
# Music Recommendation using Collaborative filtering and Matrix factorization

GACS-7401-002

The final project of Advanced Machine Learning

submitted to

Dr.Sheela Ramanna

in partial fulfilment for the degree of Master of Science

By

Hardeep singh

3089450

Singh-h18@webmail.uwinnipeg.ca

April 11, 2017

# Music Recommendation using Collaborative filtering and Matrix factorization

Hardeep Singh

140 Margate Road, R2P 1B2
Winnipeg, MB
Singh-h18@webmail.uwinnipeg.ca

**Abstract.** Collaborative filtering technique is widely used in recommendation systems. This method is able to provide recommendations to user through similarities between users. However, the main issue with this method was sparsity of the data matrix. To overcome this, matrix factorization technique is used to find dense latent factors. This paper discuss about how matrix factorization use in music recommendation systems for implicit feedback data. This paper also discuss about how latent factors of user-artist matrix learned by using alternating least squares algorithm and how models are computed for recommendations. The quality of recommendation system is evaluated by computing area under the curve in one of the experiment. It can be concluded that the matrix factorization with alternating least squares learning method produced better recommendation accuracy. This method can be implement in spark with python by using machine learning libraries.

**Keywords:** Collaborative filtering, Matrix factorization, Recommendation systems, alternating least squares.

## 1 Introduction

The importance of recommendation systems has been growing rapidly nowadays due to the huge data information available on web. These systems are becoming very popular in various fields. Every user needs a personalized system that can be match to their recent activity, what they read recently or what song they listened recently. These recent activities of users is the measure to recommend them the similar items. Recommendation systems are become more popular today and has been used in various fields. It is important today for lot of sites on internet to predict what content a user wants, in order to be competitive. There are various sites on internet such as Amazon, YouTube, and Netflix, which developed highly advanced systems for recommending new and relevant contents to users. Recommender systems play a vital role in these companies as depicted by the Netflix sponsored competition with a prize of one million dollars to improve their system [1]. These all types of systems collectively known as recommender systems [2].

Recommender systems can take multiple different approaches to achieve similar item results for a particular user. The core task of a recommender system is to predict, how a user might like an item based on the user and other users' past behavior. There are two most common techniques used to develop recommender systems: content-based and collaborative filtering. Content-based filtering compare features of different items and then make recommendations by finding items that are similar to what user liked or used previously. However, Collaborative filtering tries to give recommendation to users based on their similarity to other users.

Recommender systems rely on different types of input. Explicit feedback data's are the most convenient, which includes explicit input by users regarding their interest in products [3]. For example, Google play collects star ratings for different applications. Meanwhile, this type of feedback is not always available. The other type of data available is implicit feedback data, which is huge in size [4]. The examples of these kind of data are song listening history, browsing history, search history and purchase history. If a user is listening to a particular song many times, probability is there that the user likes that song.

Matrix factorization techniques has been proven very effective for implementing recommender systems, most significantly in Netflix prize competition [1]. Matrix factorization is the popular approach for the matrix completion problem. It is based on algorithm called latent factor models, which explains observed interactions between large numbers users and items through a relatively small latent-factors. Alternating least squares is the type of matrix factorization technique to be used in this paper.

For experiment purpose, Implicit Audioscrobbler dataset (which is now merged with last.fm) will be used in this paper. It contains large amount of user data, artist data and play counts by users for various artists. The alternating least squares matrix factorization algorithm will be used to learn user and artist latent-factors to improve prediction accuracy. The best prediction model will be searched by computing different ranks for user and artist latent-factors, which will help in recommending top artists to a user. The quality of the recommender system will be evaluated by finding Area under the curve.

The section 2. Will discuss about the problem definition. Moreover, background information of the technique used in this paper will be discussed in section 3. Furthermore section 4 will discuss about dataset analysis and the theoretical techniques used in this paper and section 5 contains code diagrams and system specifications followed by experiments and conclusion sections.

## 2 Problem description

The goal of matrix factorization and collaborative filtering is to analyze users past behavior and how it similar to other user in order to predict how they are going to act in future. Implicit feedback data come in many forms such as clicks, page views, media

streaming counts. The song listen data is to be considered in order to develop recommendation system. In our case, we have a set of non-negative feedback values associated with each pair of users and artists. Formal notation:-

- $U = (u_1, ....., u_n)$ :a group of n users
- $I = (i_1, ....., i_m)$ : a group of m artists
- $R = (r_{ui})_{n \times m}$ : a user artist observation matrix where $r_{ui} \in \mathbf{R}_{>=0}$ represents the number times that user u listened to artist i.

Notice, we do not require the $r_{ui}$ values to be integers but instead the values are allowed to be any non-negative reals. The matrix $r_{ui}$ might be very sparse because if a user, u do not interact with an artist i, 0 will be placed there. It is important to note that a value of 0 does not necessarily imply that the user does not like that artist, but could simply imply that the user does not know about the artist. Then, our goal is to recommend top artists to each user that they have not yet interacted with. Or, where $r_{ui} = 0$).

# 3 Related work

In this section we briefly review the main works in the context. Our main focus will be on collaborative filtering and matrix factorization which has been successfully applied on several real world problems, such as Netflix movie recommendation.

## 3.1 Content-based Filtering

Content-based filtering is one of the most popular and the oldest method for constructing a recommendation system. The primary source of information for such systems is content known about the items in the system. This can be size, length, color, name and so on. In the context of music, content could be the Meta information about the song such as artist, genre, and upload date [5]. Content-based recommender systems typically make recommendations by finding items that are similar to the items that the user used in the past. Predictions are typically made by constructing a profile for the particular user preferences based on the content of the items that the user previously liked, rated, listened, or interacted with [6]. The user profile must be accurate in order to predict effectively. If the items content does not have sufficient amount of features, then these systems will not recommend items that are similar to the ones already in users' profile.

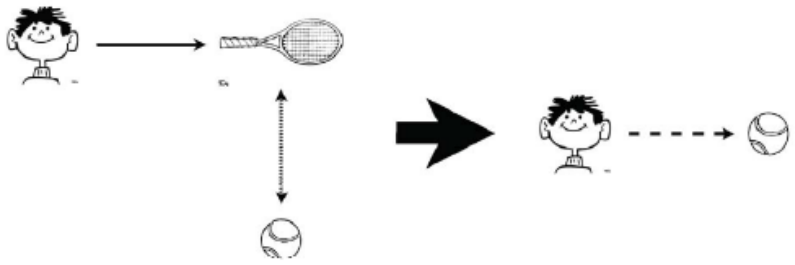

**Fig. 1.** Content-based filtering

Fig.1 shows an example of content-based filtering, where relationship is marked with full arrows. Similarity is marked with an arrow point and predictive relationship marked by dotted arrows.

This technique was used in text-based applications, to recommend a document or web-site [6].

### 3.2 Collaborative Filtering

Collaborative Filtering recommender systems make predictions by looking user-item relations, without requiring any additional information about the users or items. Companies like Google [7], Amazon [8] using the recommendation systems based on collaborative filtering. It requires large amount of information to be processed, including large-scale datasets such as ecommerce and web applications data's. Over the past few decades Collaborative filtering has been continuously improved and has been one of the most prominent recommendation method.

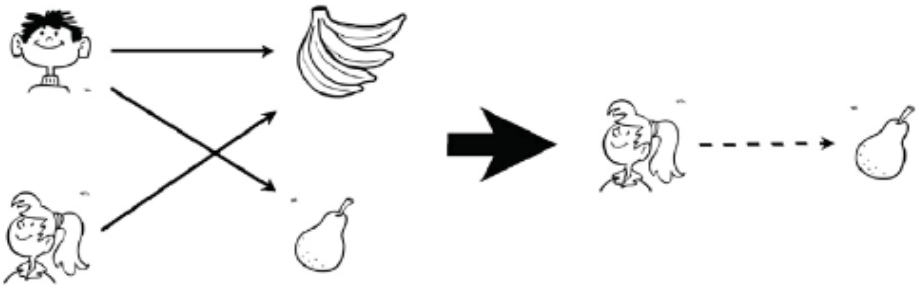

**Fig. 2.** Collaborative filtering

Fig.2 shows an example of collaborative filtering. Where the full line arrows showing the relationships that are already known. In other words, users like those items. As both

users like same item, it is considered that users have similarities. Recommendations are marked with dotted arrows.

The two most common approaches for implementing Collaborative filtering recommender systems are neighborhood-based and latent factor models.

**Neighborhood-based model.** Neighborhood-based technique makes prediction by computing similarities between user with other users and item with other items. The item based approach, calculates an item to item similarity and makes recommendations by weighting the similar items that user has used or rated before [8]. However, the user based approach look at what other users with similar behaviors when making recommendations. Neighborhood-based model combined with Pearson correlation or cosine similarity to find predicted values.

*Limitations.* When it comes to large datasets such as implicit feedback which contains very sparse ratings, neighborhood-based models experience difficulties in finding the right match and produce a weak recommendations to users. Moreover, the algorithm complexity of this model increase with number of users and number of items, which means the system will have problems in scalability.

**Latent factor model.** Latent factor model for recommendation systems utilize the user-item matrix to characterize both user and items by number of latent features. The factor vectors of users and items can be multiplied together to give predicted matrix. In the context of music, latent factors can be of users and artists which will be extracted from the large implicit feedback datasets. An effective latent factor method for recommendation systems is matrix factorization [7] which will be focus in this report. It will help in overcoming the problem of scalability in large datasets which is faced in neighborhood-based models.

## 4 Theoretical Framework

There are several things that need to be done before the recommendation can be made. First of all, we need to know about the dataset. Secondly, the matrix notation followed by the matrix factorization technique.

### 4.1 Dataset Analysis

The dataset used in this paper is publicly available song data from Audioscrobbler. Unlike a star rating dataset which contains information about the ratings provided by several users on the scale of 1 to 5 stars, the Audioscrobbler dataset only have the information about how many times a user played songs of a given artist and the names of artists. The original dataset occupies about 141k unique users, and 1.6 million unique artists. Moreover, 24.2 million users' plays of artist, along with their count. The reduced

dataset contains top 50 users with highest artist play counts. The dataset contains following 3 files.

**User_artist_data_small.txt:** It contains 3 features userId, artistId and playcount as shown in Fig.3.

| userId | artistId | playcount |
|---|---|---|
| 1059637 | 1000010 | 238 |
| 1059637 | 1000049 | 19 |
| 1059637 | 1000056 | 123 |
| 1059637 | 1000062 | 111 |
| 1059637 | 1000094 | 12 |
| 1059637 | 1000112 | 423 |
| 1059637 | 1000113 | 53 |
| 1059637 | 1000114 | 23 |

**Fig. 3.** Data format

The main focus will be on this data file. Those fields will be convert into two-dimensional matrix space. However, the result of the matrix will be sparse because the user might not played all the artists. So, the task will be to make this matrix dense in order to find the implicit data for the users who do not played other artists.

**Artist_data_small.txt:** It contains two features ArtistId and Name as shown in the Fig 4. It provides names of each artist by their ID's.

| ArtistID | Name |
|---|---|
| 1240105 | André Visior |
| 1240113 | riow arai |
| 1240132 | Outkast & Rage Against the Machine |
| 1030848 | Raver's Nature |
| 6671601 | Erguner, Kudsi |
| 1106617 | Bloque |
| 1240185 | Lexy & K. Paul |

**Fig. 4.** Data format

**Artist_alias_small.txt:** When plays are scrobbled, the client application registers the name of the artist being played. The name could be nonstandard or it could be misspelled. For example, “The Bloque”, “Bloque, and “the bloque” may appear as distinct artist IDs in the dataset, even though they are plainly the same. Artist_alias_small.txt maps artist IDs that are misspellings or variants to the canonical ID of that artist. This data file contains two features MisspelledArtistID and StandardArtistID.

### 4.2 Matrix notation

Our experiment will focus on the artist and the user entity. Artist data and user play counts are available in different data formats. Each file consists of fields that were interrelated and can be extracted to form a matrix notation that will be used to make recommendation system.

**Table 1.** Matrix notation

| User/Artist | ArtistId1 | ArtistId2 | ArtistId3 | ArtistId4 | ArtistIdn |
|---|---|---|---|---|---|
| UserId1 | 2 | - | - | 1 | 5 |
| UserId2 | 7 | 4 | 5 | - | - |
| UserId3 | - | 6 | 9 | 5 | - |
| UserIdn | - | 3 | 7 | - | 4 |

Table 1. Shows the matrix notation that will be used in this recommendation system. Where, rows represented the number users and columns represented number of artists. Cells of the table represented the play counts that how many times user listened to each artist. The matrix dimension is the number of users multiple number of artists. The missing values from the matrix can be find by using matrix factorization.

### 4.3 Matrix Factorization

Matrix factorization is a popular approach for matrix completion problem, where it summarize users and items with their latent factors. The recommendation is based on high correspondence between user and item latent factors. The recommendation of this paper is rely on implicit feedback data. This type of data usually denotes the presence and absence of an event. In context with music data, it depends on whether a user listened to an artist or not. Moreover, if listened then how many times. The indexing letters used in this paper for user and item are mentioned in section 2. For implicit dataset, the values contained by $r_{ui}$ indicate observation for user actions [3]. In our music recommender system, $r_{ui}$ indicates how many times user, u listened to artist i. For example if the value of $r_{ui}$ is 3 than it means that the user, u played artist, i three times. There will have different confidence levels among items that are indicated to be preferred by the user. In general, as $r_{ui}$ grows, it gives strong indication that user likes that item. Consequently, a set of variables $c_{ui}$ is introduced to measure confidence in observing $p_{ui}$ [3].

$$c_{ui} = 1+\alpha\, r_{ui} \tag{1}$$

Where, $\alpha$ is a parameter applicable to the implicit feedback variant matrix factorization that governs baseline confidence in preference observations. And

$$p_{ui} = \begin{cases} 1 \text{ if } r_{ui} > 0 \\ 0 \text{ if } r_{ui} = 0 \end{cases}$$

The first goal is to find the latent factor vector $x_u \in \mathbf{R}^f$ for each user u and latent factor vector $y_i \in \mathbf{R}^f$ for each user i. Then to predict user u's value for item i, we do as follows

$$p_{ui} \sim x^T_u y_i \tag{2}$$

These vectors will be known as user factors and the item factors. The vectors strive to map users and items into a common latent factor space where they can be directly compared [3]. There are two things to be different from explicit data matrix factorization.

1. We need to account for varying confidence levels.
2. Optimization should be account for all possible u,i pairs, rather than corresponding to observed data.

In general, the latent factor vectors are calculated to minimize the following cost function (root mean square error)

$$\text{Min} \sum c_{ui} ( p_{ui} - x^T_u y_i )^2 + \lambda \left( \sum || x_u ||^2 + \sum || y_i ||^2 \right) \tag{3}$$

u,i ∈ observed interactions.

Where, $x_u$ = user u's latent factor vector

$y_u$ = item i's latent factor vector

$\lambda$ = regularization parameter , it is data dependent value.

The $\lambda \left( \sum || x_u ||^2 + \sum || y_i ||^2 \right)$ term is necessary for regularizing the model such that it will not overfit the training data.

**Alternating least square (ALS)**

The approach for ALS is to fix latent factor vector $y_i$ and optimize $x_u$ latent factor vector then to fix $x_u$ and optimize $y_u$, and repeat until convergence.

*Pseudocode for ALS:*

Initialize $x_u$ , $y_i$
While (convergence is not true) do
For u = 1 ….. n, do
$x_u = ( Y^T C^u Y + \lambda I )^{-1} Y^T C^u p(u)$
end for
For u = 1….. n, do
$y_i = ( X^T C^i X + \lambda I )^{-1} X^T C^i p(i)$
end for
end while

where, $C^u$ is the n x n diagonal matrix for each user.

$p(u) \in \mathbf{R}^n$ contains all preferences for u .
Y is the n x f matrix, X is the m x f matrix.
$C^i$ is the m x m diagonal matrix for each item.
$p(i) \in \mathbf{R}^i$ contains all preference for i.
I is the identity matrix

For single machine computational cost for ALS algorithm , updating each $x_u$ will cost $O( n_u f^2 + f^3 )$ where $n_u$ is the number of artists listened by user, u. and similarly updating each $y_i$ will cost $O( n_i f^2 + f^3 )$. [3]

### 4.4 Explaining recommendations

A model is evaluated from the output of ALS algorithm. Where bestModel is used to compute recommendations. It will be explained in section 5.2.

## 5 Implementation

In this section, the system specifications will be discussed along with the code diagrams.

### 5.1 System specification

**Hardware used:**

- Intel i5 processor
- 4 GB RAM
- 750 GB Hard Disk

**Software used:**

- Apache Spark version -2.0.0-bin-hadoop2.6
- Jupyter notebook (anaconda version 4.3.1)
- Python 2.7
- Microsoft Windows 10

**Language used:** Python along with spark libraries.

### 5.2 Code diagrams with description

**Loading all three datasets**
First of all, code is for loading three data files userArtistData, artistAlias, and artistData into RDD's. Where RDD is Resilient Distributed Datasets which is a fundamental data structure of Spark.

```
In [2]: artistData = sc.textFile("artist_data_small.txt")
        artistAlias = sc.textFile("artist_alias_small.txt")
        userArtistData = sc.textFile("user_artist_data_small.txt")

        alias_data = artistAlias.collect()
        user_data = userArtistData.collect()
        artist_canonical_dict = {}
        user_list = []

        for line in alias_data:
            artist_record = line.split("\t")
            artist_canonical_dict[artist_record[0]] = artist_record[1]

        #Function to get canonical artist names
        def canonicalArtistID(line):
            line = line.split(" ")

            if line[1] in artist_canonical_dict:
                return (int(line[0]),int(artist_canonical_dict[line[1]]),int(line[2]))
            else:
                return (int(line[0]),int(line[1]),int(line[2]))

        #Getting canonical artist names
        userArtistData = userArtistData.map(canonicalArtistID)

        #Creating allArtists dataset to be used later during model evaluation process
        allArtists = userArtistData.map(lambda x:(x[1])).collect()
        allArtists = list(set(allArtists))
```

**Fig. 5.** Loading Datasets

Fig 5. Shows the code for loading datasets. Where The userArtistData RDD will only contains the canonical artist IDs.

**Splitting data into training, validation and test data**

The data will be split in such a way that training and validation data will consist of 40% each of the overall data, where test data will contain 20%. The trainData will be used to train the model. validationData will be used to perform parameter tuning and testData is used for a final evaluation of the model.

```
In [9]: trainData, validationData, testData = userArtistData.randomSplit([4, 4, 2], 13)

        print 'five instances from training data: ' + str(trainData.take(5))
        print 'five instances from validation data: ' + str(validationData.take(5))
        print 'five instances from test data: ' + str(testData.take(5))
        print 'Total number of instances in training data: ' + str(trainData.count())
        print 'Total number of instances in validation data: ' + str(validationData.count())
        print 'Total number of instances in test data: ' + str(testData.count())

        #Caching and creating ratings object
        trainData = trainData.map(lambda l: Rating(*l)).cache()
        validationData = validationData.map(lambda l: Rating(*l)).cache()
        testData = testData.map(lambda l: Rating(*l)).cache()

        five instances from training data: [(1059637, 1000049, 1), (1059637, 1000056, 1), (1059637, 1000113, 5), (1059637, 10
        00114, 2), (1059637, 1000123, 2)]
        five instances from validation data: [(1059637, 1000010, 238), (1059637, 1000062, 11), (1059637, 1000112, 423), (1059
        637, 1000289, 2), (1059637, 1000320, 21)]
        five instances from test data: [(1059637, 1000094, 1), (1059637, 1000130, 19129), (1059637, 1000139, 4), (1059637, 10
        00241, 188), (1059637, 1000263, 180)]
        Total number of instances in training data: 19817
        Total number of instances in validation data: 19633
        Total number of instances in test data: 10031
```

**Fig. 6.** Splitting data

Fig 6. Shows that the data split is done by the randomSplit function of spark. Where the userArtistData is divided into trainData (40%), ValidationData (40%) and testData (20%). In the output section, five instances of each data is printed along with instances count.

**Recommender model evaluation**

The method used for model evaluation is simple. To understand the method, suppose we have a model and some dataset of true artist plays for a set of users. This model can be used to predict the top X artist recommendations for a user and the recommendations coming from this model can be compared to the artists that user actually listened to. After that, the fraction of overlap between the top X predictions pf the model and the X artists that the user actually listened is to be calculated. This process will be repeated for all the users and average will be returned. For instance, assume a model predicted [2, 5, 4, 9] as the top X artists for a user. Assume, that user actually listened to the artist [ 2,6,7,9]. For this model, the model score will be 2/4= 0.5. This will be perform for all users to get the overall score.

[1]

[1] https://github.com/ParinSanghavi/Music-Recommender-System-using-Apache-Spark-and-Python/blob/master/prsangha_recommender.ipynb

```
In [10]: from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
         from collections import defaultdict

         #model evaluation function
         def modelEval(model, dataset):
             global trainData
             global allArtists

             #Getting nonTrainArtists for each user
             userArtists = defaultdict(list)

             for data in trainData.collect():
                 userArtists[data[0]].append(data[1])

             cvList = []

             for key in userArtists.keys():
                 userArtists[key] = list(set(allArtists) - set(userArtists[key]))
                 for artist in userArtists[key]:
                     cvList.append((key, artist))

             #Creating user,nonTrainArtists RDD
             cvData = sc.parallelize(cvList)

             userOriginal = dataset.map(lambda x:(x.user, (x.product, x.rating))).groupByKey().collect()

             #prediction on the user, nonTrainArtists RDD
             predictions = model.predictAll(cvData)
             userPredictions = predictions.map(lambda x:(x.user, (x.product, x.rating))).groupByKey().collect()
             original = {}
             predictions = {}
```

**Fig. 7.** Model evaluation

Fig 7. Shows that necessary packages are imported from the spark libraries. Packages such as ALS, MatrixFactorizationModel and Rating. The model function is named to modelEval, And it is taking input from the ALS matrix factorization. For parameter tuning, validationData parameter is used. After that, model can be evaluated using test-Data.

**Model construction**

Fig 8 shows the model build using validationData and modelEval function. The output showing the model score for rank 2, 10 and 20. Where, rank is the number of latent factors calculated using ALS. In the 2nd output the bestModel is computed using test-Data.

```
In [11]: rank_list = [2, 10, 20]

         for rank in rank_list:
             model = ALS.trainImplicit(trainData, rank, seed=345)
             modelEval(model,validationData)

         The model score for rank 2 is 0.0939183331552
         The model score for rank 10 is 0.0982729214134
         The model score for rank 20 is 0.0843559301313

In [12]: bestModel = ALS.trainImplicit(trainData, rank=10, seed=345)
         modelEval(bestModel, testData)

         The model score for rank 20 is 0.0611734435121
```

**Fig. 8.** Model construction

# 6 Experiments

## 6.1 Experiment 1

We are going to use the bestModel, which we calculated above section to predict the top 5 artists for 2 different users using recommendProducts function and by mapping the results into the real name using artistAlias.

```
The top five artists recommended to user with userID 1059637 are
0: blink-182
1: Elliott Smith
2: Taking Back Sunday
3: Incubus
4: Death Cab for Cutie
```

**Fig. 9.** Recommendation result1

Fig 9. Shows  top 5 artists recommended to userID 1059637

```
The top five artists recommended to user with userID 2020513 are
0: Rilo Kiley
1: Bishop Allen
2: Wilco
3: The Polyphonic Spree
4: Tilly and the Wall
```

**Fig. 10.** Recommendation result2

Fig 10. Shows top 5 artists recommended to userID 2020513

### 6.2 Experiment 2

In this experiment, we will try to evaluate the quality of our recommender model. User tend to play songs from artist who are more appealing. So, the plays for a user give a partial picture of what "good" and "bad" recommendations are. [10]. If recommender systems would evaluated on ability to rank good things above in a list. Than in context of our system problem is that "good" is defined as "artists the user has listened to" which recommender system received from the input dataset. It could trivially return the users previously-listened artists as top recommendations and score perfectly. This is not useful because recommender system must recommend artists that user never listened to. To avoid the above problem, we will divide data into 90% training data and 10% cross validation (cv) data. Then this cv data can be interpreted as a collection of good recommendations for each user. The recommender will then model the ranks for all the artists present in cv data. Recommender score can be calculated by comparing all artists rank from cv data to the rest of artists. Where the 1.0 is perfect score, 0.0 is the worst possible score and 0.5 is the expected value achieved from randomly ranking artists [10]. This metric is related to an information retrieval concept, called Receiver Operating Characteristic (ROC) curve.
This experiment will focus on to calculate area under the ROC curve (AUROC). Where, AUC is the probability that a randomly chosen good recommendation ranks above a randomly chosen bad recommendation. In order to compute AUC, the ALS model is trained on the training set only and the cv data is used to evaluate the model.

```
In [28]: auc = calculateAUC(validationData, bAllItemIDs, model.predictAll)
         t1 = time()
         print("The area under the curve is =",auc)

         The area under the curve is = 0.9632320255137113
```

**Fig. 11.** Evaluation model

Fig. 11 shows that a function is written to calculate AUC to evaluate the quality of the model. And the output is showing the calculated AUC.
The result is 0.9632 (approx.). It is clearly higher than the 0.5 that was expected from making recommendations randomly. It is close to maximum score 1.0. Therefore, AUC over 0.9 would be considered high.

## 7 Acknowledgement

There are few people who helped me to make this project. First of all, I would like to thank my professor Dr. Sheela Ramanna who gave me Opportunity to work on this project. Secondly, I would like to thank Parin Sanghavi whose code I used to implement this project. He gave me constant support and guidance throughout this project.

## 8 Conclusion

The matrix factorization using Alternating least square (ALS) learning method can be used to recommend music to users from implicit feedback data. The Apache-Spark with Python can be used for constructing recommender systems as we can use various machine learning libraries provided by Spark. Moreover, the quality of the recommendation system can be evaluated by computing area under the curve. Recommendation systems in future can use ALS for learning latent factor vectors for ever increasing implicit feedback data.

## 9 References

1. J. Bennet and S. Lanning, "The Netflix Prize", KDD Cup and Workshop, 2007. www.netflixprize.com.
2. Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. Springer, 2011.
3. Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In Data Mining, 2008. ICDM'08.Eighth IEEE International Conference.
4. D.W. Oard and J. Kim, "Implicit Feedback for Recommender Systems", Proc. 5th DELOS Workshop on Filtering and Collaborative Filtering, pp. 31–36, 1998.
5. Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. Knowledge and Data Engineering, IEEE Transactions on, 17(6):734{749, 2005.
6. Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Contentbased recommender systems: State of the art and trends. In Recommender systems handbook, pages 73{105. Springer, 2011.
7. Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Computer, (8):30{37, 2009.
8. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl.Item-based collabrative filtering recommendation algorithms. In Proceedings of the 10th international conference on World Wide Web,pages 285{295. ACM, 2001.
9. Original Dataset : http://www-etud.iro.umontreal.ca/~bergstrj/audioscrobbler_data.html
10. https://books.google.ca/books?id=WE_GBwAAQBAJ&pg=PA40&lpg=PA40&dq=audioscrobbler+dataset&source=bl&ots=vCDdD8DZZZ&sig=fyCisF0Fl-9Z-3g5tot7kdZ8gVg&hl=en&sa=X&sqi=2&ved=0ahUKEwiIxvnWmfzSAhWr5IMKHX60Dd8Q6AEITzAJ#v=onepage&q&f=false