

Movie Recommendation System

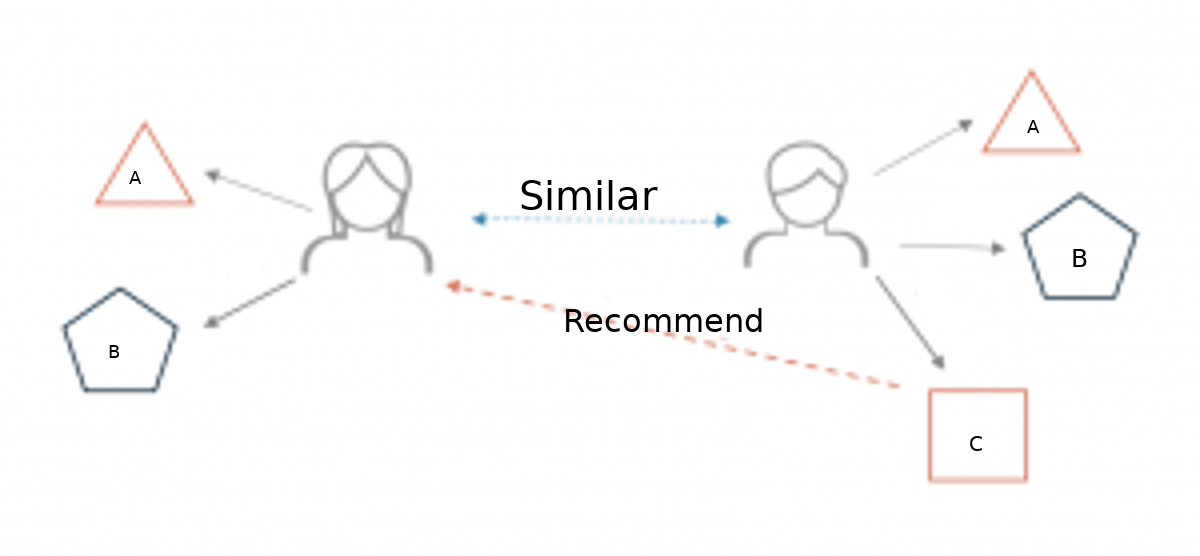
PYTHON

Tushar Kholia | Data Analytics and Machine Learning | 03-2020

**What is a recommender system?**

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon, and you notice that some items are being recommended to you or when Netflix recommends certain movies to you. They are also used by Music streaming applications such as Spotify and Deezer to recommend music that you might like.

Below is a very simple illustration of how recommender systems work in the context of an e-commerce site.



Two users buy the same items A and B from an e-commerce store. When this happens the similarity index of these two users is computed. Depending on the score the system can recommend item C to the other user because it detects that those two users are similar in terms of the items they purchase.

**Different types of recommendation engines**

The most common types of recommendation systems are **content-based** and **collaborative filtering** recommender systems. In collaborative filtering, the behavior of a group of users is used to make recommendations to other users. The recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie. There are two types of collaborative models **Memory-based** methods and **Model-based** methods. The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into two:

* **User-based collaborative filtering**: In this model, products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example, if Derrick and Dennis like the same movies and a new movie come out that Derick like, then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.
* **Item-based collaborative filtering**: These systems identify similar items based on users’ previous ratings. For example, if users A, B, and C gave a 5-star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A, B, and C.

Model-based methods are based on Matrix Factorization and are better at dealing with sparsity. They are developed using data mining, machine learning algorithms to predict users’ rating of unrated items. In this approach techniques such as dimensionality reduction are used to improve accuracy. Examples of such model-based methods include Decision trees, Rule-based Model, Bayesian Model, and latent factor models.

**Content-based systems** use metadata such as genre, producer, actor, musician to recommend items say movies or music. Such a recommendation would be for instance recommending Infinity War that featured Vin Diesel because someone watched and liked The Fate of the Furious. Similarly, you can get music recommendations from certain artists because you liked their music. Content-based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.

**Datasets to use for building recommender systems**

In this Project, we are going to use the TMDB 5000 Movie Dataset. TMDB consists of 2 datasets. The dataset for movie contains 4803 observations with 20 variables while the dataset for credits contains 4803 observations with 4 variables. We merge both the datasets to create a new data set which as well we can use for our analysis.

**Walkthrough of building a recommender system**

We are going to use the TMDB to build a simple item similarity-based recommender system. The first thing we need to do is to import pandas and numpy.

**import** **pandas** **as** **pd**   
**import** **numpy** **as** **np**

Next, we load in the data set using pandas read\_csv() utility.

**movies = pd.read\_csv('tmdb\_5000\_movies.csv')**

**credits = pd.read\_csv('tmdb\_5000\_credits.csv')**

Now let’s check the head of the data to see the data we are dealing with.

**movies.head()**

**credits.head()**

It would be nice if we can see the titles of the movie instead of just dealing with the IDs. Let’s load in the movie titles of credits dataset and merge it with movies dataset.

Since the title columns are the same, we can merge these datasets.

**movies = movies.merge(credits, on='title')**

**movies.head()**

After then, we must extract some quality feature from the dataset.

**movies = movies[['movie\_id','title','overview','genres','keywords','cast','crew']]**

Then, we do some preprocessing to check the null values and duplicates data and clean them.

**movies.isnull().sum()**

**movies.duplicated().sum()**

**movies.dropna(inplace=True)**

Then, we create a function to convert extracted features entity from dictionary to list.

**def convert(text):**

**L = []**

**for i in ast.literal\_eval(text):**

**L.append(i['name'])**

**return L**

Convert extracted features entity from dictionary to list.

**movies['genres'] = movies['genres'].apply(convert)**

**movies['keywords'] = movies['keywords'].apply(convert)**

**movies['cast'] = movies['cast'].apply(convert)**

Then, we create a function to fetch director name from crew column.

**def fetch\_director(text):**

**L = []**

**for i in ast.literal\_eval(text):**

**if i['job'] == 'Director':**

**L.append(i['name'])**

**return L**

**movies['crew'] = movies['crew'].apply(fetch\_director)**

After then, merge all the extracted feature and tag them as a single column.

**movies['tags'] = movies['overview'] + movies['genres'] +**

**movies['keywords'] + movies['cast'] + movies['crew']**

Make new database contain “movie\_id”, 'title', and 'tags' to check similarity.

**new\_df = movies[['movie\_id', 'title', 'tags']]**

After then, convert new database from list to string.

**new\_df['tags'] = new\_df['tags'].apply(lambda x:" ".join(x))**

**new\_df['tags'][0]**

Then, using nltk library we import PorterStemmer for natural language processing.

**from nltk.stem.porter import PorterStemmer**

**ps = PorterStemmer()**

Create function with name stem to merge similar words.

**def stem(text):**

**y = []**

**for i in text.split():**

**y.append(ps.stem(i))**

**return " ".join(y)**

Call function stem, give tag as parameter and merge similar kinds of words of tag column with unique names.

**new\_df['tags'] = new\_df['tags'].apply(stem)**

Import CountVectorizer using sklearn. Then, vectorize the new database.

**from sklearn.feature\_extraction.text import CountVectorizer**

**cv = CountVectorizer(max\_features=5000,stop\_words='english')**

**vectors = cv.fit\_transform(new\_df['tags']).toarray()**

**vectors**

Import cosine\_similarity using sklearn.metrics.pairwise. Then, find cosine\_similarity of vector.

**from sklearn.metrics.pairwise import cosine\_similarity**

**similarity = cosine\_similarity(vectors)**

**similarity**

Create recommend function which can choose 5 highly similar movies based on content.

**def recommend(movie):**

**movie\_index = new\_df[new\_df['title'] == movie].index[0]**

**distances = similarity[movie\_index]**

**movie\_list = sorted(list(enumerate(distances)), reverse=True, key=lambda x:x[1])[1:6]**

**for i in movie\_list:**

**print(new\_df.iloc[i[0]].title)**

Call recommend function and pass movie name as parameter.

**recommend("Iron Man")**

Then, import pickle and dump some list.

**import pickle**

**pickle.dump(new\_df, open('movies.pkl','wb'))**

**new\_df['title'].values**

**pickle.dump(new\_df.to\_dict(), open('movie\_dict.pkl', 'wb'))**

**pickle.dump(similarity, open('similarity.pkl', 'wb'))**

Create python py file, import streamlit, pickle, requests, and pandas. Link pickle files python py file.

**import streamlit as st**

**import pickle**

**import pandas as pd**

**import requests**

Define fetch\_poster function that can fetch recommended movies poster online using movie id.

**def fetch\_poster(movie\_id):**

**response = requests.get('https://api.themoviedb.org/3/movie/{}?api\_key=c60c3929624a0c9123187048c3e265f8&language=en-US'.format(movie\_id))**

**data = response.json()**

**return 'https://image.tmdb.org/t/p/w500/' + data['poster\_path']**

Define recommend function that can accept one movie name and based on similarity score can recommend 5 movies.

**def recommend(movie):**

**movie\_index = movies[movies['title'] == movie].index[0]**

**distance = similarity[movie\_index]**

**movies\_list = sorted(list(enumerate(distance)), reverse=True, key=lambda x:x[1])[1:6]**

**recommended\_movies = []**

**recommended\_posters = []**

**for i in movies\_list:**

**movie\_id = movies.iloc[i[0]].movie\_id**

**recommended\_movies.append(movies.iloc[i[0]].title)**

**# fetch poster from API**

**recommended\_posters.append(fetch\_poster(movie\_id))**

**return recommended\_movies, recommended\_posters**

Define and open all the pickle files with new name.

**movies\_dict = pickle.load(open('movie\_dict.pkl', 'rb'))**

**movies = pd.DataFrame(movies\_dict)**

**similarity = pickle.load(open('similarity.pkl', 'rb'))**

Using streamlit create a button and recommend movies.

**selected\_movie\_name = st.selectbox(**

**'How would you like to be connected?',**

**movies['title'].values)**

**if st.button('Recommend'):**

**recommended\_movie\_names, recommended\_movie\_posters = recommend(selected\_movie\_name)**

**col1, col2, col3, col4, col5 = st.columns(5)**

**with col1:**

**st.text(recommended\_movie\_names[0])**

**st.image(recommended\_movie\_posters[0])**

**with col2:**

**st.text(recommended\_movie\_names[1])**

**st.image(recommended\_movie\_posters[1])**

**with col3:**

**st.text(recommended\_movie\_names[2])**

**st.image(recommended\_movie\_posters[2])**

**with col4:**

**st.text(recommended\_movie\_names[3])**

**st.image(recommended\_movie\_posters[3])**

**with col5:**

**st.text(recommended\_movie\_names[4])**

**st.image(recommended\_movie\_posters[4])**

**How to improve the recommendation system**

This system can be improved by building a Memory-Based Collaborative Filtering based system. In this case, we’d divide the data into a training set and a test set. We’d then use techniques such as cosine similarity to compute the similarity between the movies. An alternative is to build a Model-based Collaborative Filtering system. This is based on matrix factorization. Matrix factorization is good at dealing with scalability and sparsity than the former. You can then evaluate your model using techniques such as Root Mean Squared Error (RMSE).