Movie Recommender System
Big Data UE18CS322

Work By,
Vishnu S Reddy PES2201800118
Pranav L Nambiar PES2201800290
Rithvik Rangan PES2201800150

Introduction

We plan to use the MovieLens dataset by GroupLens to perform some analysis and make movie recommendations to users using the concepts of Big Data.

Dataset Description

We are using the MovieLens 25M dataset. This has 25 million movie ratings and has a size of over 1 gigabyte. It contains 62,000 movies reviewed by over 1,62,000 users. It also includes tag genome data with 15 million relevance scores across 1,129 tags. It was released in December of 2019.

Setup

We mainly made use of pyspark on Hadoop to perform the analysis and used the packages available in it to build models. We worked on this on Google Colab and on Jupyter Notebook on Linux.

We first import OS and install JDK, spark and Findspark. We then set JAVA_HOME and SPARK_HOME.

```
import os
```

```
#Installing Spark and JDK.
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-2.4.7/spark-2.4.7-bin-hadoop2.7.tgz
!tar xf spark-2.4.7-bin-hadoop2.7.tgz
!pip install -q Findspark
!update-alternatives --set java /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.7-bin-hadoop2.7"
```

```
#Making sure that JAVA_HOME is set properly.
os.environ["JAVA_HOME"]
```

```
]: '/usr/lib/jvm/java-8-openjdk-amd64'
```

```
#Making sure that SPARK_HOME is set properly.
os.environ["SPARK_HOME"]
```

```
]: '/content/spark-2.4.7-bin-hadoop2.7'
```

PySpark is not on sys.path by default, but that doesn't mean it can't be used as a regular library. You can address this by either symlinking pyspark into your site-packages or adding pyspark to sys.path at runtime. `Findspark` does the latter.

So, we then initialize Findspark and start a spark session using the SparkSession builder.

```
import findspark
findspark.init()
```

```
#Start Apache Spark session and context
import pyspark
from pyspark.sql import SQLContext
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('BigDataProject').getOrCreate()
```

Reading the Data

We mount our drive which contains the dataset to Colab and then read the dataset using spark.read.

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
ue).
```

```
!ls "drive/My Drive/Big_Data_Movie_Recommender"
```

```
Data  Results
```

```
DATA_PATH = "drive/My Drive/Big_Data_Movie_Recommender/Data"
RESULTS_PATH = "drive/My Drive/Big_Data_Movie_Recommender/Results"
```

```
ratings = spark.read.option("header", "true").csv(DATA_PATH+"/ratings.csv")
ratings.show(5)
```

```
+------+-------+------+----------+
|userId|movieId|rating| timestamp|
+------+-------+------+----------+
|     1|    296|   5.0|1147880044|
|     1|    306|   3.5|1147868817|
|     1|    307|   5.0|1147868828|
|     1|    665|   5.0|1147878820|
|     1|    899|   3.5|1147868510|
+------+-------+------+----------+
only showing top 5 rows
```

```
movies = spark.read.option("header", "true").csv(DATA_PATH+"/movies.csv")
movies.show(5)
```

```
+-------+--------------------+--------------------+
|movieId|               title|              genres|
+-------+--------------------+--------------------+
|      1|    Toy Story (1995)|Adventure|Animati...|
|      2|      Jumanji (1995)|Adventure|Childre...|
|      3|Grumpier Old Men ...|      Comedy|Romance|
|      4|Waiting to Exhale...|Comedy|Drama|Romance|
|      5|Father of the Bri...|              Comedy|
+-------+--------------------+--------------------+
only showing top 5 rows
```

Ratings and the movies are stored in the ratings and the movies object, respectively.

Exploratory Data Analysis

The Most Popular Movies

We find the most popular movies by counting the number of ratings a movie has received. We make use of Pyspark SQL functions to get the most popular movies.

```
from pyspark.sql.functions import *

most_popular = ratings\
.groupBy("movieId")\
.agg(count("userId"))\
.withColumnRenamed("count(userId)", "num_ratings")\
.sort(desc("num_ratings"))
```

```
#To show the top 10 most popular movies.
most_popular.show(10)
```

```
+-------+-----------+
|movieId|num_ratings|
+-------+-----------+
|    356|      81491|
|    318|      81482|
|    296|      79672|
|    593|      74127|
|   2571|      72674|
|    260|      68717|
|    480|      64144|
|    527|      60411|
|    110|      59184|
|   2959|      58773|
+-------+-----------+
only showing top 10 rows
```

As you can see only the movieId is shown above and not the movie name. So we make use of the join function and join it with the movies object based on the movieId.

```
#Showing the top 15 movies.
most_popular_movies = most_popular.join(movies, ["movieId"])
most_popular_movies = most_popular_movies \
.sort(desc("num_ratings"))
most_popular_movies.show(15)
```

```
+-------+-----------+--------------------+--------------------+
|movieId|num_ratings|               title|              genres|
+-------+-----------+--------------------+--------------------+
|    356|      81491| Forrest Gump (1994)|Comedy|Drama|Roma...|
|    318|      81482|Shawshank Redempt...|         Crime|Drama|
|    296|      79672| Pulp Fiction (1994)|Comedy|Crime|Dram...|
|    593|      74127|Silence of the La...|Crime|Horror|Thri...|
|   2571|      72674|  Matrix, The (1999)|Action|Sci-Fi|Thr...|
|    260|      68717|Star Wars: Episod...|Action|Adventure|...|
|    480|      64144|Jurassic Park (1993)|Action|Adventure|...|
|    527|      60411|Schindler's List ...|           Drama|War|
|    110|      59184|   Braveheart (1995)|    Action|Drama|War|
|   2959|      58773|   Fight Club (1999)|Action|Crime|Dram...|
|    589|      57379|Terminator 2: Jud...|       Action|Sci-Fi|
|   1196|      57361|Star Wars: Episod...|Action|Adventure|...|
|      1|      57309|    Toy Story (1995)|Adventure|Animati...|
|   4993|      55736|Lord of the Rings...|    Adventure|Fantasy|
|     50|      55366|Usual Suspects, T...|Crime|Mystery|Thr...|
+-------+-----------+--------------------+--------------------+
only showing top 15 rows
```

Top Rated Movies

We find the average ratings of movies and sort them in the descending order of their average rating.

```
top_rated = ratings\
.groupBy("movieId")\
.agg(avg(col("rating")))\
.withColumnRenamed("avg(rating)", "avg_rating")\
.sort(desc("avg_rating"))
```

```
top_rated_movies = top_rated.join(movies, ['movieId']).sort(desc("avg_rating"))
top_rated_movies.show(10)
```

```
+-------+----------+--------------------+--------------------+
|movieId|avg_rating|               title|              genres|
+-------+----------+--------------------+--------------------+
| 175095|       5.0|The Exhibitionist...|               Drama|
| 136914|       5.0|         RISE (2014)|Crime|Drama|Thriller|
| 139289|       5.0|The Shattering (2...|              Horror|
| 136782|       5.0|The Girl is in Tr...|            Thriller|
| 159050|       5.0|The Pilot's Wife ...|  (no genres listed)|
| 169338|       5.0|Brad Williams: Da...|              Comedy|
| 194280|       5.0|        Panic (1982)|       Horror|Sci-Fi|
| 200296|       5.0|The Control Group...|              Horror|
| 187951|       5.0|Father of Lights ...|         Documentary|
| 159904|       5.0|Living on Love (1...|      Comedy|Romance|
+-------+----------+--------------------+--------------------+
only showing top 10 rows
```

As we can see, none of these movies are well known even though they have 5 starts. The reason for this is because these movies have very few ratings.

```
top_rated = ratings\
.groupBy("movieId")\
.agg(count("userId"), avg(col("rating")))\
.withColumnRenamed("count(userId)", "num_ratings")\
.withColumnRenamed("avg(rating)", "avg_rating")


top_rated_movies = top_rated.join(movies, ['movieId']).sort(desc("avg_rating"), desc("num_ratings"))
top_rated_movies.show(50)
```

```
+-------+-----------+----------+--------------------+--------------------+
|movieId|num_ratings|avg_rating|               title|              genres|
+-------+-----------+----------+--------------------+--------------------+
| 165787|          3|       5.0|Lonesome Dove Chu...|             Western|
| 118268|          3|       5.0|Borrowed Time (2012)|               Drama|
| 179731|          3|       5.0|Sound of Christma...|               Drama|
| 148298|          3|       5.0|       Awaken (2013)|Drama|Romance|Sci-Fi|
| 201821|          2|       5.0| Civilisation (1969)|  (no genres listed)|
| 159797|          2|       5.0|La muerte de Jaim...|         Documentary|
| 184903|          2|       5.0|     Joy Road (2011)|         Crime|Drama|
| 133297|          2|       5.0|Genius on Hold (2...|  (no genres listed)|
| 168020|          2|       5.0|Christmas Angel (...|Comedy|Drama|Romance|
| 183357|          2|       5.0|    Bidder 70 (2013)|  (no genres listed)|
| 169232|          2|       5.0|Christmas on Salv...|  (no genres listed)|
| 179559|          2|       5.0|The Memory Book (...|       Drama|Romance|
```

We can see that these movies have around 2-3 ratings and that is the reason for them to get such a high average rating. So we put a check condition to display movies with high average ratings and the minimum number of ratings has to be 500. We then get some sensible movies which are popular and have high ratings.

```
top_rated_movies.where("num_ratings > 500").show(20)
```

```
+-------+-----------+------------------+--------------------+--------------------+
|movieId|num_ratings|        avg_rating|               title|              genres|
+-------+-----------+------------------+--------------------+--------------------+
| 171011|       1124| 4.483096085409253|Planet Earth II (...|         Documentary|
| 159817|       1747| 4.464796794504865| Planet Earth (2006)|         Documentary|
|    318|      81482| 4.413576004516335|Shawshank Redempt...|         Crime|Drama|
| 170705|       1356| 4.398598820058997|Band of Brothers ...|     Action|Drama|War|
|    858|      52498| 4.324336165187245|Godfather, The (1...|         Crime|Drama|
| 179135|        659| 4.289833080424886|Blue Planet II (2...|         Documentary|
|     50|      55366| 4.284353213163313|Usual Suspects, T...|Crime|Mystery|Thr...|
|   1221|      34188|4.2617585117585115|Godfather: Part I...|         Crime|Drama|
| 163809|        546| 4.258241758241758|Over the Garden W...|Adventure|Animati...|
|   2019|      13367|  4.25476920775043|Seven Samurai (Sh...|Action|Adventure|...|
| 142115|        564|  4.24822695035461|The Blue Planet (...|         Documentary|
|    527|      60411| 4.247579083279535|Schindler's List ...|           Drama|War|
|   1203|      16569| 4.243014062405697| 12 Angry Men (1957)|               Drama|
|    904|      20162| 4.237947624243627|  Rear Window (1954)|    Mystery|Thriller|
|   2959|      58773| 4.228310618821568|   Fight Club (1999)|Action|Crime|Dram...|
|   1193|      36058|4.2186616007543405|One Flew Over the...|               Drama|
|    750|      26714| 4.215804447106386|Dr. Strangelove o...|          Comedy|War|
|   5618|      22719| 4.212267265284564|Spirited Away (Se...|Adventure|Animati...|
| 166024|       1030| 4.210194174757282|      Whiplash (2013)|  (no genres listed)|
|    912|      26890| 4.206563778356267|   Casablanca (1942)|       Drama|Romance|
+-------+-----------+------------------+--------------------+--------------------+
only showing top 20 rows
```

All these movies are popular and are have high ratings. When we search for high rated movies on IMDB, these are the ones we get.

Most Polarizing Movies – Marmite Movies

Marmite movies are those which people either love or hate. We can find these movies by looking for the ones which have the highest standard deviation in the ratings. Standard deviation is a measure of how much the data varies from the mean.

```
ratings_stddev = ratings\
.groupBy("movieId")\
.agg(count("userId").alias("num_ratings"),
     avg(col("rating")).alias("avg_rating"),
     stddev(col("rating")).alias("std_rating")
    )\
.where("num_ratings > 500")

marmite_movies = ratings_stddev.join(movies, ['movieId'])
marmite_movies.sort(desc("std_rating")).show(15)
```

```
+-------+-----------+------------------+------------------+--------------------+--------------------+
|movieId|num_ratings|        avg_rating|        std_rating|               title|              genres|
+-------+-----------+------------------+------------------+--------------------+--------------------+
|  74754|        670| 2.403731343283582|1.6649650528666515|    Room, The (2003)|Comedy|Drama|Romance|
|  62912|        611|2.5106382978723403|1.4888552380190527|High School Music...|             Musical|
|  98203|       1569|2.5242192479286167|1.4560043846864676|Twilight Saga: Br...|Adventure|Drama|F...|
|  27899|        616|2.7767857142857144| 1.445519028350731|What the #$*! Do ...|Comedy|Documentar...|
|  91104|       1896|2.3285864978902953| 1.442346851125679|Twilight Saga: Br...|Adventure|Drama|F...|
|   1924|       2210| 2.613348416289593| 1.417228312413465|Plan 9 from Outer...|       Horror|Sci-Fi|
|  78772|       2857|2.3773188659432973| 1.405303363298029|Twilight Saga: Ec...|Fantasy|Romance|T...|
|  46062|       1165| 2.584978540772532|1.3779333361047366|High School Music...|Children|Comedy|D...|
|  81535|        759|2.7259552042160737|1.3775904753516586|Saw VII 3D - The ...|Horror|Mystery|Th...|
|  72407|       2842| 2.351161154116819| 1.377464981075965|Twilight Saga: Ne...|Drama|Fantasy|Hor...|
|  78174|        700|2.3392857142857144|1.3729814903705226|Sex and the City ...|Comedy|Drama|Romance|
|  61123|        690| 2.508695652173913|1.3661615663530424|High School Music...|Comedy|Drama|Musi...|
| 100083|        592|2.3597972972972974|1.3601552462375284|     Movie 43 (2013)|              Comedy|
|  63992|       6115| 2.391414554374489|1.3592448248808724|     Twilight (2008)|Drama|Fantasy|Rom...|
|   4255|       1721|2.1406159209761766|1.3546798615900133|Freddy Got Finger...|              Comedy|
+-------+-----------+------------------+------------------+--------------------+--------------------+
only showing top 15 rows
```

We can see that these movies are quite popular, and some people love it and a few hate it. We also make sure that the number of ratings here are more than 500 because we would not want movies which have 2 ratings like one with 1 star and one with 5 stars to come up here in this list.

Visualizations

We make use of koalas for doing the visualizations because they convert the spark data objects to something remarkably similar to Pandas data frames making it very easy to operate and plot graphs with it.

Quoting them on their docs –

The Koalas project makes data scientists more productive when interacting with big data, by implementing the pandas Data Frame API on top of Apache Spark. pandas is the de facto standard (single-node) Data Frame implementation in Python, while Spark is the de facto standard for big data processing. With this package, you can:

- Be immediately productive with Spark, with no learning curve, if you are already familiar with pandas.
- Have a single codebase that works both with pandas (tests, smaller datasets) and with Spark (distributed datasets).

We first install and import all the required packages.

```
#Installing koalas. Needed for some visualizations.
!pip install koalas > /dev/null
```

```
!pip install seaborn > /dev/null
```

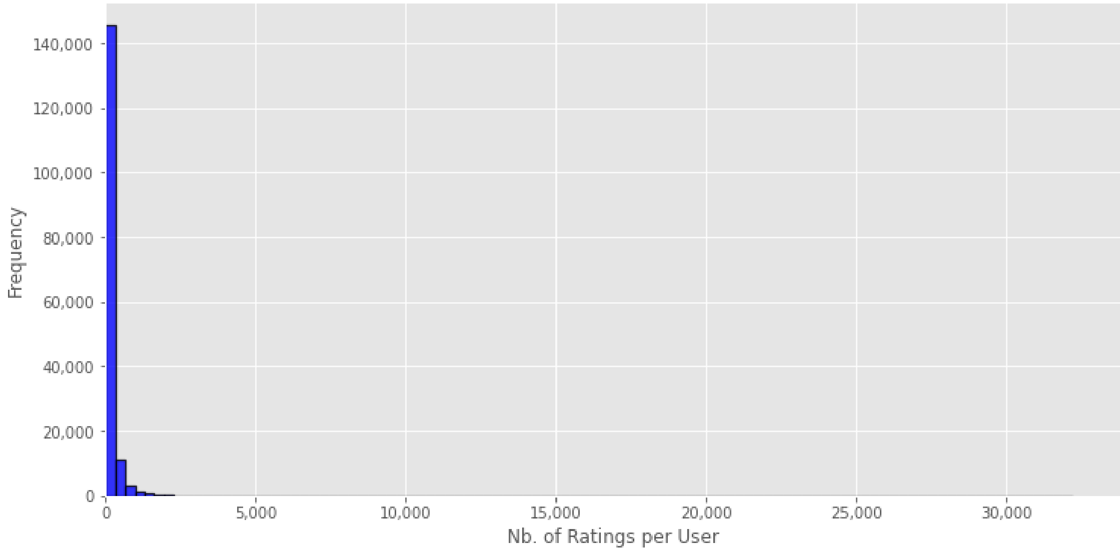
```
#Importing packages needed for visualizations
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
import random
from pprint import pprint
from matplotlib.lines import Line2D
import databricks.koalas as ks

#Set-up
plt.style.use('ggplot')
```

Number of Ratings vs Users

In this plot we hope to visualize the number of ratings given by users. We convert the spark data object to a Koalas data frame and then use this matplotlib package to plot the graph.

```
ks.set_option('compute.default_index_type', 'sequence')
ks.set_option('compute.ops_on_diff_frames', True)
dfRatingsKdf = ratings.to_koalas()
```

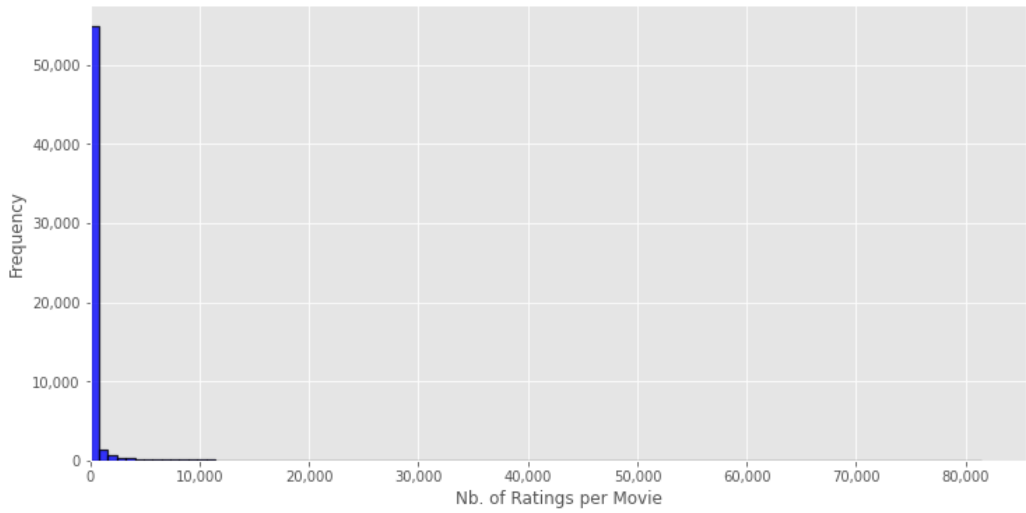
```
f, ax = plt.subplots(figsize=(12,6))
userRatingGroup = dfRatingsKdf.groupby("userId")['rating'].count()
userRatingGroup.hist(bins=100, color='blue', edgecolor='black',
                     linewidth=1.25, alpha=0.78, ax=ax)
ax.set_xlabel('Nb. of Ratings per User')
ax.set_xlim(0.0)
ax.set_xticklabels(['{:,}'.format(int(x)) for x in ax.get_xticks().tolist()])
ax.set_yticklabels(['{:,}'.format(int(x)) for x in ax.get_yticks().tolist()])
plt.show()
```



We can see from this graph that the greatest number of users have rated zero or very few movies.

Number of Ratings per Movie

```
f, ax = plt.subplots(figsize=(12,6))
movieRatingGroup = dfRatingsKdf.groupby("movieId")['rating'].count()
movieRatingGroup.hist(bins=100, color='blue', edgecolor='black',
                      linewidth=1.25, alpha=0.78, ax=ax)
ax.set_xlabel('Nb. of Ratings per Movie')
ax.set_xlim(0.0)
ax.set_xticklabels(['{:,}'.format(int(x)) for x in ax.get_xticks().tolist()])
ax.set_yticklabels(['{:,}'.format(int(x)) for x in ax.get_yticks().tolist()])
plt.show()
```

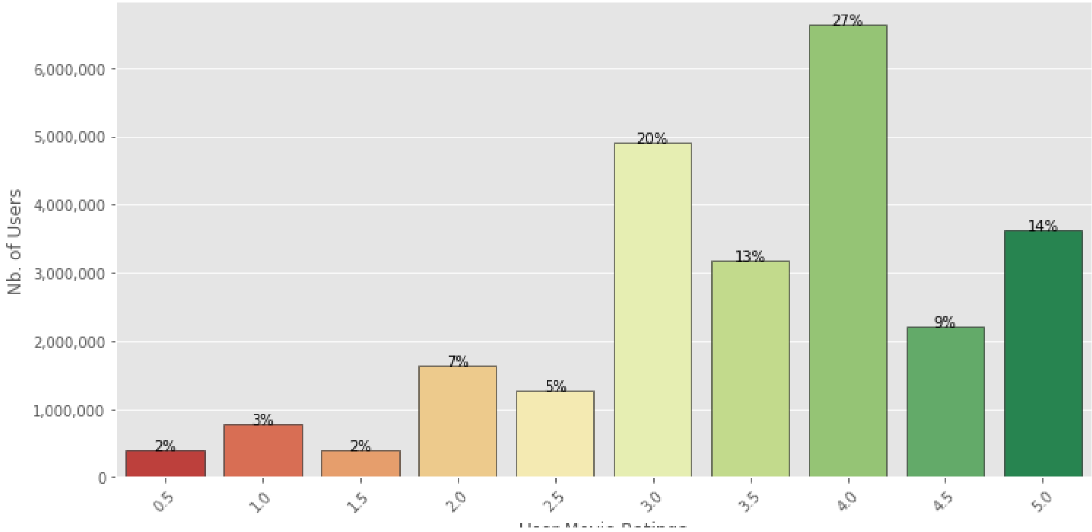


We can see from the above graph that most of the movies have very few to no ratings.

How Users Rate?

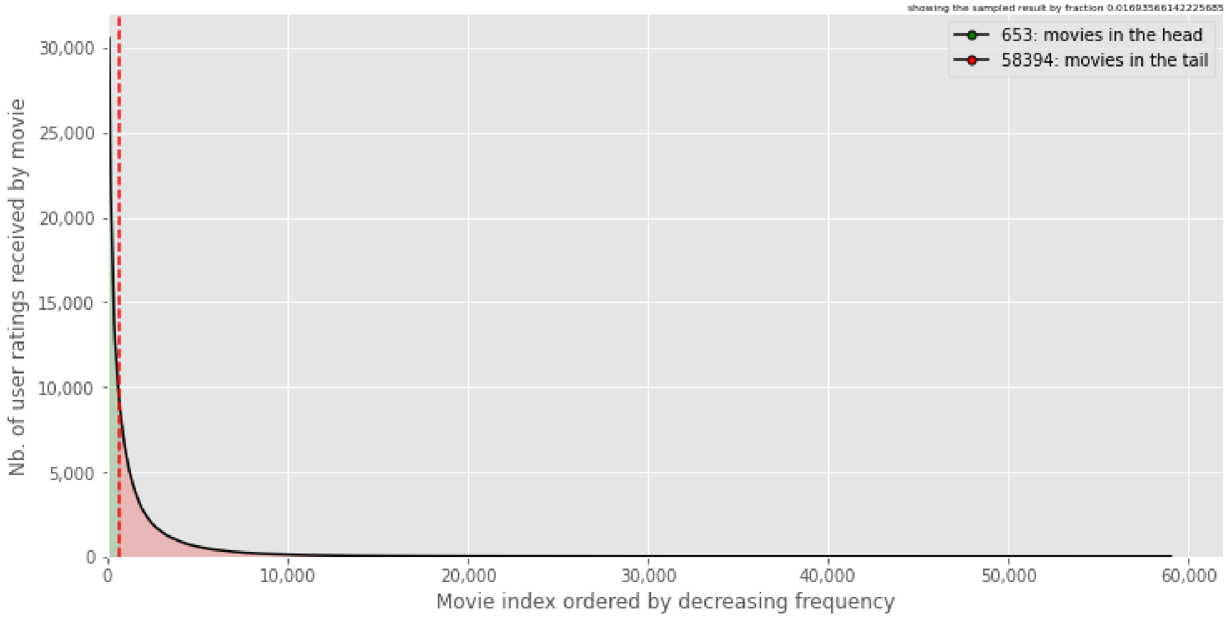
We wanted to see which rating is most used. Are most of the users critics or do they watch movies for fun and entertainment?

```
movieRatingDistGroup = dfRatingsKdf['rating'].value_counts() \
                                              .sort_index() \
                                              .reset_index() \
                                              .to_pandas()

# Create Matplotlib Figure
fig, ax = plt.subplots(figsize=(12,6))

# Main Figure: Seaborn Barplot
sns.barplot(data=movieRatingDistGroup, x='index', y='rating',
            palette='RdYlGn', edgecolor="black", ax=ax)

# Set Xaxis and Yaxis
ax.set_xlabel("User-Movie Ratings")
ax.set_ylabel('Nb. of Users')
ax.xaxis.set_tick_params(rotation=45)

# Thousand separator on Yaxis labels
ax.set_yticklabels(['{:,}'.format(int(x)) for x in ax.get_yticks().tolist()])

# Add percentage text on top of each bar
total = float(movieRatingDistGroup['rating'].sum())
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,
            height+350,
            '{0:.0%}'.format(height/total),
            ha="center")

# Display plot
plt.show()
```



We can see from the above graph that majority of the users are not critics. They simple chose to go with 3, 4 or 5.

Number of User Ratings Received by Movies



We can see that only 653 movies lie within the top 0.5% of the movies, taking number of ratings into consideration. The remaining form the 'long tail'. Chris Anderson, in his book, The Long Tail: Why the Future of Business is Selling Less of More, argues that products in low demand or that have a low sales volume can collectively build a better market share than their relatively few but popular rivals, provided the product distribution is large enough. In this regard, an online marketspace alleviates competition for shelf space, allowing immeasurable number of products to be sold. He notes that Amazon, Apple, and Yahoo are some of the businesses applying this strategy.

A good recommendation engine should exploit this strategy and recommend more items from the long tail to the users.

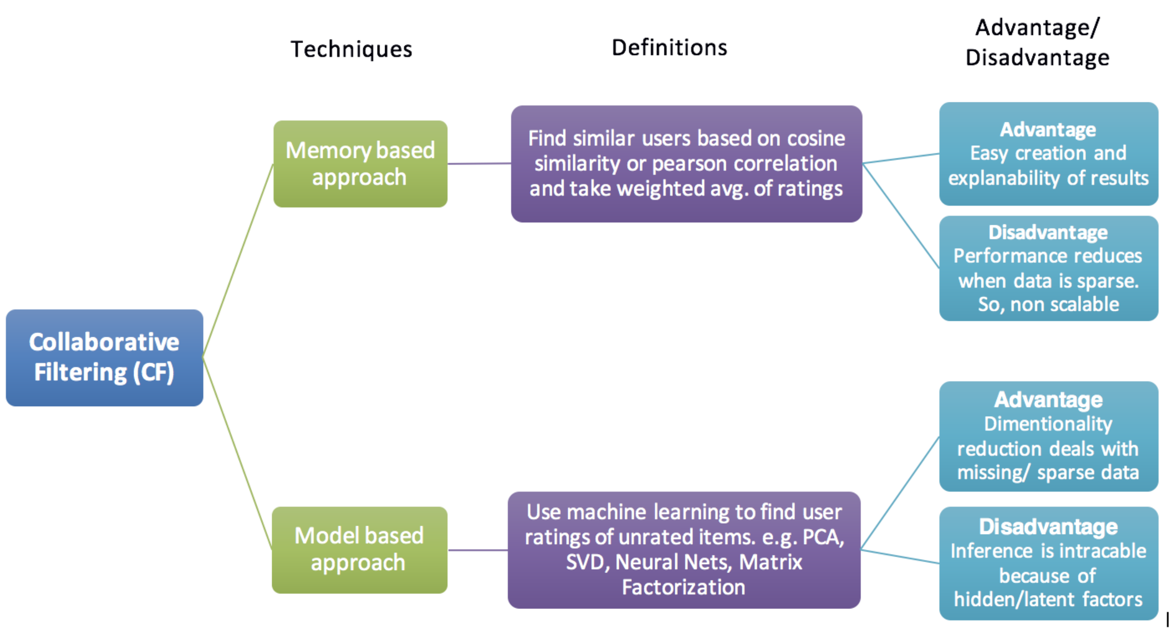
Recommendation System

There are two main approaches to recommendation systems:

- Collaborative Filtering: Based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. It does not rely on machine analysable content and therefore, it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself.

- Content Based Filtering: Based on a description of the item and a profile of the user's preferences. These methods are best suited to situations where there is known domain knowledge, that is, data on an item (name, location, description, etc.), but not on the user. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on an item's features. In this system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes.

Our project only deals with the collaborative filtering approach. Content based filtering requires domain knowledge about the items at hand. The genome-scores.csv and genome-tags.csv files of the MovieLens dataset contain this information. The relevance score of each movie with respect to each of the user-submitted 1129 tags has been given, which was precomputed with the help of machine learning, as described in this paper. One could use an algorithm such as the tf–idf (Term Frequency - Inverse Document Frequency) representation (also called vector space representation) to create a content-based profile of users based on a weighted vector of item features, which here, are the tags. However, this is beyond the present scope and shall not be explored in this project.



There are two kinds of collaborative filtering, as shown in the diagram:

- Memory-Based / Neighborhood-Based Approach: Rating data is used to compute the similarity between the users or items. It is a non-parametric approach, that is, we don't attempt to learn any parameter using an optimization algorithm. This algorithm calculates the similarity between two users or items, and produces a prediction for the user by taking the weighted average of all the ratings. Similarity computation between items or users is an important part of this approach. Multiple measures, such as Pearson correlation and vector cosine-based similarity are used for this.

The Pearson correlation similarity of two users x, y is defined as:

$$\text{simil}(x, y) = \frac{\sum\limits_{i \in I_{xy}} (r_{x,i} - \bar{r_x})(r_{y,i} - \bar{r_y})}{\sqrt{\sum\limits_{i \in I_{xy}} (r_{x,i} - \bar{r_x})^2} \sqrt{\sum\limits_{i \in I_{xy}} (r_{y,i} - \bar{r_y})^2}}$$

The cosine-based approach defines the cosine-similarity between two users x and y as:

$$\text{simil}(x, y) = \cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \times ||\vec{y}||} = \frac{\sum\limits_{i \in I_{xy}} r_{x,i} r_{y,i}}{\sqrt{\sum\limits_{i \in I_x} r_{x,i}^2} \sqrt{\sum\limits_{i \in I_y} r_{y,i}^2}}$$

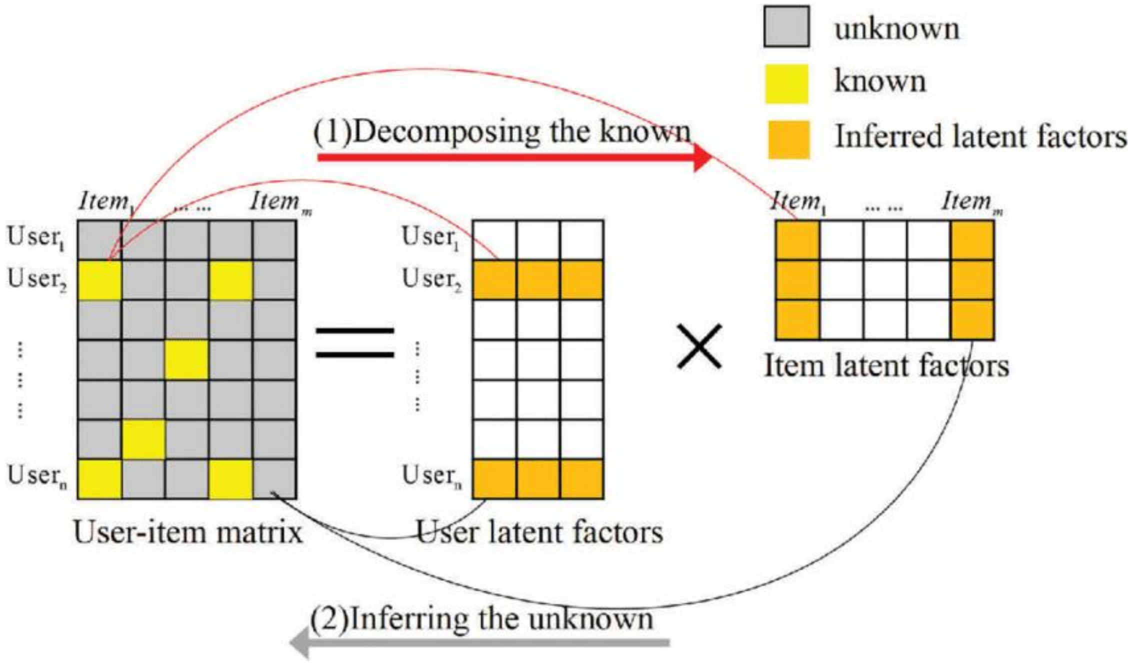
There are two subcategories to this approach are:

1)Item-Item based filtering: “Users who liked this item also liked …”

2)User-Item based Filtering: “Users who are similar to you also liked …”

The disadvantage of this method is that its performance decreases for a sparse data, which hinders scalability of this approach for most of the real-world problems.

- Model-Based Approach: Models are developed using machine learning algorithms to predict a user’s rating of unrated items. We can use matrix factorization algorithms such as Alternating Least Squares to split the ratings matrix into a user matrix and item matrix, where the preferences of the user and the properties of the item are defined by a number of hidden/latent factors.



```
from pyspark.ml.recommendation import ALS
```

```
from pyspark.sql.types import IntegerType
ratings = ratings.withColumn("userId",ratings['userId'].cast(IntegerType())) \
                    .withColumn("movieId",ratings['movieId'].cast(IntegerType())) \
                    .withColumn("rating",ratings['rating'].cast(IntegerType()))
```

```
df_train, df_test = (ratings.randomSplit([0.7, 0.3], seed = 1))
```

After splitting the ratings dataset into training and testing sets, we build the Alternating Least Squares Model.

```
als = ALS(rank=10, maxIter=5, seed=0, userCol= "userId", itemCol= "movieId", ratingCol="rating")
    # Rank is the number of latent/hidden factors
```

Collaborative Filtering suffers from the Cold Start Problem, which describes the difficulty of making recommendations when the items are new. This is due to the lack of past history of interaction of the users with the item.

Content-based filtering is less prone to this problem because the recommendations are made based on the feature the items possess, so even if no interaction exists for a new item, its features will allow for a recommendation to be made.

Since the split was random, the testing dataset could have movies that have never been seen in the training dataset, as a result of which no predictions can be made for them. This, in turn, won't let us compute the regression metric scores later to evaluate the prediction on the test dataset. Possible workarounds are to remove the entries for the items in the test dataset which have not been seen in the train dataset, or to impute their prediction to some average value. We have opted for the former, by setting the ColdStartStrategy of our model to 'drop'.

```
als.setColdStartStrategy('drop')
```

We define the Root Mean Square Error metric to evaluate our predictions.

```
from pyspark.ml.evaluation import RegressionEvaluator
reg_eval = RegressionEvaluator(predictionCol="prediction", labelCol="rating", metricName="rmse")
```

We now fit our ALS model to the train dataset.

```
als_model = als.fit(df_train)
```

We backup our model so that we can load it for future use.

```
from pyspark.ml.recommendation import ALSModel
als_model.save(RESULTS_PATH+"ALS_MovieLens_1")
als_model = ALSModel.load(RESULTS_PATH+"ALS_MovieLens_1")
```

Our ratings matrix is very sparse, as a result of which neighbourhood based collaborative filtering will not be efficient. But we can apply our similarity metrics to the newly obtained itemFactors and userFactors datasets, which are complete matrices, to obtain recommendations based on item-item similarity and user-user similarity.

We opt for the cosine similarity metric which is a part of the scipy package. The output of this metric is 0 when the vectors being compared are the same. We register our user defined 'distance' function with the sparkContext in order to use it on pyspark data frames.

```
from pyspark.sql.types import DoubleType
import numpy as np
import scipy
import scipy.spatial

def distance(v1, v2):
    v1 = np.array(v1)
    v2 = np.array(v2)
    return float(scipy.spatial.distance.cosine(v1, v2))

spark.udf.register("distance", distance, DoubleType())
```

We implement a function to recommend movies similar to a movie, by selecting those movies whose vectors in 10-D space are closest to the movie vector for which we want to make the recommendation.

```
def recommendation_by_i2i(movie_id):
    return (als_model
     .itemFactors
     .filter(F.col("id") == movie_id)
     .alias("t1")
     .crossJoin(als_model.itemFactors.alias("t2"))
     .withColumn("similarity", F.expr("distance(t1.features, t2.features)"))
     .join(movies, F.col("t2.id") == F.col("movieId"))
     .orderBy(F.asc("similarity"))
     .select("movieId", "title", "similarity")
    )
```

Suppose a user really likes the movie _Blade Runner (1982)_(movieId=541). We now try to recommend movies like it.

```
recommendation_by_i2i(541).show(20, False)
```

```
+-------+---------------------------------------------+--------------------+
|movieId|title                                        |similarity          |
+-------+---------------------------------------------+--------------------+
|541    |Blade Runner (1982)                          |0.0                 |
|1080   |Monty Python's Life of Brian (1979)          |0.012101173928981468|
|73914  |Sometimes a Great Notion (1970)              |0.01240904214166283 |
|6104   |Monty Python Live at the Hollywood Bowl (1982)|0.013113274085467697|
|5965   |Duellists, The (1977)                        |0.013625690509695643|
|1218   |Killer, The (Die xue shuang xiong) (1989)    |0.014535267911132355|
|148288 |Who Am I This Time? (1982)                   |0.015166192408404777|
|1136   |Monty Python and the Holy Grail (1975)       |0.015209555968390687|
|145755 |The Dark Glow of the Mountain (1985)         |0.015227092839303014|
|152292 |Mojin: The Lost Legend (2015)                |0.01574341465655449 |
|62049  |1984 (1956)                                  |0.016103574573950064|
|6461   |Unforgiven, The (1960)                       |0.017456390540486644|
|1214   |Alien (1979)                                 |0.01755678923504267 |
|86318  |Goddess, The (1958)                          |0.017728778975932502|
|924    |2001: A Space Odyssey (1968)                 |0.017867493902059994|
|1233   |Boot, Das (Boat, The) (1981)                 |0.018440178775429916|
|1208   |Apocalypse Now (1979)                        |0.018452899143339585|
|158595 |Labyrinth (2002)                             |0.018827403110057483|
|128127 |Commandos (1968)                             |0.018827404448030438|
|150479 |The Battle of Sutjeska (1973)                |0.018827404448030438|
+-------+---------------------------------------------+--------------------+
only showing top 20 rows
```

We observe that the recommendations are fairly relevant. For example, both *1984* and *Blade Runner* are movies set in future dystopias and *2001: A Space Odyssey* and *Blade Runner* are works of science fiction.

Now, we implement a function to find users similar to a particular user, i.e., the users for whom the vectors in 10-D space are closest to the vector of the user in question. All users who are within a 0.03 similarity are included and recommendations are made for the current user based on the top movies that the users like him have watched.

```
top_rated_movies_by_user = (ratings
                            .filter("rating = 4 or rating = 5 or rating = 4.5")
                            .groupBy("userId")
                            .agg(F.collect_set("movieId").alias("top_movies")))
top_rated_movies_by_user.show()
```

```
+------+--------------------+
|userId|          top_movies|
+------+--------------------+
|100010|[647, 1047, 1, 60...|
|100140|[2348, 2313, 1189...|
|100227|[6, 3, 662, 62, 7...|
|100263|[2105, 5995, 3300...|
|100320|[6873, 1719, 2160...|
|100553|[5995, 2160, 5679...|
|100704|[74458, 4886, 412...|
|100735|[110, 2000, 74458...|
|100768|[41, 306, 1450, 1...|
| 10096|[784, 832, 1, 839...|
|100964|[62081, 48744, 13...|
|101021|[2859, 858, 1950,...|
|101122|[60069, 64614, 40...|
|101205|[1, 352, 141, 307...|
|101261|[33794, 37380, 43...|
|101272|[189333, 140956, ...|
|102113|[74458, 81845, 33...|
|102521|[110, 350, 356, 2...|
|102536|[3, 783, 1, 141, ...|
|102539|[110, 185, 2231, ...|
+------+--------------------+
only showing top 20 rows
```

We now make recommendations for the userId = 1.

```
recommendation_by_u2u(1).show(20, False)
```

```
+-------+-----------------------------------------------------------+
|movieId|title                                                      |
+-------+-----------------------------------------------------------+
|2160   |Rosemary's Baby (1968)                                     |
|1207   |To Kill a Mockingbird (1962)                               |
|1215   |Army of Darkness (1993)                                    |
|79132  |Inception (2010)                                           |
|589    |Terminator 2: Judgment Day (1991)                          |
|7445   |Man on Fire (2004)                                         |
|87192  |Attack the Block (2011)                                    |
|6385   |Whale Rider (2002)                                         |
|179135 |Blue Planet II (2017)                                      |
|2011   |Back to the Future Part II (1989)                          |
|57274  |[REC] (2007)                                               |
|64839  |Wrestler, The (2008)                                       |
|2867   |Fright Night (1985)                                        |
|103688 |Conjuring, The (2013)                                      |
|194004 |Halloween (2018)                                           |
|4993   |Lord of the Rings: The Fellowship of the Ring, The (2001)|
|104841 |Gravity (2013)                                             |
|187541 |Incredibles 2 (2018)                                       |
|85788  |Insidious (2010)                                           |
|99114  |Django Unchained (2012)                                    |
+-------+-----------------------------------------------------------+
only showing top 20 rows
```

When compared to the movies that userId 1 has watched, these are fairly relevant recommendations, belonging to the drama/comedy/sci-fi genres.

We now use the model to obtain predictions on the train dataset and evaluate the predictions.

```
training_predictions_df = als_model.transform(df_train)
reg_eval.evaluate(training_predictions_df)
```

```
]: 0.8193309774106281
```

```
training_predictions_df.show()
```

```
+------+-------+------+----------+----------+
|userId|movieId|rating| timestamp|prediction|
+------+-------+------+----------+----------+
| 32855|    148|     4|1029309135| 2.3990471|
| 26480|    148|     2| 915406133| 1.9864883|
| 38199|    148|     2| 835601960| 2.4336302|
|159730|    148|     3| 842162037|  2.716928|
| 33354|    148|     3| 938886119| 2.6935844|
| 47989|    148|     2| 833173771| 2.9645228|
| 72337|    148|     2| 944246202|  2.777789|
|151614|    148|     1| 878170956|  2.731505|
|  5055|    148|     3| 842463284| 2.8496578|
|108767|    148|     3|1276969740| 2.5595648|
| 21531|    148|     3| 834035555| 3.0218282|
| 38679|    148|     3| 853421750| 2.5657732|
| 99684|    148|     3|1027645782| 2.9732146|
| 35969|    148|     2| 835094487|  2.794639|
| 54331|    148|     2| 954702916| 2.9416816|
| 77130|    148|     1| 831284829| 1.1798544|
| 29943|    148|     3|1049216998| 2.8596456|
|117168|    148|     4| 835820190| 3.2516797|
| 28229|    148|     1| 833850593| 2.6150947|
| 31376|    148|     2| 901681963| 2.5119638|
+------+-------+------+----------+----------+
only showing top 20 rows
```

Since we configured our model to not predict the user ratings for the movies it hasn't encountered before, there are no NaN values in the prediction column, which lets us compute the RMSE metric to evaluate our model's performance.

```
+------+-------+------+---------+----------+
|userId|movieId|rating|timestamp|prediction|
+------+-------+------+---------+----------+
|     0|      0|     0|        0|         0|
+------+-------+------+---------+----------+
```

```
reg_eval.evaluate(validation_predictions_df)
```

```
]: 0.8576732625272104
```

We observe that our model performs slightly worse on the test dataset, but this is expected.

```
validation_predictions_df.show()
```

```
+------+-------+------+----------+----------+
|userId|movieId|rating| timestamp|prediction|
+------+-------+------+----------+----------+
|     1|   1250|     4|1147868414| 3.3892672|
|     1|   2161|     3|1147868609| 2.9195266|
|     1|  27266|     4|1147879365| 3.3538353|
|     1|   2843|     4|1147868891|   3.80054|
|     1|    306|     3|1147868817| 4.1096406|
|     1|   3448|     4|1147868480|  3.211211|
|     1|   4308|     3|1147868534| 3.3079157|
|     1|   4973|     4|1147869080|  3.985975|
|     1|   5767|     5|1147878729| 3.6437132|
|     1|   5912|     3|1147878698| 3.2279365|
|     1|   5952|     4|1147868053| 3.3478596|
|     1|   6016|     5|1147869090| 3.8530746|
|     1|   6377|     4|1147868469|  3.213149|
|     1|   6539|     3|1147868461| 2.9053173|
|     1|   7234|     4|1147868869|  3.627199|
|     1|   7361|     5|1147880055|  3.841207|
|     1|   7938|     2|1147878063|  3.478118|
|     1|   8154|     5|1147868865| 3.4568613|
|    10|   1962|     3|1227570828| 2.7332594|
|    10|   2915|     3|1227570836| 2.8876076|
+------+-------+------+----------+----------+
only showing top 20 rows
```

Now, we make predictions for the ratings given by the user with Id = 1.

```
predictions=als_model.transform(ratings)
predictions.createOrReplaceTempView("predictions_sql")
movies.createOrReplaceTempView("movies_sql")
```

We have converted the predictions and movies data frames to SQL tables, which allows us to use Spark SQL in order to query the tables for the required data.

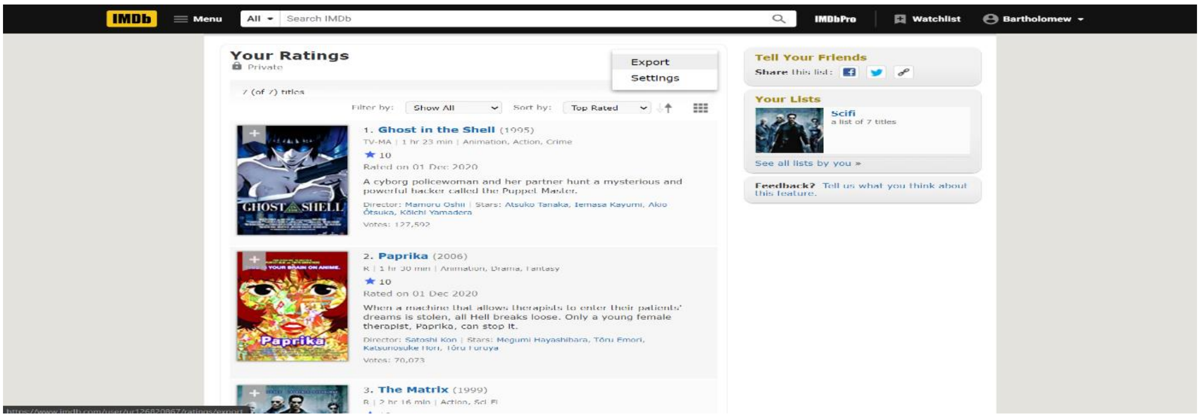
```
spark.sql(
"""select p.userId, p.movieId, p.rating, p.prediction, m.title, m.genres
from predictions_sql p
join movies_sql m on p.movieId=m.movieId
where p.userId == 1
order by p.prediction desc
""").show(20,False)
```

You can see the prediction in the next page.

```
-------------------------------------+
|userId|movieId|rating|prediction|title
enres                                |
+------+-------+------+----------+------------------------------------------------------------
-------------------------------------+
|1     |306    |3     |4.1096406 |Three Colors: Red (Trois couleurs: Rouge) (1994)
rama                                 |
|1     |307    |5     |4.0995245 |Three Colors: Blue (Trois couleurs: Bleu) (1993)
rama                                 |
|1     |296    |5     |4.0422907 |Pulp Fiction (1994)
omedy|Crime|Drama|Thriller           |
|1     |4973   |4     |3.985975  |Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)
omedy|Romance                        |
|1     |665    |5     |3.8845673 |Underground (1995)
omedy|Drama|War                      |
|1     |6016   |5     |3.8530746 |City of God (Cidade de Deus) (2002)
ction|Adventure|Crime|Drama|Thriller|
|1     |7361   |5     |3.841207  |Eternal Sunshine of the Spotless Mind (2004)
rama|Romance|Sci-Fi                  |
|1     |3949   |5     |3.8231428 |Requiem for a Dream (2000)
rama                                 |
|1     |2843   |4     |3.80054   |Black Cat, White Cat (Crna macka, beli macor) (1998)
omedy|Romance                        |
|1     |5878   |4     |3.7910283 |Talk to Her (Hable con Ella) (2002)
rama|Romance                         |
|1     |8014   |3     |3.7812328 |Spring, Summer, Fall, Winter... and Spring (Bom yeoreum gaeu
rama                                 |
|1     |2351   |4     |3.734074  |Nights of Cabiria (Notti di Cabiria, Le) (1957)
rama                                 |
|1     |1175   |3     |3.733309  |Delicatessen (1991)
omedy|Drama|Romance                  |
|1     |2692   |5     |3.7194128 |Run Lola Run (Lola rennt) (1998)
ction|Crime                          |
|1     |4144   |5     |3.7120512 |In the Mood For Love (Fa yeung nin wa) (2000)
rama|Romance                         |
|1     |2068   |2     |3.695225  |Fanny and Alexander (Fanny och Alexander) (1982)
rama|Fantasy|Mystery                 |
|1     |6954   |3     |3.6941934 |Barbarian Invasions, The (Les invasions barbares) (2003)
omedy|Crime|Drama|Mystery|Romance    |
|1     |7323   |3     |3.690628  |Good bye, Lenin! (2003)
omedy|Drama                          |
|1     |1237   |5     |3.682242  |Seventh Seal, The (Sjunde inseglet, Det) (1957)
rama                                 |
|1     |1217   |3     |3.6799335 |Ran (1985)
rama|War                             |
+------+-------+------+----------+------------------------------------------------------------
```

Prediction Based on User's IMDb Ratings

One can export his/her IMDb ratings by choosing the Export option as shown below. This results in a download prompt for a CSV file which contains all ratings with other details.



We have to pre-process the data to include the new user ratings in the dataset. There is no userId=0, so we assign 0 to be the new user. We also binarize the movie ratings, i.e., all ratings greater than 3 are set to 1, which indicates that the user enjoyed the movie, and such movies are included for recommendation, while those below 3 are set to 0, which indicates that user disliked the movie.

Popularity Model

In this model, we recommend the most popular movies to all the users. We make use of the logarithmic scaling factor which penalizes movies with fewer ratings.

The recommended movies are shown below.

movieId	sumRating	nbRatings	meanLogUserRating	title
318	251444.5	56981	48.32200642241223	Shawshank Redemption, The (1994)
296	233476.0	55743	45.77335501425202	Pulp Fiction (1994)
858	159121.0	36803	45.45532488277934	Godfather, The (1972)
50	165948.5	38720	45.27630330905071	Usual Suspects, The (1995)
527	179594.0	42294	45.23353708263301	Schindler's List (1993)
593	216082.0	52049	45.08516477258331	Silence of the Lambs, The (1991)
2571	210869.0	50745	45.022654401198956	Matrix, The (1999)
2959	173603.5	41049	44.92452813293522	Fight Club (1999)
260	198706.0	48264	44.40023986999179	Star Wars: Episode IV - A New Hope (1977)
356	231598.0	57226	44.33476644811912	Forrest Gump (1994)
1196	166377.0	40139	43.937653026516884	Star Wars: Episode V - The Empire Strikes Back (1980)
1198	158013.0	38320	43.51842643850247	Raiders of the Lost Ark (Indiana Jones and the Raiders of th 1)
2858	154837.0	37697	43.28114869723798	American Beauty (1999)
4993	160127.5	39166	43.23746802953712	Lord of the Rings: The Fellowship of the Ring, The (2001)
1221	102362.0	24041	42.95072215253253	Godfather: Part II, The (1974)
7153	145912.5	35685	42.861866228538794	Lord of the Rings: The Return of the King, The (2003)
58559	121521.5	29146	42.86179531810681	Dark Knight, The (2008)
608	136872.0	33300	42.80152950706202	Fargo (1996)
1193	106061.5	25146	42.736947520338674	One Flew Over the Cuckoo's Nest (1975)

The popularity model recommender can be used to solve the cold start problem caused by new users with no previous user interaction to give recommendations.

Alternating Least Squares

We perform ALS as we've already done above.

```
%%time
tempALS = ALS(maxIter=10, rank=10, regParam=0.1, nonnegative=True,
              userCol='userId', itemCol='movieId', ratingCol='rating',
              coldStartStrategy='drop', implicitPrefs=False, seed=SEED)

mlALSFitted = tempALS.fit(dfRatingsTrain)
```

```
CPU times: user 63.7 ms, sys: 44.8 ms, total: 108 ms
Wall time: 59.6 s
```

```
mlALSFitted.save(RESULTS_PATH+"/ALS_MovieLens_25M")
```

```
mlALSFitted = ALSModel.load(RESULTS_PATH+"/ALS_MovieLens_25M")
```

RMSE

```
%%time
predictions = mlALSFitted.transform(dfRatingsTest)
evaluator = RegressionEvaluator(metricName='rmse', labelCol='rating',
                                predictionCol='prediction')
rmse = evaluator.evaluate(predictions)
print('RMSE (Test Set):', rmse)
```

```
RMSE (Test Set): 0.8147259683304076
CPU times: user 66.4 ms, sys: 25.4 ms, total: 91.8 ms
Wall time: 36.1 s
```

Now, we generate top 20 movie recommendations for all the users.

```
resultsALS = mlALSFitted.recommendForAllUsers(20)

resultsALS = resultsALS.withColumn('recommendations',
              udf_format_recommendations(F.col("recommendations"))) \
              .toDF('userId', 'predictions')
```

```
# Downloaded Packages (not available by Default)
import databricks.koalas
```

```
resultsALSExpanded = resultsALS \
                        .withColumn("movieId", F.explode("predictions")) \
                        .drop('predictions') \
                        .join(dfMovies, "movieId")

resultsALSKdf = resultsALSExpanded.to_koalas()

MostRecommendedMoviesForAllUsers = resultsALSKdf.groupby(["movieId", "title"])['userId'].count()
MostRecommendedMoviesForAllUsers = MostRecommendedMoviesForAllUsers.sort_values(ascending=False)
```

The movies recommended the greatest number of times are shown below.

movieId	title	
203882	Dead in the Water (2006)	144255
183947	NOFX Backstage Passport 2	142233
194434	Adrenaline (1990)	136780
196787	The Law and the Fist (1964)	119244
165689	Head Trauma (2006)	98235
192089	National Theatre Live: One Man, Two Guvnors (2011)	96258
143422	2 (2007)	90441
166812	Seeing Red: Stories of American Communists (1983)	86752
117352	A Kind of America 2 (2008)	84742
194334	Les Luthiers: El Grosso Concerto (2001)	82590
121919	The Good Mother (2013)	74576
128667	Wiseguy (1996)	74542
197355	Once Upon a Ladder (2016)	72643
165559	Ο Θανάσης στη χώρα της σφαλιάρας (1976)	70270

The recommendations to the IMDb dataset we gave to the model are shown below.

```
%%time
predictionsPerso = resultsALS.filter(F.col("userId")==0) \
                             .select(F.explode("predictions") \
                             .alias("movieId")) \
                             .join(dfMovies.select(["movieId", "title"]),
                                   "movieId") \
                             .join(dfRatings.filter(F.col("userId")==0),
                                   ['movieId'], how='left')

predictionsPerso.select(["title"]).show(10, truncate=False)
```

```
+------------------------------------------+
|title                                     |
+------------------------------------------+
|The Country Cousin (1936)                 |
|Foster (2018)                             |
|Cássia (2015)                             |
|Insane (2016)                             |
|Olga (2004)                               |
|Argo (2004)                               |
|.hack Liminality In the Case of Yuki Aihara|
|NOFX Backstage Passport 2                 |
|.hack Liminality In the Case of Kyoko Tohno|
|Red, Honest, in Love (1984)               |
+------------------------------------------+
only showing top 10 rows
```

Potential Improvements to the Recommender System

- **Hybrid recommender systems**: The major weakness of collaborative filtering is that forgoing the actual characteristics of items (for movies, meta-information such as genres, actor/actresses, director, country of origin) hurts both recommender accuracy and interpretability of recommendations. In practice, industrial recommender systems use hybrid approaches that combine both user similarity (collaborative filtering) and item characteristics (content-based approach).
- **Reducing dataset size and user-item segmentation**: One reason why our collaborative filtering model struggled to generate sound recommendations for all users is that our model training included every user (e.g. single rating users, users with few ratings or many ratings). A more ideal strategy would be to segment our ratings data into seperate but homogeneous user datasets and train different recommender systems on those separate data chunks, potentially improving our results.