BUILDING RECOMMENDATION SYSTEM USING MOVIELENS DATA- PROJECT DEOCUMENT ON GITHUB

A Project Report Submitted in Partial Fulfilment of the Requirements for the Award of the Degree of Bachelor of Science (Research) in Computer Science with Specialization in Cloud Computing and Big data. It's a complete documentation to build recommendation model.

Submitted by

Ms. Aishwarya

School of Computer Science and Applications

REVA University, Bangalore

SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS

REVA University

Kattigenahalli, Yelahanka, Bangalore – 560 064

November 2019

ABSTRACT

- Recommendation Systems are engines which has been deployed in many platforms. Ever heard about Netflix, Amazon Prime, YouTube and Spotify? How do they know what you are interested in! It is predicted by Recommendation System in the field of movies, songs, items, videos etc. according to your interest. This has also reduced dependencies. Our Recommendation System predicts depending on each user's interest by his/her previous/current search list, based on the genres it keeps suggesting the movies which the user would like to watch next. Content based prediction is assigned based on the particular user interest. The weight values are estimated from a set of algorithms obtained from social network graph which captures human judgement about similarity of items. In our model it is movie genres.

TABLE OF CONTENTS

INTRODUCTION

BASIC INTRODUCTION OF PROJECT

Most of us are very much familiar with this website called MovieLens- using this data we are building a recommendation system which will recommend 'Movies' based on user preferences/interest this is just like the concept used in YouTube, Netflix, Amazon Prime and any other recommendation system. Also, our recommendation system aims to provide suitable suggestions to users with improved quality and accurate assumptions about the information item, social elements, products or services following the data which is flexible and is more based on the user's interest. The tools we are using to build recommendation system is efficient and easy to access, this is all because we are using **Google Colab** notebook where project will be built on cloud and you need not worry about your physical device since, Google Colab is cloud based where we can access our project from anywhere. This project mainly gives a platform for movies which was never lightened before and also it will increase the weightage of movies which were never suggested but individual user interest wise stand on top list.

STATEMENT OF PROBLEM:

GOAL: To accomplish our above problem we are building a recommendation system which recommends movies based on the interest of the individual user. This model will fulfil and also help them discover movies of their interest and also increase the weightage of movies which were never discovered/suggested before.

SYSTEM ANALYSIS

PROPOSED SYSTEM DESIGN

We are using Google Collaboration to build our model which will let us access the data from anywhere across the world since we can access over data cloud. And this is the requirement which can adopt to increase the use-case of a model.

SYSTEM DEVELOPMENT

HARDWARE REQUIREMENT: Laptop/Desktop, Keyboard.

SOFTWARE REQUIREMENT: Google Collaboratory, Google Drive, Gmail.

SYSTEM DESIGN

ARCHITECTURE:

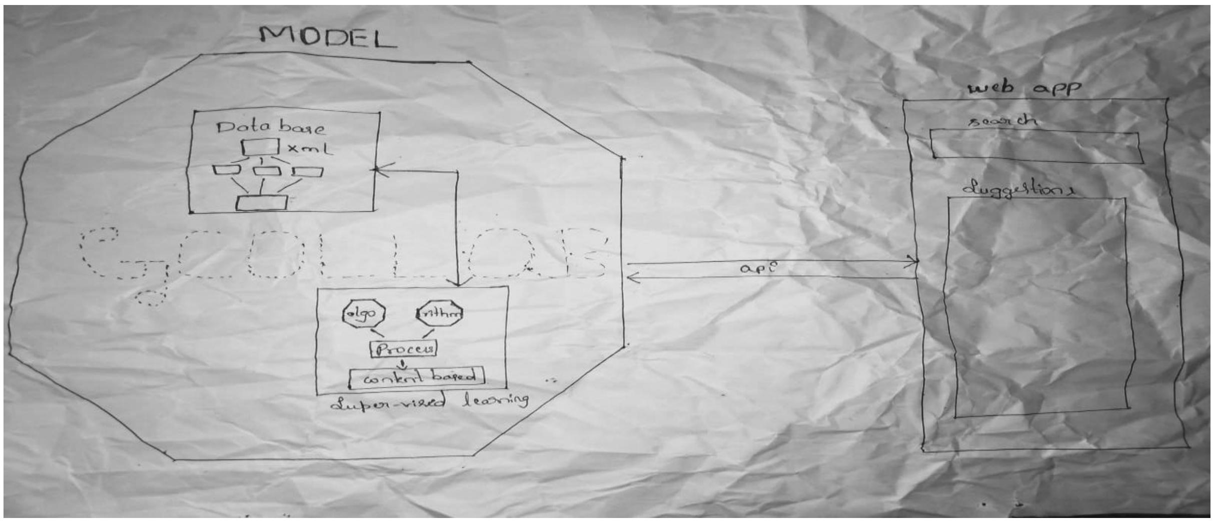


Fig 1.0

ARCHITECTURE EXPLANATION

When a user gives a movie name as an input, which will trigger google colab data set (imported from google drive) where the parsed data has been stored and based on that our input movie Genre it will analyse all the movies of the same genres in the database and that’s where Content-based algorithm comes into picture. we followed a protocol where we need to make ‘To-Do’ list(phase) which help us to stick on our project.

There are mainly FIVE phases to build the model, they are:

1. Planning
2. Designing
3. Data parsing
4. Data processing[ALGORITHM]
5. Execution phase

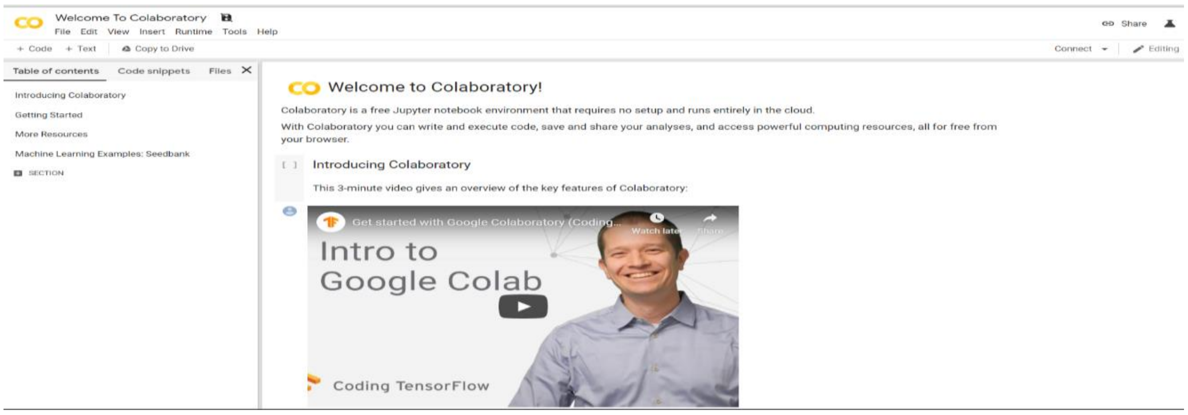
IMPLEMENTATION

PLANING: In this phase we started thinking about our model, by asking few questions to our self.

- how to start this project?
- Which platform to work on
- How to deal with data?
- Which libraries to import?
- Which algorithm to be used?
- Finally, how to execute this project?

DESIGNING: In this phase we started mapping our project according to our plan and started organizing it. Designing is the main phase in ML because this is where we give a kick start i.e., actual start for the model.

First basic requirement is to make a Gmail account and have access to your Google Drive. Make sure your required data is in drive. Now, search for Google Colaboratory and click on the first link. Google Colab's welcome page will be opened as shown in fig 1.2



Welcome to Colaboratory!

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

Introducing Colaboratory

This 3-minute video gives an overview of the key features of Colaboratory:

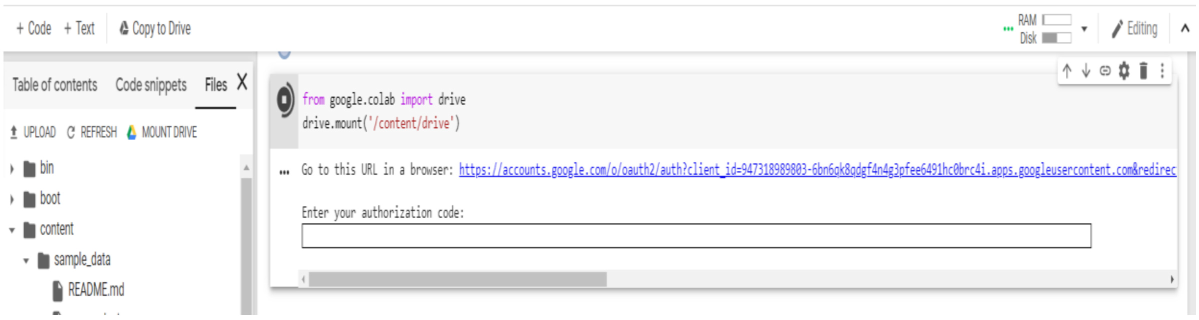
fig 1.1

Now, you've to scroll a bit and then right after the video there will be a command which helps in mounting Google drive into the Colaboratory.

```
from google.colab import drive

drive.mount('/content/drive')
```

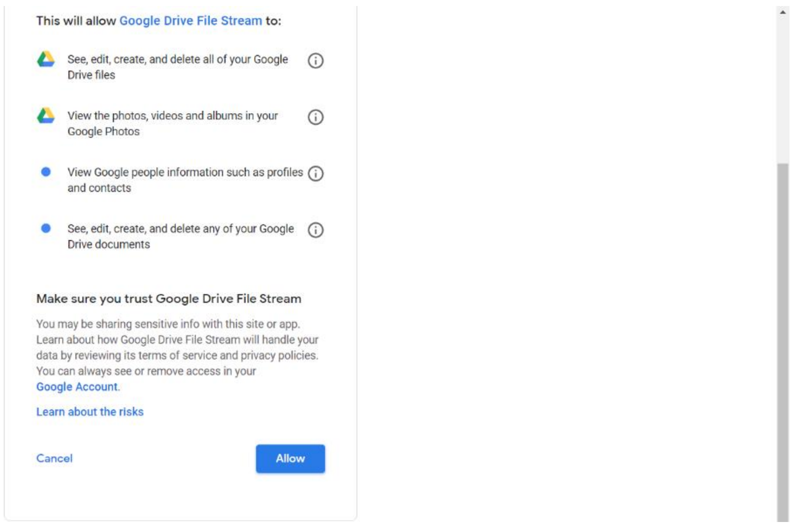
After running this command, you can see a link. Click on that link and it will direct you to your Google Account. Select your Google Account and click on Next. You'll be prompted to another page where terms and conditions will be mentioned. Click on Allow and then you can see a code on your screen. Copy the code and paste it in colab as in fig 1.2



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirec

Enter your authorization code:
```

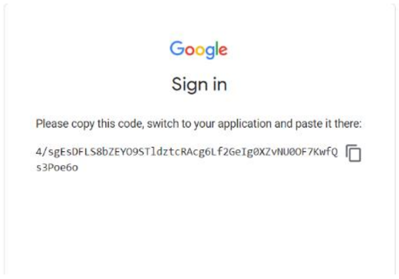

This will allow **Google Drive File Stream** to:

- See, edit, create, and delete all of your Google Drive files
- View the photos, videos and albums in your Google Photos
- View Google people information such as profiles and contacts
- See, edit, create, and delete any of your Google Drive documents

Make sure you trust Google Drive File Stream

You may be sharing sensitive info with this site or app. Learn about how Google Drive File Stream will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.

Learn about the risks



Sign in

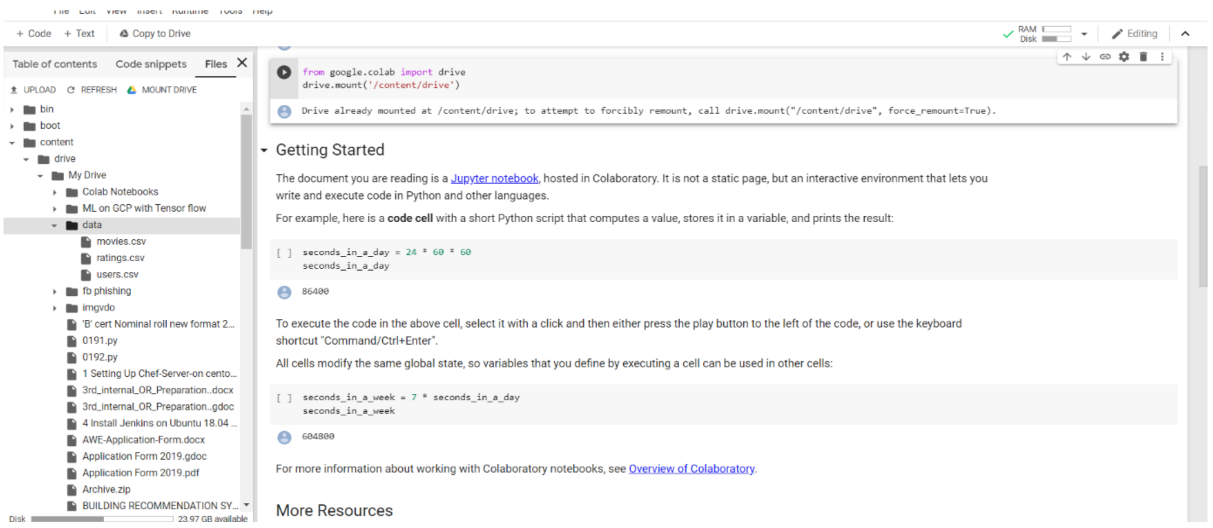
Please copy this code, switch to your application and paste it there:

```
4/sgEsDFLS8bZEYO9STldztcRAcg6Lf2GeIg0XZvNU0OF7KwfQ
s3Poe6o
```

fig 1.2

Now, go to files and click on Mount Drive. You will be able to see a folder by name **Content**. Click on that and then click on MyDrive subfolder. Here you will be able to see three csv files which you have put in Drive folder before.

Right click on the .csv file, copy the path then paste it in the cell. As you can see there are 3 Data Sets in data folder as shown in fig 1.3



- bin
- boot
- content
  - drive
    - My Drive
      - Colab Notebooks
      - ML on GCP with Tensor flow
      - data
        - movies.csv
        - ratings.csv
        - users.csv
      - fb phishing
      - imgvdo
      - 'B' cert Nominal roll new format 2...
      - 0191.py
      - 0192.py
      - 1 Setting Up Chef-Server-on cento...
      - 3rd_internal_OR_Preparation..docx
      - 3rd_internal_OR_Preparation..gdoc
      - 4 Install Jenkins on Ubuntu 18.04 ...
      - AWE-Application-Form.docx
      - Application Form 2019.gdoc
      - Application Form 2019.pdf
      - Archive.zip
      - BUILDING RECOMMENDATION SY...

Disk 23.97 GB available

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Getting Started

The document you are reading is a Jupyter notebook, hosted in Colaboratory. It is not a static page, but an interactive environment that lets you write and execute code in Python and other languages.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
    seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter".

All cells modify the same global state, so variables that you define by executing a cell can be used in other cells:

```
[ ] seconds_in_a_week = 7 * seconds_in_a_day
    seconds_in_a_week
```

```
604800
```

For more information about working with Colaboratory notebooks, see Overview of Colaboratory.

More Resources

fig 1.3

DATA PARSING: It’s a phase where we collect the data from MovieLens which is required for the project. There’s no null values and garbage values, and hence making it more accurate dataset which can be easily used for further analysis.

This 3 data set has to be merged after parsing/cleaning. Those three data-sets are **USERS, MOVIE, RATING** from MovieLens. Soon after this we started analysing the data by creating the word cloud, by looking at its description and information. There were few attributes which was common in this dataset and hence, we were able to merge these. First, we need to import few necessary libraries like:

Pandas - In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

NumPy- NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Matplotlib- Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Example Command to import dataset:

```
ratings = pd.read_csv('/content/drive/My Drive/data/ratings.csv', sep='\t', encoding
='latin-1', usecols=['user_id', 'movie_id', 'rating'])
```

/content/drive/My Drive/data/ratings.csv' → This is the path of your Data Set in Google Drive.

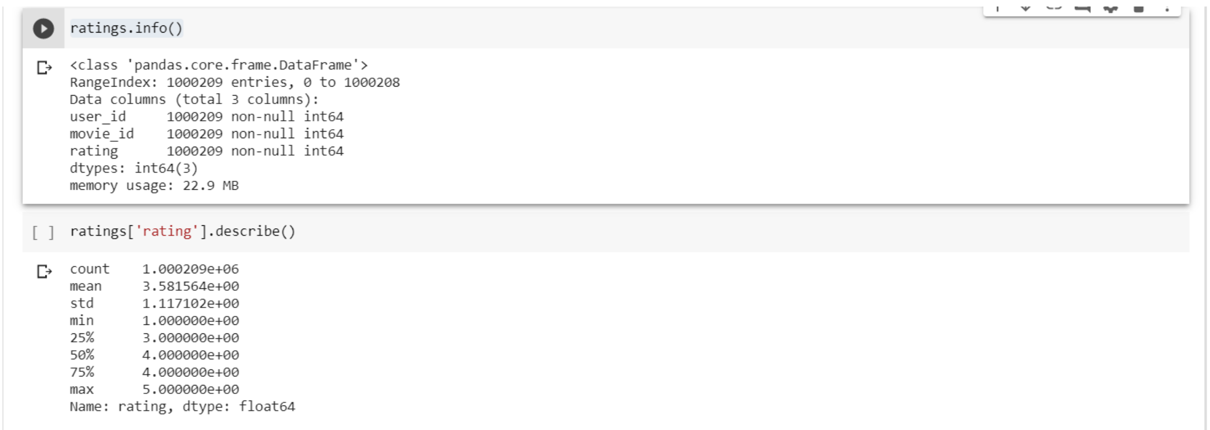
We should follow the same syntax for importing other datasets.

To mention the data inside the dataset we should just mention the variable name which you assigned while importing data. For example, here we have imported ratings data and have assigned ratings as the variable. So, to see the data inside ratings we just need to write the variable name.

```
ratings
```

We need to do the same with other two datasets also.

And now to check the information and summary statistics of rating as in fig 1.4



```
ratings.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 3 columns):
user_id      1000209 non-null int64
movie_id     1000209 non-null int64
rating       1000209 non-null int64
dtypes: int64(3)
memory usage: 22.9 MB

[ ] ratings['rating'].describe()

count    1.000209e+06
mean     3.581564e+00
std      1.117102e+00
min      1.000000e+00
25%      3.000000e+00
50%      4.000000e+00
75%      4.000000e+00
max      5.000000e+00
Name: rating, dtype: float64
```

Fig 1.4

Now, let’s explore the data by checking if there are certain words that feature more often in Movie Titles by creating a word cloud before that join the 3 dataset.

To join three data sets and then display top 20 movies with highest ratings:

```
# Join all 3 files into one dataframe
dataset = pd.merge(pd.merge(movies, ratings),users)
# Display 20 movies with highest ratings
dataset[['title','genres','rating']].sort_values('rating', ascending=False).head(20)
```

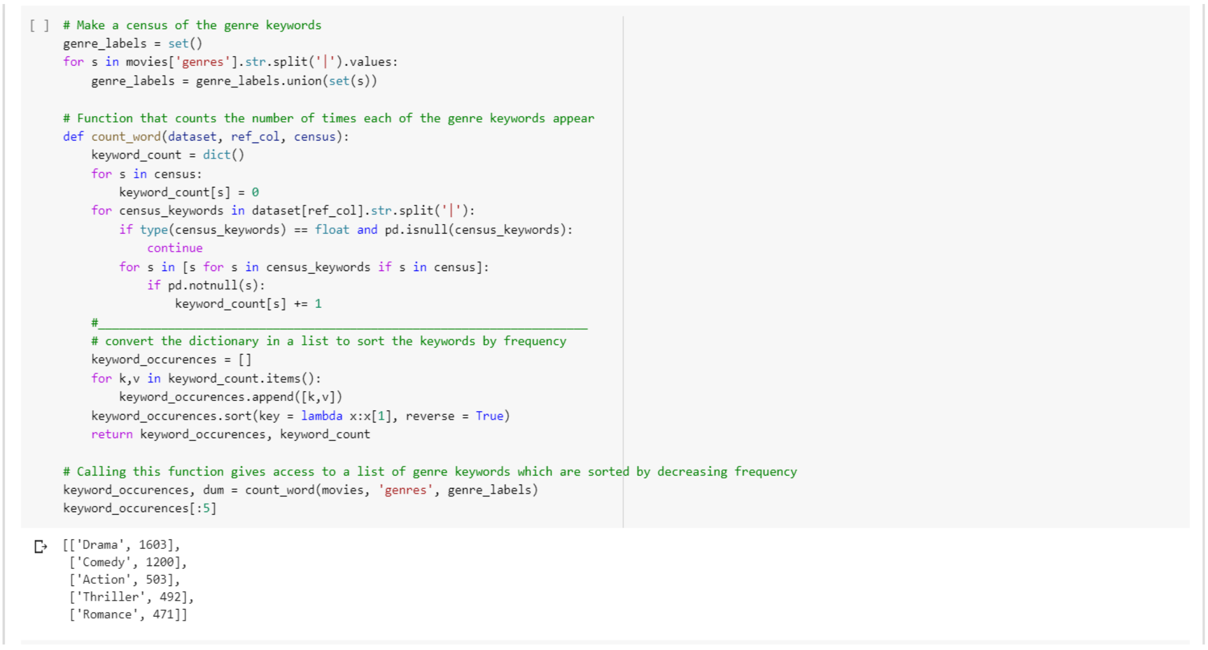
Since its content based, The genres variable will surely be important while building the recommendation engines, so in this recommendation system is based on genre which will lead our model to suggest films.

```
%matplotlib inline
import wordcloud
from wordcloud import WordCloud, STOPWORDS

# Create a wordcloud of the movie titles
movies['title'] = movies['title'].fillna("").astype('str')
title_corpus = ' '.join(movies['title'])
title_wordcloud = WordCloud(stopwords=STOPWORDS, background_color='blac
k', height=2000, width=4000).generate(title_corpus)

# Plot the wordcloud
plt.figure(figsize=(16,8))
plt.imshow(title_wordcloud)
plt.axis('off')
plt.show()
```

This above algorithm visually helped us to see which genres are used mostly in all the movies. And the most repeated genre will be large in size in the below shown fig 1.5



```
[ ] # Make a census of the genre keywords
    genre_labels = set()
    for s in movies['genres'].str.split('|').values:
        genre_labels = genre_labels.union(set(s))

    # Function that counts the number of times each of the genre keywords appear
    def count_word(dataset, ref_col, census):
        keyword_count = dict()
        for s in census:
            keyword_count[s] = 0
        for census_keywords in dataset[ref_col].str.split('|'):
            if type(census_keywords) == float and pd.isnull(census_keywords):
                continue
            for s in [s for s in census_keywords if s in census]:
                if pd.notnull(s):
                    keyword_count[s] += 1
        #______________________________________________________________________
        # convert the dictionary in a list to sort the keywords by frequency
        keyword_occurences = []
        for k,v in keyword_count.items():
            keyword_occurences.append([k,v])
        keyword_occurences.sort(key = lambda x:x[1], reverse = True)
        return keyword_occurences, keyword_count

    # Calling this function gives access to a list of genre keywords which are sorted by decreasing frequency
    keyword_occurences, dum = count_word(movies, 'genres', genre_labels)
    keyword_occurences[:5]
```

```
[['Drama', 1603],
 ['Comedy', 1200],
 ['Action', 503],
 ['Thriller', 492],
 ['Romance', 471]]
```

fig 1.5

So as per above, we are splitting data based on genre and now each film will be classified, which will be later sorted according to descending order where its genres and number count will be printed together as a set. You can see this in above screenshot.

You can also see that if same genres are repeating in many films those genres will be grouped together and the number of particular genres will keep increasing as shown in the algorithm as +1.

The genres will be sorted based on number of times its repeated in the movies and at last list them in decreasing order as keyword_occurences and keyword_count.

The top 5 genres are, in that respect order: Drama, Comedy, Action, Thriller, and Romance. To make the genres visually appealing let's create a word cloud of genres, after that it will look something like below fig 1.6

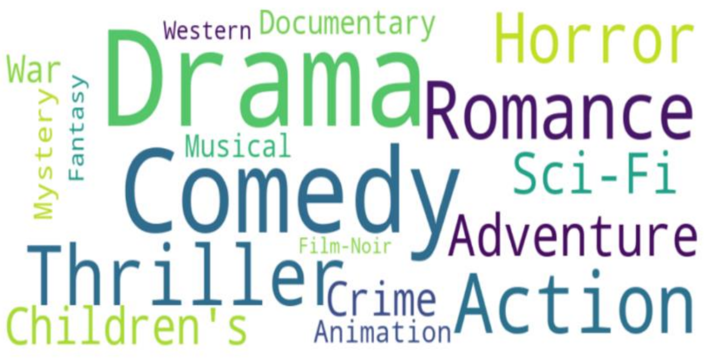


fig 1.6

DATA PROCESSING: Our machine learning is based on supervised learning and we are using a method which will suggest based on movie genres. And to do this we have following methods to adopt which will help us build recommendation system. Here there are 2 main methods:

1. Content based:

The Content-Based Recommender relies on the similarity of the items being recommended. The basic idea is that if you like an item, then you will also like a “similar” item. It generally works well, when it's easy to determine the context/properties of each item. A content-based recommender works with data that the user provides, either explicitly movie ratings for the MovieLens dataset. Based on that data, a user profile is generated, which is then used to make suggestions to the user.

Collaborative based:

The Collaborative Filtering Recommender is entirely based on the past behaviour and not on the context. More specifically, it is based on the similarity in preferences, tastes and choices of two or more users(clustering similar user). It analyses how similar the tastes of one user is to another and makes recommendations on the basis of that.

WE ARE USING CONTENT BASED SYSTEM.

Step 1:

```
# Break up the big genre string into a string array
movies['genres'] = movies['genres'].str.split('|')
# Convert genres to string value
movies['genres'] = movies['genres'].fillna("").astype('str')
```

We are using movie.csv file since it holds genres attribute. So here the genres are to be converted into strings. Step 2:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words=
'english')
tfidf_matrix = tf.fit_transform(movies['genres'])
```

Since we are dealing with **TfidfVectorizer** function we have to import sklearn library where text file will be converted into feature vector so we can take input.

Step 3:

```
from sklearn.metrics.pairwise import linear_kernel
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
cosine_sim[:4, :4]
```

We will be using the **Cosine Similarity** to calculate a numeric quantity that denotes the similarity between two movies. 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 **linear_kernel** instead of cosine_similarities since it is much faster.

Step 3:

```
# Build a 1-dimensional array with movie titles
titles = movies['title']
indices = pd.Series(movies.index, index=movies['title'])

def genre_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:21]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]
```

This step is to write a function that returns the 20 most similar movies based on the cosine similarity score. As you can see we are enumerating and sorting based on genre_recommendations. Which will return the output by title name of a movie.

EXECUTION:

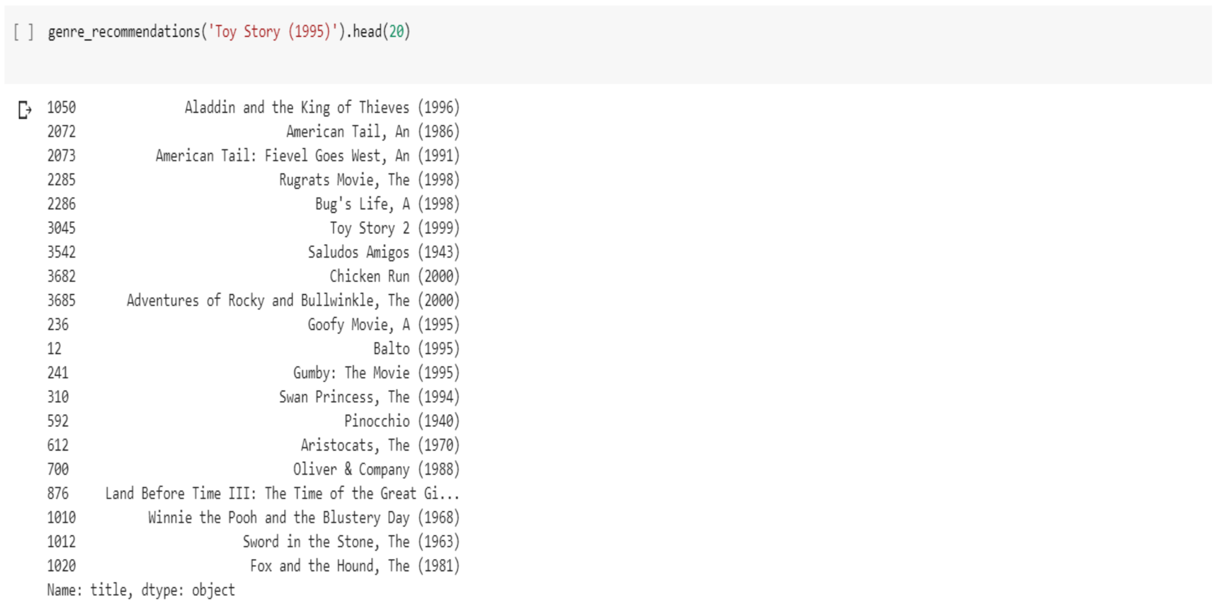
Since, our content-based recommendation system is based on genres, we have to use syntax as

genre_recommendations(‘movie_name(with year)’) . head()

for example, lets take a movie by name Toy Story (1995) the film genres are animation, comedy, children so the out put movie suggestion will all be related to these 3 genres when top 20 films are suggested if those films contain similar genres. So, this is how recommendation system works by reading whole data set and executing more or less accurate output based on user input as in below fig 1.7

```
genre_recommendations('Toy story (1995)').head(20)
```

OUTPUT:



```
[ ] genre_recommendations('Toy Story (1995)').head(20)

    1050             Aladdin and the King of Thieves (1996)
    2072                           American Tail, An (1986)
    2073         American Tail: Fievel Goes West, An (1991)
    2285                          Rugrats Movie, The (1998)
    2286                               Bug's Life, A (1998)
    3045                                 Toy Story 2 (1999)
    3542                              Saludos Amigos (1943)
    3682                                 Chicken Run (2000)
    3685       Adventures of Rocky and Bullwinkle, The (2000)
    236                                Goofy Movie, A (1995)
    12                                          Balto (1995)
    241                             Gumby: The Movie (1995)
    310                           Swan Princess, The (1994)
    592                                    Pinocchio (1940)
    612                              Aristocats, The (1970)
    700                             Oliver & Company (1988)
    876      Land Before Time III: The Time of the Great Gi...
    1010          Winnie the Pooh and the Blustery Day (1968)
    1012                        Sword in the Stone, The (1963)
    1020                        Fox and the Hound, The (1981)
    Name: title, dtype: object
```

fig 1.7

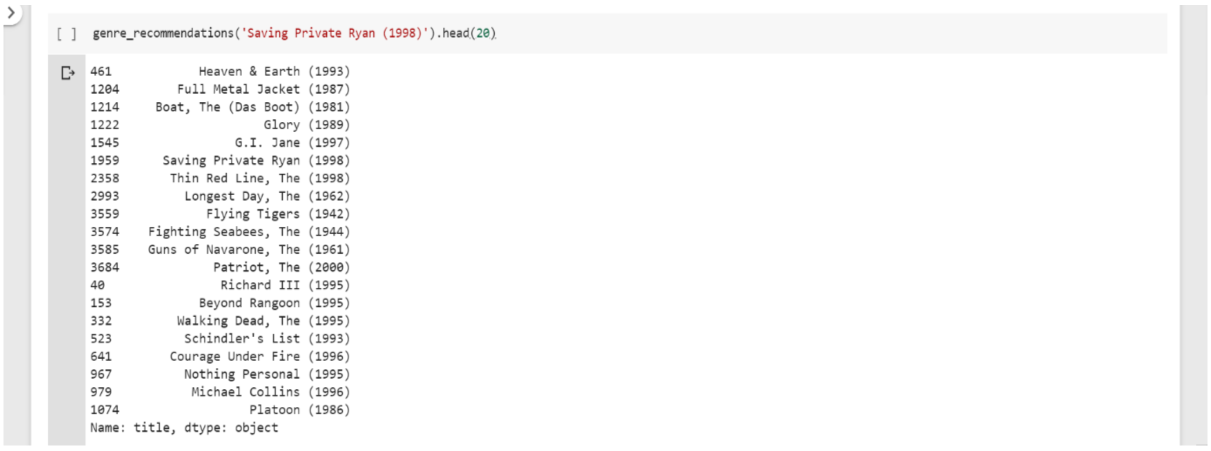
Aladdin and the king of thieves (1996), America tail, an(1986) is based genre like animation| children| comedy| so this film genre completely matches with the **toy story(1995)** genres,

Where as **fox and the hound, the (1981)** genre is animation| children so as you can see that the genres of **Toys story** have those genres too. So, this is how genres-based works by analysing the genres of each films and same goes to other outputs.

As per planning we estimated to reach 80% of accuracy and by evaluating the output by our self we can say that we reached to 82% of accuracy, and this is how Machine learning recommendation system works.

EVALUATION

Whatever we planned in the beginning of project in planning stage, those requirements have been evaluated and executed in the final stage where our model is able to recommend movies.



```
genre_recommendations('Saving Private Ryan (1998)').head(20)
```

```
461              Heaven & Earth (1993)
1204          Full Metal Jacket (1987)
1214      Boat, The (Das Boot) (1981)
1222                      Glory (1989)
1545                 G.I. Jane (1997)
1959        Saving Private Ryan (1998)
2358         Thin Red Line, The (1998)
2993           Longest Day, The (1962)
3559              Flying Tigers (1942)
3574     Fighting Seabees, The (1944)
3585     Guns of Navarone, The (1961)
3684               Patriot, The (2000)
40                  Richard III (1995)
153               Beyond Rangoon (1995)
332           Walking Dead, The (1995)
523             Schindler's List (1993)
641           Courage Under Fire (1996)
967             Nothing Personal (1995)
979              Michael Collins (1996)
1074                     Platoon (1986)
Name: title, dtype: object
```

fig 1.8

The evaluation step is more or less same like execution phase and above pictures shows and other example of our recommendation system working. And this phase is the final phase which follows the planning phase and fulfils the designing phase.

ACCURACY

Our goal at the very beginning was to reach 80% accuracy. We, are now able to reach 82% of accuracy.

Overall, here are some pros for using content-based recommendation:

- No need for data on other users, thus no cold-start or sparsity problems.
- Can recommend to users with unique tastes.
- Can recommend new & unpopular items.
- Can provide explanations for recommended items by listing content-features that caused an item to be recommended (in this case, movie genres)

CONCLUSION

Our model is now built and trained and is able to predict the data which is based on supervised learning. We took a test data to find whether our model reaches our target accuracy as above based on people interest/profile. We can use the same algorithm for any other recommendation systems of different fields.

Recommender systems are a powerful new technology for extracting additional value for a business from its user databases. These systems help users find items they want to buy from a business. Recommender systems benefit users by enabling them to find items they like. Conversely, they help the business by generating more sales. Recommender systems are rapidly becoming a crucial tool in E-commerce on the Web. Recommender systems are being stressed by the huge volume of user data in existing corporate databases, and will be stressed even more by the increasing volume of user data available on the Web. New technologies are needed that can dramatically improve the scalability of recommender systems.

In this paper we presented and experimentally evaluated a new algorithm for content-based recommender systems. Our results show that genre-based techniques hold the promise of allowing content-based algorithms to scale to large data sets and at the same time produce high-quality recommendations.

THANK YOU