RECFLIX

(*Movies Recommendation System*)

A PROJECT REPORT

Submitted by

MUJTABA SHAIKH

WAISULLAH YOUSOFI

AMBARI FATIMA KHAN

SALMAN KHAN

OWAIS SHAIKH

Under the esteemed guidance of

Prof. CYRUS LENTIN

in partial fulfilment for the award of the degree

of

M. Sc. in DATA SCIENCE & ARTIFICIAL INTELLIGENCE

RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ART'S, SCIENCE & COMMERCE (AUTONOMOUS), GHATKOPAR W

December-2021

ABSTRACT

Recommendation systems are becoming increasingly important in today's extremely busy world. A classic problem people have is finding a good/interesting movie to watch without doing a lot of exploration. To overcome this problem, recommender systems based on Machine Learning is used which helps user to find movies that they are most likely to enjoy. There are 2 main types of recommender systems: Content Filtering (finds similarities between movies by their data), and Collaborative Filtering (finds similar movies by ratings and other data from users). This project uses Streamlit to create a web application that allows the user to choose a movie and uses Content Filtering to recommend movies that are most similar to the one chosen by the user.

INTRODUCTION

Recommendation System

A recommender system, or a recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.

Recommender systems are used in a variety of areas, with commonly recognised examples taking the form of playlist generators for video and music services, product recommenders for online stores, or content recommenders for social media platforms and open web content recommenders. These systems can operate using a single input, like music, or multiple inputs within and across platforms like news, books, and search queries. There are also popular recommender systems for specific topics like restaurants and online dating. Recommender systems have also been developed to explore research articles and experts, collaborators, and financial services.

Recommender systems usually make use of either or both collaborative filtering and content-based filtering (also known as the personality-based approach), as well as other systems such as knowledge-based systems. Collaborative filtering approaches build a model from a user's past behaviour (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that the user may have an interest in.

Types of collaborative filtering are Model and Memory based. The memory-based approach uses user rating data to compute the similarity between users or items. Typical examples of this approach are neighbourhood-based CF and item-based/user-based top-N recommendations. In Model based approach, models are developed using different data mining, machine learning algorithms to predict users' rating of unrated items. There are many model-based CF algorithms. Content-based filtering approaches utilize a series of discrete, pre-

tagged characteristics of an item in order to recommend additional items with similar properties. Most recommender systems now use a hybrid approach, combining collaborative filtering, content-based filtering, and other approaches. There is no reason why several different techniques of the same type could not be hybridized.

Fig : Types of Recommender system

Content based filtering is used in our recflix project.

***Content-Based Recommendation system*:** Content-Based systems recommends items to the customer similar to previously high-rated items by the users. It uses the features and properties of the item. From these properties, it can calculate the similarity between the items.

In a content-based recommendation system, first, we need to create a profile for each item, which represents the properties of those items. From the user profiles are inferred for a particular user. We use these user profiles to recommend the items to the users from the catalogue.

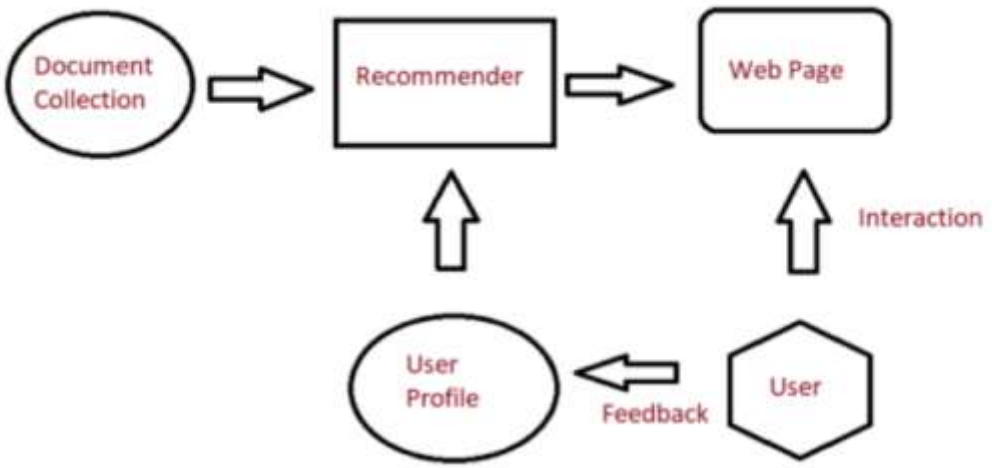

Fig: Content-Based Recommendation System

We used classification approach in the recommendation systems , as we can use the Decision Tree for finding out whether a user wants to watch a movie or not, like at each level we can apply a certain condition to refine our recommendation. For example:

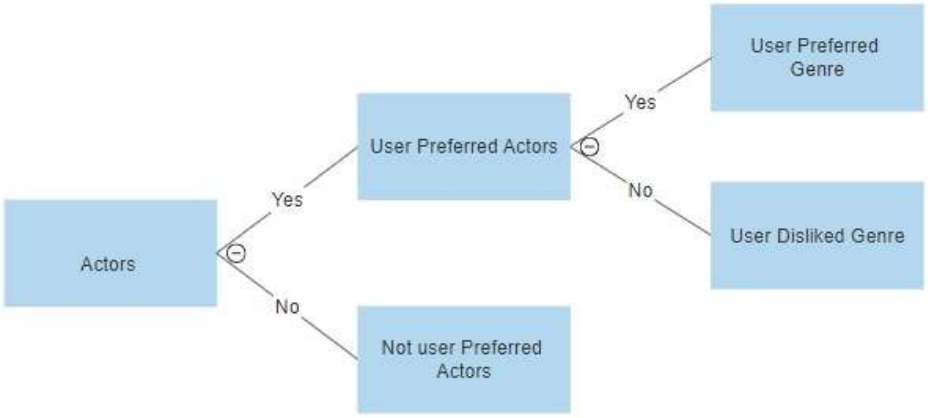

Fig : Decision Tree

INTRODUCTION OF DATA SET

The movie dataset, which is originally from Kaggle, was cleaned and provided by Udacity. According Kaggle introduction page, the data contains information that are provided from The Movie Database (TMDb). It collects 5000+ movies basic move information and movie matrices, including user ratings, popularity and revenue data. These metrics can be seen as how successful these movies are. The movie basic information contained like cast, director, keywords, runtime, genres, etc. And the whole dataset duration covers from 1960 to 2015.In this dataset, two csv-files are provided: tmdb_5000_movies.csv and tmdb_5000_credits.csv. The movies file contains 20 variables on 4,803 movies.

CONTENT FILTERING

Content Filtering is performed on The Movies Dataset, which contains metadata for over 45,000 movies. The main idea behind Content Filtering is that if a user likes a movie, then they will probably like other movies that are similar to it. Movie similarity is determined by the following metadata: Cast, Director, Keywords, and Genre. The metadata is text based, so some basic natural language processing is required before it can compare movies.

Processing the data involves the following steps:

1. Removing stop words such as the, and, or, etc. They add no relevant information, so they are not useful for most types of analysis.
2. Finding the director of the movie from the cast so the director has their own category.
3. Removing spaces between words. This is done so that names and other multiple word terms like "Johnny Depp" and "Johnny Galecki" are not treated the same.

4. Creating tag column which includes combined information viz. overview of the movie, genres, actor's name, cast and crew details.

OBJECTIVE

The main objective of recommendation systems is to provide recommendations based on recorded information on the user's preferences. A recommendation system is a platform that provides its users with various contents based on their preferences and likings. Objective of this project is to build a recommendation engine that recommends movies to users. It also aims at reducing the time taken by to user to do research about movies.

LIBRARIES USED

Streamlit: is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. We can instantly develop web apps and deploy them easily using Streamlit. Streamlit allows you to write an app the same way you write a python code. Streamlit makes it seamless to work on the interactive loop of coding and viewing results in the web app.

Scikit-learn: is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy .

Recommendation is done based on the number of times words have occurred. We converted text into vectors using vectorization technique Bag of words and used scikit learn library "CountVectorizer".

***CountVectorizer*:** is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors (for using in further text analysis).

Using the Cosine Similarity

We used the Cosine Similarity from Sklearn, as the metric to compute the similarity between two movies. Cosine similarity is a metric used to measure how similar two items are. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The output value ranges from 0–1.

The python Cosine Similarity or cosine kernel, computes similarity as the normalized dot product of input samples X and Y. We used the Sklearn cosine_similarity to find the cos θ for the two vectors in the count matrix. In matrix representation the cosine similarity of movie 0 with movie 0 will 1.

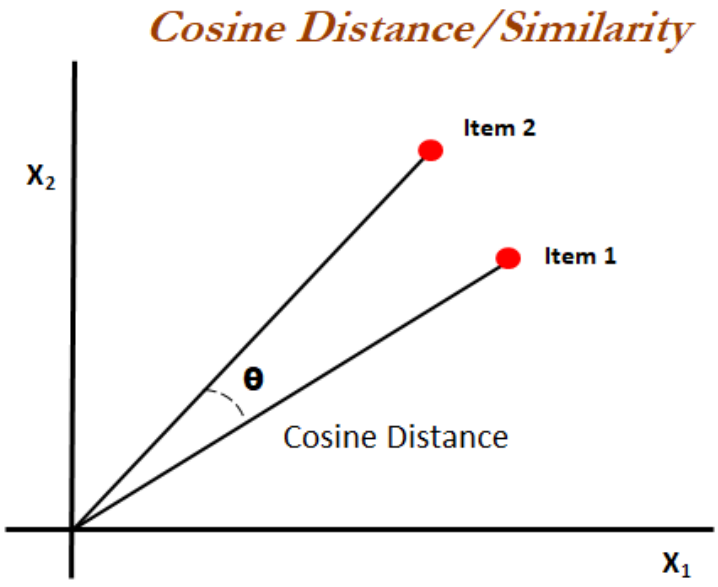

Fig: Cosine Similarity

SOFTWARE USED

Jupyter notebook: The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Its uses include data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more.

Pycharm: Pycharm is a smart code editor. The editor provides first-class support for Python, JavaScript, CoffeeScript, TypeScript, CSS, popular template language and more. Take advantage of language-aware code completion, error detection, and on-the-fly code fixes. Tools like Python, Django, Anaconda, Wakatime, Kite etc. are integrated with Pycharm.

Postman: Postman is an application that allows us to test APIs utilizing a graphical user interface. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated. Postman include the collection feature and the possibility to create different testing environments. Postman is a user-friendly tool that helps us optimize our time when executing tests.

ALGORITHM

Algorithm of feature selection and vector generation method

Step 1: Start

Step 2: Set i = 1

Step 3: Documents in category I (input)

Step 4: Tokenize and stem

Step 5: Compute $X^2 (t^i_j, c_i)$ for unique item j

Step 6: Sort all $X^2 (t^i_j, c_i)$ in descending order

Step 7: Select top M terms to constructs FV^i

Step 8: If i > N_e → ***Go to step 9***

Else i++ and → ***Go to step 3***

Step 9: Component all FV^i and then remove duplicated terms

to generate terms to generate FV

Step 10: End

CONCLUSION

The API developed was tested satisfactorily using postman software and we were able to generate desired recommendation system.

REFERENCE

[1] https://www.kaggle.com/erikbruin/movie-recommendation-systems-for-tmdb/report

[2] https://en.wikipedia.org/wiki/Collaborative_filtering

[3] https://www.geeksforgeeks.org/recommendation-system-in-python/

[4] https://en.wikipedia.org/wiki/Cosine_similarity

[5] https://golden.com/wiki/Streamlit-5KMKYAB

[6] https://www.oreilly.com/library/view/statistics-for-machine/9781788295758/eb9cd609-e44a-40a2-9c3a-f16fc4f5289a.xhtml