

CS 556 Big Data Analysis

**Project : Recommendation Engine with**

**Collaborative Filtering, ALS and Autoencoders**

**CANBERK ASLAN**

# **Abstract**

In this work, i tried to implement 3 different method for Recommendation Engine. These methods are Collaborative Filtering with SVD , Alternating Least Square(ALS) and Autoencoders. Kaggle Netflix Data, Movie Lens 100K Data and 10M Data were used for this Project. Hadoop and Spark frameworks were used for experimenting the data I/O Operations.

# **Introduction**

A recommendation engine filters the data using different algorithms and recommends the most relevant items to users. It first captures the past behaviour of a customer and based on that, recommends products which the users might be likely to buy or watch or etc.

Recommendation systems are used not only for movies, but on multiple other products and services like Amazon (Books, Items), Pandora/Spotify (Music), Google (News, Search), YouTube (Videos) etc.

My focus in this project was to gain experience in taking up and solving a movie recommendation system problem for well-known Kaggle Netflix Movie Lens Datasets. After obtaining the datasets, as a first step, i focused on pre-processing data. Following that, i tried three different methodologies for recommendation predictions about movies with using Hadoop, Spark and TensorFlow Frameworks. The Kaggle Netflix Data are separated to four different part. Each parts are about 450 Mb size. Total size of movie ratings are around 2 Gb size for Netflix Movie Data. For Movie Lens 10 M Data, size of the ratings file is around 250 Mb size. Finally, 100 K Movie Lens data about 2 Mb size.

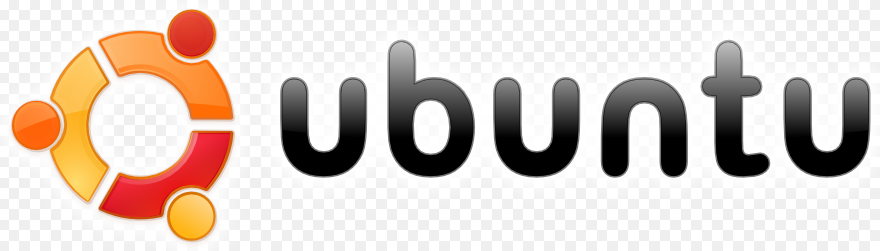
Recommendation systems can be consisting of explicit and implicit datasets. I used Implicit Data in this system. It consists of user ratings and the recommendations are depends on these ratings.

# **Environment and Data**

## **2.1 Environment**

In the starting period of Project, Unix Ubuntu 18.4 O.S. used for the base.

Ubuntu is an open source software operating system that runs from the desktop.



For easy environment installations and configurations, i decided to use Docker System at my Ubuntu environment. I tried to use “uhopper” for both Hadoop and Spark environments. For configuring, first of all installed the Docker for Unix. After installation i tried to implement Dockerfiles for Hadoop environments like below

* **HDFS:**

Namenodes, datanode

* **YARN:**

Resourcemanager, nodemanager

* **Spark submitter**

Hadoop Configurations: The *hadoop* configuration is controlled via the following environment variable groups:

a. CORE\_CONF: affects /etc/hadoop/core-site.xml

b. HDFS\_CONF: affects /etc/hadoop/hdfs-site.xml

1. YARN\_CONF: affects /etc/hadoop/yarn-site.xml
2. HTTPFS\_CONF: affects /etc/hadoop/httpfs-site.xml
3. KMS\_CONF: affects /etc/hadoop/KMS-site.xml

MULTI\_NETWORK: configure the Hadoop cluster in such a way to be reachable from multiple networks, specifically the following properties are set:

* In ***/etc/hadoop/hdfs-site.xml:***

dfs.namenode.rpc-bind-host = 0.0.0.0

dfs.namenode.servicerpc-bind-host = 0.0.0.0

dfs.namenode.http-bind-host = 0.0.0.0

dfs.namenode.https-bind-host = 0.0.0.0

dfs.client.use.datanode.hostname = true

dfs.datanode.use.datanode.hostname = true

* In ***/etc/hadoop/yarn-site.xml***:

yarn.resourcemanager.bind-host = 0.0.0.0

yarn.nodemanager.bind-host = 0.0.0.0

yarn.nodemanager.bind-host = 0.0.0.0

* In ***/etc/hadoop/mapred-site.xml***:

yarn.nodemanager.bind-host = 0.0.0.0

**Components**

* **namenode**

The hadoop-namenode image starts a Hadoop NameNode. (single instance)

* Additional environment variables:

CLUSTER\_NAME: name of the HDFS cluster (used during the initial formatting)

* Volumes:

/hadoop/dfs/name: HDFS filesystem name directory

* Mandatory configuration:

CLUSTER\_NAME: cluster name

* Docker-compose template:

*namenode:*

*image: uhopper/hadoop-namenode*

*hostname: namenode*

*container\_name: namenode*

*domainname: hadoop*

*net: hadoop*

volumes:

- <NAMENODE-VOLUME>:/hadoop/dfs/name

environment:

- GANGLIA\_HOST=<GMOND-RECEIVER-HOST>

- CLUSTER\_NAME=<CLUSTER-NAME>

For testing env -> http://<CONTAINER\_IP>:50070 to see the webui.

* **datanode**

The hadoop-datanode image starts an Hadoop DataNode. (multiple instances)

* Volumes:

/hadoop/dfs/data: HDFS filesystem data directory

* Mandatory configuration:

CORE\_CONF\_fs\_defaultFS: HDFS address (i.e. hdfs://<NAMENODE-HOST>:8020)

* Docker-compose template:

*datanode1:*

*image: uhopper/hadoop-datanode*

*hostname: datanode1*

*container\_name: datanode1*

*domainname: hadoop*

*net: hadoop*

*volumes:*

*- <DATANODE-VOLUME>:/hadoop/dfs/data*

*environment:*

*- GANGLIA\_HOST=<GMOND-RECEIVER-HOST>*

*- CORE\_CONF\_fs\_defaultFS=hdfs://<NAMENODE-HOST>:8020*

* **resourcemanager**

The hadoop-resourcemanager image starts an Hadoop ResourceManager. (single instance)

* Mandatory configuration:

CORE\_CONF\_fs\_defaultFS: HDFS address (i.e. hdfs://<NAMENODE-HOST>:8020)

* Docker-compose template:

resourcemanager:

image: uhopper/hadoop-resourcemanager

hostname: resourcemanager

container\_name: resourcemanager

domainname: hadoop

net: hadoop

environment:

- GANGLIA\_HOST=<GMOND-RECEIVER-HOST>

- CORE\_CONF\_fs\_defaultFS=hdfs://<NAMENODE-HOST>:8020

- YARN\_CONF\_yarn\_log\_\_\_aggregation\_\_\_enable=true

For testing environment http://<CONTAINER\_IP>:8088 to see the webui.

* **nodemanager**

The hadoop-nodemanager image starts an Hadoop NodeManager. (multiple instances)

* Mandatory configuration:

CORE\_CONF\_fs\_defaultFS: HDFS address (i.e. hdfs://<NAMENODE-HOST>:8020)

YARN\_CONF\_yarn\_resourcemanager\_hostname: resourcemanager host

* Docker-compose template:

nodemanager1:

image: uhopper/hadoop-nodemanager

hostname: nodemanager1

container\_name: nodemanager1

domainname: hadoop

net: hadoop

environment:

- GANGLIA\_HOST=<GMOND-RECEIVER-HOST>

- CORE\_CONF\_fs\_defaultFS=hdfs://<NAMENODE-HOST>:8020

- YARN\_CONF\_yarn\_resourcemanager\_hostname=<RESOURCEMANAGER-HOST>

- YARN\_CONF\_yarn\_log\_\_\_aggregation\_\_\_enable=true

- YARN\_CONF\_yarn\_nodemanager\_remote\_\_\_app\_\_\_log\_\_\_dir=/app-logs

* **spark**

The Hadoop-spark image is an utility container which provides a Spark environment configured for the Hadoop cluster.

The image itself doesn't specify any command since no service are exposed. You are expected to specify it yourself via docker run uhopper/hadoop-spark <command>.

A common approach is to keep the container alive using tail -f /var/log/dmesg as command and then connect to it via docker exec -ti spark bash to have a spark environment.

* Mandatory configuration:

CORE\_CONF\_fs\_defaultFS: HDFS address (i.e. hdfs://<NAMENODE-HOST>:8020)

YARN\_CONF\_yarn\_resourcemanager\_hostname: resourcemanager host

* Docker-compose template:

spark:

image: uhopper/hadoop-spark

hostname: spark

container\_name: spark

domainname: hadoop

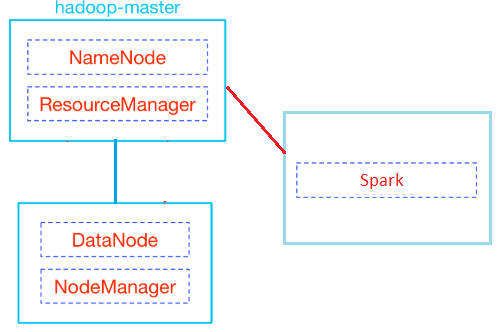
net: hadoop

environment:

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- YARN\_CONF\_yarn\_resourcemanager\_hostname=resourcemanager

command: tail -f /var/log/dmesg



After trying to apply these configuration, unfortunately Spark server did not work properly. I tried to solve this problem but i saw that it seems like a bug.

So i decided to install the system standalone version on my Ubuntu environment. I quitted from Docker installation and configuration.

When i tried to install my system to Ubuntu, i saw that Python 3.7 is not compatible with TensorFlow. So i installed another Python (version 3.6) for TensorFlow Framework.

After completing these configurations for standalone Unix Environment, i tried to work at GPU instead of CPU for performance issues. But my system was crashed because of Nvidia Cuda Driver Installations.

So finally, i switched to Microsoft Windows 10 O.S. for fresh start

## **2.2** **Data**

In my environment i used Kaggle Netflix Data for movie ratings. The size of all data is about 2Gb and divided into four main parts.

Each parts consists of “**UserId:MovieID,Ratings,Timestamp**” formatted data.

Other data for Movie Lens (10M and 100K) are one file. Files consists of “**UserId::MovieID::Ratings::Timestamp**” formatted data.

## **2.3 Final Environment and Data**

* *Software Environment*
  + Hadoop 2.7 Environment
  + Spark 2.3.0 Framework
  + Implementation Lang: Python 3.7 for SVD CF and ALS, Python 3.6 for Autoencoders
  + TensorFlow Framework
* *O.S.*
  + Windows 10
* *Infra Structure*
  + Processor: Intel i7 5500U and Intel i7 7780HQ
  + Ram: 8GB DDR3L and 16GB DDR4
  + GPU tried but had issues: Nvidia GTX960M and Nvidia 1050Ti
* *Data*
  + Kaggle Netflix Data, Movie Lens 100K Data and 10M Data
  + Train Set : % 80 , Test Set : %20

# **Pre-processing and Algorithms For Recommendation**

For Kaggle Netflix Data, data must be pre-processed because of the format of the lines and sparsity.

In pre-processing implementation, I used python pandas framework. The data consist of “Userid:” column and after this column, a lot of line for this user which like “Movieid,Rating,Timestamp”. I convert every line to “Userid,MovieId,Rating,Timestamp” format.

After formatting lines, I deleted sparse data lines (which does not have rating value). Output of the new file will be written to same directory.

For MovieLens Data, I don’t have to do pre-process operation. Because the format of the data is ready like “Userid::MovieId::Ratings::Timestamp” and does not have any sparse data for ratings.

After completing Pre-Processing operations, I applied the algorithms.

## **3.1** **Colloborative Filtering with SVD**

Collaborative Filtering (CF) is a method of making automatic predictions about the interests of a user by learning its preferences (or taste) based on information of his engagements with a set of available items, along with other users’ engagements with the same set of items. CF assumes that, if a *person A* has the same opinion as *person B*on some set of issues *X={x1,x2,…}*, then *A* is more likely to have *B*‘s opinion on a new issue *y* than to have the opinion of any other person that doesn’t agree with *A*on *X.*

The CF techniques are broadly divided into 2-types:

#### **3.1.1. Memory Based Approach**

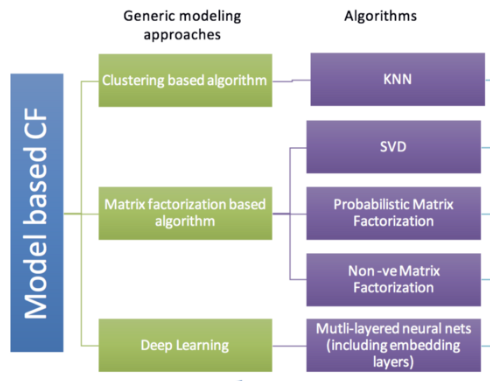
Memory-Based Collaborative Filtering approaches can be divided into two main sections: user-item filtering and item-item filtering. A user-item filtering takes a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked. In contrast, item-item filtering will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations.

*Item-Item Collaborative Filtering*: “Users who liked this item also liked …”  
*User-Item Collaborative Filtering*: “Users who are similar to you also liked …”

As no training or optimization is involved, it is an easy to use approach. But its performance decreases when we have sparse data which hinders scalability of this approach for most of the real-world problems.

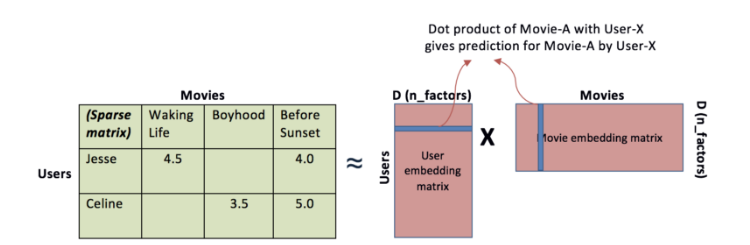
#### **3.1.2. Model Based Approach**

In this approach, CF models are developed using machine learning algorithms to predict user’s rating of unrated items. As per my understanding, the algorithms in this approach can further be broken down into 3 sub-types.



Matrix Factorization Based Algorithm which is SVD used for this Project implementation.

The idea behind the matrix factorization is that attitudes or preferences of a user can be determined by a small number of hidden factors. We can call these factors as Embeddings.



Intuitively, we can understand embeddings as low dimensional hidden factors for items and users. For example we have 5 dimensional (i.e. D or n\_factors = 5 in above figure) embeddings for both items and users (# 5 chosen randomly). Then for user-X & movie A, we can say the those 5 numbers might represent 5 different characteristics about the movie, like

1. how much movie-A is sci-fi intense
2. how recent is the movie
3. how much special effects are in movie A
4. how dialogue driven is the movie
5. how CGI driven is the movie.

Likewise, 5 numbers in user embedding matrix might represent, (i) how much does user-X like sci-fi movie (ii) how much does user-X like recent movies …and so on. In above figure, a higher number from dot product of user-X and movie-A matrix means that movie-A is a good recommendation for user-X.

SVD is a matrix decomposition technique that has mathematically originated from linear algebra. It decomposes any matrix into 3 matrices

{{U}\in{R^{m \times k}}} , {{\Sigma}\in{R^{k \times k}}} and {{V}\in{R^{m \times k}}} such that {{\;A}} = {{U}} \times {{\Sigma }} \times {{{V}}^{{T}}}.

It comes with stronger guarantees than Matrix Factorization’s:

* {{\Sigma }} is a diagonal matrix having the singular values of A on its diagonal. A common convention is to list the singular values in {{\Sigma }} in a descending order.
* Data must be linear
* U and V are orthonormal matrices (their columns are orthogonal and their norm equals 1)
* SVD solution is unique.

So if we assume A is a matrix of user-item ratings (users as rows, items as columns), that means that:

* Each row in U (or V) corresponds to a user (or item) characteristics. So for example, {{{U}}_{i,k}} can be interpreted as the strength of membership of user i to the kthcategory, and {{{V}}_{j,k}} can be interpreted as the strength of membership of user i to the samekth Since U and V form orthonormal bases, it follows that (for each k) the overall strength of the kth effect on ratings is already “deducted” from the values of the kth column in U or V.
* Each rating (value in {{{A}}_{{{ij}}}}) is explained by a set of independent categories / effects. For a specific useri and itemj, the rating {A_{ij}} is a decomposed into the following summation \mathop \sum \limits_{k = 1}^K \left[ {{u_{i,k}}*\;{\Sigma _{k,k}}*v_{j,k}^T} \right] where each value of k determines:
* {{{u}}_{{{i}},{{k}}}} – user\_i‘s kth latent factor
* {{v}}_{{{j}},{{k}}}^T – item\_j‘s kth latent factor
* {{{\Sigma }}_{{{k}},k}} – the overall weight (magnitude of effect) of the kth factor

It follows that any rating given by user i to item j is affected by K factors, each, in turn, is decomposed into: similarity between user I and item j in this dimension ({u_{i,k}}*v_{j,k}^T), and the overall effect of this dimension ({\Sigma _{k,k}}) on ratings across all users and items.

* Since \Sigma is ordered by the size of the singular values of A in descending order,

the accumulative sum \mathop \sum \limits_{k = 1}^K {\Sigma_{k,k}} accounts for the total variance (effect on the ratings) explained by the k strongest effects (or categories). Therefore, it is easy to roughly estimate what should be the rank of A (how many factors affect the ratings).

* The main goal of SVD is minimize the sum of reconstruction errors.

## **3.2** **ALS (Alternating Least Square)**

Alternating Least Square (ALS) is also a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is implemented in Apache Spark ML and built for a larges-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.

* 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.
* ALS is implemented in Apache Spark ML
* ALS is good at a larges-scale collaborative filtering problems.
* ALS is a good at solving scalability and sparseness of the Ratings data.

Just like other machine learning algorithms, ALS has its own set of hyper-parameters. We probably want to tune its hyper-parameters via hold-out validation or cross-validation.

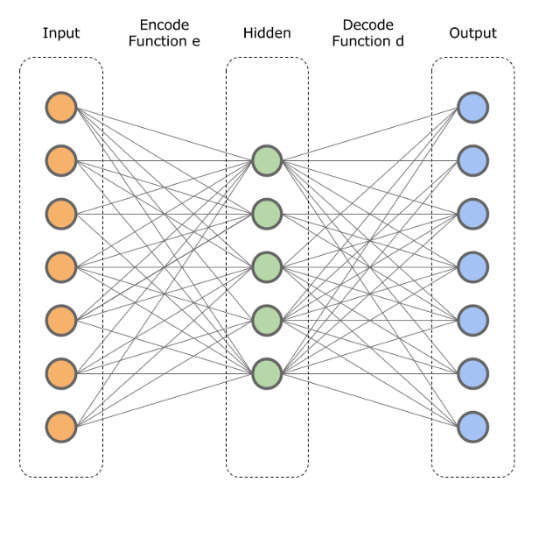
Most important hyper-params in Alternating Least Square (ALS):

* 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)

## **3.3 Autoencoders**

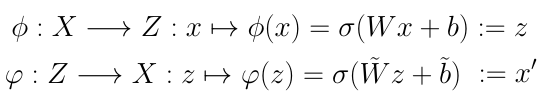
An Autoencoder is a deep learning neural network architecture that achieves state of the art performance in the area of collaborative filtering and it is an artificial neural network used to learn a representation (encoding) for a set of input data, usually to a achieve dimensionality reduction.

Architecturally, the form of an Autoencoder is a feedforward neural network having an input layer, one hidden layer and an output layer. The output layer has the same number of neurons as the input layer for the purpose of reconstructing it’s own inputs. This makes an Autoencoder a form of unsupervised learning, which means no labelled data are necessary — only a set of input data instead of input-output pairs.



It is useful that an Autoencoder has a smaller hidden layer than the input layer. This effect forces the model to create a compressed representation of the data in the hidden layer by learning correlations in the data.

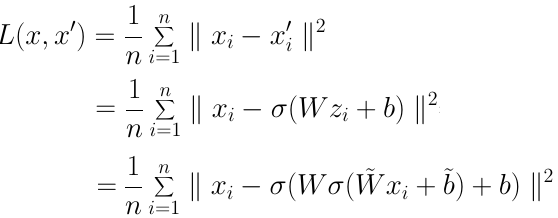
The transition from the input to the hidden layer is called the encoding step and the transition from the hidden to the output layer is called the decoding step. We can also define these transitions mathematically as a mapping:



The mapping is realized by multiplying the input data vector *x* with a weight matrix, adding a bias term and applying to the resulting vector a non linear operation **σ**such as sigmoid, tanh or rectified linear unit.

At training time for data, the encoder takes a input data sample**x** and maps it to the so called hidden or latent representation **z.**Then the decoder maps **z**to the outputvector **x’**which is the exact representation of the input data **x**.

Having the output **x’**the training consists of applying stochastic gradient descent to minimize a predefined loss such as a mean squared error:



Tensorflow framework used for implementing the Autoencoders for recommendation system.

2 different network model were used for Project. In the first Neural Network 100 Epochs used with Relu and Adam Optimizer and the layers like below.

**Encoder: (Moviesize x 500) \* ( 500 x 100) => 2 Layers**

**Decoder: (100 x 500) \* (500 x Moviesize) => 2 Layers**

In the second Neural Network 100 Epochs used with Sigmoid and Adam Optimizer and the layers like below.

**Encoder: (Moviesize x 10) => 1Layers**

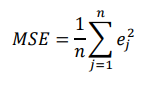
**Decoder: (10 x Moviesize) => 1 Layers**

# **Discussion and Results**

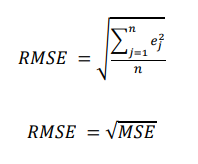
As I mentioned before parts , I applied 3 different algorithms to our movie data. I tried to explain how they are working and now I want to show the results and performance metrics of these algorithms.

Before results and discussion, I want to give a brief explanation about what is MSE and RMSE for system’s results. Because system’s results parameters have MSE and RMSE values.

**MSE:** Simply, the mean square error tells you how close a regression curve is to a set point. The MSE measures the performance of a machine learning model, the predictor, is always positive, and it can be said that the predictors that are close to zero have a better performance.



**RMSE:** It is a quadratic metric that measures the magnitude of the error, often used in finding the distance between a machine learning model and the values ​​predicted by the estimator. RMSE is the standard deviation of prediction errors (residues). That is, the remains are a measure of how far the regression line is from the data points; RMSE is a measure of how much of these remains are spreading. In other words, it tells you how dense the data is around the line that best fits the data. The RMSE value can vary from 0 to R. Negative-oriented scores, ie those with lower values, perform better. A RMSE value of zero means that the model has not made any errors. RMSE has the advantage of punishing large errors further, so it may be more appropriate in some situations. RMSE prevents the use of unwanted absolute values ​​in many mathematical calculations.

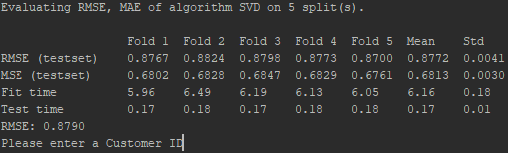


For all data, I split %80 for Training, %20 for Testing.

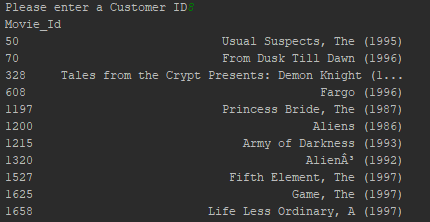
## **4.1 CF with SVD & Results**

For **Collaborative Filtering with SVD**, I used Surprise and Panda Frameworks. Cross validation and SVD algorithms are in the libraries at Surprise Framework. Called with specific parameters for implementation. In the structure of the code first of all, all data was read and %80 of data was used for training and %20 of data was used for testing. Lastly the RMSE and MSE values are calculated depends on recommendations(predictions). The duration of calculations **approximately 402.3910 seconds** for Movie Lens Rating Data. After that I added an interactive output for taking a customer id from user and try to recommend 30 top movies depends on selected customer and selected customer’s ratings.

Example like below



After entering Customer Id a sample of recommendations.



The output values of our measure values like that for SVD,

Mean Test RMSE Value : 0.8752

Mean Train RMSE Value : 0.8673

Mean Test MSE Value : 0.7202

Mean Train MSE Value : 0.706

## **4.2 ALS & Results**

For **ALS (Alternating Least Square) Method,** I used Spark Framework. In my environment I could use only one Spark Server. So I don’t have any distributed architecture. So outputs of the algorithm has not too much difference between SVD results. Again the data has been splitted into two parts %80 trainin %20 test .

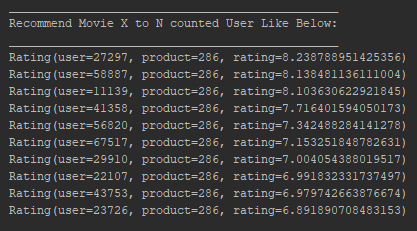
The duration of calculations **approximately 484.5570 seconds** for Movie Lens Rating Data. ALS library has Rank and NumberofIterations values for hyper-parameters. Since v1.1, Spark Framework scale the regularization parameter lambda in solving each least squares problem by the number of ratings the user generated in updating user factors, or the number of ratings the product received in updating product factors. This approach is named **“ALS-WR”** and discussed in the paper “[Large-Scale Parallel Collaborative Filtering for the Netflix Prize](http://dx.doi.org/10.1007/978-3-540-68880-8_32)”( <https://link.springer.com/chapter/10.1007%2F978-3-540-68880-8_32>) . It makes lambda less dependent on the scale of the dataset, so we can apply the best parameter learned from a sampled subset to the full dataset and expect similar performance.

NumberofIterations: the maximum number of iterations to run (defaults to 10)

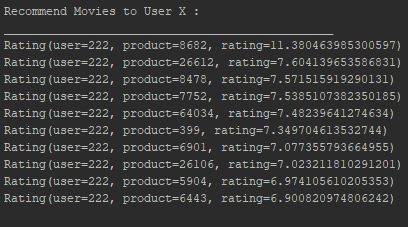
rank: the number of latent factors in the model (defaults to 10)

Again i am trying to recommend a specific customer to different movies depends on his/her ratings. In addition to that i tried to recommend a specific movie to different customers depends on their ratings.

Recommendation of Movie X to N Customer



Recommendation of Movies to Customer X



After that i tried to get the measurements of my ALS Recommendation System prediction results

Mean Test RMSE: 0.8181

Mean Train RMSE: 0.7728

Mean Train MSE: 0.5973

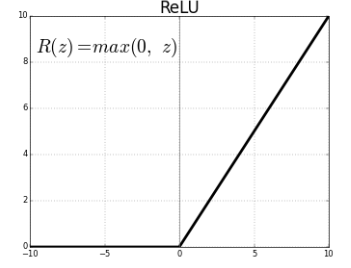
Mean Test MSE: 0.6693

## **4.3 Autoencoders & Results**

For **AutoEncoders,** I used TensorFlow Framework.In addition to that used panda framework. An **autoencoder** neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. Again the data was splitted in to 2 parts for %80 training, %20 for testing.

Before the system architecture I want to give a brief explanation about Relu and Sigmoid functions

**The Relu** **(Rectified Linear Unit -Activation Func)** is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning.



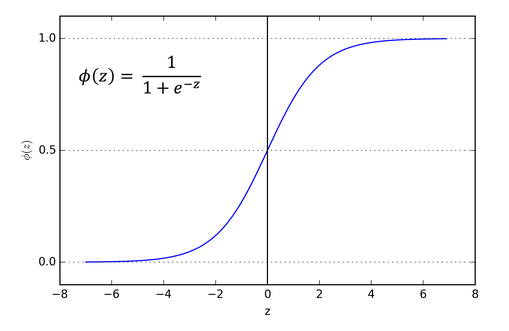
As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

**Range:**[ 0 to infinity)

The function and its derivative **both are** **monotonic**.

But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

**The Sigmoid Function** curve looks like a S-shape.



The main reason why we use sigmoid function is because it exists between **(0 to 1).**Therefore, it is especially used for models where we have to **predict the probability** as an output.Since probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice.

The function is **differentiable**.That means, we can find the slope of the sigmoid curve at any two points.

The function is **monotonic**but function’s derivative is not.

The logistic sigmoid function can cause a neural network to get stuck at the training time.

The **softmax function** is a more generalized logistic activation function which is used for multiclass classification.

I applied two different Autoencoder architecture to Movie Recommendation system. First system architecture like below

**Encoder:** (Moviesize x 500) \* ( 500 x 100) => 2 Layers

**Decoder:** (100 x 500) \* (500 x Moviesize) => 2 Layers

**Relu Function** with Dropout and **Adam Optimizer** for decreasing the error and 100 Epochs applied for results.

Second system architecture like below

**Encoder:** (Moviesize x 10) => 1Layers

**Decoder:** (10 x Moviesize) => 1 Layers

**Sigmoid Function** with Dropout and **Adam Optimizer** for decreasing the error and 100 Epochs applied for results.

**(I selected Adam Optimizer for 2 architecture, because before work experiments, i saw that Adam Optimizer gives much more accurate results)**

**First System Prediction Results**

epoch:0 rmse(train):3.6543 rmse(test):3.6614

epoch:1 rmse(train):3.5380 rmse(test):3.5466

epoch:2 rmse(train):3.2796 rmse(test):3.2902

epoch:3 rmse(train):2.8622 rmse(test):2.8747

|||

epoch:97 rmse(train):1.5155 rmse(test):1.5405

epoch:98 rmse(train):1.5356 rmse(test):1.5603

epoch:99 rmse(train):1.5434 rmse(test):1.5680

**Second System Prediction Results**

epoch:0 rmse(train):3.7122 rmse(test):3.7198

epoch:1 rmse(train):3.7024 rmse(test):3.7106

epoch:2 rmse(train):3.6964 rmse(test):3.7051

epoch:3 rmse(train):3.6921 rmse(test):3.7012

|||

epoch:97 rmse(train):2.8615 rmse(test):2.8737

epoch:98 rmse(train):2.8510 rmse(test):2.8634

epoch:99 rmse(train):2.8403 rmse(test):2.8529

## **4.4 Comparisons & Results**

After getting all results from our 3 different system, you can find the result of the efficiency and errors when the systems trying to predict some outputs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MovieLens Rating Data** | | | | |
|  | **CF with SVD** | **ALS** | **AutoEncoder 1st** | **AutoEncoder 2nd** |
| **Train RMSE** | 0.8673 | 0.7728 | 1.5434 | 2.8403 |
| **Test RMSE** | 0.8752 | 0.8181 | 1.568 | 2.8529 |
| **Train MSE** | 0.706 | 0.5973 | 2.3716 | 8.06 |
| **Test MSE** | 0.7202 | 0.6693 | 2.4586 | 8.139 |
| **Time Durations** | ~ 6.5 Mins | ~ 7 ,5 Mins | Over 15 Mins | Over 10 Mins |

The outcome is best algorithm is ALS for recommendation systems (in 3 algorithms)

The problems were like below

* **Cold-start problem**: Refers to when movies added to the catalogue have either none or very little interactions while recommender rely on the movie’s interactions to make recommendations

New items have no ratings

New users have no history

* **Data Sparsity**
* **Dimensionality:** Reduction of dimensionality provides

Discover hidden correlations/topics (Words that occur commonly together)

Remove redundant and noisy features (Not all words are useful)

Easier storage and processing of the data

* In SVD algorithm, the Dimensionality Reduction can be applied easily and it’s very efficient. But data must be linear at SVD Algorithm and difficult to interpret and these are a disadvantage of this algorithm(So sparsity must be corrected manually).

It’s basic forms means fitting some line to the data, measuring the sum of squared distances from all points to the line and trying to get an optimal fit for missing points.

* In ALS algorithm used matrix factorization algorithm and runs in a parallel fashion. Good at larges-scale collaborative filtering problems.The Dimensionality Reduction can be applied easil too. In addition to that ALS is a good at solving scalability and sparseness of the data.

 We used the same idea at fitting like SVD but iteratively alternate between optimizing U and fixing V and vice versa. It is an iterative optimization process where we for every iteration try to arrive closer and closer to a factorized representation of our original data.

* Auto Encoders are a kind of unsupervised Neural Network. It can solve Cold-Start Problem, Dimensionality Problem and Data Sparsity problem. But the problem about autoencoders, you must apply big amount of data for decreasing the error and must construct your system before experiments. I could not use them efficiently because of hardware limitations (Processing big amount of data) . But if I had suitable infrastructure, I can solve these problems more easy than before algorithms.

# **Future Works**

From the results we can experiment that the best algorithm is ALS. But

because of the environment(hardware) and problems about cluster mechanism at O.S. or Docker, I could not construct a distributed Hadoop or Spark Environment. If I could succeed it, the recommendation errors will be decreasing and the performance of the system will be increasing certainly. So in future works the system can be improved with distributed architecture.

In addition to that my Autoencoder layers and architectures are not work as good as ALS and CF with SVD. So in the working on more accurate layers and node counts may can be a future work too.

# **References**

<https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1>

<https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html>

<https://link.springer.com/chapter/10.1007%2F978-3-540-68880-8_32>

<https://datasciencemadesimpler.wordpress.com/tag/alternating-least-squares/#SVD>

<https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html>

<https://github.com/databricks/spark-training/blob/master/website/movie-recommendation-with-mllib.md>

<https://bugra.github.io/work/notes/2014-04-19/alternating-least-squares-method-for-collaborative-filtering/>

<https://openreview.net/forum?id=SkNQeiRpb>

<https://towardsdatascience.com/deep-autoencoders-for-collaborative-filtering-6cf8d25bbf1d>

<https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe>

<https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0>

<http://kontext.tech/docs/DataAndBusinessIntelligence/p/install-hadoop-300-in-windows-single-node>

<https://mc.ai/recommender-system-on-the-movielens-dataset-using-an-autoencoder-using-tensorflow-in-python/>

<https://lms.ozyegin.edu.tr/pluginfile.php/153665/mod_resource/content/0/07-recsys1.pdf>

<https://medium.com/@SnazzyHam/how-to-get-up-and-running-with-clouderas-quickstart-docker-container-732c04ed0280>

<http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/>

<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>