DISTRIBUTED AND SCALABLE DATA ENGINEERING
FINAL PROJECT REPORT
On

"COLLABORATIVE FILTERING USING THE NETFLIX DATA"

Submitted By
HEMENDRA JAMPALA (00695281)

Under The Guidance of
VAHID BEHZADAN (Assistant Professor)

UNIVERSITY OF NEW HAVEN
Tagliatela College of Engineering
Department of Data Science
300 Boston Post Rd, West Haven, CT 06516

Problem Statement:-

Movie Recommendation - Predictions about how much someone is going to enjoy a movie based on their movie preferences.

Approach:-

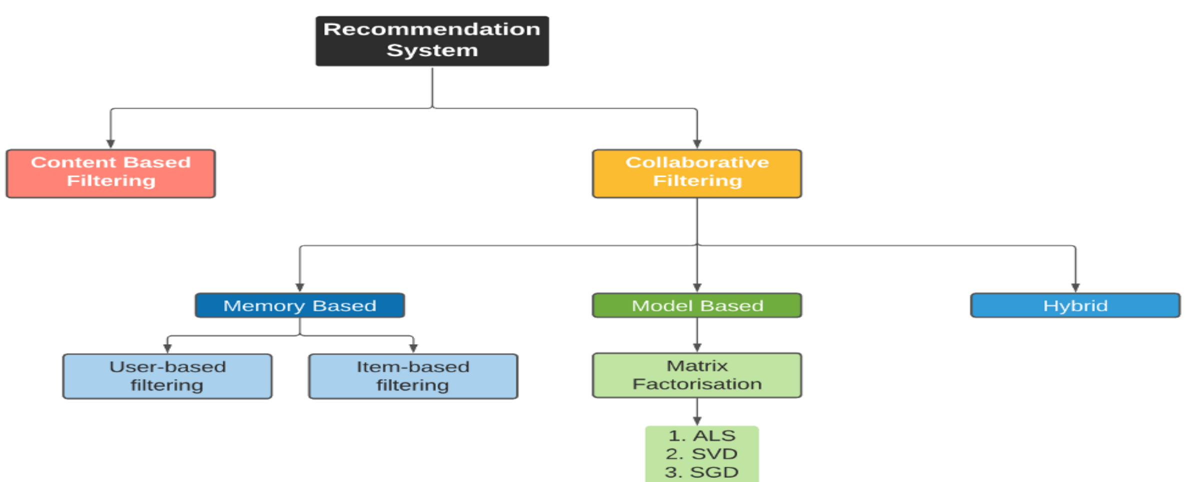
- Analyze the NETFLIX data using SPARK and, based on the outcomes of this analysis,
- Developing a feasible and efficient implementation of the **collaborative filtering algorithm in SPARK**.
- Executing program on Amazon EMR to get the rating predictions and evaluate those ratings by comparing them to the provided true ratings.

RECOMMENDER SYSTEMS:-

A Recommender System makes prediction based on user's historical behaviors like view, search or purchase histories.

For example, Amazon can recommend new shopping items to buy, Netflix can recommend new movies to watch, and Google can recommend news that a user might be interested in.

There are two ways to gather user preference data to recommend items, the first method is to ask for explicit ratings from a user, typically on a concrete rating scale (such as rating a movie from one to five stars) making it easier to make extrapolations from data to predict future ratings. However, the drawback with explicit data is that it puts the responsibility of data collection on the user, who may not want to take time to enter ratings. On the other hand, implicit data is easy to collect in large quantities without any extra effort on the part of the user. Unfortunately, it is much more difficult to work with.

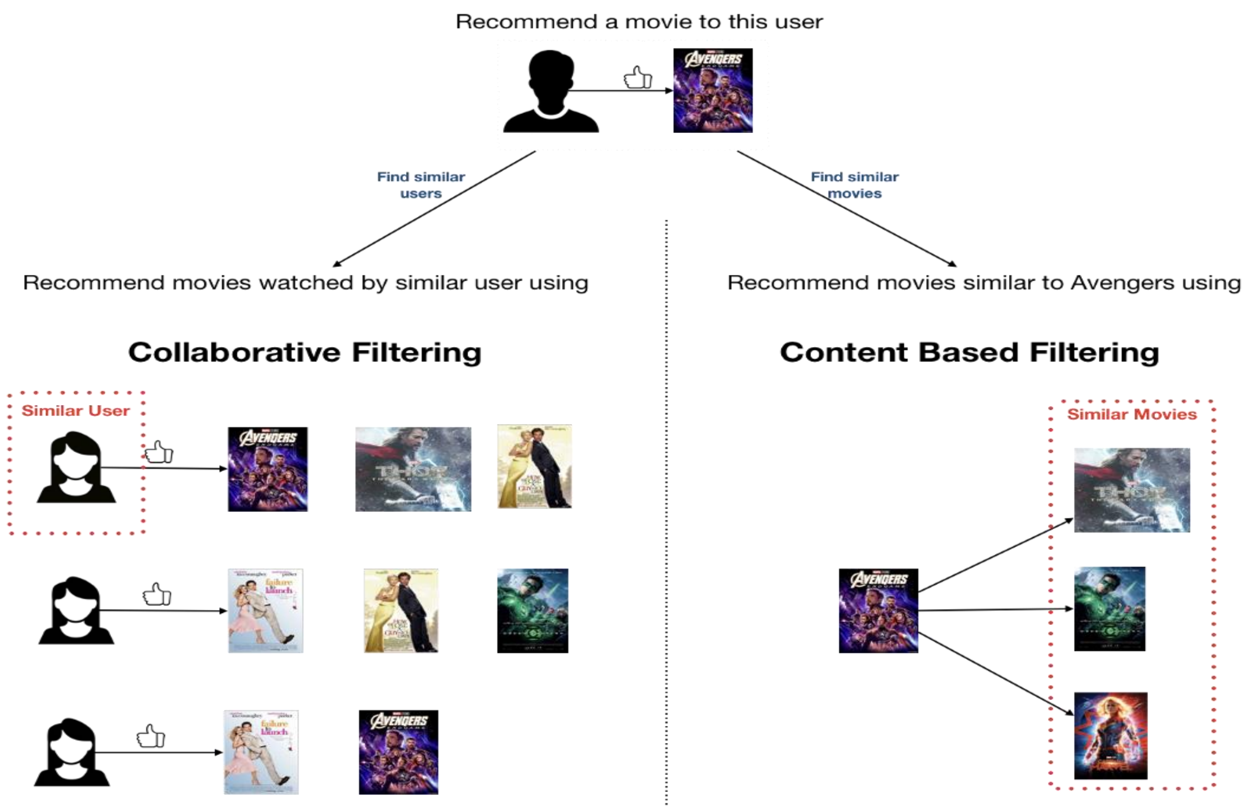


Two common approaches of recommender system are,

- Content Based Filtering (CBF)

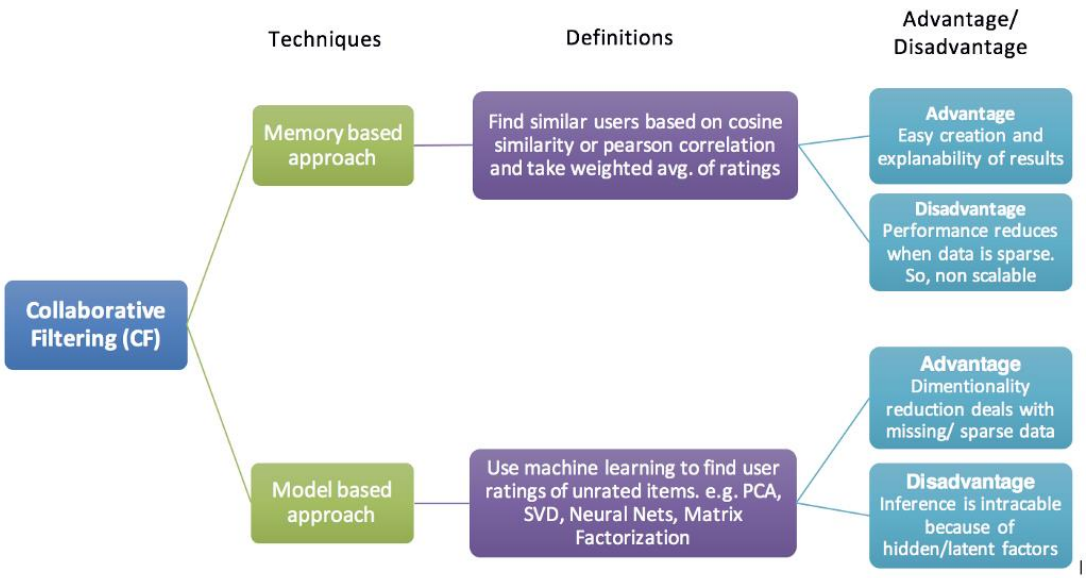
  The main idea behind CBF is to recommend items similar to the items previously liked by the user.

- Collaborative Filtering (CF)
  Collaborative filtering aggregates the past behavior of all users.
  It recommends items to a user based on the items liked by another set of users whose likes (and dislikes) are similar to the user under consideration.



COLLABORATIVE FILTERING:-

Collaborative filtering is commonly used for recommender systems.



The memory-based approach:-

User-based Filtering and Item-based Filtering are the two ways to approach memory-based collaborative filtering.

User-based Filtering: To recommend items to user u1 in the user-user based neighborhood approach first a set of users whose likes and dislikes similar to the useru1 is found using a similarity metrics which captures the intuition that sim(u1, u2) >sim(u1, u3) where user u1 and u2 are similar and user u1 and u3 are dissimilar. Similar user is called the neighbourhood of user u1.

Item-based Filtering: To recommend items to user u1 in the item-item based neighborhood approach the similarity between items liked by the user and other items are calculated.

user-user based CF

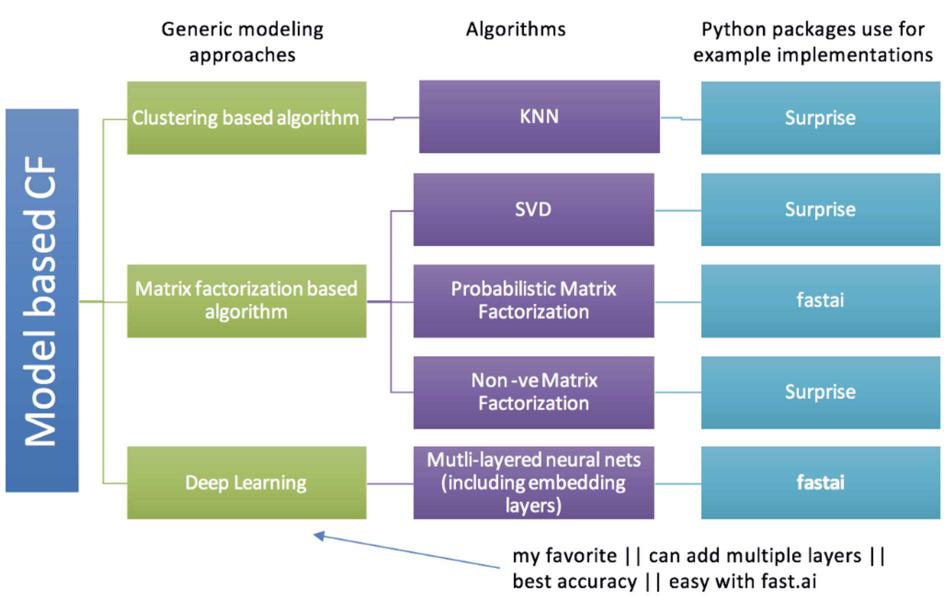
It recommends items to a user based on the items liked by another set of users whose likes (and dislikes) are similar to the user under consideration.

item-item based CF

Recommend an item to a user, the similarity between items liked by the user and other items are calculated.

The model-based approach:-

Latent factor model based collaborative filtering learns the (latent) user and item profiles (both of dimension K) through matrix factorization by minimizing the RMSE (Root Mean Square Error) between the available ratings y and their predicted values yˆ. Here each item i is associated with a latent (feature) vector xi, each user u is associated with a latent (profile) vector theta(u), and the rating yˆ(ui) is expressed as Image for post.



History of Spark:-

Building large-scale machine learning models has never been simple. Our first data processing jobs were built on Hadoop MapReduce using the Java API. Hadoop MapReduce's execution model was simply not a great fit for the highly-iterative machine learning algorithms that we were trying to implement.

Apache Spark was originally developed at UC Berkeley explicitly for the use case of large-scale machine learning. Early in Spark's development, the team realized that Spark could be a general data processing platform, so they carved out different pieces of functionality into separate subprojects, all relying on common facilities provided by Spark Core. The machine learning capabilities became a library called MLlib, and there are libraries for streaming, SQL, and graph processing as well.

Compared to Hadoop, Spark is much better suited for building large-scale machine learning problems. Beyond better performance, the developer experience when using Spark is much better than when developing against Hadoop. Spark's Scala, Java, and Python APIs are famously well-conceived and provide a functional programming data model that is declarative and high-level. Another significant advantage of using Spark as a platform has been getting access to scalable library implementations of common **machine learning algorithms via MLlib**.

COLLABORATIVE FILTERING USING PYSPARK

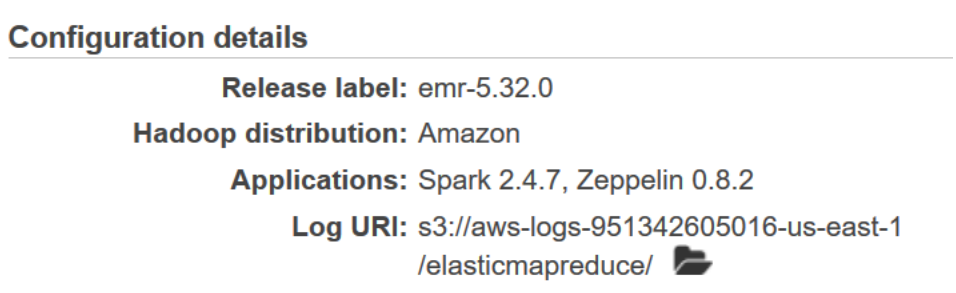
Setup Environment to run Python Scripts

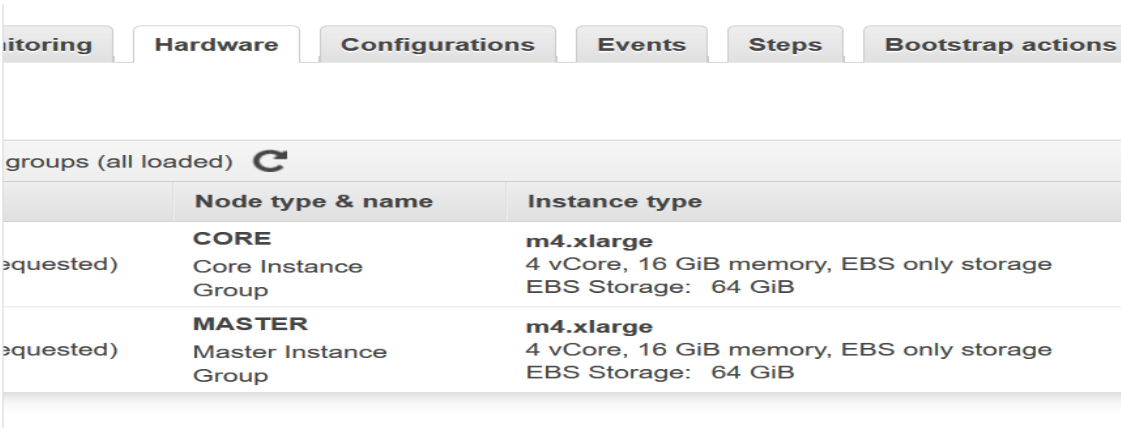
The project is done in Jupyter notebook created on AWS EMR Cluster.

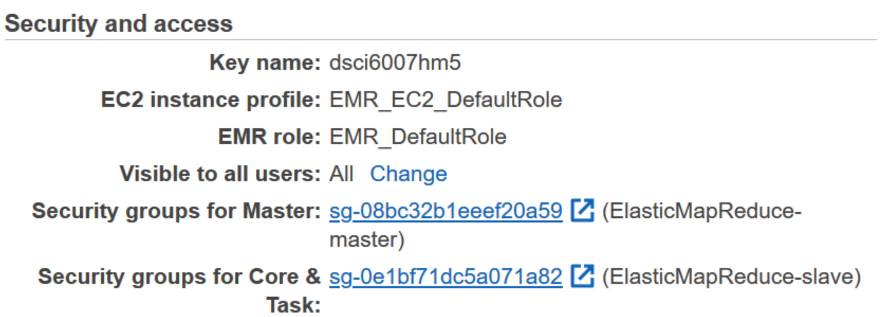
Steps to create EMR cluster:-

With the help of the AWS educate account provided as a part of the coursework we had the opportunity to work with AWS.

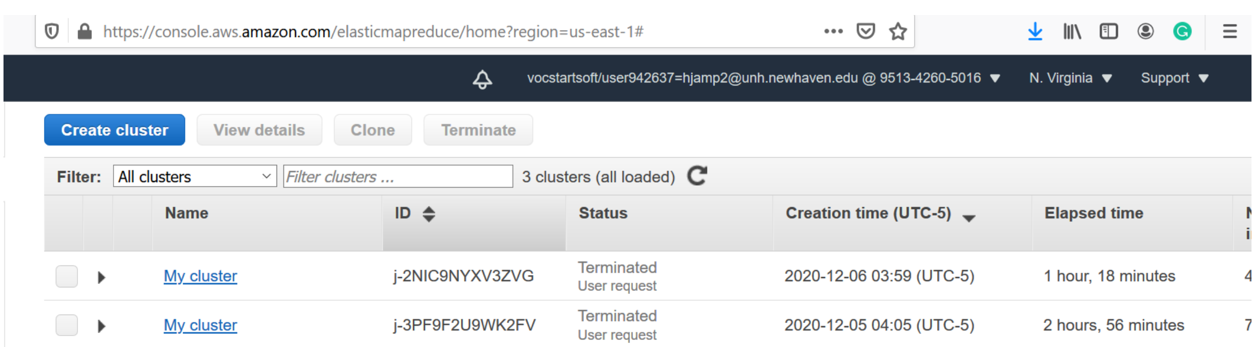
I have Created EMR cluster with the below specified applications.







Under Security and access, choose the EC2 key pair that you created in Create an Amazon EC2 Key Pair.



Once clicked on create cluster, public DNS will be created as below.

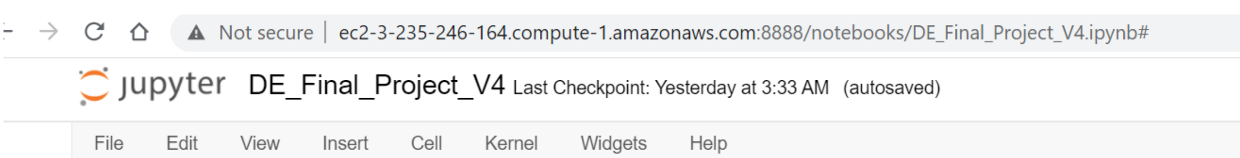
Master public DNS: ec2-54-237-71-230.compute-1.amazonaws.com.

Then connect to the Master Node Using SSH.

Commands to run to launch the Jupyter Notebook with required python packages

- ssh -i C:\Users\madhuyen\Downloads\dsci6007hm5.pem hadoop@ec2-54-237-71-230.compute-1.amazonaws.com
- sudo yum update
- sudo pip3 install pyyaml ipython jupyter ipyparallel pandas boto3 seaborn
- sudo yum install python3-devel
- sudo pip3 install scikit-surprise
- add these 2 commands in .bashrc
- export PYSPARK_DRIVER_PYTHON=/usr/local/bin/jupyter
- export PYSPARK_DRIVER_PYTHON_OPTS="notebook --no-browser --ip=0.0.0.0 --port=8888 "
- source ~/.bashrc
- pyspark
- After successfully running above commands jupyter browser link will be generated as below
- http://127.0.0.1:8888/?token=7f242244c2653be172aae837620a29505c14878af8cd3d49
- With master node instance we can run Jupyter notebook in browsers as below
- ec2-54-237-71-230.compute1.amazonaws.com:8888/?token=7f242244c2653be172aae837620a29505c14878af8cd3d49
- **Note:** We should have updated ports based on our application requirement
- For SSH 20, for Jupyter 8888 and for Zeppelin 8890 created in Security Inbound rules.

Jupyter Notebook launches as shown below:



Analyzing the Netflix Data

➢ The text data files are stored in S3. Import the text files and formed the Spark data frame.

Dataset Loading:- I have stored my data in AWS S3 service.

```
#location of data files
dbfs_dir = 's3://dsci-6007-hj-final2/Netflix/'
movieTitles_filename = dbfs_dir + 'movie_titles.txt'
trainingRatings_filename = dbfs_dir + 'TrainingRatings.txt'
testingRatings_filename = dbfs_dir + 'TestingRatings.txt'
```

DataAnalysis:-

(a) How many distinct items and how many distinct users are there in the test set?

```
In [4]: # getting the  movie count as unique to plot
movie_Unique_count = movieTitles_df.select("ID").distinct().count()
print("There are %s distinct_movies in Movie Titles dataset: "% (movie_Unique_count))

tr_movie_Unique_count = trainingRatings_df.select("movieId").distinct().count()
tr_user_Unique_count = testingRatings_df.select("userId").distinct().count()
tr_rating_Unique_count = testingRatings_df.select("rating").distinct().count()
print ('There are %s distinct_movies and %s distinct_Users and %s distinct_Ratings in the Training dataset' % (tr_movie_Unique_cc

ts_movie_Unique_count = testingRatings_df.select("movieId").distinct().count()
ts_user_Unique_count = testingRatings_df.select("userId").distinct().count()
ts_rating_Unique_count = testingRatings_df.select("rating").distinct().count()
print ('There are %s distinct_movies and %s distinct_Users and %s distinct_Ratings in the Testing dataset' % (ts_movie_Unique_cou

There are 17769 distinct_movies in Movie Titles dataset:
There are 1821 distinct_movies and 27555 distinct_Users and 5 distinct_Ratings in the Training dataset
There are 1701 distinct_movies and 27555 distinct_Users and 5 distinct_Ratings in the Testing dataset
```

```
In [14]: import seaborn as sns
sns.set_theme(style="darkgrid")
ax = sns.countplot(x="rating", data=pandas_tr_df)
ax.set_title('Rating Distribution of Training Data')
ax.set_ylabel('RatingCount')

Out[14]: Text(0, 0.5, 'RatingCount')
```

(b) The collaborative filtering approaches lives from finding many similar users (for a user-user model) or many similar items (item-item model):

(item-item model):

```
In [22]: #Item Based approach
sim_options = {
    'name': 'pearson',
    'user_based': 'False'
}

clf = KNNBasic(sim_options = sim_options)
cross_validate(clf, dataset, measures=['MAE'], cv=5, verbose=True)
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Evaluating MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MAE (testset)     0.7963  0.7972  0.7983  0.7978  0.7984  0.7976  0.0008
Fit time          2.50    4.18    2.48    2.48    2.52    2.83    0.68
Test time         9.24    6.87    7.34    6.65    6.69    7.36    0.97
```

user-user model

```
In [23]: #User Based approach
sim_options = {
    'name': 'MSD',
    'user_based': 'True'
}

clf = KNNBasic(sim_options = sim_options)
cross_validate(clf, dataset, measures=['MAE'], cv=5, verbose=True)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MAE (testset)     0.7874  0.7911  0.7874  0.7884  0.7867  0.7882  0.0016
Fit time          0.57    0.84    0.85    0.85    0.82    0.79    0.11
Test time         7.01    7.59    6.93    6.81    7.00    7.07    0.27

Out[23]: {'test_mae': array([0.78740071, 0.79108821, 0.7873836 , 0.78835323, 0.78665412]),
```

Compute the **Pearson correlation** coefficient between all pairs of users (or items).Only common users (or items) are taken into account. The Pearson correlation coefficient can be seen as a mean-centered cosine similarity, and is defined as:

$$\text{pearson\_sim}(u, v) = \frac{\sum\limits_{i \in I_{uv}} (r_{ui} - \mu_u) \cdot (r_{vi} - \mu_v)}{\sqrt{\sum\limits_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \cdot \sqrt{\sum\limits_{i \in I_{uv}} (r_{vi} - \mu_v)^2}}$$

or

$$\text{pearson\_sim}(i, j) = \frac{\sum\limits_{u \in U_{ij}} (r_{ui} - \mu_i) \cdot (r_{uj} - \mu_j)}{\sqrt{\sum\limits_{u \in U_{ij}} (r_{ui} - \mu_i)^2} \cdot \sqrt{\sum\limits_{u \in U_{ij}} (r_{uj} - \mu_j)^2}}$$

Compute the **Mean Squared Difference similarity** between all pairs of users (or items). Only common users (or items) are taken into account. The Mean Squared Difference is defined as:

$$\text{msd}(u, v) = \frac{1}{|I_{uv}|} \cdot \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2$$

or

$$\text{msd}(i, j) = \frac{1}{|U_{ij}|} \cdot \sum_{u \in U_{ij}} (r_{ui} - r_{uj})^2$$

If we find similar users, then we only have to do the process once for user.

From the set of similar users we can estimate all the blanks in the utility matrix for User.

If we work from similar items, we have to compute similar items for almost all items, before we can estimate.

On the other hand, item-item similarity often provides more reliable information, because of the phenomenon observed above, namely that it is easier to find movies of the same rating than it is to find users that like only movies of a single rating.

Whichever method we choose, we should precompute preferred movies for each user, rather than waiting until we need to make a decision.

Since the utility matrix evolves slowly, it is generally sufficient to compute it infrequently and assume that it remains fixed between re computations.

Problem 3: Collaborative Filtering Implementation:

MemoryBased:-

```
benchmark = []
# Iterate over all algorithms
for algorithm in [KNNBaseline(), KNNBasic(), KNNWithMeans(), KNNWithZScore()]:
    # Perform cross validation
    results = cross_validate(algorithm, dataset, measures=['RMSE'], cv=5, verbose=False)

    # Get results & append algorithm name
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp = tmp.append(pd.Series([str(algorithm).split(' ')[0].split('.')[-1]], index=['Algorithm']))
    benchmark.append(tmp)

pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse')
```

```
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
```

Out[25]:

Algorithm	test_rmse	fit_time	test_time
KNNBaseline	0.920993	3.442311	11.020222
KNNWithMeans	0.923492	1.002225	8.027472
KNNWithZScore	0.924000	1.498159	9.153796
KNNBasic	0.992764	0.781038	7.294846

Model Based ALS:-

```
for rank in ranks:
  # Set the rank here:
  als.setRank(rank)

  # Create the model with these parameters.
  model = als.fit(trainingRatings_df)

  # Run the model to create a prediction. Predict against the validation_df.
  predict_df = model.transform(testingRatings_df)
  print("Rank:",rank)

  # Remove NaN values from prediction (due to SPARK-14489)
  predicted_ratings_df = predict_df.filter(predict_df.prediction != float('nan'))
  print("predicted_ratings_df")
  predicted_ratings_df.show(3, truncate=False)

  # Run the previously created RMSE evaluator, reg_eval, on the predicted_ratings_df DataFrame
  error = reg_eval.evaluate(predicted_ratings_df)
  errors[err] = error
  models[err] = model
  print('For rank %s the RMSE is %s' % (rank, error))
  if error < min_error:
    min_error = error
    best_rank = err
  err += 1
```

```
For rank 8 the RMSE is 0.8616004838570359
For rank 8 the MSE is 0.8616004838570359
Rank: 12
predicted_ratings_df
+-------+-------+------+----------+
|movieId|userId |rating|prediction|
+-------+-------+------+----------+
|28     |2358799|3.0   |3.6669366 |
|156    |973051 |5.0   |3.9271145 |
|851    |1189060|3.0   |3.5178668 |
+-------+-------+------+----------+
only showing top 3 rows

For rank 12 the RMSE is 0.8681582347312298
For rank 12 the MSE is 0.8681582347312298

The best model was trained with rank 8 with respect to RMSE
The best model was trained with rank 8 with respect to MSE
```