MOVIE RECOMMENDATION SYSTEM

Project Description

- Recommendation engines play a critical role in customer engagement and retention for online media and entertainment industry.
- Recommendation system is a system that seeks to predict or filter preferences according to the user’s choices. It helps to personalize user experience to find something they like.
- Recommendation Engines are the programs which basically compute the similarities between two entities and on that basis, they give us the targeted output.
- This movie recommendation system recommends movies to a user or a client by evaluating data set.

Problem Statement

- Build a movie recommendation system based on the user ratings.
- General recommendation system: As for recommendation, for this method, we always recommend those movies with the highest average rating and more than certain number of ratings.
- User based recommendation system: Used similarities between the user ratings and predicted recommendations for the user.

Objectives:

Implement a collaborative-filtering approach to recommend movies to user

Predict the rating that a user would give to a movie that he has not yet rated.

Recommend top 10 movies suitable to the user

Minimize the difference between predicted and actual rating (RMSE)

Dataset Description

The data set we used for this project is from this source : https://www.kaggle.com/netflix-inc/netflix-prize-data/data

For this project, I have utilized four datasets :

- The first dataset is **TRAINING DATASET FILE** where it provides a directory containing 17770 files, one per movie. The first line of each file contains the movie id followed by a colon. Each subsequent line in the file corresponds to a rating from a customer and its date in the following format: Customer-ID, Rating, Date.
- The second dataset is **MOVIES FILE** where there is Movie information in "movie_titles.txt" is in the following format: Movie ID, Year Of Release, Title.
- The Third dataset is **QUALIFYING AND PREDICTION DATASET FILE** where it consists of lines indicating a movie id, followed by a colon, and then customer ids and rating dates, one per line for that movie id. The movie and customer ids are contained in the training set. Of course the ratings are withheld. There are no empty lines in the file.
- The fourth dataset is the **PROBE DATASET FILE** where it allows you to test your system before you submit a prediction set based on the qualifying dataset, we have provided a probe dataset in the file "probe.txt". This text file contains lines indicating a movie id, followed by a colon, and then customer ids, one per line for that movie id.

Spark MLlib

- MLlib is Spark's scalable machine learning library consisting of common learning algorithms and utilities.
- It includes classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives.
- Spark ML also works well with model training and production, so those models trained can easily be deployed to production.

Collaborative Filtering Movie Recommendation Systems

Collaborative Filtering is commonly used for recommender systems. It tackles the similarities between the users and items to perform recommendations. The algorithm constantly finds the relationships between the users and does the recommendations.

- Apache Spark ML implements alternating least squares (ALS) for collaborative filtering, a very popular algorithm for making recommendations.
- Implemented Alternating Least Square(ALS) with Spark. ALS is a matrix factorization technique to perform collaborative filtering. The objective function of ALS uses L1 regularization and optimizes the loss functions using Gradient Descent.
- We train the ALS model by tuning the below hyper-parameters:

  Rank , Reg Parameters & Maximum iterations.
- To measure the errors for each value of rank and store the index of error values, best rank values and best iteration value.

Implementation

Step 1 : Loading the Netflix data on Spark.

Installing and Importing all the relevant libraries.

```
In [ ]: import os
        from pyspark import SparkConf, SparkContext
        from itertools import cycle, islice
        import matplotlib.pyplot as plt
        from pyspark.sql import SparkSession
        import numpy as np
        from datetime import datetime as dt
        from time import time
        from pyspark.mllib.recommendation import ALS
        import math
        from collections import OrderedDict
        from google.colab import drive
        import sys
        %matplotlib inline
```

Implementation

Step 2 : Collecting the data and mounting the data file to Google Collaboratory.

```
In [ ]: !ls   #Listing all  files in the colab directory

combined_data_1.txt  combined_data_4.txt  probe.txt
combined_data_2.txt  data.csv             qualifying.txt
combined_data_3.txt  movie_titles.csv     README
```

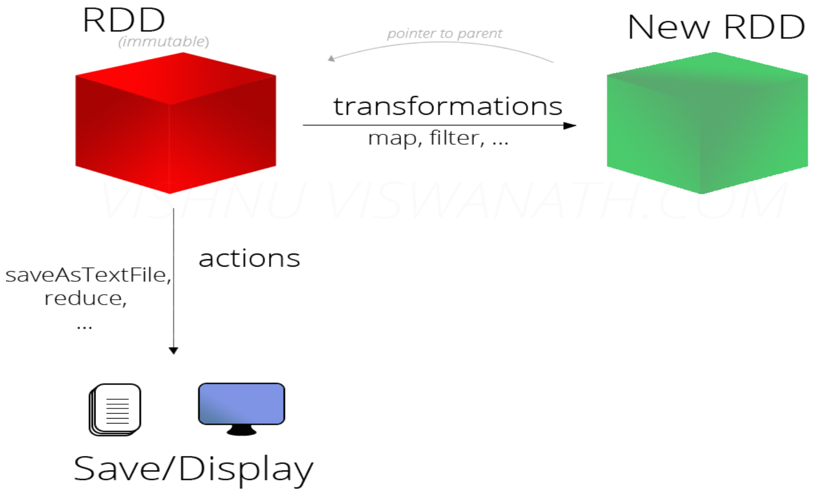
Step 3 : Starting the spark kernel :

```
In [ ]: conf = SparkConf().setMaster("local").setAppName("Project")
        sc = SparkContext(conf=conf)
```

Implementation

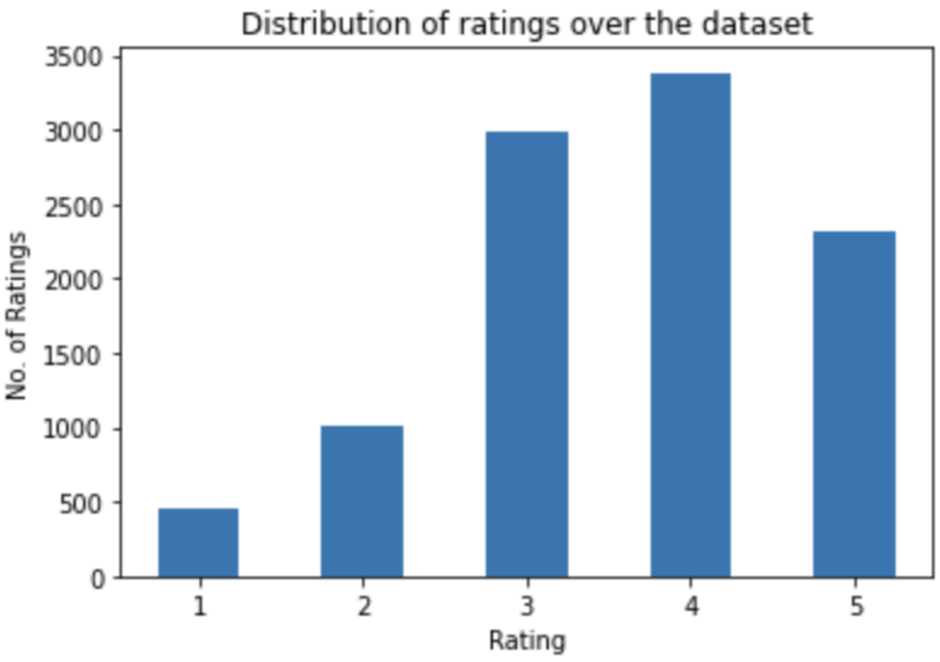
Step 4 : Merging all the data files into a single file and reading the data.

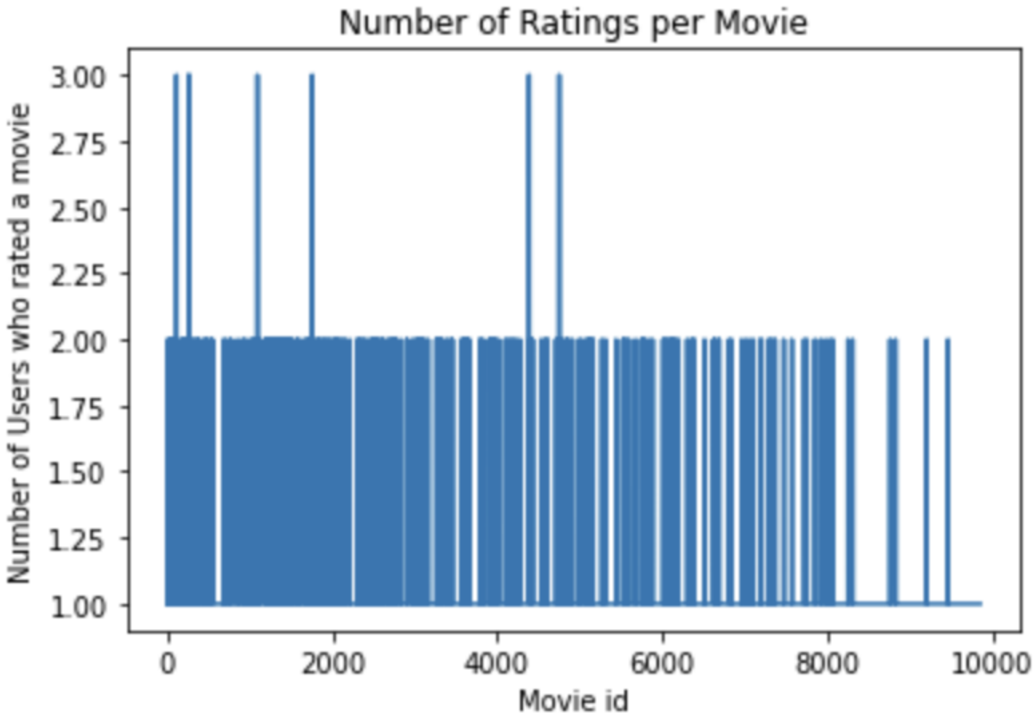
Step 5 : Creating RDDs (Resilient Distributed Dataset) is a low-level object and are highly efficient in performing distributed tasks.



Step 6 : Performing Exploratory Data analysis on the dataset to check the duplicate entries and unnecessary columns. The dataset has been cleaned.

Plotting graphs based on ratings distribution





Implementation

Step 7 : To Validate and perform test splitting for training Machine Learning model and finding best hyper parameter to find RMSE on test data.(We are using ALS Algorithm here.)

#Machine Learning model

```
In [ ]: iterations = 5     # No of iterations
regularization_parameter = 0.1     # Setting regularisation parameter
ranks = [4, 8, 12]      # Hyperparameter
errors = [0, 0, 0]      # To measure the error for each value of rank
err = 0                 # To store index of error values

min_error = float('inf')   # To store lowest error value, setting initial value as infinite
best_rank = -1             # To store best rank value
best_iteration = -1        # To store best iteration value
start = time()
for rank in ranks:
    model = ALS.train(training_RDD, rank, iterations=iterations, lambda_=regularization_parameter)   #Training the mode
    predictions = model.predictAll(validation_for_predict_RDD).map(lambda x: ((x[0], x[1]), x[2]))   #Predicting the ra
    actual_and_pred_rating = validation_RDD.map(lambda x: ((int(x[0]), int(x[1])), float(x[2]))).join(predictions)
    error = math.sqrt(actual_and_pred_rating.map(lambda x: (x[1][0] - x[1][1])**2).mean())          #Calculating mean
    errors[err] = error
    err += 1
    print('For rank {0} the RMSE is {1}'.format(rank, error))
    if error < min_error:                                                                          #To find best rank
        min_error = error
        best_rank = rank

end = time() - start
print('The best model was trained with rank',best_rank)

print("Model trained in {0} seconds".format(round(end,3)))

For rank 4 the RMSE is 0.884054551449582
For rank 8 the RMSE is 0.8713566797793022
For rank 12 the RMSE is 0.8688835515815428
The best model was trained with rank 12
```

Implementation

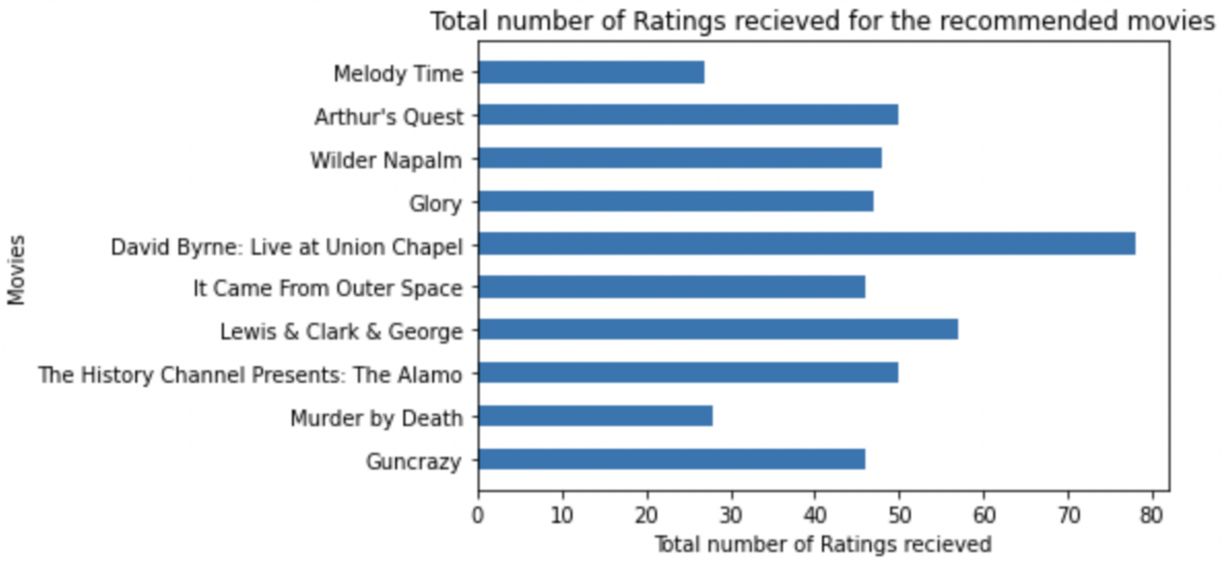
Step 8 : Now , after finding the best hyper parameter. We are creating a new user to get movie recommendations, with a new user ID .

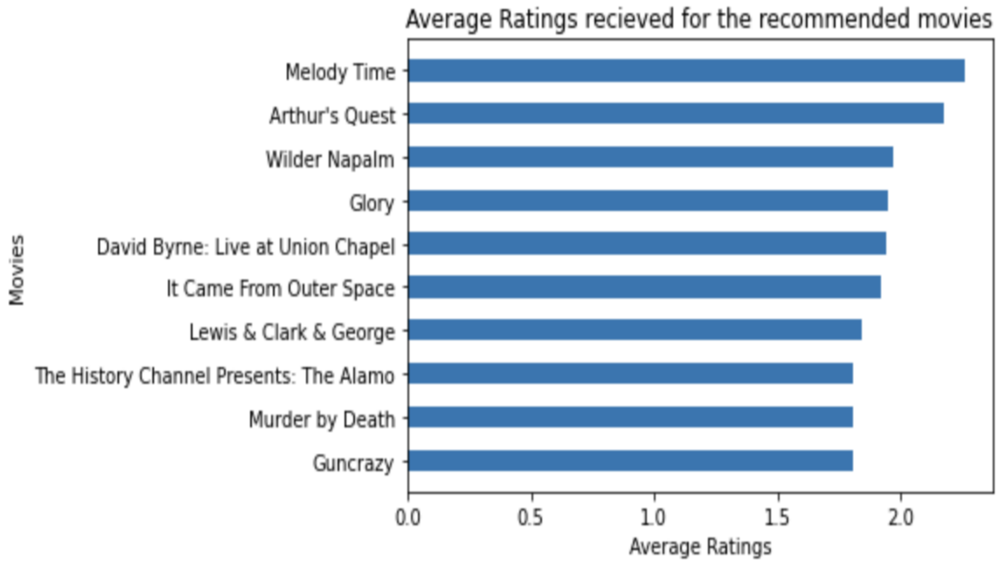
Step 9 : Training the model with the merged data to get a new model to predict ratings for movies not yet watched by the user.

Step 10 : Recommending top 10 movies to the new user.

Visualization

- Creating visualisation of recommended movies(x-axis) vs movie counts(y-axis).





Conclusion and Learnings

- Learnt to develop a recommendation system using Spark with the help of Python
- Recommended movies for the users based on the previous ratings provided by them.
- Movie recommendations have been pretty accurate for specific users.

Thank you!