

Report On

# “Movielens Recommendation System”

By

Manish Shinde, D17C, 56

Akshay Thite, D17C, 65

Abhay Tiwari, D17C, 66

Under the Guidance of

Mrs Sujata Khedkar

Name of Lab Teacher : Mrs. Sujata Khedkar



Department of Computer Engineering  
Vivekanand Education Society's Institute of Technology  
2020-21

# **1.Problem Definition and Scope of Project**

## **1.1 Introduction**

A recommendation system is a type of information filtering system which attempts to predict the preferences of a user, and make suggestions based on these preferences. There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. The content of such platforms varies from movies, music, books and videos, to friends and stories on social media platforms, to products on e-commerce websites, to people on professional and dating websites, to search results returned on Google.

Two main approaches are widely used for recommender systems. One is content-based filtering, where we try to profile the users interests using information collected, and recommend items based on that profile. The other is collaborative filtering, where we try to group similar users together and use information about the group to make recommendations to the use

## **1.2 Problem Definition and Scope of Project**

Given a set of users with their previous ratings for a set of movies, can we predict the rating they will assign to a movie they have not previously rated? The system provides the recommendation of the movies which they have not watched on the basis of the movies that they have rated and also the rating of other users.

## **1.3 Users of the system**

Any user having the application to watch movies installed in their device.

## **1.4 Dataset**

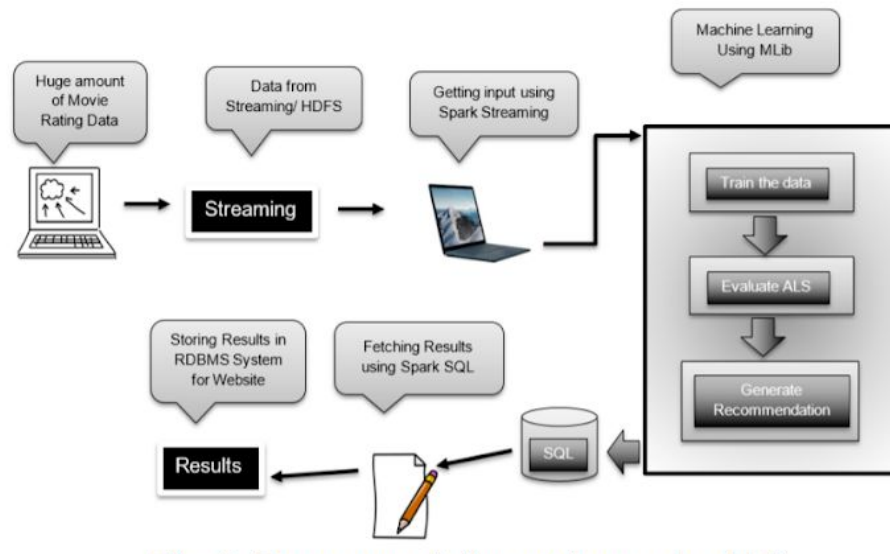
Dataset consists of the movies which users have watched and its rating. The second dataset used consists of all the data related to the movies

# **2. Literature Review :**

Various collaborative filtering algorithms can be used to find the estimated rating of a movie for users which they have not watched. In this system the ALS collaborative filtering model is used to find the estimated rating of the movie. The movies with the highest estimated rating will be provided to the user as recommendation. The parameters of the model need to be decided appropriately.

### 3. Conceptual System Design

#### 3.1 Conceptual System Design- CSD Diagram with explanation of each module.



- Get the data from the dataset in the form of RDD and convert this data to Dataframe
- Design a ALS model with suitable parameters
- Train the ALS model with the data
- Provide the output of top 20 movies with the highest estimated rating.

#### 3.2 Methodology:

##### 3.2.1 Data Gathering / Loading

Data is Loaded from a text file in the form of RDD using the SparkSession.

##### 3.2.2 Data Preprocessing ,Descriptive Analysis

Data loaded in the form of RDD is converted into the Dataframe so that various relational algebra operations can be performed on it. This is because in RDD the data is stored in the form of records and in the Dataframe it is stored in the form of columns.

##### 3.2.3 Filtering

Movies which are rated by more than 100 users are considered to find the estimated rating of the movies for a particular user which he has not watched.

### 3.2.4 Classification/ clustering etc

ALS model (Alternating least square model) is used for recommendation of movies to the users.  
ALS model :

Alternating Least Square (ALS) is a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is implemented in Apache Spark ML and built for large-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets.

Some high-level ideas behind ALS are:

- Its objective function is slightly different than Funk SVD: ALS uses **L2 regularization** while Funk uses **L1 regularization**
- Its training routine is different: ALS minimizes **two loss functions alternatively**; It first holds user matrix fixed and runs gradient descent with item matrix; then it holds item matrix fixed and runs gradient descent with user matrix
- Its scalability: ALS runs its gradient descent in **parallel** across multiple partitions of the underlying training data from a cluster of machines

Most important hyper-params in Alternating Least Square (ALS):

- maxIter: the maximum number of iterations to run (defaults to 10)
- rank: the number of latent factors in the model (defaults to 10)
- regParam: the regularization parameter in ALS (defaults to 1.0)

1. A new user inputs his/her favorite movies, then system create new user-movie interaction samples for the model
2. System retrains ALS model on data with the new inputs
3. System creates movie data for inference (in my case, I sample all movies from the data)
4. System make rating predictions on all movies for that user
5. System outputs top N movie recommendations for that user based on the ranking of movie rating predictions

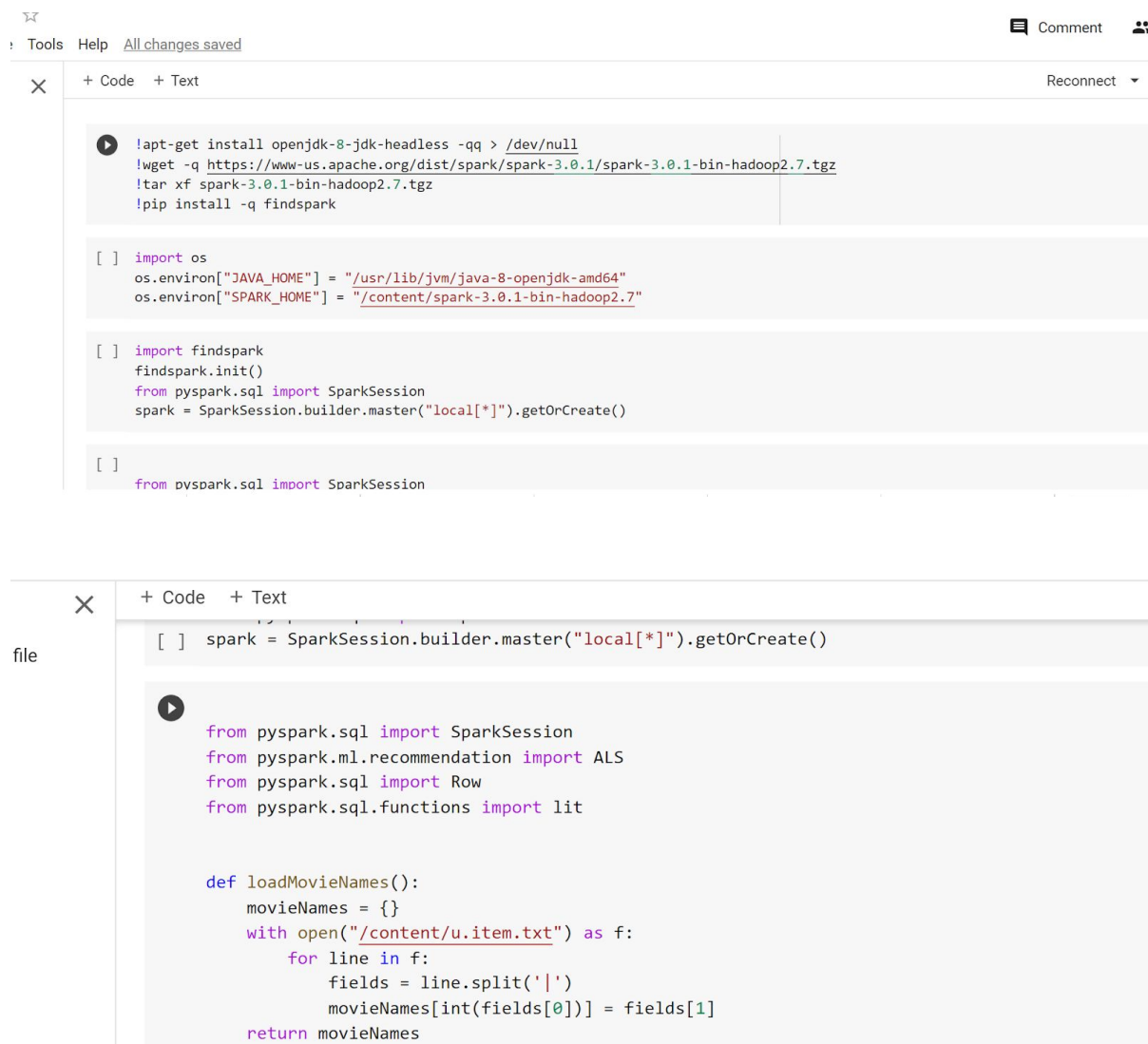
### 3.2.5 Visualizations

The approximate ratings of the movies which a user has not watched is found out and among them the highest rated 20 movies are recommended to that user as an output.

## 4. Technology Used

- Python
- Pyspark - SparkSession
- Alternating least square collaborative filtering model

## 5. Implementation



```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist/spark/spark-3.0.1/spark-3.0.1-bin-hadoop2.7.tgz
!tar xf spark-3.0.1-bin-hadoop2.7.tgz
!pip install -q findspark

[ ] import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.1-bin-hadoop2.7"

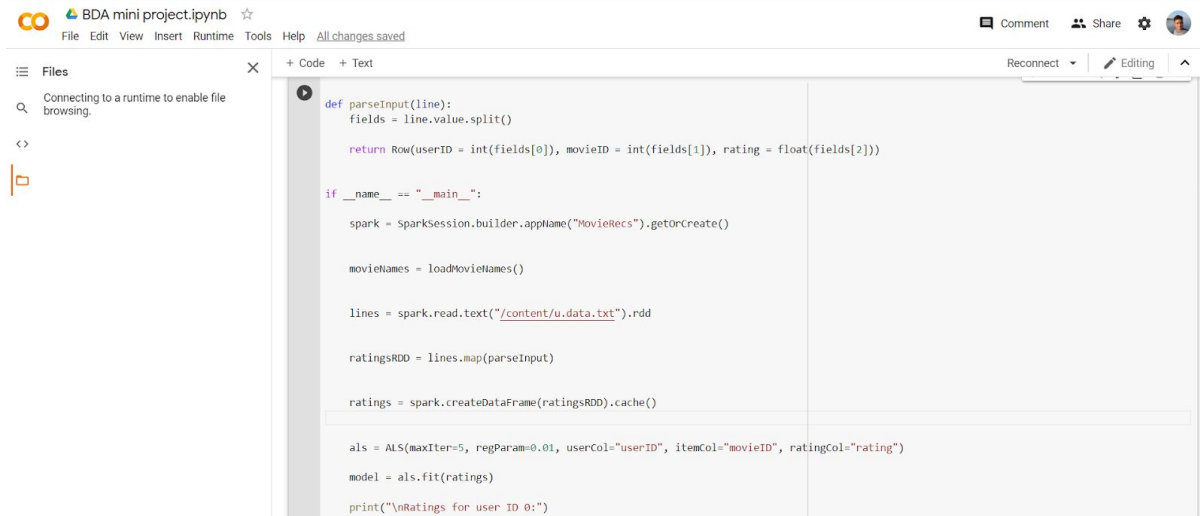
[ ] import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()

[ ]
from pyspark.sql import SparkSession

[ ] spark = SparkSession.builder.master("local[*]").getOrCreate()

from pyspark.sql import SparkSession
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row
from pyspark.sql.functions import lit

def loadMovieNames():
    movieNames = {}
    with open("/content/u.item.txt") as f:
        for line in f:
            fields = line.split('|')
            movieNames[int(fields[0])] = fields[1]
    return movieNames
```



BDA mini project.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Files

Connecting to a runtime to enable file browsing.

```
def parseInput(line):
    fields = line.value.split()
    return Row(userID = int(fields[0]), movieID = int(fields[1]), rating = float(fields[2]))

if __name__ == "__main__":
    spark = SparkSession.builder.appName("MovieRecs").getOrCreate()

    movieNames = loadMovieNames()

    lines = spark.read.text("/content/u.data.txt").rdd

    ratingsRDD = lines.map(parseInput)

    ratings = spark.createDataFrame(ratingsRDD).cache()

    als = ALS(maxIter=5, regParam=0.01, userCol="userID", itemCol="movieID", ratingCol="rating")
    model = als.fit(ratings)
    print("\nRatings for user ID 0:")
```



Files

Connecting to a runtime to enable file browsing.

```
userRatings = ratings.filter("userID = 0")
for rating in userRatings.collect():
    print(movieNames[rating['movieID']], rating['rating'])

Ratings for user ID 0:
Star wars (1977) 5.0
Empire Strikes Back, The (1980) 5.0
Gone with the Wind (1939) 1.0

print("\nTop 20 recommendations:")

ratingCounts = ratings.groupBy("movieID").count().filter("count > 100")
popularMovies = ratingCounts.select("movieID").withColumn('userID', lit(0))
recommendations = model.transform(popularMovies)

topRecommendations = recommendations.sort(recommendations.prediction.desc()).take(20)
for recommendation in topRecommendations:
    print(movieNames[recommendation['movieID']], recommendation['prediction'])
```

## 6.Results and conclusion

The screenshot shows a Jupyter Notebook titled "BDA mini project.ipynb". The interface includes a top menu bar with options like File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu, there's a sidebar on the left with a "Files" tab and a search bar. The main area is divided into two sections: a code editor and an output area. The code editor contains the text `spark.stop()`. The output area displays the result of the code execution, which is a list of 20 movie recommendations, each followed by a rating. The movies are listed in descending order of rating.

```
Top 20 recommendations:
Blade Runner (1982) 5.547658443450928
Trainspotting (1996) 5.543519973754883
Fish Called Wanda, A (1988) 5.337456703186035
Terminator, The (1984) 5.324463367462158
Alien (1979) 5.25009298324585
Blues Brothers, The (1980) 5.146204948425293
Star Wars (1977) 5.0042033195495605
Stand by Me (1986) 4.985867500305176
Empire Strikes Back, The (1980) 4.9666523933410645
Seven (Se7en) (1995) 4.9551167488098145
Raising Arizona (1987) 4.938271999359131
Usual Suspects, The (1995) 4.879996299743652
2001: A Space Odyssey (1968) 4.843062877655029
Star Trek: The Wrath of Khan (1982) 4.805322647094727
Pulp Fiction (1994) 4.801003932952881
Terminator 2: Judgment Day (1991) 4.755334377288818
Jaws (1975) 4.732297897338867
Princess Bride, The (1987) 4.629415988922119
Back to the Future (1985) 4.628874778747559
GoodFellas (1990) 4.5846357345581055
```

Movies Recommendation System was executed successfully using the ALS model and it provided the recommendations of top 20 movies to the users depending on their previous choices i.e. the ratings given to the movies.