

Final Project Report on  
COLLABORATIVE FILTERING USING NETFLIX DATA

Prepared By:  
Bijaya Dhital

Submitted to:  
Vahid Behzadan (Assistant Professor)



University of  
New Haven

Tagliatela College of Engineering  
Department of Data Science

Date: 15 December 2021

## Introduction

Netflix is a subscription based streaming service that allows its members to watch TV shows and movies without commercials on an internet-connected devices. Netflix presents a large variety of TV shows, movies, documentaries across a wide range of genre and languages. It suggests users to watch different TV shows and movies. Netflix use their recommendations system that is based on a machine-learning algorithm that considers your past choices in movies, the types of genres you like, and what moves were watched by users that had similar tastes like yours. Recommender systems at Netflix span various algorithmic approaches like reinforcement learning, neural networks, causal modelling, probabilistic graphical models, matrix factorization, ensembles, bandits.

Netflix's recommendation systems have been developed by hundreds of engineers that analyze the habits of millions of users based on multiple factors. Whenever a user accesses Netflix services, the recommendations system estimates the probability of a user watching a particular title based on different factors such as viewer interaction with Netflix, time duration, the device user used, the time of the day a viewer watches and so on. A recommendation system is a subclass of Information filtering Systems that seeks to predict the rating or the preference a user might give to an item. In simple words, it is an algorithm that suggests relevant items to users for example in the case of Netflix which movie to watch.

In addition, Netflix collaborative-based filtering recommends the new watch list to users based on the interest and preference of other similar users. Recommender systems or recommendation systems forms a class of data and information filtering system that aids in predicting the taste and preferences of a user. Collaborative filtering tackles the similarities between the users and items to perform recommendations. Meaning that the algorithm constantly finds the relationships between the users and in-turns does the recommendations.

## Project Goal

In this project, I will be predicting 100,000 movie ratings for users in a subset of the original NETFLIX data issued for the NETFLIX prize. This challenge aimed at substantially improving the accuracy of predictions about how much someone is going to enjoy a movie based on their movie preferences.

Goal:

Analyze the NETFLIX data using SPARK, and based on the outcome of this analysis, develop a feasible and efficient implementation of the collaborative filtering algorithm in SPARK.

## Motivation

Collaborative filtering tackles the similarities between the users and items to perform recommendations. Meaning that the algorithm constantly finds the relationships between the users

and in-turns does the recommendations. You can use this technique to build recommenders that give suggestions to a user based on the likes and dislikes of similar users. The motivation for collaborative filtering comes from the idea that people often get the best recommendations from someone with tastes like themselves. Collaborative filtering encompasses techniques for matching people with similar interests and making recommendations on this basis.

In this project, I want to improve the accuracy of the predictions to know how much a user is going to enjoy a movie based on their movie preferences. The reason behind using recommendation systems is that people have too many options to choose from and might not be able to go through them all. Netflix, for example, has a huge collection of movies. Although the amount of available information increased, a new problem arose as people had a hard time selecting the items they want to see. Recommender system helps to uncover this problem. the application of this system improves customer satisfaction and increases sales for the business. In

## **Approach**

The key idea behind Collaborative Filtering is that similar users share similar interest, people with similar interest tends to like similar items. Collaborative filtering is commonly used for recommender systems. Hence those items are recommended to similar set of users.

Content-based filtering made their predictions based on the genes of your watch history Movies streaming services use both simultaneously to get the best results.

In this Project, I will be using following approach

- Analyze the input Netflix data using SPARK
- Implement Collaborative filtering algorithm.
- Implement the approach in a Jupyter Notebook using SPARK on AWS EMR cluster.
- Run the implementation to predict the ratings for all user item pairs.
- Compute the Mean Absolute error and the root mean squared error for the prediction.

## **Recommender System**

A recommender system is a type of information filtering system. By drawing from huge data sets, the system's algorithm can pinpoint accurate user preferences. Once you know what your users like, you can recommend them new, relevant content. Netflix, YouTube, Tinder, and Amazon are all examples of recommender systems in use. The systems entice users with relevant suggestions based on the choices they make.

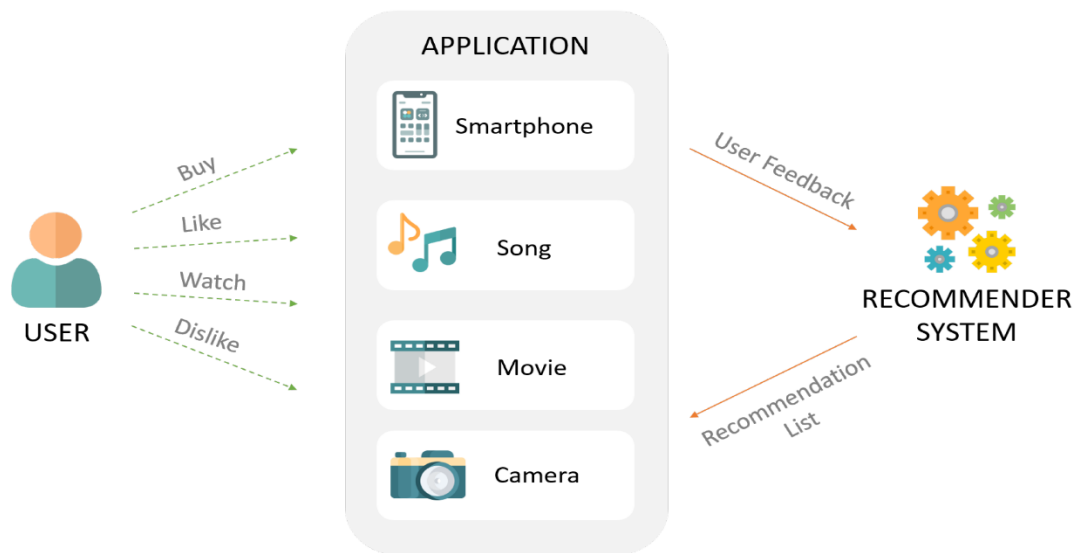
Advantage of recommender system

- Increase in sales thanks to personalized offers.
- Enhanced customer experience.
- More time spent on the platform.

- Customer retention thanks to users feeling understood

Using machine learning, recommender systems provide you with suggestions in a few ways:

- Collaborative Filtering
- Content-based Filtering
- Hybrid (Combination of Both)



## Collaborative Filtering

A collaborative filtering recommender system analyzes similarities between users and/or item interactions. Once the system identifies similarities, it serves users recommendations. In general, users see items that similar users liked. Collaborative filtering is a technique used by recommender systems. Collaborative filtering filters information by using the interactions and data collected by the system from other users. It's based on the idea that people who agreed in their evaluation of certain items are likely to agree again in the future. The concept is simple: when we want to find a new movie to watch we'll often ask our friends for recommendations. Naturally, we have greater trust in the recommendations from friends who share tastes like our own.

Most collaborative filtering systems apply the so-called similarity index-based technique. In the neighborhood-based approach, several users are selected based on their similarity to the active user. Inference for the active user is made by calculating a weighted average of the ratings of the selected users.

Collaborative-filtering systems focus on the relationship between users and items. The similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items.

There are different types of collaborative filtering systems including:

- Item-item Collaborative Filtering
- User-user Collaborative Filtering

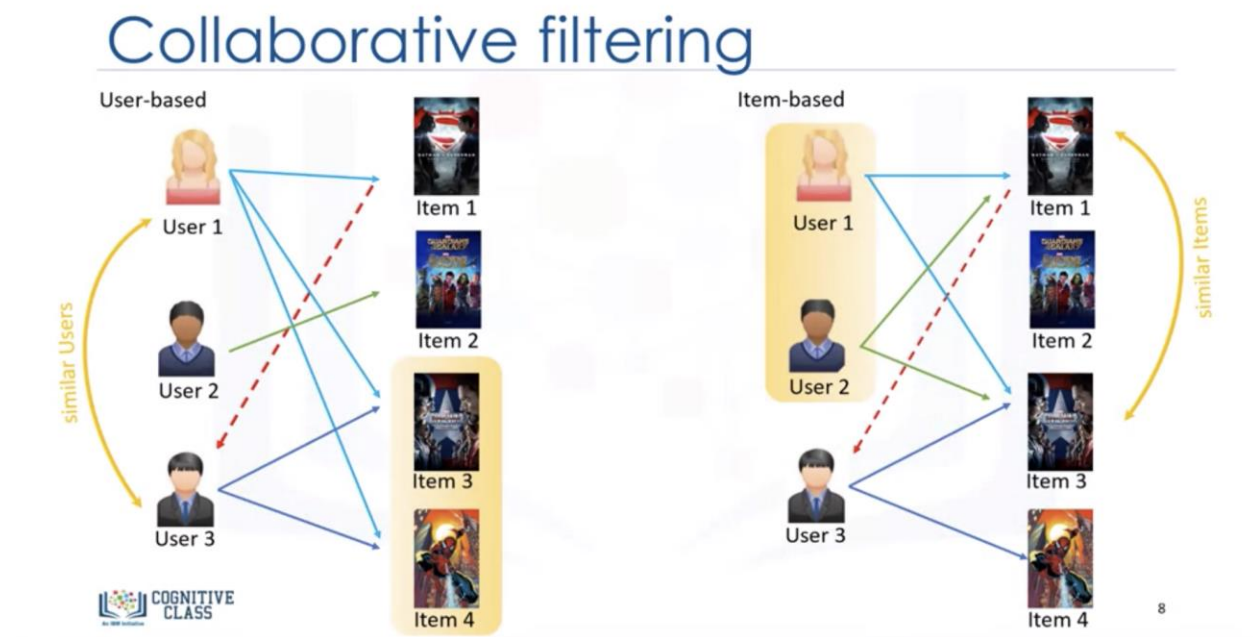
### Item-item Collaborative Filtering

An item-item filtering algorithm analyzes product associations taken from user ratings. Users then see recommendations based on how they rate individual products. For example, you rate a movie as a 10/10. Now, you will see the top-rated movies with similar attributes.

### User-user Collaborative Filtering

The other kind of collaborative filtering takes the similarity of user tastes into consideration. So, user-user collaborative filtering doesn't serve you items with the best ratings. Instead, you join a cluster of other people with similar tastes, and you see content based on historic choices.

Let's say you use Netflix for the first time. You play a Sci-Fi Movie. The system clusters you with other users who also like Sci-Fi. Then the Netflix recommendation system shows you other movies chosen by users in your cluster. The more choices you make, the more relevant the results.



## **Back to Project (Collaborative Filtering using PySpark)**

Steps used for project using SPARK.

### **Amazon EMR**

The project is done in Jupyter Notebook created from Amazon EMR Cluster.

1. Login into Amazon web services, create a cluster from EMR.
2. Click on Create cluster and fill in the following


#### **Configuration details**

---

**Release label:** emr-5.34.0

**Hadoop distribution:** Amazon 2.10.1

**Applications:** Hive 2.3.8, Hue 4.9.0, Mahout 0.13.0, Pig 0.17.0, Tez 0.9.2

**Log URI:** s3://aws-logs-149376158574-us-east-1/elasticmapreduce/ 

**EMRFS consistent view:** Disabled

**Custom AMI ID:**

After Creating the EMR Spark cluster, follow these steps:

1. Open the Security Groups for the Master node, add 2 inbound rules for SSH and Custom TCP Port 8888.
2. SSH to the master node.
3. Run `sudo pip3 install pyyaml ipython jupyter ipyparallel pandas boto -U`
4. Append the following two lines to `.bashrc` file:
  - `export PYSPARK_DRIVER_PYTHON=/usr/local/bin/jupyter`
  - `export PYSPARK_DRIVER_PYTHON_OPTS="notebook --no-browser --ip=0.0.0.0 --port=8888"`
  - Then run `source ~/.bashrc` followed by `pyspark`.
  - connections on port 8888 should be allowed (in Security Groups) and browser to the master node: 8888 and look for token. (i.e., your URL should look something like `http://ec2-52-329-72-376.compute-1.amazonaws.com:8888/?token=7g625954d2364be169gy852649a25648ad8bd3s46`)

## Netflix Dataset

The provided Dataset contains information about the Netflix movies and ratings given by users.

**Descriptions.txt:** describe the provided dataset

**Movie\_title.txt:** it has rows with MovieID, YearOfRelease and Title

**TrainingRatings.txt:** it contains lines having the format MovieID, UserID and Rating. The dataset has 3.25 million ratings.

**TestingRatings.txt:** it contains lines having the format MovieID, UserID and Rating. It has 100,000 ratings.

### Steps for uploading and analyzing the data

- Uploading the data from S3 bucket

```
: import os

# Change to the location of data files
dbfs_dir = 's3://bijayadhitalidsci6007netflix/Netflix/Netflix/'
movie_titles_filename = dbfs_dir + '/movie_titles.txt'
TrainingRatings_filename = dbfs_dir + '/TrainingRatings.txt'
TestingRatings_filename = dbfs_dir + '/TestingRatings.txt'
```

- Specifying the DataFrame Schema

```
from pyspark.sql.types import *

movie_titles_df_schema = StructType(
    [StructField('ID', IntegerType()),
     StructField('movie_year', IntegerType()),
     StructField('movie_title', StringType())]
)
TrainingRatings_df_schema = StructType(
    [StructField('movie_Id', IntegerType()),
     StructField('user_Id', IntegerType()),
     StructField('movie_ratings', FloatType())]
)
TestingRatings_df_schema = StructType(
    [StructField('movie_Id', IntegerType()),
     StructField('user_Id', IntegerType()),
     StructField('movie_ratings', FloatType())]
)
```

- Cache the data

Rather than reading the data again and again from S3 bucket. Lets cache both movie DataFrame and rating DataFrame in memory.

There are 17770 movietitle, 3255352 trainingRatings and 100478 testingRatings in the datasets

Movie\_titles:

ID	movie_year	movie_title
1	2003	Dinosaur Planet
2	2004	Isle of Man TT 20...
3	1997	Character

only showing top 3 rows

TrainingRatings:

movie_Id	user_Id	movie_ratings
8	1744889	1.0
8	1395430	2.0
8	1205593	4.0

only showing top 3 rows

TestingRatings:

movie_Id	user_Id	movie_ratings
8	573364	1.0
8	2149668	3.0
8	1089184	3.0

only showing top 3 rows

- Average estimated overlap

2675	2149668	3.0
2755	2149668	3.0
2913	2149668	5.0
2955	2149668	5.0
3151	2149668	4.0
3253	2149668	4.0
3274	2149668	2.0
3290	2149668	5.0
3355	2149668	5.0
3538	2149668	4.0
4847	2149668	3.0
4849	2149668	3.0

only showing top 20 rows

movie_Id	user_Id	movie_ratings
992	306466	3.0
992	765331	4.0
992	41412	3.0
992	1331887	3.0
992	1276913	3.0
992	887273	4.0
992	1663216	3.0
992	1778851	5.0
992	2630287	5.0
992	530441	5.0

only showing top 10 rows

movie\_ids\_with\_avg\_ratings\_df:

user_Id	count	average
1896167	24	3.375
128389	42	3.0238095238095237
1552084	31	3.7419354838709675

only showing top 3 rows

8	1765381	4.0
8	433803	3.0
8	1148143	2.0
8	1174811	5.0
8	1684516	3.0
8	754781	4.0
8	567025	4.0
8	1623132	4.0
8	1567095	3.0
8	1666394	5.0

only showing top 20 rows

movie_Id	user_Id	movie_ratings
8	1744889	1.0
8	1395430	2.0
8	1205593	4.0
8	1488844	4.0
8	1447354	1.0
8	306466	4.0
8	1331154	4.0
8	1818178	3.0
8	991725	4.0
8	1987434	4.0

only showing top 10 rows

movie\_ids\_with\_avg\_ratings\_df:

movie_Id	count	average
4190	9	3.4444444444444446
3220	155	2.7870967741935484
3149	25	2.96

only showing top 3 rows



For accurately estimating the similarity, let's pick a user from the test set and extract the item the user rated in the training set, compare it with other users on the training set. Let's look for comparison statistics with the overall average of items of that user and perform the same for other users in the test set to get an idea of estimated average overlap. Now continue the same for items instead of users to get the estimated average overlap of users for items.

- Splitting the dataset into training and the validation

```
# Splitting the dataset

(split_60_df, split_a_20_df) = TrainingRatings_df.randomSplit([7.5, 2.5])

# Dataset Cache for performance
train_df = split_60_df.cache()
validation_df = split_a_20_df.cache()

print('Training: {0}, validation: {1}\n'.format(
    train_df.count(), validation_df.count()))
)
train_df.show(3)
validation_df.show(3)

Training: 2442110, validation: 813242

+-----+-----+-----+
|movie_Id|user_Id|movie_ratings|
+-----+-----+-----+
|      8|   1333|          3.0|
|      8|   3321|          1.0|
|      8|   3363|          2.0|
+-----+-----+-----+
only showing top 3 rows

+-----+-----+-----+
|movie_Id|user_Id|movie_ratings|
+-----+-----+-----+
|      8|      7|          5.0|
|      8|   5980|          3.0|
|      8|  13197|          3.0|
+-----+-----+-----+
only showing top 3 rows
```

- Model based ALS for Mean square error (MSE) and Root mean square error (RMSE).

```
als = ALS(maxIter=10, regParam=0.5, userCol="user_Id",
          itemCol = "movie_Id", ratingCol = "movie_ratings", coldStartStrategy = "drop")

model = als.fit(train_df)
#Generating Predictions
prediction = model.transform(validation_df)
evaluate1 = RegressionEvaluator(metricName="mse", labelCol="movie_ratings",
                               predictionCol="prediction")

evaluate2 = RegressionEvaluator(metricName="rmse", labelCol="movie_ratings",
                               predictionCol="prediction")

rmse = evaluate1.evaluate(prediction)
mse = evaluate2.evaluate(prediction)

print('The model had a MAE on the test set of {0}'.format(test_MAE))
print('The model had a RMSE on the test set of {0}'.format(test_RMSE))

Mean squared error = 1.0335312185835566
Root-mean-square error = 1.068186779786811
```

- Movie ratings

```
: from pyspark.sql import Row
my_user_id = 1

myRatedMovies = [
    (my_user_id, 12293, 5),
    (my_user_id, 10947, 5),
    (my_user_id, 2290, 5),
    (my_user_id, 14648, 3),
    (my_user_id, 14185, 4),
    (my_user_id, 11812, 5),
    (my_user_id, 11088, 3),
    (my_user_id, 25468, 5),
    (my_user_id, 5326, 4),
    (my_user_id, 3928, 2)

my_ratings_df = sqlContext.createDataFrame(myRatedMovies, ['user_Id', 'movie_Id', 'movie_ratings'])
print('My movie ratings:')
display(my_ratings_df.limit(10))
my_ratings_df.limit(10).show()
```

My movie ratings:

DataFrame[user\_Id: bigint, movie\_Id: bigint, movie\_ratings: bigint]

user_Id	movie_Id	movie_ratings
1	12293	5
1	2290	5
1	11812	5
1	25468	5
1	10947	5
1	14648	3
1	14185	4
1	11088	3
1	5326	4
1	3928	2

## References

<https://ir.netflix.net/ir-overview/profile/default.aspx>

<https://help.netflix.com/en/node/412>

<https://towardsdatascience.com/tensorflow-for-recommendation-model-part-1-19f6b6dc207d>