network concludes with two Dense layers, with the final one using a sigmoid activation function to predict the probability of each genre. This is the best for multi-label classification. We used this information to give the user recommendations of the most popular movies in the genre the model predicted.

5 Results

Our end product is a versatile and intelligent movie recommendation system that delivers on two pivotal functionalities, aimed at enhancing the user experience in discovering cinematic content. Firstly, our **Embedding system** allows for genre classification. The model was able to predict the genres of a given overview with an accuracy of about 50%.This is somewhat impressive considering that all movies have 1 - 3 genres and our model almost never misclassified, but instead was missing one or two genres in its predictions. It uses user input text that can be a string of keywords or a sentence describing mood or theme and our algorithm processes the input through word embedding techniques to ascertain the most likely genre categorization. Upon this determination, the system then presents the user with a list of the most popular movies within the identified genre, thus streamlining the discovery process.

Secondly, the **self made system** offers a more targeted recommendation feature. When provided with a specific movie title and a genre, it utilizes TF-IDF vectors as well as a recommendation algorithm, to scour its extensive movie database and generate a curated list of 10 movies that share thematic similarities with the input title within the given genre. This functionality recognizes the user's genre preference and considers the nuances of their selected movie to find similar cinematic experiences, thereby creating a more tailored selection.

Both features of our system work well in helping try find your next favorite movie.

References

[1] R. Banik, "The movie dataset." https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset, 2017. Username: rounakbanik, Accessed: November 2023.

[2] "Word embeddings." https://www.tensorflow.org/text/guide/word_embeddings, 2023. Accessed: November 2023.

[3] "sklearn.preprocessing.multilabelbinarizer." https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBinarizer.html, 2023. Accessed: November 2023.

[4] "tf.keras.preprocessing.text.tokenizer | tensorflow v2.14.0." https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer, 2023. Accessed: November 2023.

[5] K. Team, "Keras documentation: Lstm layer." https://keras.io/api/layers/recurrent_layers/lstm/, 2023. Accessed: November 2023.