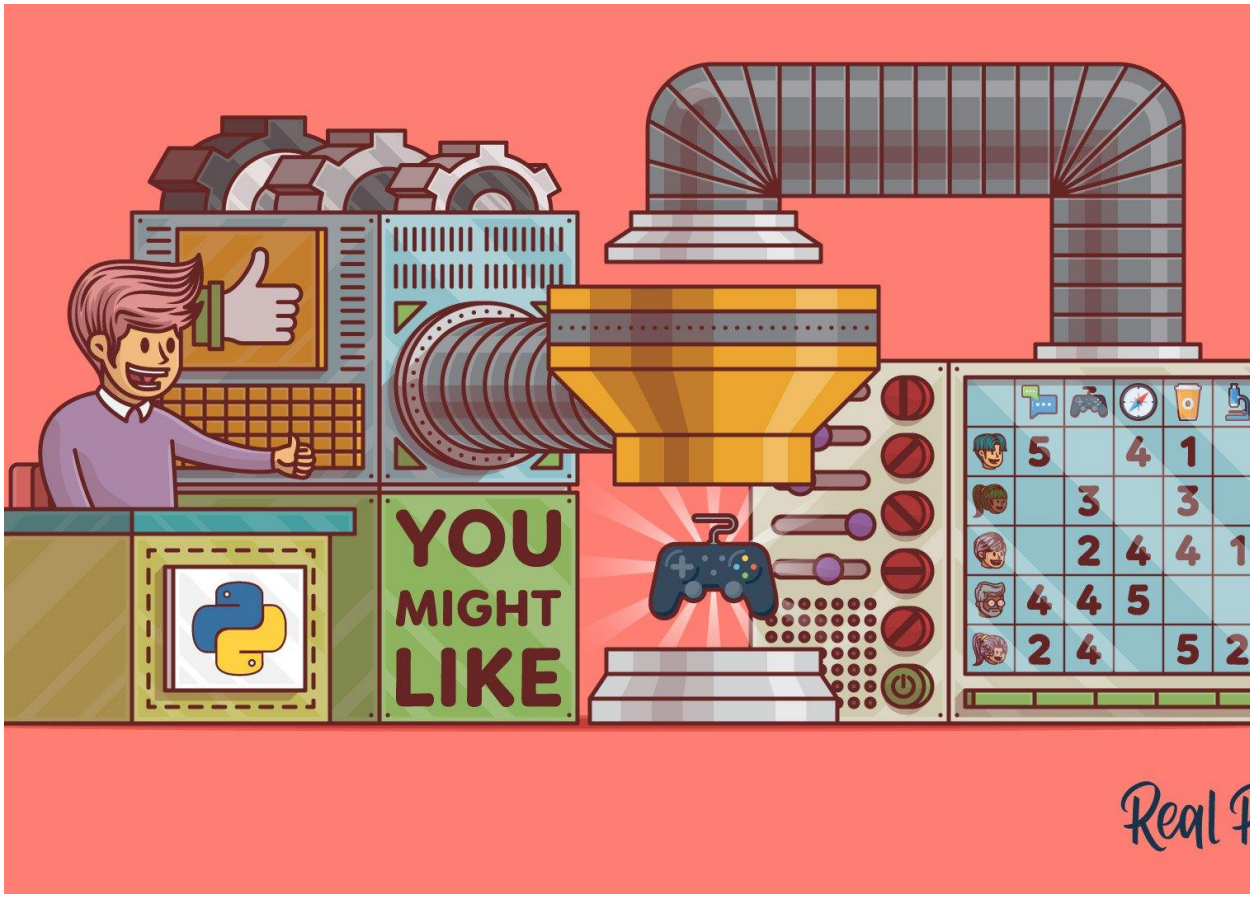
Movie/Web Series Recommendation System



Introduction

Recommender systems are the systems that are designed to recommend things to the user based on many different factors. These systems predict the most likely product that the users are most likely to purchase and are of interest to. Companies like Netflix, Amazon, etc. use recommender systems to help their users to identify the correct product or movies for them.

The recommender system deals with a large volume of information present by filtering the most important information based on the data provided by a user and other factors that take care of the user's preference and interest. It finds out the match between user and item and imputes the similarities between users and items for recommendation.

Both the users and the services provided have benefited from these kinds of systems. The quality and decision-making process has also improved through these kinds of systems.

Why the Recommendation Systems?

- Benefits users in finding items of their interest.
- Help item providers in delivering their items to the right user.
- Identity products that are most relevant to users.
- Personalized content.
- Help websites to improve user engagement.

Problem statement and objectives

Problem Statement

YouTube has undoubtedly emerged as a top platform for videos, vlogs, and so on. They have now planned to launch its own OTT platform “YouTube Watch” wherein they will be featuring movies, web series, documentaries and so on specifically. Netflix, Amazon prime video being its top competitors, YT now wants to build a movie/ series recommendation system based on the previous watch history, preferred genre, and other behavioral insights of the user.

YT is part of Google and YT itself is huge in India and they have humungous amount of data of Indian audience. YT after 2016 particularly grew exponentially when internet got cheaper and became more mainstream in Tier 1 and Tier 2 cities. Now, not only are the people using YT to entertain themselves, but they are also using it to create content and entertain others such as from Cooking Channels of Dadaji's made by their grandchildren to farmers. YT is already filled with tones and tones of Indian content; your task is to use that data and create a recommendations model using Content-Based Filtering and collaborative filter.

Objective

Focus on developing a decision system that is a combination of a machine learning based system, which specifically requires Collaborative and Content Filtering or a clustering model requiring understanding of viewers as well as the content developers and their behavior, to fine tune the underwriting algorithm to differentiate and find similarities between viewers. The solution should address the users' concern of getting recommended more content which they prefer consuming.

Work Done

1. **Gather Data using Google YouTube API v3**
2. **Exploratory Data Analysis (EDA)**
3. **Data Preprocessing**

Data Cleaning

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. We have removed duplicate columns, some empty brackets and comma operators from the obtained database to make it a relational database. For data cleaning, we have installed Pandas which is a software library written for the Python programming language for data manipulation and analysis. In addition, we have removed tags, special characters and digits.

Keyword Extraction

From all the fields present in the dataset, what we are mostly interested in is the title, description and tags which will become our source of text for keyword extraction. We created a field that combines both body and title so we have it in one field and then removed the tags, special characters and digits from it and also converted it to lower case.

Creating Vocabulary and Word Counts for IDF

We now need to create the vocabulary and start the counting process. We can use the CountVectorizer to create a vocabulary from all the text in our df ['text'] followed by the counts of words in the vocabulary. While cv.fit(...) would only create the vocabulary, cv.fit_transform(...) creates the vocabulary and returns a term-document matrix which is what we want. With this, each column in the matrix represents a word in the vocabulary while each row represents the document in our dataset where the values in this case are the word counts.

Note that with this representation, counts of some words could be 0 if the word did not appear in the corresponding document.

Notice that in the code above, we are passing two parameters to CountVectorizer, max_df and stop_words. The first is just to ignore all words that have appeared in 85% of the documents, since those may be unimportant. The later, is a custom stop words list. You can also use stop words that are native to sklearn by setting stop_words='english', but we personally find this to be quite limited. We used a custom stop word list for this project.

TfidfTransformer to Compute Inverse Document Frequency (IDF)

We compute the IDF values by taking the sparse matrix from CountVectorizer (word_count_vector) to generate the IDF when you invoke tfidf_transformer.fit(...). An extremely important point to note here is that the IDF should always be based on a large corpora and should be representative of texts you would be using to extract keywords. This is why we are using texts from 1,20,000 data points to compute the IDF instead of just a handful.

Computing TF-IDF and Extracting Keywords

Once we have our IDF computed, we are now ready to compute TF-IDF and then extract top keywords from the TF-IDF vectors.

The next step is to compute the tf-idf value for a given document in our test set by invoking tfidf_transformer.transform(...). This generates a vector of tf-idf scores. Next, we sort the words in the vector in descending order of tf-idf values and then iterate over them to extract the top-n keywords (top 5 keywords in our case).

4. Modelling

Self Supervised Learning

Method of machine learning that can be regarded as an intermediate form between supervised and unsupervised learning.

It is a type of autonomous learning using artificial neural networks that does not necessarily require sample data classified in advance by humans.

Label Generation for Supervised Learning

K means clustering is done based on processed text. K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.

A cluster refers to a collection of data points aggregated together because of certain similarities. You'll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster. Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The 'means' in the K-means refers to averaging of the data; that is, finding the centroid.

Calculation of optimal value of K

Silhouette score is used to evaluate the quality of clusters created using clustering algorithms such as K-Means in terms of how well samples are clustered with other samples that are similar to each other. The Silhouette score is calculated for each sample of different clusters.

Model Training - DecisionTreeClassifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Some advantages of decision trees are:

1. Simple to understand and to interpret. Trees can be visualised.
2. Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
3. The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
4. Able to handle both numerical and categorical data. However scikit-learn implementation does not support categorical variables for now. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
5. Able to handle multi-output problems.
6. Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
7. Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
8. Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

****DecisionTreeClassifier is a class capable of performing multi-class classification on a dataset.**

Architecture