**Report**

## **Table of Contents**

1. Brief Introduction to Recommendation System
2. Setting up
3. About the Dataset
4. Our Strategy to Build a Movie Recommendation Model based on title, overview and genre
5. Implementation

**1. INTRODUCTION**

There are three main approaches in recommender system.

1. Demographic Filtering
2. Content based Filtering
3. Collaborative Filtering

To personalise our recommendations more, I am going to build an engine that computes similarity between movies based on certain metrics and suggests movies that are most similar to a particular movie that a user liked. Since we will be using movie metadata (or content) to build this engine, this is also known as **Content Based Filtering.**

**2.SETUP**

Let's get started by importing the required packages.

**3.DATASET**

### **Content**

This dataset consists of the following files:

movies\_metadata.csv: The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

keywords.csv: Contains the movie plot keywords for our MovieLens movies. Available in the form of a stringified JSON Object.

credits.csv: Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

links.csv: The file that contains the TMDB and IMDB IDs of all the movies featured in the Full MovieLens dataset.

links\_small.csv: Contains the TMDB and IMDB IDs of a small subset of 9,000 movies of the Full Dataset.

ratings\_small.csv: The subset of 100,000 ratings from 700 users on 9,000 movies.

The Full MovieLens Dataset consisting of 26 million ratings and 750,000 tag applications from 270,000 users on all the 45,000 movies in this dataset can be accessed [here](https://grouplens.org/datasets/movielens/latest/)

**4. APPROACH**

Here I used content based filtering technique.In content based filtering, we recommend user, similar movies based on what users have seen in the past or what users like. To achieve this, we can use various parameters like movie title, cast, genre, movie overview, votes etc

I had first started with doing EDA on the given dataset to understand all features important wrt to our recommendation generation. Due to huge data set and multiple genres couldn't get much effective results wrt to graphical representation and it took too long to process too.

I created a few helper functions too, to ease my task and calculations.

Data Cleaning was followed by removing the null values.

I had Generated WordCloud which gave me fare idea about totally different genres and their importance also I figured out Keyword Occurrences.

I used user id, Movie Id and ratings to get deeper statistical information.

I decided to create my own dataframe which will consist of only important features which I feel will be helpful. **Main Task:**

#### **I first tried to build a system which can recommend the movies according to the movie overview**

NLP:

Here our my steps to deal with data using nlp

1. Data preprocessing
2. Converting into proper data types- converted overview into string
3. Replaces nan values with empty string
4. Generated WordCloud
5. Count Of Stopwords and removal of it
6. Conversion into Lowercase
7. The first pre-processing step which we will do is transform our texts into lower case. This avoids having multiple copies of the same words
8. Found out number of numerics
9. Punctuation removal
10. Common words removal
11. Spelling correction
12. Tokenization
13. Tfidf vectorization
14. Sparse matrix

#### 

I had build Content Based Recommenders based on: Movie Overviews and title using the cosine similarity

**Cosine Similarity**

I will be using the Cosine Similarity to calculate a numeric quantity that denotes the similarity between two movies. Mathematically, it is defined as follows:

*cosine*(*x*,*y*)=*x*.*y*⊺||*x*||.||*y*||

Since we have used the TF-IDF Vectorizer, calculating the Dot Product will directly give us the Cosine Similarity Score. Therefore, we will use sklearn's **linear\_kernel** instead of cosine\_similarities since it is much faster.

I calculate the **similarity scores** for the overview of each movie entry in the dataset

for that I use a text encoding technique known as **word\_to\_vec** in order to convert each overview into numeric embeddings. This technique is used to convert textual data into numeric vectors

Then we compute **Term Frequency-Inverse Document Frequency (TF-IDF)** vectors for each overview.

This will give us a relative frequency of word in the document. To achieve this, I use scikit-learn which gives a built-in TfIdfVectorizer class that produces the TF-IDF matrix

Now I create a *get\_recommendation*s function which accepts a movie title from the user and recommends similar movies to the user based on the title by taking the cosine similarity scores of the most similar movies.

This function will recommend the 10 most similar movies

### **Future scope:**

So, my plan was to try using other properties like movie genre, ratings, keywords, crew etc which I couldn’t execute due to lack of time Taking important features like movie genres, caste of the movie, keywords etc will help to build a much more accurate recommender system What I plan on doing is creating a metadata dump for every movie which consists of genres, director, main actors and keywords. I can then use a Countvectorizer to create our count matrix as we did in the Description Recommender. The remaining steps are similar to what we did earlier: we calculate the cosine similarities and return movies that are most similar**.**

### **Summary:**

**So, thus I have successfully been able to develop a recommender system which basically uses word\_to\_vec encodings along with tf\_idf / CostVectorizer to determine the frequency of words and calculate the matricesFurther I used cosine\_similarity score to evaluate the similarity of words and then accurately recommend the most 10 similar movies to the user based on the title of the movie which the user gives as an input to the system**