Recommendation System Using Deep Learning

Report submitted for the partial fulfillment of the requirements for the degree of Bachelor of Technology in

Computer Science and Engineering

Submitted by

Name and Roll Number

Tirthankar Ghosh(CSE2014/011)
11700114091
Subham Misra(CSE2014/008)
11700114080
Sourik Barman(CSE2014/006)
11700114071
Angshuman Midya(CSE2014/022)
11700114006

Under the Guidance of Mr. **Koushik Mallick (Assistant Professor(CSE),RCCIIT)**

श्रमम् बिना न किमपि साध्यम्

RCC Institute of Information Technology
Canal South Road, Beliaghata, Kolkata – 700 015
[Affiliated to West Bengal University of Technology]

Acknowledgement

We would like to express our sincere gratitude to Mr. Koushik Mallick of the department of Computer Science and Engineering, whose role as project guide was invaluable for the project. We are extremely thankful for the keen interest he took in advising us, for the books and reference materials provided for the moral support extended to us.

Last but not the least we convey our gratitude to all the teachers for providing us the technical skill that will always remain as our asset and to all non-teaching staff for the gracious hospitality they offered us.

Place: RCCIIT, Kolkata

Date: 16.05.2018

Tirthankar Ghosh

Subham Misra

Sourik Barman

Angshuman Midya

Department of Computer Science and Engineering
RCCIIT, Beliaghata,
Kolkata – 700 015,
West Bengal, India

Approval

This is to certify that the project report entitled "**Recommendation System Using Deep Learning"** prepared under my supervision by *Tirthankar Ghosh*, *Subham Misra*, *Sourik Barman and Angshuman Midya* be accepted in partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering.

It is to be understood that by this approval, the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn thereof, but approves the report only.

………………………………………..	… ………………………………………
Dr.Siddhartha Bhattacharyya.	Mr.Koushik Mallick
HOD,CSE	Assistant Professor
RCC Institute of Information Technology	RCC Institute of Information Technology

INDEX:

Introduction:

The explosive growth in the amount of available digital information and the number of visitors to the Internet have created a potential challenge of information overload which hinders timely access to items of interest on the Internet. Information retrieval systems, such as Google, DevilFinder and Altavista have partially solved this problem but prioritization and personalization (where a system maps available content to user's interests and preferences) of information were absent. This has increased the demand for recommender systems more than ever before. Recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of dynamically generated information according to user's preferences, interest, or observed behavior about item . Recommender system has the ability to predict whether a particular user would prefer an item or not based on the user's profile.

Recommender systems are beneficial to both service providers and users . They reduce transaction costs of finding and selecting items in an online shopping environment . Recommendation systems have also proved to improve decision making process and quality . In e-commerce setting, recommender systems enhance revenues, for the fact that they are effective means of selling more products. In scientific libraries, recommender systems support users by allowing them to move beyond catalog searches. Therefore, the need to use efficient and accurate recommendation techniques within a system that will provide relevant and dependable recommendations for users cannot be over-emphasized.

Literature Review:

Recommender system is defined as a decision making strategy for users under complex information environments . Also, recommender system was defined from the perspective of E-commerce as a tool that helps users search through records of knowledge which is related to users' interest and preference . Recommender system was defined as a means of assisting and augmenting the social process of using recommendations of others to make choices when there is no sufficient personal knowledge or experience of the alternatives . Recommender systems handle the problem of information overload that users normally encounter by providing them with personalized, exclusive content and service recommendations. Recently, various approaches for building recommendation systems have been developed, which can utilize either collaborative filtering, content-based filtering or hybrid filtering . Collaborative filtering technique is the most mature and the most commonly implemented. Collaborative filtering recommends items by identifying other users with similar taste; it uses their opinion to recommend items to the active user. Collaborative recommender systems have been implemented in different application areas. GroupLens is a news-based architecture which employed collaborative methods in assisting users to locate articles from massive news database . Ringo is an online social information filtering system that uses collaborative filtering to build users profile based on their ratings on music albums . Amazon uses topic diversification algorithms to improve its recommendation . The system uses collaborative filtering method to overcome scalability issue by generating a table of similar items offline through the use of item-to-item matrix. The system then recommends other products which are similar online according to the users' purchase history. On the other hand, content-based techniques match content resources to user characteristics. Content-based filtering techniques normally base their predictions on user's information, and they ignore contributions from other users as with the case of collaborative techniques . Fab relies heavily on the ratings of different users in order to create a training set and it is an example of content-based recommender system. Some other systems that use content-based filtering to help users find information on the Internet include Letizia . The system makes use of a user interface that assists users in browsing the Internet; it is able to track the browsing pattern of a user to predict the pages that they may be interested in. Pazzani et al. designed an intelligent agent that attempts to predict which web pages will interest a user by using naive Bayesian classifier. The agent allows a user to provide training instances by rating different pages as either hot or cold. Jennings and Higuchi describe a neural network that models the interests of a user in a Usenet news environment.

Despite the success of these two filtering techniques, several limitations have been identified. Some of the problems associated with content-based filtering techniques are limited content analysis, overspecialization and sparsity of data . Also, collaborative approaches exhibit cold-start, sparsity and scalability problems. These problems usually reduce the quality of recommendations. In order to mitigate some of the problems identified, Hybrid filtering, which combines two or more filtering techniques in different ways in order to increase the accuracy and performance of recommender systems has been proposed . These techniques combine two or more filtering approaches in order to harness their strengths while leveling out their corresponding weaknesses . They can be classified based on their operations into weighted hybrid, mixed hybrid, switching hybrid, feature-combination hybrid, cascade hybrid, feature-augmented hybrid and meta-level hybrid . Collaborative filtering and content-based filtering approaches are widely used today by implementing content-based and collaborative techniques differently and the results of their prediction later combined or adding the characteristics of content-based to collaborative filtering and vice versa. Finally, a general unified model which incorporates both content-based and collaborative filtering properties could be developed . The problem of sparsity of data and cold-start was addressed by combining the ratings, features and demographic information about items in a cascade hybrid recommendation technique in . In Ziegler et al. , a hybrid collaborative filtering approach was proposed to exploit bulk taxonomic information designed for exacting product classification to address the data sparsity problem of CF recommendations, based on the generation of profiles via inference of super-topic score and topic diversification. A hybrid recommendation technique is also proposed in Ghazantar and Pragel-Benett , and this uses the content-based profile of individual user to find similar users which are used to make predictions. In Sarwar et al. , collaborative filtering was combined with an information filtering agent. Here, the authors proposed a framework for integrating the content-based filtering agents and collaborative filtering. A hybrid recommender algorithm is employed by many applications as a result of new user problem of content-based filtering techniques and average user problem of collaborative filtering . A simple and straightforward method for combining content-based and collaborative filtering was proposed by Cunningham et al. . A music recommendation system which combined tagging information, play counts and social relations was proposed in Konstas et al. . In order to determine the number of neighbors that can be automatically connected on a social platform, Lee and Brusilovsky embedded social information into collaborative filtering algorithm. A Bayesian mixed-effects model that integrates user ratings, user and item features in a single unified framework was proposed by Condiff et al..

Objective:

On the Internet, where the number of choices is overwhelming, there is need to filter, prioritize and efficiently deliver relevant information in order to alleviate the problem of information overload, which has created a potential problem to many Internet users. Recommender systems solve this problem by searching through large volume of dynamically generated information to provide users with personalized content and services. This paper explores the different characteristics and potentials of different prediction techniques in recommendation systems in order to serve as a compass for research and practice in the field of recommendation systems.

System Design:

Phases of recommendation process

1. Information collection phase:

This collects relevant information of users to generate a user profile or model for the prediction tasks including user's attribute, behaviors or content of the resources the user accesses. A recommendation agent cannot function accurately until the user profile/model has been well constructed. The system needs to know as much as possible from the user in order to provide reasonable recommendation right from the onset. Recommender systems rely on different types of input such as the most convenient high quality explicit feedback, which includes explicit input by users regarding their interest in item or implicit feedback by inferring user preferences indirectly through observing user behavior . Hybrid feedback can also be obtained through the combination of both explicit and implicit feedback. In E-learning platform, a user profile is a collection of personal information associated with a specific user. This information includes cognitive skills, intellectual abilities, learning styles, interest, preferences and interaction with the system. The user profile is normally used to retrieve the needed information to build up a model of the user. Thus, a user profile describes a simple user model. The success of any recommendation system depends largely on its ability to represent user's current interests. Accurate models are indispensable for obtaining relevant and accurate recommendations from any prediction techniques.

2. Explicit feedback:

The system normally prompts the user through the system interface to provide ratings for items in order to construct and improve his model. The accuracy of recommendation depends on the quantity of ratings provided by the user. The only shortcoming of this method is, it requires effort from the users and also, users are not always ready to supply enough information. Despite the fact that explicit feedback requires more effort from user, it is still seen as providing more reliable data, since it does not involve extracting preferences from actions, and it also provides transparency into the recommendation process that results in a slightly higher perceived recommendation quality and more confidence in the recommendations .

3. Implicit feedback:

The system automatically infers the user's preferences by monitoring the different actions of users such as the history of purchases, navigation history, and time spent on some web pages, links followed by the user, content of e-mail and button clicks among others. Implicit feedback reduces the burden on users by inferring their user's preferences from their behavior with the system. The method though does not require effort from the user, but it is less accurate. Also, it has also been argued that implicit preference data might in actuality be more objective, as there is no bias arising from users responding in a socially desirable way and there are no self-image issues or any need for maintaining an image for others .

4. Hybrid feedback:

The strengths of both implicit and explicit feedback can be combined in a hybrid system in order to minimize their weaknesses and get a best performing system. This can be achieved by using an implicit data as a check on explicit rating or allowing user to give explicit feedback only when he chooses to express explicit interest.

5. Learning phase:

It applies a learning algorithm to filter and exploit the user's features from the feedback gathered in information collection phase.

6. Prediction/recommendation phase:

It recommends or predicts what kind of items the user may prefer. This can be made either directly based on the dataset collected in information collection phase which could be memory based or model based or through the system's observed activities of the user.

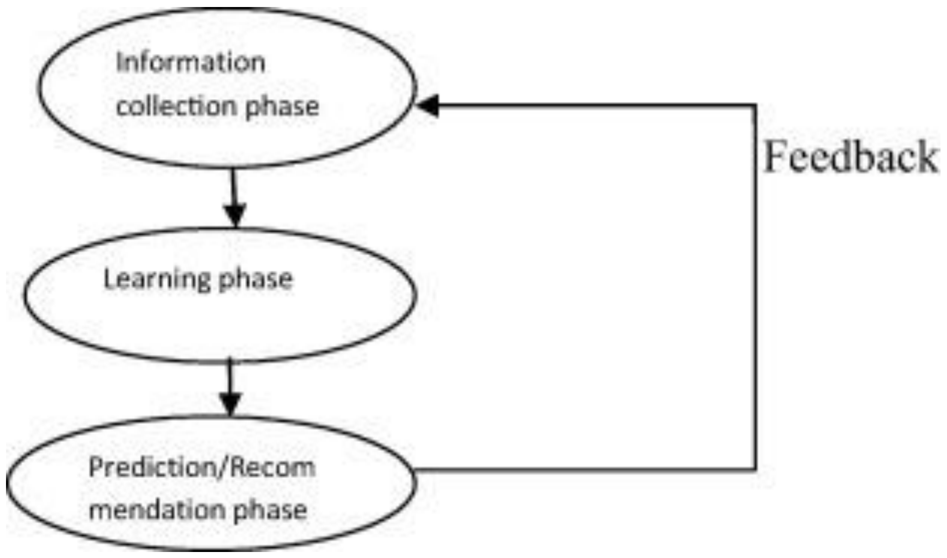


Fig:1

Recommendation filtering techniques:

The use of efficient and accurate recommendation techniques is very important for a system that will provide good and useful recommendation to its individual users. This explains the importance of understanding the features and potentials of different recommendation techniques. Below figure shows the anatomy of different recommendation filtering techniques.

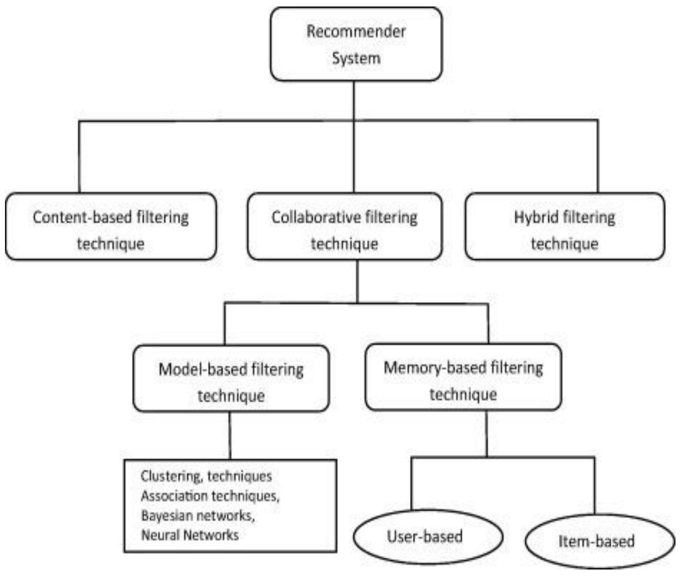

Recommender System
Content-based filtering technique
Collaborative filtering technique
Hybrid filtering technique
Model-based filtering technique
Memory-based filtering technique
Clustering, techniques Association techniques, Bayesian networks, Neural Networks
User-based
Item-based

Fig:2

1. **Content-based filtering:**

Content-based technique is a domain-dependent algorithm and it emphasizes more on the analysis of the attributes of items in order to generate predictions. When documents such as web pages, publications and news are to be recommended, content-based filtering technique is the most successful. In content-based filtering technique, recommendation is made based on the user profiles using features extracted from the content of the items the user has evaluated in the past . Items that are mostly related to the positively rated items are recommended to the user. CBF uses different types of models to find similarity between documents in order to generate meaningful recommendations. It could use Vector Space Model such as Term Frequency Inverse Document Frequency (TF/IDF) or Probabilistic models such as Naïve Bayes Classifier , Decision Trees or Neural Networks to model the relationship between different documents within a corpus. These techniques make recommendations by learning the underlying model with either statistical analysis or machine learning techniques. Content-based filtering technique does not need the profile of other users since they do not influence recommendation. Also, if the user profile changes, CBF technique still has the potential to adjust its recommendations within a very short period of time. The major disadvantage of this technique is the need to have an in-depth knowledge and description of the features of the items in the profile.

Pros and Cons of content-based filtering techniques:

CB filtering techniques overcome the challenges of CF. They have the ability to recommend new items even if there are no ratings provided by users. So even if the database does not contain user preferences, recommendation accuracy is not affected. Also, if the user preferences change, it has the capacity to adjust its recommendations in a short span of time. They can manage situations where different users do not share the same items, but only identical items according to their intrinsic features. Users can get recommendations without sharing their profile, and this ensures privacy . CBF technique can also provide explanations on how recommendations are generated to users. However, the techniques suffer from various problems as discussed in the literature . Content based filtering techniques are dependent on items' metadata. That is, they require rich description of items and very well organized user profile before recommendation can be made to users. This is called limited content analysis. So, the effectiveness of CBF depends on the availability of descriptive data. Content overspecialization is another serious problem of CBF technique. Users are restricted to getting recommendations similar to items already defined in their profiles.

Examples of content-based filtering systems:

News Dude is a personal news system that utilizes synthesized speech to read news stories to users. TF-IDF model is used to describe news stories in order to determine the short-term recommendations which is then compared with the Cosine Similarity Measure and finally supplied to a learning algorithm (NN). CiteSeer is an automatic citation indexing that uses various heuristics and machine learning algorithms to process documents. Today, CiteSeer is among the largest and widely used research paper repository on the web.

LIBRA is a content-based book recommendation system that uses information about book gathered from the Web. It implements a Naïve Bayes classifier on the information extracted from the web to learn a user profile to produce a ranked list of titles based on training examples supplied by an individual user. The system is able to provide explanation on any recommendations made to users by listing the features that contribute to the highest ratings and hence allowing the users to have total confidence on the recommendations provided to users by the system.

2. Collaborative filtering:

Collaborative filtering is a domain-independent prediction technique for content that cannot easily and adequately be described by metadata such as movies and music. Collaborative filtering technique works by building a database (user-item matrix) of preferences for items by users. It then matches users with relevant interest and preferences by calculating similarities between their profiles to make recommendations . Such users build a group called neighborhood. An user gets recommendations to those items that he has not rated before but that were already positively rated by users in his neighborhood. Recommendations that are produced by CF can be of either prediction or recommendation. Prediction is a numerical value, Rij, expressing the predicted score of item j for the user i, while Recommendation is a list of top N items that the user will like the most as shown in below figure. The technique of collaborative filtering can be divided into two categories: memory-based and model-based .

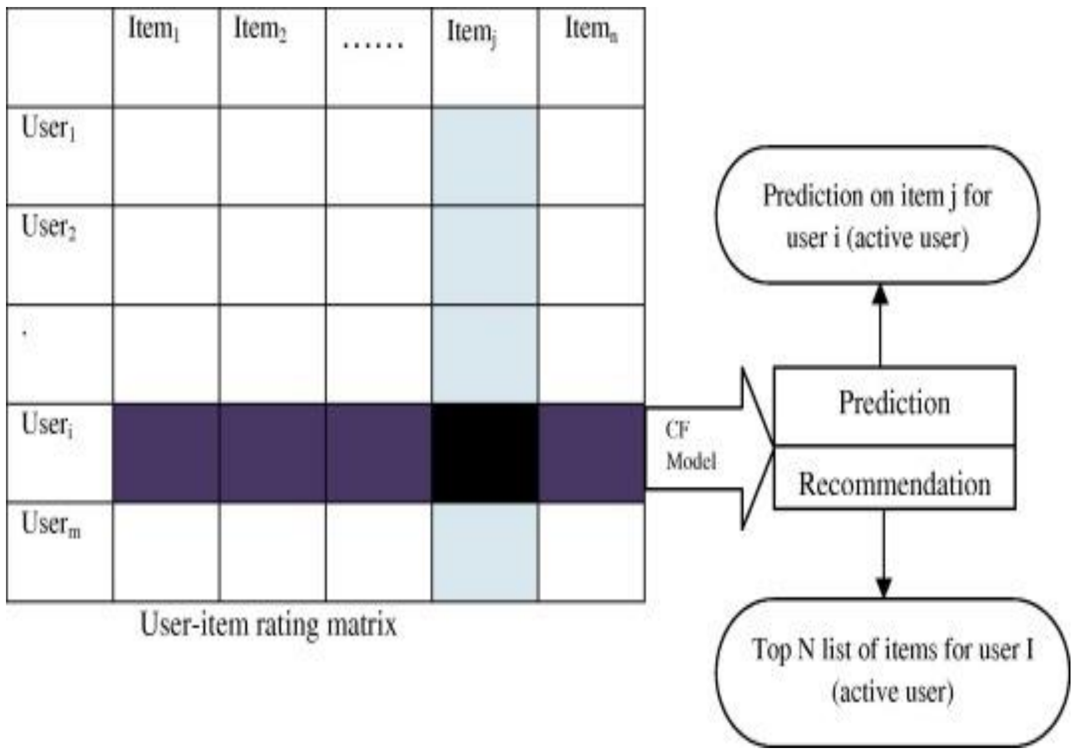


Fig:3

2.1. Memory based techniques:

The items that were already rated by the user before play a relevant role in searching for a neighbor that shares appreciation with him. Once a neighbor of a user is found, different algorithms can be used to combine the preferences of neighbors to generate recommendations. Due to the effectiveness of these techniques, they have achieved widespread success in real life applications. Memory-based CF can be achieved in two ways through user-based and item-based techniques. User based collaborative filtering technique calculates similarity between users by comparing their ratings on the same item, and it then computes the predicted rating for an item by the active user as a weighted average of the ratings of the item by users similar to the active user where weights are the similarities of these users with the target item. Item-based filtering techniques compute predictions using the similarity between items and not the similarity between users. It builds a model of item similarities by retrieving all items rated by an active user from the user-item matrix, it determines how similar the retrieved items are to the target item, then it selects the k most similar items and their corresponding similarities are also determined. Prediction is made by taking a weighted average of the active users rating on the similar items k. Several types of similarity measures are used to compute similarity between item/user. The two most popular similarity measures are correlation-based and cosine-based. Pearson correlation coefficient is used to measure the extent to which two variables linearly relate with each other and is defined as

$$s(a,u) = \frac{\sum_{i=1}^{n} (r_{a,i} - \overline{r_a})(r_{u,i} - \overline{r_u})}{\sqrt{\sum_{i=1}^{n} (r_{a,i} - \overline{r_a})^2} \sqrt{\sum_{i=1}^{n} (r_{u,i} - \overline{r_u})^2}}$$

From the above equation,

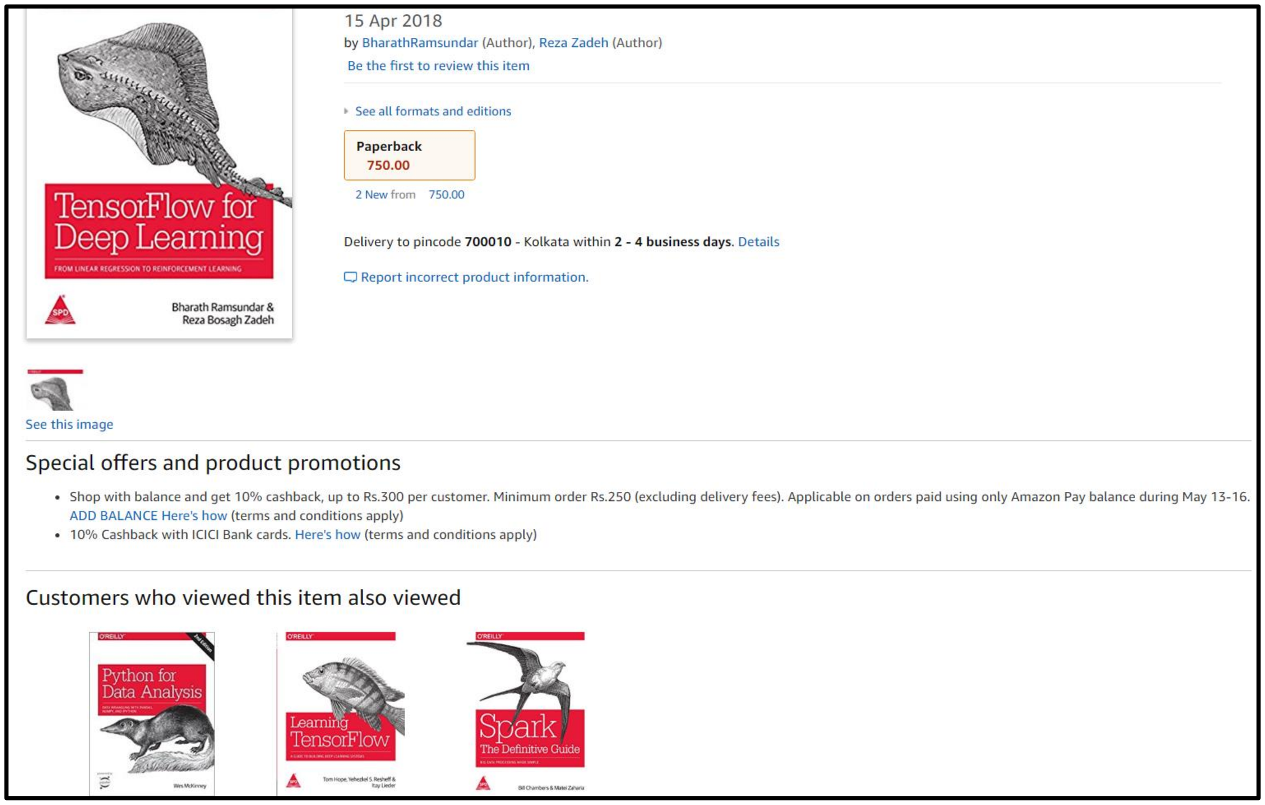
S(a,u)is the mean rating given by user a while n is the total number of items in the user-item space. Also, prediction for an item is made from the weighted combination of the selected neighbors' ratings, which is computed as the weighted deviation from the neighbors' mean. The general prediction formula is

$$p(a,i) = \overline{r_a} + \frac{\sum_{i=1}^{n} (r_{u,i} - \overline{r_u}) \times s(a,u)}{\sum_{i=1}^{n} s(a,u)}$$

Cosine similarity is different from Pearson-based measure in that it is a vector-space model which is based on linear algebra rather that statistical approach. It measures the similarity between two n-dimensional vectors based on the angle between them. Cosine-based measure is widely used in the fields of information retrieval and texts mining to compare two text documents, in this case, documents are represented as vectors of terms. The similarity between two items u and v can be defined as

$$s(\overrightarrow{u}, \overrightarrow{v}) = \frac{\overrightarrow{u} \cdot \overrightarrow{v}}{|\overrightarrow{u}| * |\overrightarrow{v}|} = \frac{\sum_i r_{u,i} r_{v,i}}{\sqrt{\sum_i r_{u,i}^2} \times \sqrt{\sum_i r_{v,i}^2}}$$

Similarity measure is also referred to as similarity metric, and they are methods used to calculate the scores that express how similar users or items are to each other. These scores can then be used as the foundation of user- or item-based recommendation generation. Depending on the context of use, similarity metrics can also be referred to as correlation metrics or distance metrics.



15 Apr 2018

by BharathRamsundar (Author), Reza Zadeh (Author)

Be the first to review this item

See all formats and editions

Paperback
750.00

2 New from 750.00

Delivery to pincode **700010** - Kolkata within **2 - 4 business days**. Details

Report incorrect product information.

See this image

Special offers and product promotions

- Shop with balance and get 10% cashback, up to Rs.300 per customer. Minimum order Rs.250 (excluding delivery fees). Applicable on orders paid using only Amazon Pay balance during May 13-16. ADD BALANCE Here's how (terms and conditions apply)
- 10% Cashback with ICICI Bank cards. Here's how (terms and conditions apply)

Customers who viewed this item also viewed

TensorFlow for Deep Learning
FROM LINEAR REGRESSION TO REINFORCEMENT LEARNING
Bharath Ramsundar & Reza Bosagh Zadeh
Python for Data Analysis
Learning TensorFlow
Spark The Definitive Guide

Fig:4

2.2. Model-based techniques:

This technique employs the previous ratings to learn a model in order to improve the performance of Collaborative filtering Technique. The model building process can be done using machine learning or data mining techniques. These techniques can quickly recommend a set of items for the fact that they use pre-computed model and they have proved to produce recommendation results that are similar to neighborhood-based recommender techniques. Examples of these techniques include Dimensionality Reduction technique such as Singular Value Decomposition (SVD), Matrix Completion Technique, Latent Semantic methods, and Regression and Clustering. Model-based techniques analyze the user-item matrix to identify relations between items; they use these relations to compare the list of top-N recommendations. Model based techniques resolve the sparsity problems associated with recommendation systems.

The use of learning algorithms has also changed the manner of recommendations from recommending what to consume by users to recommending when to actually consume a product. It is therefore very important to examine other learning algorithms used in model-based recommender systems:

Association rule: Association rules mining algorithms extract rules that predict the occurrence of an item based on the presence of other items in a transaction. For instance, given a set of transactions, where each transaction is a set of items, an association rule applies the form A → B, where A and B are two sets of items . Association rules can form a very compact representation of preference data that may improve efficiency of storage as well as performance. Also, the effectiveness of association rule for uncovering patterns and driving personalized marketing decisions has been known for sometimes . However, there is a clear relation between this method and the goal of a Recommendation System but they have not become mainstream.

Clustering: Clustering techniques have been applied in different domains such as, pattern recognition, image processing, statistical data analysis and knowledge discovery. Clustering algorithm tries to partition a set of data into a set of sub-clusters in order to discover meaningful groups that exist within them. Once clusters have been formed, the opinions of other users in a cluster can be averaged and used to make recommendations for individual users. A good clustering method will produce high quality clusters in which the intra-cluster similarity is high, while the inter-cluster similarity is low. In some clustering approaches, a user can have partial participation in different clusters, and recommendations are then based on the average across the clusters of participation which is weighted by degree of participation. K-means and Self-Organizing Map (SOM) are the most commonly used among the different clustering methods. K-means takes an input parameter, and then partitions a set of n items into K clusters . The Self-Organizing Map (SOM) is a method for an unsupervised learning, based on artificial neurons clustering technique. Clustering techniques can be used to reduce the candidate set in collaborative-based algorithms.

Decision tree: Decision tree is based on the methodology of tree graphs which is constructed by analyzing a set of training examples for which the class labels are known. They are then applied to classify previously unseen examples. If trained on very high quality data, they have the ability to make very accurate predictions. Decision trees are more interpretable than other classifier such as Support Vector machine (SVM) and Neural Networks because they combine simple questions about data in an understandable manner. Decision trees are also flexible in handling items with mixture of real-valued and categorical features as well as items that have some specific missing features.

Artificial Neural network: ANN is a structure of many connected neurons (nodes) which are arranged in layers in systematic ways. The connections between neurons have weights associated with them depending on the amount of influence one neuron has on another. There are some advantages in using neural networks in some special problem situations. For example, due to the fact that it contains many neurons and also assigned weight to each connection, an artificial neural network is quite robust with respect to noisy and erroneous data sets. ANN has the ability of estimating nonlinear functions and capturing complex relationships in data sets also, they can be efficient and even operate if part of the network fails. The major disadvantage is that it is hard to come up with the ideal network topology for a given problem and once the topology is decided this will act as a lower bound for the classification error.

Link analysis: Link Analysis is the process of building up networks of interconnected objects in order to explore pattern and trends. It has presented great potentials in improving the accomplishment of web search. Link analysis consists of PageRank and HITS algorithms. Most link analysis algorithms handle a web page as a single node in the web graph.

Regression: Regression analysis is used when two or more variables are thought to be systematically connected by a linear relationship. It is a powerful and diversity process for analyzing associative relationships between dependent variable and one or more independent variables. Uses of regression contain curve fitting, prediction, and testing systematic hypotheses about relationships between variables. The curve can be useful to identify a trend within dataset, whether it is linear, parabolic, or of some other forms.

Bayesian Classifiers: They are probabilistic framework for solving classification problems which is based on the definition of conditional probability and Bayes theorem. Bayesian classifiers consider each attribute and class label as random variables. Given a record of N features (A1, A2, …, AN), the goal of the classifier is to predict class Ck by finding the value of Ck that maximizes the posterior probability of the class given the data P(Ck|A1, A2, …, AN) by applying Bayes' theorem, $P(Ck|A1, A2, \dots, AN) \propto P(A1, A2, \dots, AN|Ck)P(Ck)$. The most commonly used Bayesian classifier is known as the Naive Bayes Classifier. In order to estimate the conditional probability, P(A1, A2, …, AN|Ck), a Naive Bayes Classifier assumes the probabilistic independence of the attributes that is, the presence or absence of a particular attribute is unrelated to the presence or absence of any other. This assumption leads to P(A1, A2,

…, AN|Ck) = P(A1|Ck)P(A2|Ck)… P(AN|Ck). The main benefits of Naive Bayes classifiers are that they are robust to isolated noise points and irrelevant attributes, and they handle missing values by ignoring the instance during probability estimate calculations. However, the independence assumption may not hold for some attributes as they might be correlated. In this case, the usual approach is to use Bayesian Networks. Bayesian classifiers may prove practical for environments in which knowledge of user preferences changes slowly with respect to the time needed to build the model but are not suitable for environments in which users preference models must be updated rapidly or frequently. It is also successful in model-based recommendation systems because it is often used to derive a model for content-based recommendation systems.

Matrix completion techniques: The essence of matrix completion technique is to predict the unknown values within the user-item matrices. Correlation based K-nearest neighbor is one of the major techniques employed in collaborative filtering recommendation systems. They depend largely on the historical rating data of users on items. Most of the time, the rating matrix is always very big and sparse due to the fact that users do not rate most of the items represented within the matrix. This problem always leads to the inability of the system to give reliable and accurate recommendations to users. Different variations of low rank models have been used in practice for matrix completion especially toward application in collaborative filtering . Formally, the task of matrix completion technique is to estimate the entries of a matrix, $M \in Rm \times n$, when a subset, $\Omega C\{(i,j):1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n\}$

of the new entries is observed, a particular set of low rank matrices,

$M\hat{}=UVT$, where $U \in Rm \times k$ and $V \in Rm \times k$ and $k \ll \min(m,n)$

. The most widely used algorithm in practice for recovering M from partially observed matrix using low rank assumption is Alternating Least Square (ALS) minimization which involves optimizing over U and V in an alternating manner to minimize the square error over observed entries while keeping other factors fixed. Candes and Recht proposed the use of matrix completion technique in the Netflix problem as a practical example for the utilization of the technique. Keshavan et al. used SVD technique in an OptSpace algorithm to deal with matrix completion problem. The result of their experiment showed that SVD is able provide a reliable initial estimate for spanning subspace which can be further refined by gradient descent on a Grassmannian manifold. Model based techniques solve sparsity problem. The major drawback of the techniques is that the model building process is computationally expensive and the capacity of memory usage is highly intensive. Also, they do not alleviate the cold-start problem.

<u>Pros and Cons of collaborative filtering techniques:</u>

Collaborative Filtering has some major advantages over CBF in that it can perform in domains where there is not much content associated with items and where content is difficult for a computer system to analyze (such as opinions and ideal). Also, CF technique has the ability to provide serendipitous recommendations, which means that it can recommend items that are relevant to the user even without the content being in the user's profile. Despite the success of CF techniques, their widespread use has revealed some potential problems such as follows.

Cold-start problem:

This refers to a situation where a recommender does not have adequate information about a user or an item in order to make relevant predictions. This is one of the major problems that reduce the performance of recommendation system. The profile of such new user or item will be empty since he has not rated any item; hence, his taste is not known to the system.

Data sparsity problem:

This is the problem that occurs as a result of lack of enough information, that is, when only a few of the total number of items available in a database are rated by users. This always leads to a sparse user-item matrix, inability to locate successful neighbors and finally, the generation of weak recommendations. Also, data sparsity always leads to coverage problems, which is the percentage of items in the system that recommendations can be made .

Scalability:

This is another problem associated with recommendation algorithms because computation normally grows linearly with the number of users and items . A recommendation technique that is efficient when the number of dataset is limited may be unable to generate satisfactory number of recommendations when the volume of dataset is increased. Thus, it is crucial to apply recommendation techniques which are capable of scaling up in a successful manner as the number of dataset in a database increases. Methods used for solving scalability problem and speeding up recommendation generation are based on Dimensionality reduction techniques, such as Singular Value Decomposition (SVD) method, which has the ability to produce reliable and efficient recommendations.

Synonymy:

Synonymy is the tendency of very similar items to have different names or entries. Most recommender systems find it difficult to make distinction between closely related items such as the difference between e.g. baby wear and baby cloth. Collaborative Filtering systems usually find no match between the two terms to be able to compute their similarity. Different methods, such as automatic term expansion, the construction of a thesaurus, and Singular Value Decomposition (SVD), especially Latent Semantic Indexing are capable of solving the synonymy problem. The shortcoming of these methods is that some added terms may have different meanings from what is intended, which sometimes leads to rapid degradation of recommendation performance.

Examples of collaborative systems:

Ringo is a user-based CF system which makes recommendations of music albums and artists. In Ringo, when a user initially enters the system, a list of 125 artists is given to the user to rate according to how much he likes listening to them. The list is made up of two different sections. The first session consists of the most often rated artists, and this affords the active user opportunity to rate artists which others have equally rated, so that there is a level of similarities between different users' profiles. The second session is generated upon a random selection of items from the entire user-item matrix, so that all artists and albums are eventually rated at some point in the initial rating phases.

GroupLens is a CF system that is based on client/server architecture; the system recommends Usenet news which is a high volume discussion list service on the Internet. The short lifetime of Netnews, and the underlying sparsity of the rating matrices are the two main challenges addressed by this system. Users and Netnews are clustered based on the existing news groups in the system, and the implicit ratings are computed by measuring the time the users spend reading Netnews.

Amazon.com is an example of e-commerce recommendation engine that uses scalable item-to-item collaborative filtering techniques to recommend online products for different users. The computational algorithm scales independently of the number of users and items within the database. Amazon.com uses an explicit information collection technique to obtain information from users. The interface is made up of the following sections, your browsing history, rate these items, and improve your recommendations and your profile. The system predicts users interest based on the items he/she has rated. The system then compares the users browsing pattern on the system and decides the item of interest to recommend to the user. Amazon.com popularized feature of "people who bought this item also bought these items". Example of Amazon.com item-to-item contextual recommendation interface is shown in below figure

Algorithms and Mathematical Derivation:

RECOMMENDATION CLASS USING TEST TRAIN DATA:

ALGORITHM FOR POPULARITY BASED RECOMMENDATION

```
#Class for Popularity based Recommender System model

class popularity_recommender_py():

    def __init__(self):

        self.train_data = None

        self.user_id = None

        self.item_id = None

        self.popularity_recommendations = None


   #Create the popularity based recommender system model

    def create(self, train_data, user_id, item_id):

        self.train_data = train_data

        self.user_id = user_id

        self.item_id = item_id


 #Get a count of user_ids for each unique song as recommendation score

        train_data_grouped = train_data.groupby([self.item_id]).agg({self.user_id:
'count'}).reset_index()

        train_data_grouped.rename(columns = {'user_id': 'score'},inplace=True)
        #Sort the songs based upon recommendation score
    train_data_sort = train_data_grouped.sort_values(['score', self.item_id], ascending = [0,1])

    #Generate a recommendation rank based upon score
    train_data_sort['Rank'] = train_data_sort['score'].rank(ascending=0, method='first')

    #Get the top 10 recommendations
    self.popularity_recommendations = train_data_sort.head(10)

#Use the popularity based recommender system model to
#make recommendations
def recommend(self, user_id):
    user_recommendations = self.popularity_recommendations

    #Add user_id column for which the recommendations are being generated
    user_recommendations['user_id'] = user_id

    #Bring user_id column to the front
    cols = user_recommendations.columns.tolist()
    cols = cols[-1:] + cols[:-1]
    user_recommendations = user_recommendations[cols]

    return user_recommendations
```

```
#Class for Item similarity based Recommender System model
class item_similarity_recommender_py():
    def __init__(self):
        self.train_data = None
        self.user_id = None
        self.item_id = None
        self.cooccurence_matrix = None
        self.songs_dict = None
        self.rev_songs_dict = None
        self.item_similarity_recommendations = None

    #Get unique items (songs) corresponding to a given user
    def get_user_items(self, user):
        user_data = self.train_data[self.train_data[self.user_id] == user]
        user_items = list(user_data[self.item_id].unique())

        return user_items

    #Get unique users for a given item (song)
    def get_item_users(self, item):
        item_data = self.train_data[self.train_data[self.item_id] == item]
        item_users = set(item_data[self.user_id].unique())

        return item_users
```

```
#Get unique items (songs) in the training data
def get_all_items_train_data(self):
    all_items = list(self.train_data[self.item_id].unique())

    return all_items

#Construct cooccurence matrix
def construct_cooccurence_matrix(self, user_songs, all_songs):

    ####################################
    #Get users for all songs in user_songs.
    ####################################
    user_songs_users = [ ]
    for i in range(0, len(user_songs)):
        user_songs_users.append(self.get_item_users(user_songs[i]))

    ###############################################
    #Initialize the item cooccurence matrix of size
    #len(user_songs) X len(songs)
    ###############################################
    cooccurence_matrix = np.matrix(np.zeros(shape=(len(user_songs), len(all_songs))), float)

    #############################################################
    #Calculate similarity between user songs and all unique songs
```

```
#in the training data
###############################################################
for i in range(0,len(all_songs)):
    #Calculate unique listeners (users) of song (item) i
    songs_i_data = self.train_data[self.train_data[self.item_id] == all_songs[i]]
    users_i = set(songs_i_data[self.user_id].unique())

    for j in range(0,len(user_songs)):

        #Get unique listeners (users) of song (item) j
        users_j = user_songs_users[j]

        #Calculate intersection of listeners of songs i and j
        users_intersection = users_i.intersection(users_j)

        #Calculate cooccurence_matrix[i,j] as Jaccard Index
        if len(users_intersection) != 0:
            #Calculate union of listeners of songs i and j
            users_union = users_i.union(users_j)

            cooccurence_matrix[j,i] = float(len(users_intersection))/float(len(users_union))
        else:
            cooccurence_matrix[j,i] = 0
        return cooccurence_matrix



    #Use the cooccurence matrix to make top recommendations
    def generate_top_recommendations(self, user, cooccurence_matrix, all_songs, user_songs):
        print("Non zero values in cooccurence_matrix :%d" %
np.count_nonzero(cooccurence_matrix))

        #Calculate a weighted average of the scores in cooccurence matrix for all user songs.
        user_sim_scores = cooccurence_matrix.sum(axis=0)/float(cooccurence_matrix.shape[0])
        user_sim_scores = np.array(user_sim_scores)[0].tolist()

        #Sort the indices of user_sim_scores based upon their value
        #Also maintain the corresponding score
        sort_index = sorted(((e,i) for i,e in enumerate(list(user_sim_scores))), reverse=True)

        #Create a dataframe from the following
        columns = ['user_id', 'song', 'score', 'rank']
        #index = np.arange(1) # array of numbers for the number of samples
        df = pandas.DataFrame(columns=columns)

        #Fill the dataframe with top 10 item based recommendations
        rank = 1
        for i in range(0,len(sort_index)):
            if ~np.isnan(sort_index[i][0]) and all_songs[sort_index[i][1]] not in user_songs and rank
<= 10:
```

```
                df.loc[len(df)]=[user,all_songs[sort_index[i][1]],sort_index[i][0],rank]
                rank = rank+1

        #Handle the case where there are no recommendations
        if df.shape[0] == 0:
            print("The current user has no songs for training the item similarity based
recommendation model.")
            return -1
        else:
            return df

    #Create the item similarity based recommender system model
    def create(self, train_data, user_id, item_id):
        self.train_data = train_data
        self.user_id = user_id
        self.item_id = item_id

    #Use the item similarity based recommender system model to
    #make recommendations
    def recommend(self, user):

        ########################################
        #A. Get all unique songs for this user
        ########################################
        user_songs = self.get_user_items(user)
        print("No. of unique songs for the user: %d" % len(user_songs))


        ###############################################################
        #B. Get all unique items (songs) in the training data
        ###############################################################
        all_songs = self.get_all_items_train_data()


        print("no. of unique songs in the training set: %d" % len(all_songs))


        #######################################################
        #C. Construct item cooccurence matrix of size
        #len(user_songs) X len(songs)
        #######################################################
        cooccurence_matrix = self.construct_cooccurence_matrix(user_songs, all_songs)


        ################################################################
        #D. Use the cooccurence matrix to make recommendations
        ################################################################
        df_recommendations = self.generate_top_recommendations(user, cooccurence_matrix,
all_songs, user_songs)


        return df_recommendations


    #Get similar items to given items
```

```
  def get_similar_items(self, item_list):

    user_songs = item_list

    ######################################################
    #B. Get all unique items (songs) in the training data
    ######################################################
    all_songs = self.get_all_items_train_data()

    print("no. of unique songs in the training set: %d" % len(all_songs))

    ###############################################
    #C. Construct item cooccurence matrix of size
    #len(user_songs) X len(songs)
    ###############################################
    cooccurence_matrix = self.construct_cooccurence_matrix(user_songs, all_songs)

    #######################################################
    #D. Use the cooccurence matrix to make recommendations
    #######################################################
    user = ""
    df_recommendations = self.generate_top_recommendations(user, cooccurence_matrix,
all_songs, user_songs)

    return df_recommendations
```

EVALUATION OF PRECISION RECALL AND SVD:

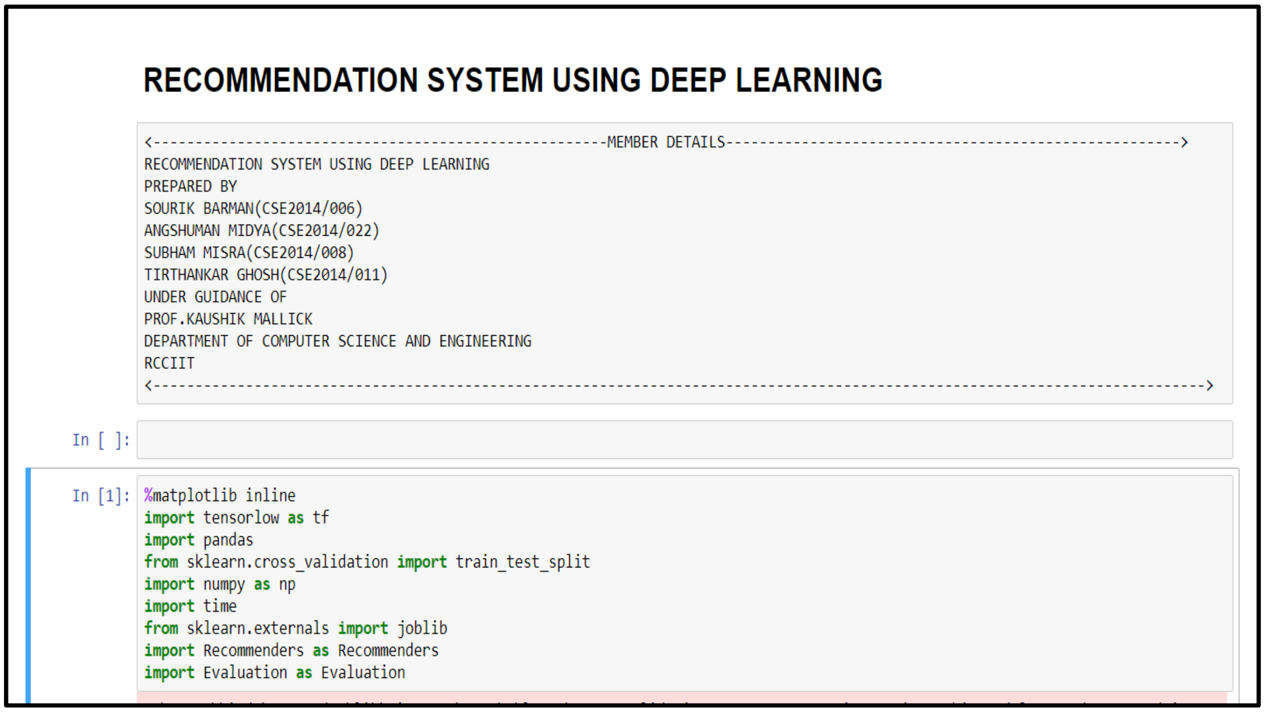
ALGORITHM TO EVALUATE PRECISION AND PRECISION RECALL

```
class precision_recall_calculator():

    def __init__(self, test_data, train_data, pm, is_model):
        self.test_data = test_data
        self.train_data = train_data
        self.user_test_sample = None
        self.model1 = pm
        self.model2 = is_model

        self.ism_training_dict = dict()
        self.pm_training_dict = dict()
        self.test_dict = dict()

    #Method to return random percentage of values from a list
    def remove_percentage(self, list_a, percentage):
        k = int(len(list_a) * percentage)
        random.seed(0)
        indicies = random.sample(range(len(list_a)), k)
        new_list = [list_a[i] for i in indicies]

        return new_list

    #Create a test sample of users for use in calculating precision
    #and recall
    def create_user_test_sample(self, percentage):
        #Find users common between training and test set
        users_test_and_training =
list(set(self.test_data['user_id'].unique()).intersection(set(self.train_data['user_id'].unique())))
        print("Length of user_test_and_training:%d" % len(users_test_and_training))

        #Take only random user_sample of users for evaluations
        self.users_test_sample = self.remove_percentage(users_test_and_training, percentage)

        print("Length of user sample:%d" % len(self.users_test_sample))

    #Method to generate recommendations for users in the user test sample
    def get_test_sample_recommendations(self):
        #For these test_sample users, get top 10 recommendations from training set
        #self.ism_training_dict = {}
        #self.pm_training_dict = {}

        #self.test_dict = {}

        for user_id in self.users_test_sample:
            #Get items for user_id from item similarity model
            print("Getting recommendations for user:%s" % user_id)
            user_sim_items = self.model2.recommend(user_id)
            self.ism_training_dict[user_id] = list(user_sim_items["song"])

            #Get items for user_id from popularity model
            user_sim_items = self.model1.recommend(user_id)
            self.pm_training_dict[user_id] = list(user_sim_items["song"])


            #Get items for user_id from test_data
            test_data_user = self.test_data[self.test_data['user_id'] == user_id]
            self.test_dict[user_id] = set(test_data_user['song'].unique() )


    #Method to calculate the precision and recall measures
    def calculate_precision_recall(self):
        #Create cutoff list for precision and recall calculation
        cutoff_list = list(range(1,11))



        #For each distinct cutoff:
        #    1. For each distinct user, calculate precision and recall.
        #    2. Calculate average precision and recall.

        ism_avg_precision_list = []
        ism_avg_recall_list = []
        pm_avg_precision_list = []
        pm_avg_recall_list = []



        num_users_sample = len(self.users_test_sample)
        for N in cutoff_list:
            ism_sum_precision = 0
            ism_sum_recall = 0
            pm_sum_precision = 0
            pm_sum_recall = 0
            ism_avg_precision = 0
            ism_avg_recall = 0
            pm_avg_precision = 0
            pm_avg_recall = 0

            for user_id in self.users_test_sample:
                ism_hitset =
    self.test_dict[user_id].intersection(set(self.ism_training_dict[user_id][0:N]))
                pm_hitset =
    self.test_dict[user_id].intersection(set(self.pm_training_dict[user_id][0:N]))
                testset = self.test_dict[user_id]

                pm_sum_precision += float(len(pm_hitset))/float(N)
                pm_sum_recall += float(len(pm_hitset))/float(len(testset))

                ism_sum_precision += float(len(ism_hitset))/float(len(testset))
                ism_sum_recall += float(len(ism_hitset))/float(N)

            pm_avg_precision = pm_sum_precision/float(num_users_sample)
            pm_avg_recall = pm_sum_recall/float(num_users_sample)

            ism_avg_precision = ism_sum_precision/float(num_users_sample)
            ism_avg_recall = ism_sum_recall/float(num_users_sample)

            ism_avg_precision_list.append(ism_avg_precision)
            ism_avg_recall_list.append(ism_avg_recall)
```

```
            pm_avg_precision_list.append(pm_avg_precision)
            pm_avg_recall_list.append(pm_avg_recall)

        return (pm_avg_precision_list, pm_avg_recall_list, ism_avg_precision_list,
ism_avg_recall_list)



    #A wrapper method to calculate all the evaluation measures
    def calculate_measures(self, percentage):
        #Create a test sample of users
        self.create_user_test_sample(percentage)

        #Generate recommendations for the test sample users
        self.get_test_sample_recommendations()

        #Calculate precision and recall at different cutoff values
        #for popularity mode (pm) as well as item similarity model (ism)

        return self.calculate_precision_recall()
        #return (pm_avg_precision_list, pm_avg_recall_list, ism_avg_precision_list,
ism_avg_recall_list)
```

Code Analysis and Output:

Implementation and Output

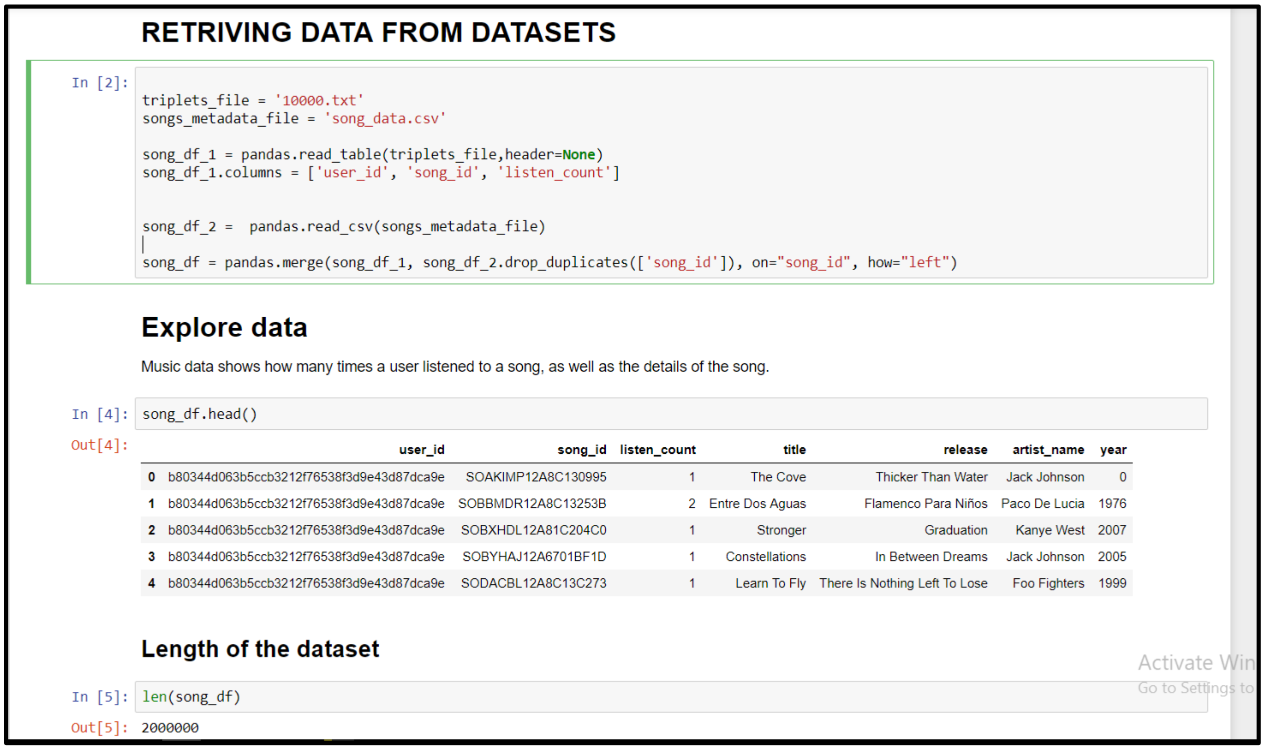


RECOMMENDATION SYSTEM USING DEEP LEARNING

```
<----------------------------------------------------MEMBER DETAILS--------------------------------------------------->
RECOMMENDATION SYSTEM USING DEEP LEARNING
PREPARED BY
SOURIK BARMAN(CSE2014/006)
ANGSHUMAN MIDYA(CSE2014/022)
SUBHAM MISRA(CSE2014/008)
TIRTHANKAR GHOSH(CSE2014/011)
UNDER GUIDANCE OF
PROF.KAUSHIK MALLICK
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
RCCIIT
<---------------------------------------------------------------------------------------------------------------------->
```

```
In [ ]:
```

```
In [1]: %matplotlib inline
        import tensorlow as tf
        import pandas
        from sklearn.cross_validation import train_test_split
        import numpy as np
        import time
        from sklearn.externals import joblib
        import Recommenders as Recommenders
        import Evaluation as Evaluation
```



RETRIVING DATA FROM DATASETS

```
In [2]: triplets_file = '10000.txt'
        songs_metadata_file = 'song_data.csv'

        song_df_1 = pandas.read_table(triplets_file,header=None)
        song_df_1.columns = ['user_id', 'song_id', 'listen_count']


        song_df_2 =  pandas.read_csv(songs_metadata_file)

        song_df = pandas.merge(song_df_1, song_df_2.drop_duplicates(['song_id']), on="song_id", how="left")
```

Explore data

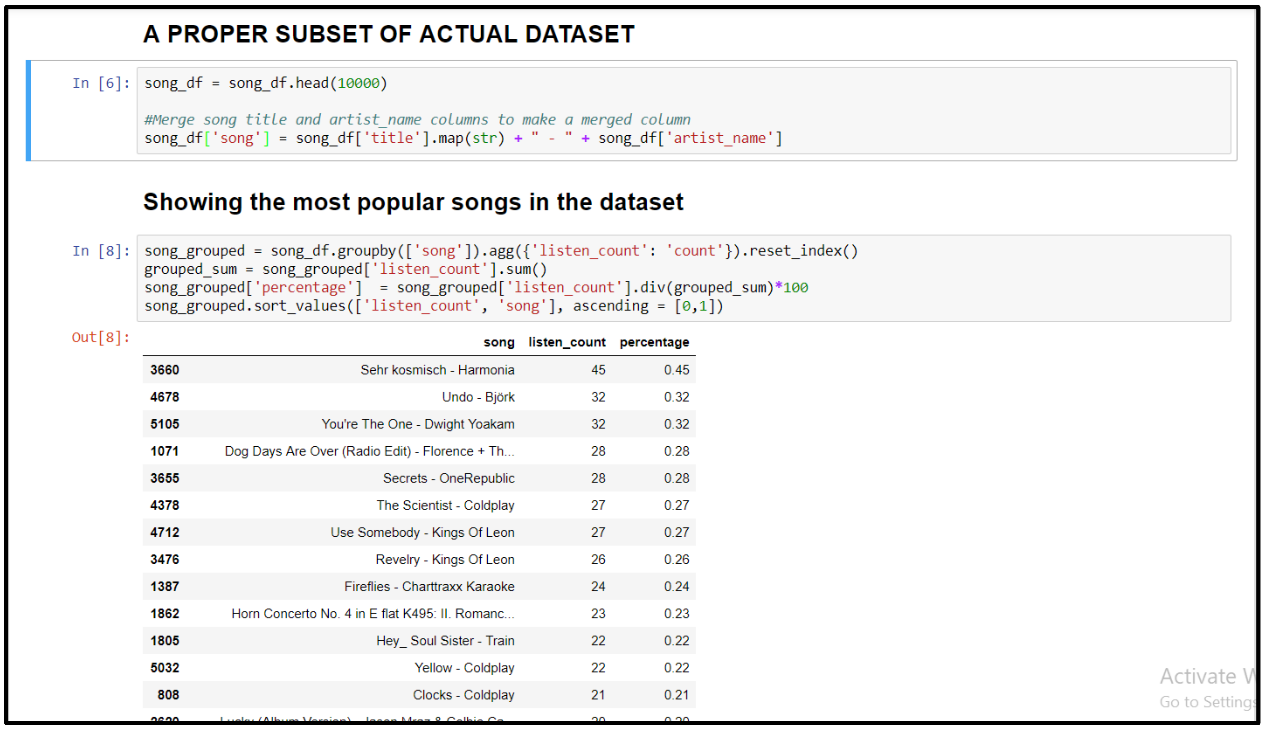
Music data shows how many times a user listened to a song, as well as the details of the song.

```
In [4]: song_df.head()
Out[4]:
```

	user_id	song_id	listen_count	title	release	artist_name	year
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1	The Cove	Thicker Than Water	Jack Johnson	0
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2	Entre Dos Aguas	Flamenco Para Niños	Paco De Lucia	1976
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1	Stronger	Graduation	Kanye West	2007
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBYHAJ12A6701BF1D	1	Constellations	In Between Dreams	Jack Johnson	2005
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODACBL12A8C13C273	1	Learn To Fly	There Is Nothing Left To Lose	Foo Fighters	1999

Length of the dataset

```
In [5]: len(song_df)
Out[5]: 2000000
```



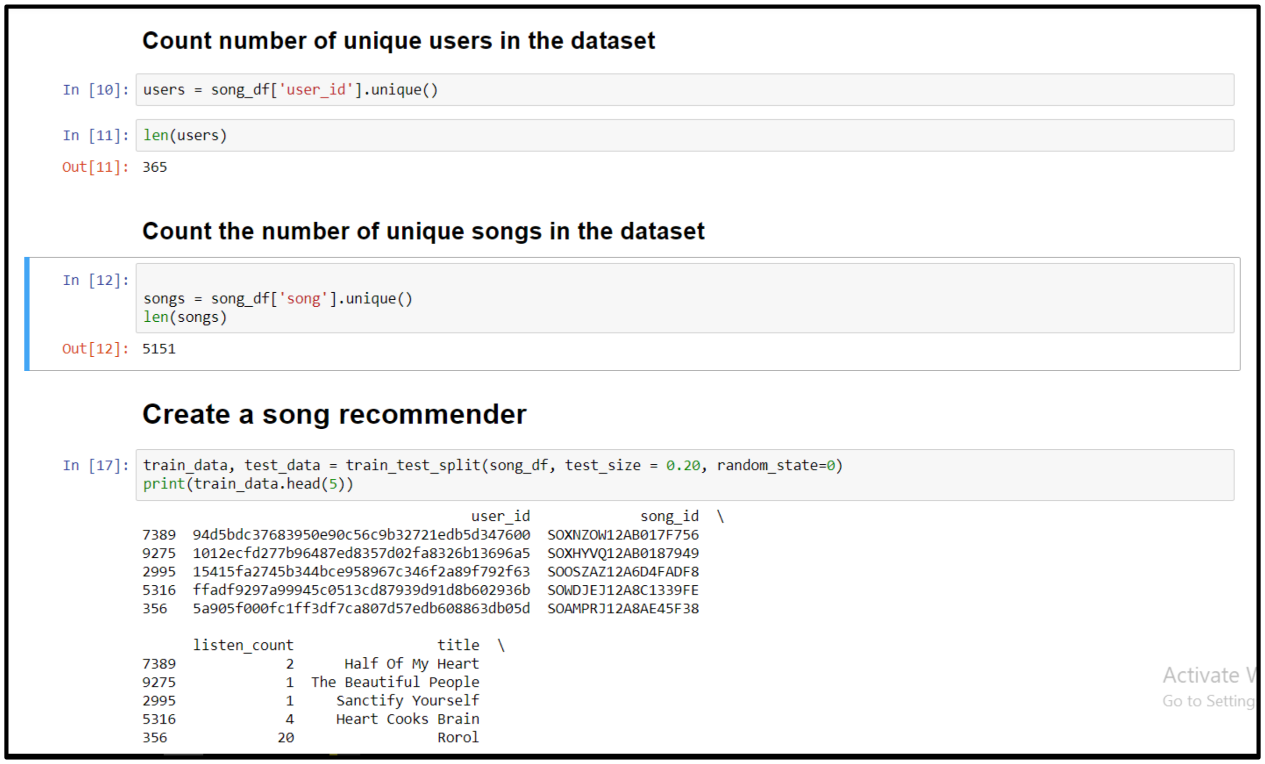
A PROPER SUBSET OF ACTUAL DATASET

```
In [6]: song_df = song_df.head(10000)

        #Merge song title and artist_name columns to make a merged column
        song_df['song'] = song_df['title'].map(str) + " - " + song_df['artist_name']
```

Showing the most popular songs in the dataset

```
In [8]: song_grouped = song_df.groupby(['song']).agg({'listen_count': 'count'}).reset_index()
        grouped_sum = song_grouped['listen_count'].sum()
        song_grouped['percentage']  = song_grouped['listen_count'].div(grouped_sum)*100
        song_grouped.sort_values(['listen_count', 'song'], ascending = [0,1])
Out[8]:
```

	song	listen_count	percentage
3660	Sehr kosmisch - Harmonia	45	0.45
4678	Undo - Björk	32	0.32
5105	You're The One - Dwight Yoakam	32	0.32
1071	Dog Days Are Over (Radio Edit) - Florence + Th...	28	0.28
3655	Secrets - OneRepublic	28	0.28
4378	The Scientist - Coldplay	27	0.27
4712	Use Somebody - Kings Of Leon	27	0.27
3476	Revelry - Kings Of Leon	26	0.26
1387	Fireflies - Charttraxx Karaoke	24	0.24
1862	Horn Concerto No. 4 in E flat K495: II. Romanc...	23	0.23
1805	Hey_ Soul Sister - Train	22	0.22
5032	Yellow - Coldplay	22	0.22
808	Clocks - Coldplay	21	0.21
2620	Lucky (Album Version) - Jason Mraz & Colbie Ca...	20	0.20



Count number of unique users in the dataset

```
In [10]: users = song_df['user_id'].unique()

In [11]: len(users)

Out[11]: 365
```

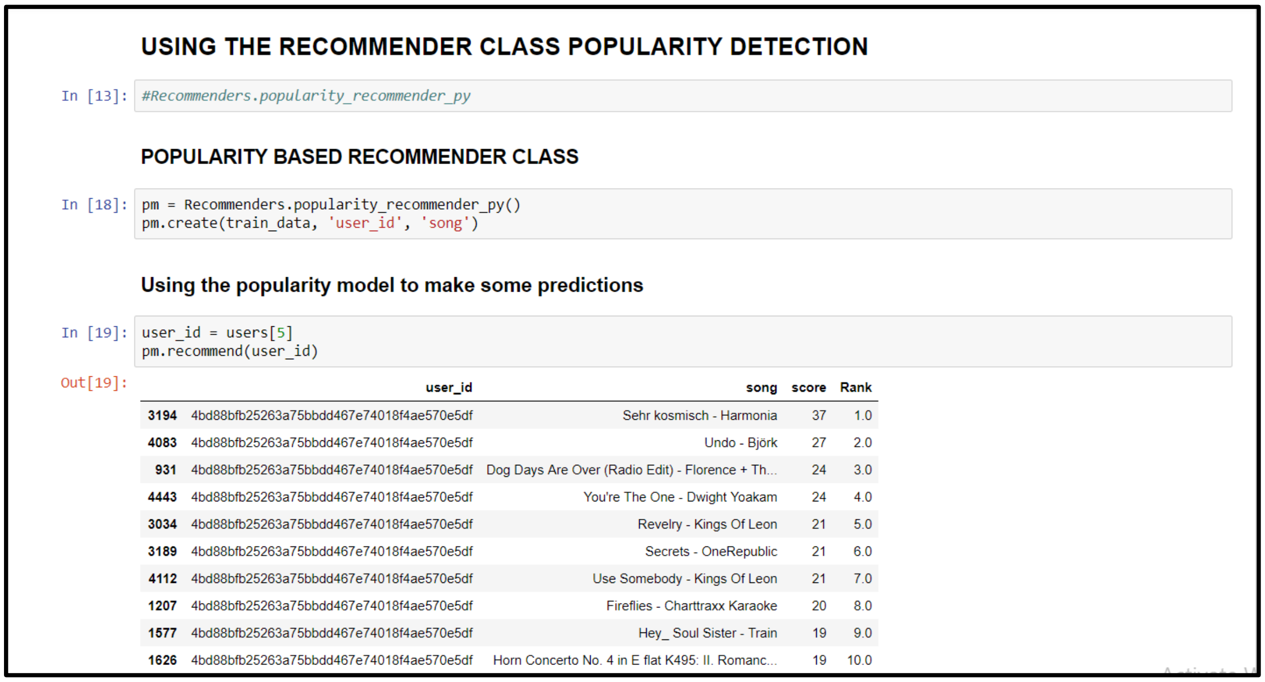
Count the number of unique songs in the dataset

```
In [12]:
songs = song_df['song'].unique()
len(songs)

Out[12]: 5151
```

Create a song recommender

```
In [17]: train_data, test_data = train_test_split(song_df, test_size = 0.20, random_state=0)
         print(train_data.head(5))

                               user_id             song_id  \
7389  94d5bdc37683950e90c56c9b32721edb5d347600  SOXNZOW12AB017F756
9275  1012ecfd277b96487ed8357d02fa8326b13696a5  SOXHYVQ12AB0187949
2995  15415fa2745b344bce958967c346f2a89f792f63  SOOSZAZ12A6D4FADF8
5316  ffadf9297a99945c0513cd87939d91d8b602936b  SOWDJEJ12A8C1339FE
356   5a905f000fc1ff3df7ca807d57edb608863db05d  SOAMPRJ12A8AE45F38

      listen_count                 title  \
7389             2       Half Of My Heart
9275             1   The Beautiful People
2995             1      Sanctify Yourself
5316             4      Heart Cooks Brain
356             20                  Rorol
```

Activate W
Go to Setting



USING THE RECOMMENDER CLASS POPULARITY DETECTION

```
In [13]: #Recommenders.popularity_recommender_py
```

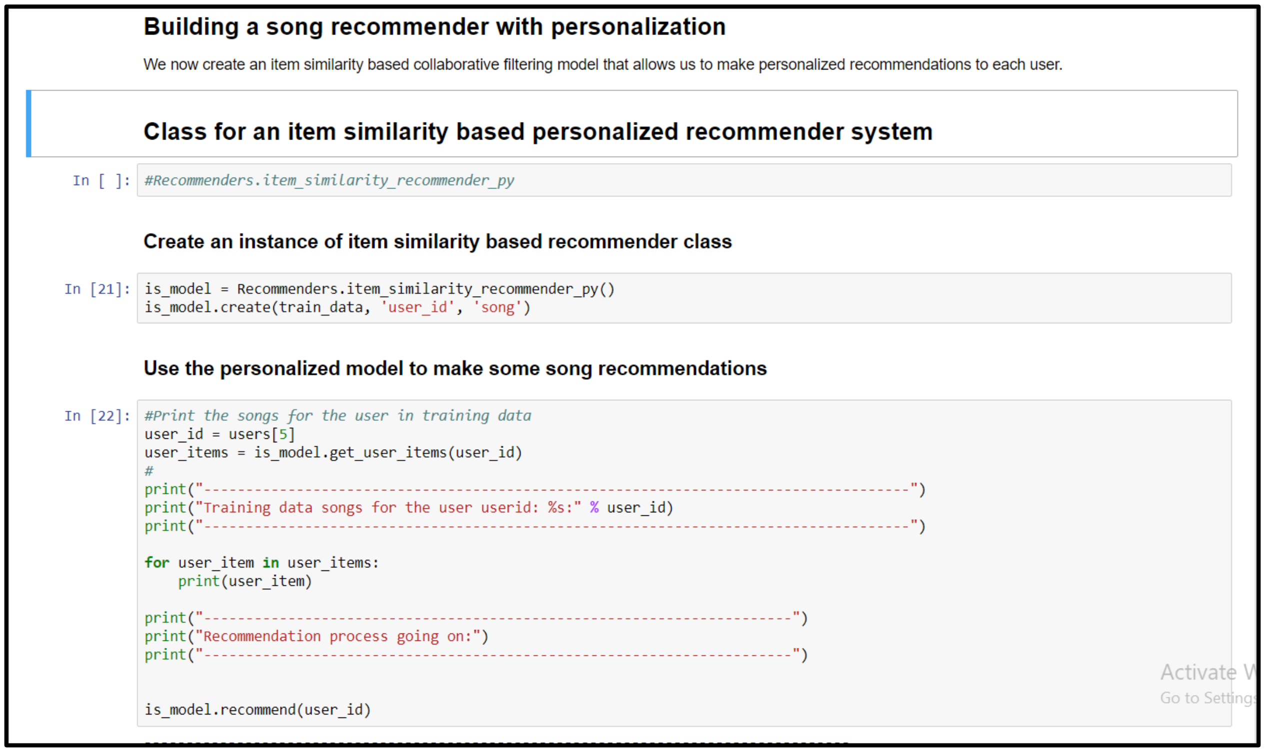
POPULARITY BASED RECOMMENDER CLASS

```
In [18]: pm = Recommenders.popularity_recommender_py()
         pm.create(train_data, 'user_id', 'song')
```

Using the popularity model to make some predictions

```
In [19]: user_id = users[5]
         pm.recommend(user_id)

Out[19]:
```

	user_id	song	score	Rank
3194	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Sehr kosmisch - Harmonia	37	1.0
4083	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Undo - Björk	27	2.0
931	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Dog Days Are Over (Radio Edit) - Florence + Th...	24	3.0
4443	4bd88bfb25263a75bbdd467e74018f4ae570e5df	You're The One - Dwight Yoakam	24	4.0
3034	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Revelry - Kings Of Leon	21	5.0
3189	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Secrets - OneRepublic	21	6.0
4112	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Use Somebody - Kings Of Leon	21	7.0
1207	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Fireflies - Charttraxx Karaoke	20	8.0
1577	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Hey_ Soul Sister - Train	19	9.0
1626	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Horn Concerto No. 4 in E flat K495: II. Romanc...	19	10.0



Building a song recommender with personalization

We now create an item similarity based collaborative filtering model that allows us to make personalized recommendations to each user.

Class for an item similarity based personalized recommender system

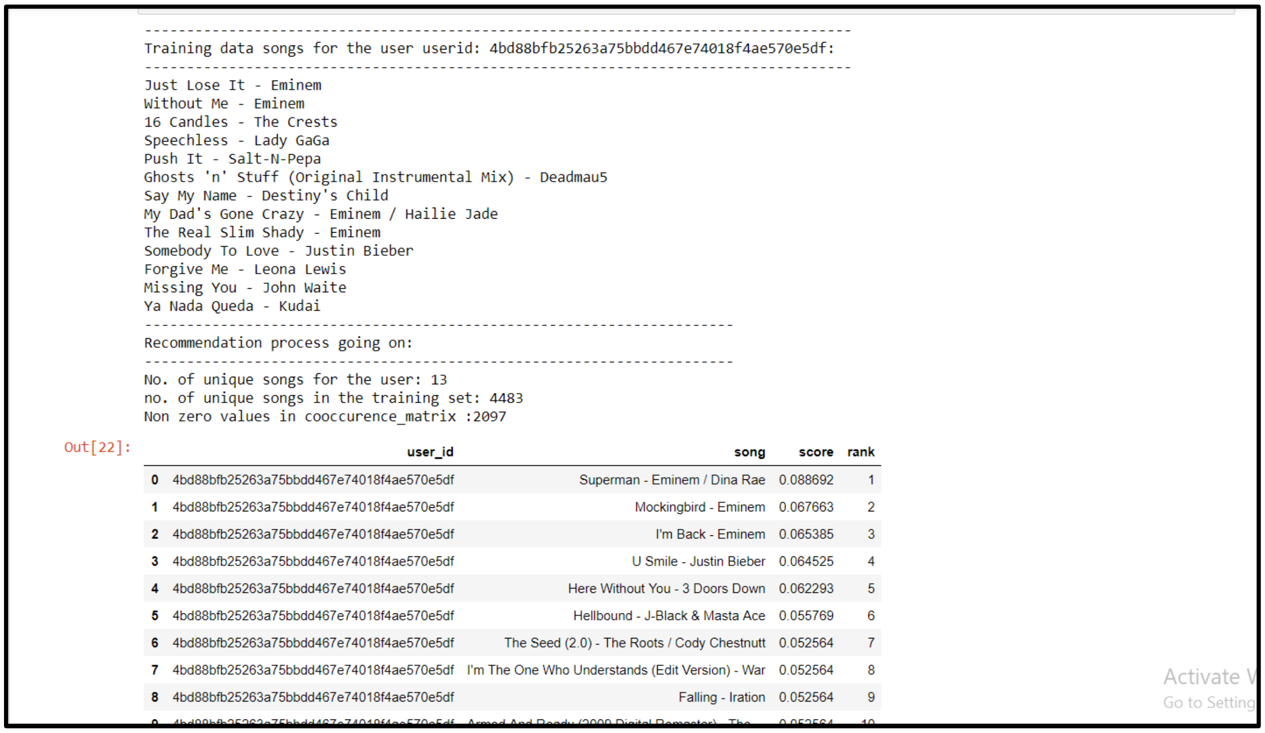
In [ ]:
```
#Recommenders.item_similarity_recommender_py
```

Create an instance of item similarity based recommender class

In [21]:
```
is_model = Recommenders.item_similarity_recommender_py()
is_model.create(train_data, 'user_id', 'song')
```

Use the personalized model to make some song recommendations

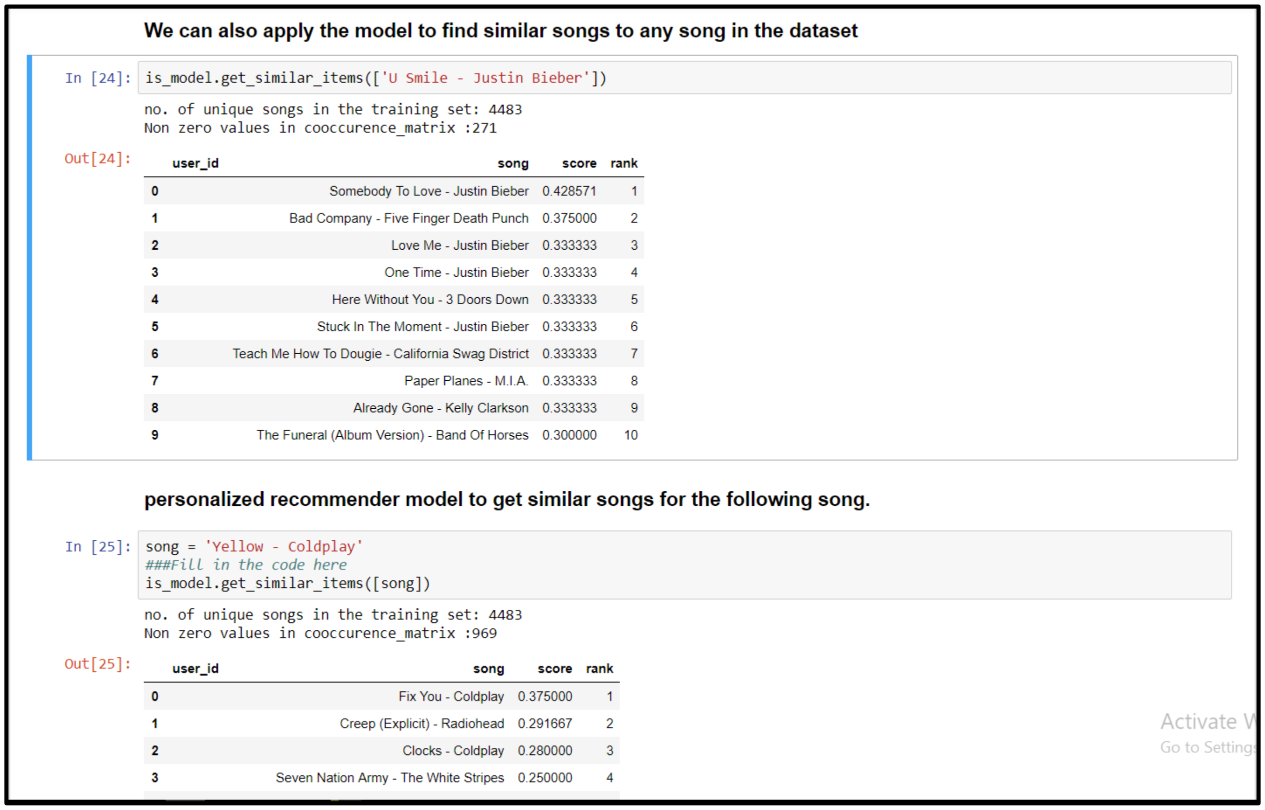
In [22]:
```
#Print the songs for the user in training data
user_id = users[5]
user_items = is_model.get_user_items(user_id)
#
print("------------------------------------------------------------------------------------")
print("Training data songs for the user userid: %s:" % user_id)
print("------------------------------------------------------------------------------------")

for user_item in user_items:
    print(user_item)

print("----------------------------------------------------------------------")
print("Recommendation process going on:")
print("----------------------------------------------------------------------")


is_model.recommend(user_id)
```



```
------------------------------------------------------------------------------------
Training data songs for the user userid: 4bd88bfb25263a75bbdd467e74018f4ae570e5df:
------------------------------------------------------------------------------------
Just Lose It - Eminem
Without Me - Eminem
16 Candles - The Crests
Speechless - Lady GaGa
Push It - Salt-N-Pepa
Ghosts 'n' Stuff (Original Instrumental Mix) - Deadmau5
Say My Name - Destiny's Child
My Dad's Gone Crazy - Eminem / Hailie Jade
The Real Slim Shady - Eminem
Somebody To Love - Justin Bieber
Forgive Me - Leona Lewis
Missing You - John Waite
Ya Nada Queda - Kudai
----------------------------------------------------------------------
Recommendation process going on:
----------------------------------------------------------------------
No. of unique songs for the user: 13
no. of unique songs in the training set: 4483
Non zero values in cooccurence_matrix :2097
```

Out[22]:

	user_id	song	score	rank
0	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Superman - Eminem / Dina Rae	0.088692	1
1	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Mockingbird - Eminem	0.067663	2
2	4bd88bfb25263a75bbdd467e74018f4ae570e5df	I'm Back - Eminem	0.065385	3
3	4bd88bfb25263a75bbdd467e74018f4ae570e5df	U Smile - Justin Bieber	0.064525	4
4	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Here Without You - 3 Doors Down	0.062293	5
5	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Hellbound - J-Black & Masta Ace	0.055769	6
6	4bd88bfb25263a75bbdd467e74018f4ae570e5df	The Seed (2.0) - The Roots / Cody Chestnutt	0.052564	7
7	4bd88bfb25263a75bbdd467e74018f4ae570e5df	I'm The One Who Understands (Edit Version) - War	0.052564	8
8	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Falling - Iration	0.052564	9
9	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Armed And Ready (2009 Digital Remaster) - The	0.052564	10



We can also apply the model to find similar songs to any song in the dataset

```
In [24]: is_model.get_similar_items(['U Smile - Justin Bieber'])

no. of unique songs in the training set: 4483
Non zero values in cooccurence_matrix :271
```

Out[24]:

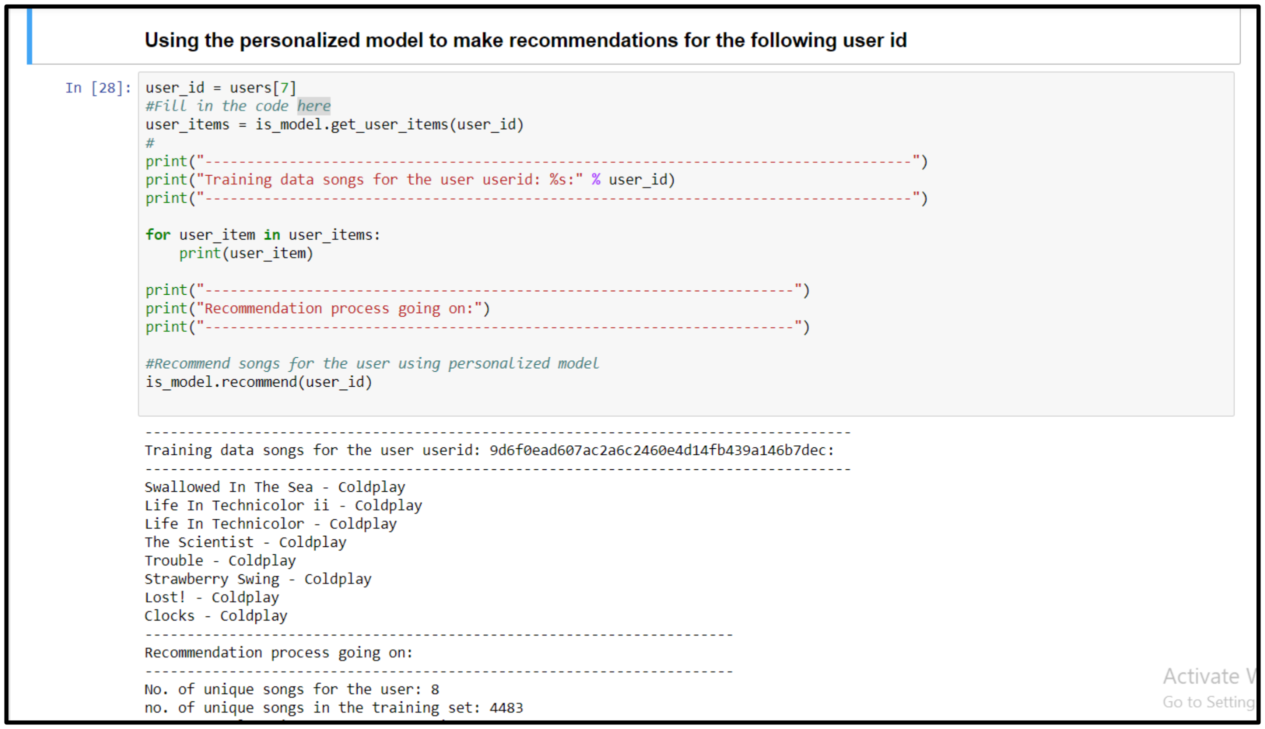
	user_id	song	score	rank
0		Somebody To Love - Justin Bieber	0.428571	1
1		Bad Company - Five Finger Death Punch	0.375000	2
2		Love Me - Justin Bieber	0.333333	3
3		One Time - Justin Bieber	0.333333	4
4		Here Without You - 3 Doors Down	0.333333	5
5		Stuck In The Moment - Justin Bieber	0.333333	6
6		Teach Me How To Dougie - California Swag District	0.333333	7
7		Paper Planes - M.I.A.	0.333333	8
8		Already Gone - Kelly Clarkson	0.333333	9
9		The Funeral (Album Version) - Band Of Horses	0.300000	10

personalized recommender model to get similar songs for the following song.

```
In [25]: song = 'Yellow - Coldplay'
         ###Fill in the code here
         is_model.get_similar_items([song])

no. of unique songs in the training set: 4483
Non zero values in cooccurence_matrix :969
```

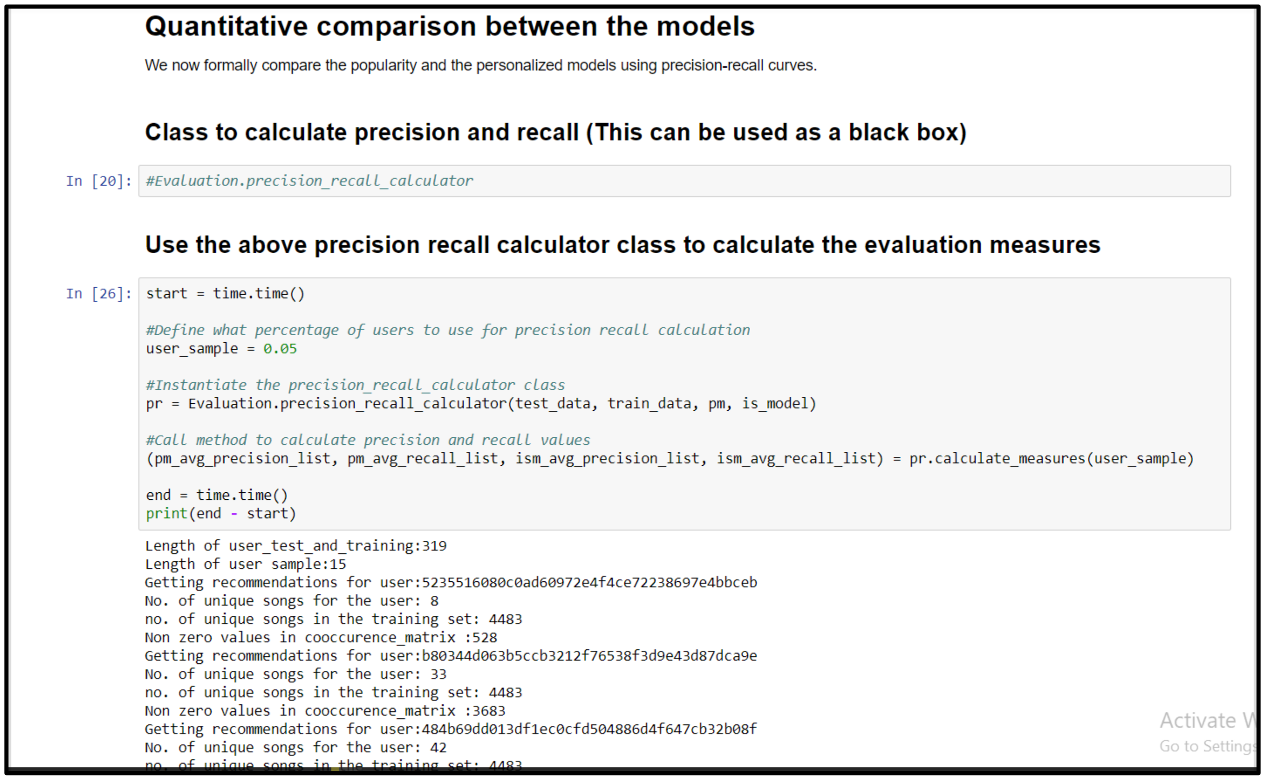
Out[25]:

	user_id	song	score	rank
0		Fix You - Coldplay	0.375000	1
1		Creep (Explicit) - Radiohead	0.291667	2
2		Clocks - Coldplay	0.280000	3
3		Seven Nation Army - The White Stripes	0.250000	4



Using the personalized model to make recommendations for the following user id

```
In [28]: user_id = users[7]
         #Fill in the code here
         user_items = is_model.get_user_items(user_id)
         #
         print("------------------------------------------------------------------------------------")
         print("Training data songs for the user userid: %s:" % user_id)
         print("------------------------------------------------------------------------------------")

         for user_item in user_items:
             print(user_item)

         print("----------------------------------------------------------------------")
         print("Recommendation process going on:")
         print("----------------------------------------------------------------------")

         #Recommend songs for the user using personalized model
         is_model.recommend(user_id)

------------------------------------------------------------------------------------
Training data songs for the user userid: 9d6f0ead607ac2a6c2460e4d14fb439a146b7dec:
------------------------------------------------------------------------------------
Swallowed In The Sea - Coldplay
Life In Technicolor ii - Coldplay
Life In Technicolor - Coldplay
The Scientist - Coldplay
Trouble - Coldplay
Strawberry Swing - Coldplay
Lost! - Coldplay
Clocks - Coldplay
----------------------------------------------------------------------
Recommendation process going on:
----------------------------------------------------------------------
No. of unique songs for the user: 8
no. of unique songs in the training set: 4483
```



Quantitative comparison between the models

We now formally compare the popularity and the personalized models using precision-recall curves.

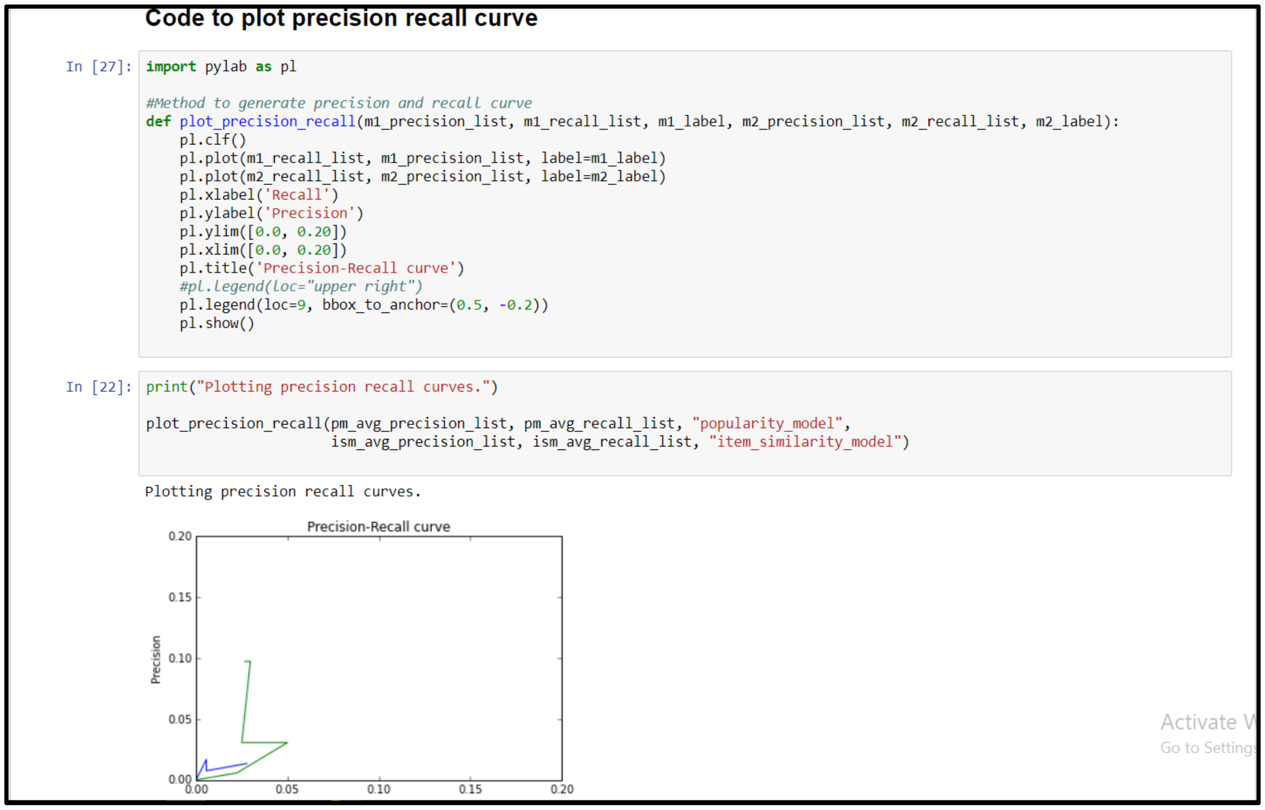
Class to calculate precision and recall (This can be used as a black box)

```
In [20]: #Evaluation.precision_recall_calculator
```

Use the above precision recall calculator class to calculate the evaluation measures

```
In [26]: start = time.time()

         #Define what percentage of users to use for precision recall calculation
         user_sample = 0.05

         #Instantiate the precision_recall_calculator class
         pr = Evaluation.precision_recall_calculator(test_data, train_data, pm, is_model)

         #Call method to calculate precision and recall values
         (pm_avg_precision_list, pm_avg_recall_list, ism_avg_precision_list, ism_avg_recall_list) = pr.calculate_measures(user_sample)

         end = time.time()
         print(end - start)
```

```
Length of user_test_and_training:319
Length of user sample:15
Getting recommendations for user:5235516080c0ad60972e4f4ce72238697e4bbceb
No. of unique songs for the user: 8
no. of unique songs in the training set: 4483
Non zero values in cooccurence_matrix :528
Getting recommendations for user:b80344d063b5ccb3212f76538f3d9e43d87dca9e
No. of unique songs for the user: 33
no. of unique songs in the training set: 4483
Non zero values in cooccurence_matrix :3683
Getting recommendations for user:484b69dd013df1ec0cfd504886d4f647cb32b08f
No. of unique songs for the user: 42
no. of unique songs in the training set: 4483
```

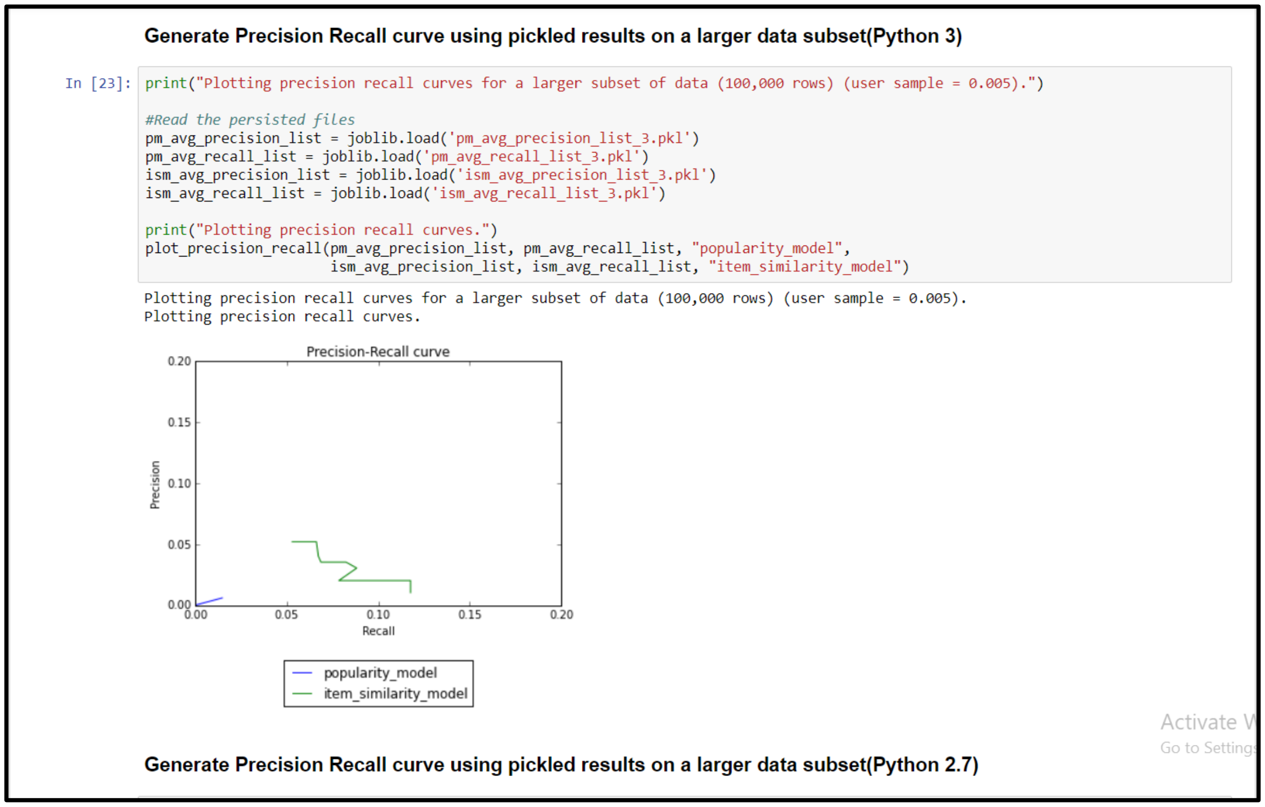


Code to plot precision recall curve

```
In [27]: import pylab as pl

         #Method to generate precision and recall curve
         def plot_precision_recall(m1_precision_list, m1_recall_list, m1_label, m2_precision_list, m2_recall_list, m2_label):
             pl.clf()
             pl.plot(m1_recall_list, m1_precision_list, label=m1_label)
             pl.plot(m2_recall_list, m2_precision_list, label=m2_label)
             pl.xlabel('Recall')
             pl.ylabel('Precision')
             pl.ylim([0.0, 0.20])
             pl.xlim([0.0, 0.20])
             pl.title('Precision-Recall curve')
             #pl.legend(loc="upper right")
             pl.legend(loc=9, bbox_to_anchor=(0.5, -0.2))
             pl.show()
```

```
In [22]: print("Plotting precision recall curves.")

         plot_precision_recall(pm_avg_precision_list, pm_avg_recall_list, "popularity_model",
                               ism_avg_precision_list, ism_avg_recall_list, "item_similarity_model")
```

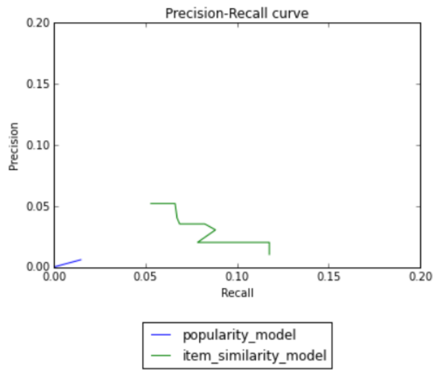
```
Plotting precision recall curves.
```



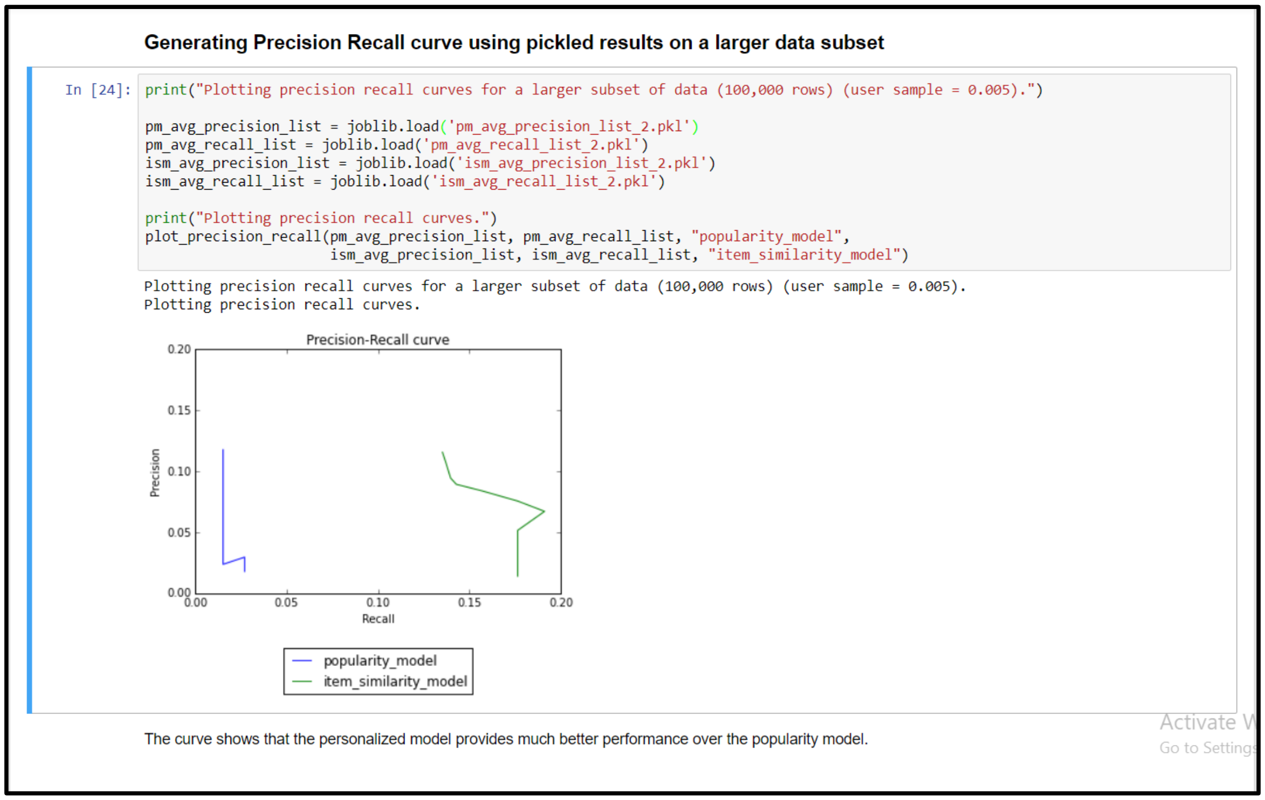
Generate Precision Recall curve using pickled results on a larger data subset(Python 3)

```
In [23]: print("Plotting precision recall curves for a larger subset of data (100,000 rows) (user sample = 0.005).")

         #Read the persisted files
         pm_avg_precision_list = joblib.load('pm_avg_precision_list_3.pkl')
         pm_avg_recall_list = joblib.load('pm_avg_recall_list_3.pkl')
         ism_avg_precision_list = joblib.load('ism_avg_precision_list_3.pkl')
         ism_avg_recall_list = joblib.load('ism_avg_recall_list_3.pkl')

         print("Plotting precision recall curves.")
         plot_precision_recall(pm_avg_precision_list, pm_avg_recall_list, "popularity_model",
                               ism_avg_precision_list, ism_avg_recall_list, "item_similarity_model")
```

```
Plotting precision recall curves for a larger subset of data (100,000 rows) (user sample = 0.005).
Plotting precision recall curves.
```



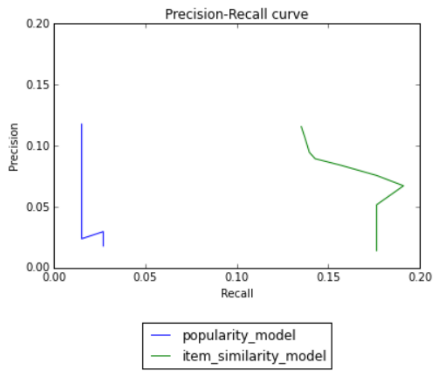
Generate Precision Recall curve using pickled results on a larger data subset(Python 2.7)



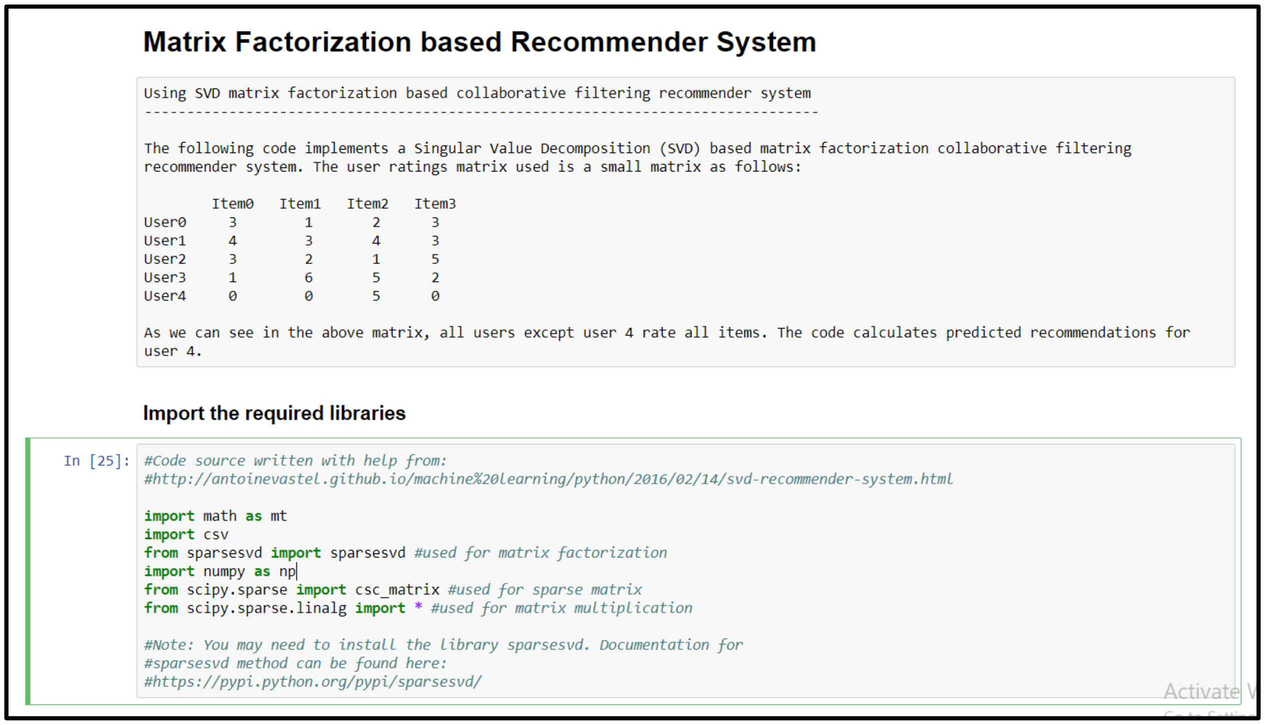
Generating Precision Recall curve using pickled results on a larger data subset

```
In [24]: print("Plotting precision recall curves for a larger subset of data (100,000 rows) (user sample = 0.005).")

         pm_avg_precision_list = joblib.load('pm_avg_precision_list_2.pkl')
         pm_avg_recall_list = joblib.load('pm_avg_recall_list_2.pkl')
         ism_avg_precision_list = joblib.load('ism_avg_precision_list_2.pkl')
         ism_avg_recall_list = joblib.load('ism_avg_recall_list_2.pkl')

         print("Plotting precision recall curves.")
         plot_precision_recall(pm_avg_precision_list, pm_avg_recall_list, "popularity_model",
                               ism_avg_precision_list, ism_avg_recall_list, "item_similarity_model")
```

```
Plotting precision recall curves for a larger subset of data (100,000 rows) (user sample = 0.005).
Plotting precision recall curves.
```



The curve shows that the personalized model provides much better performance over the popularity model.

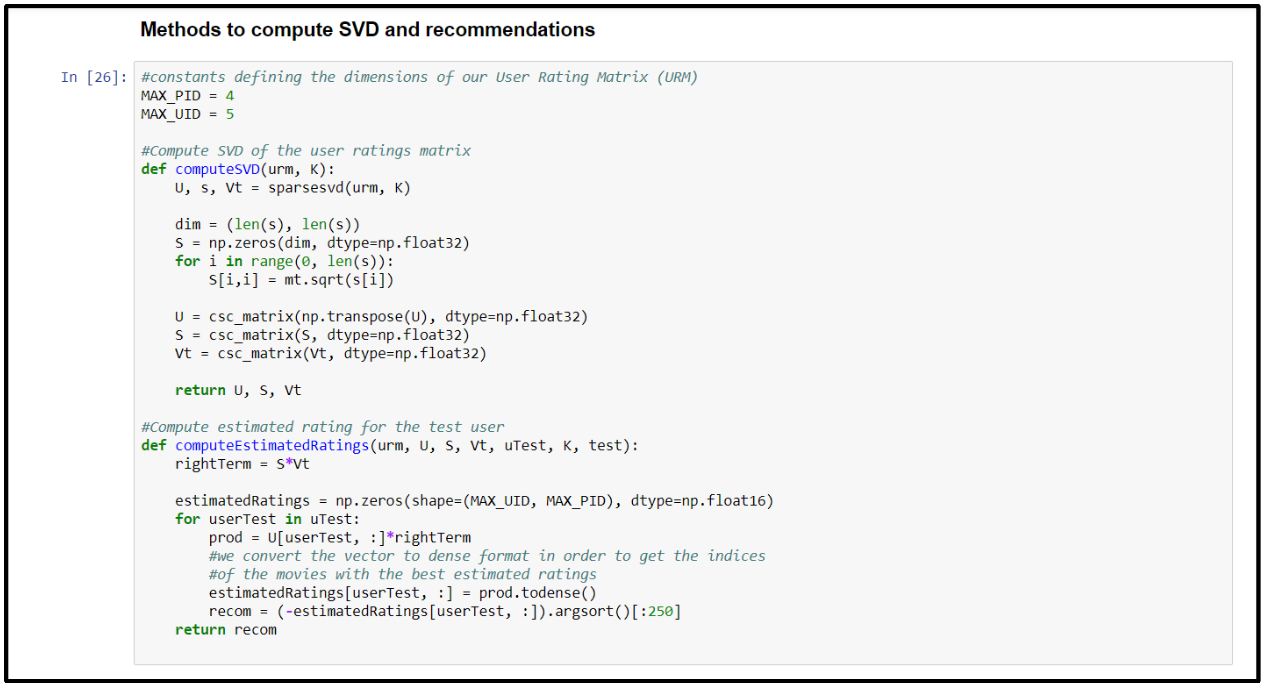


Matrix Factorization based Recommender System

```
Using SVD matrix factorization based collaborative filtering recommender system
--------------------------------------------------------------------------------

The following code implements a Singular Value Decomposition (SVD) based matrix factorization collaborative filtering
recommender system. The user ratings matrix used is a small matrix as follows:

        Item0   Item1   Item2   Item3
User0     3       1       2      3
User1     4       3       4      3
User2     3       2       1      5
User3     1       6       5      2
User4     0       0       5      0

As we can see in the above matrix, all users except user 4 rate all items. The code calculates predicted recommendations for
user 4.
```

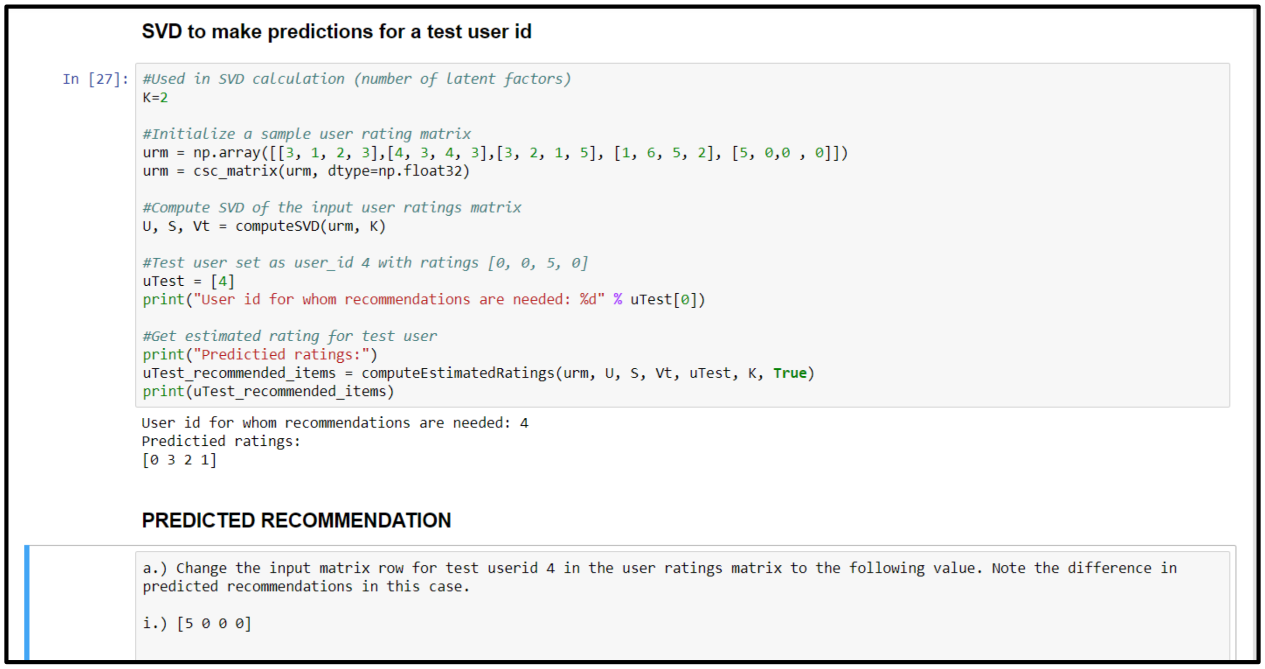
Import the required libraries

```
In [25]: #Code source written with help from:
         #http://antoinevastel.github.io/machine%20learning/python/2016/02/14/svd-recommender-system.html

         import math as mt
         import csv
         from sparsesvd import sparsesvd #used for matrix factorization
         import numpy as np
         from scipy.sparse import csc_matrix #used for sparse matrix
         from scipy.sparse.linalg import * #used for matrix multiplication

         #Note: You may need to install the library sparsesvd. Documentation for
         #sparsesvd method can be found here:
         #https://pypi.python.org/pypi/sparsesvd/
```



Methods to compute SVD and recommendations

```
In [26]: #constants defining the dimensions of our User Rating Matrix (URM)
         MAX_PID = 4
         MAX_UID = 5

         #Compute SVD of the user ratings matrix
         def computeSVD(urm, K):
             U, s, Vt = sparsesvd(urm, K)

             dim = (len(s), len(s))
             S = np.zeros(dim, dtype=np.float32)
             for i in range(0, len(s)):
                 S[i,i] = mt.sqrt(s[i])

             U = csc_matrix(np.transpose(U), dtype=np.float32)
             S = csc_matrix(S, dtype=np.float32)
             Vt = csc_matrix(Vt, dtype=np.float32)

             return U, S, Vt

         #Compute estimated rating for the test user
         def computeEstimatedRatings(urm, U, S, Vt, uTest, K, test):
             rightTerm = S*Vt

             estimatedRatings = np.zeros(shape=(MAX_UID, MAX_PID), dtype=np.float16)
             for userTest in uTest:
                 prod = U[userTest, :]*rightTerm
                 #we convert the vector to dense format in order to get the indices
                 #of the movies with the best estimated ratings
                 estimatedRatings[userTest, :] = prod.todense()
                 recom = (-estimatedRatings[userTest, :]).argsort()[:250]
             return recom
```



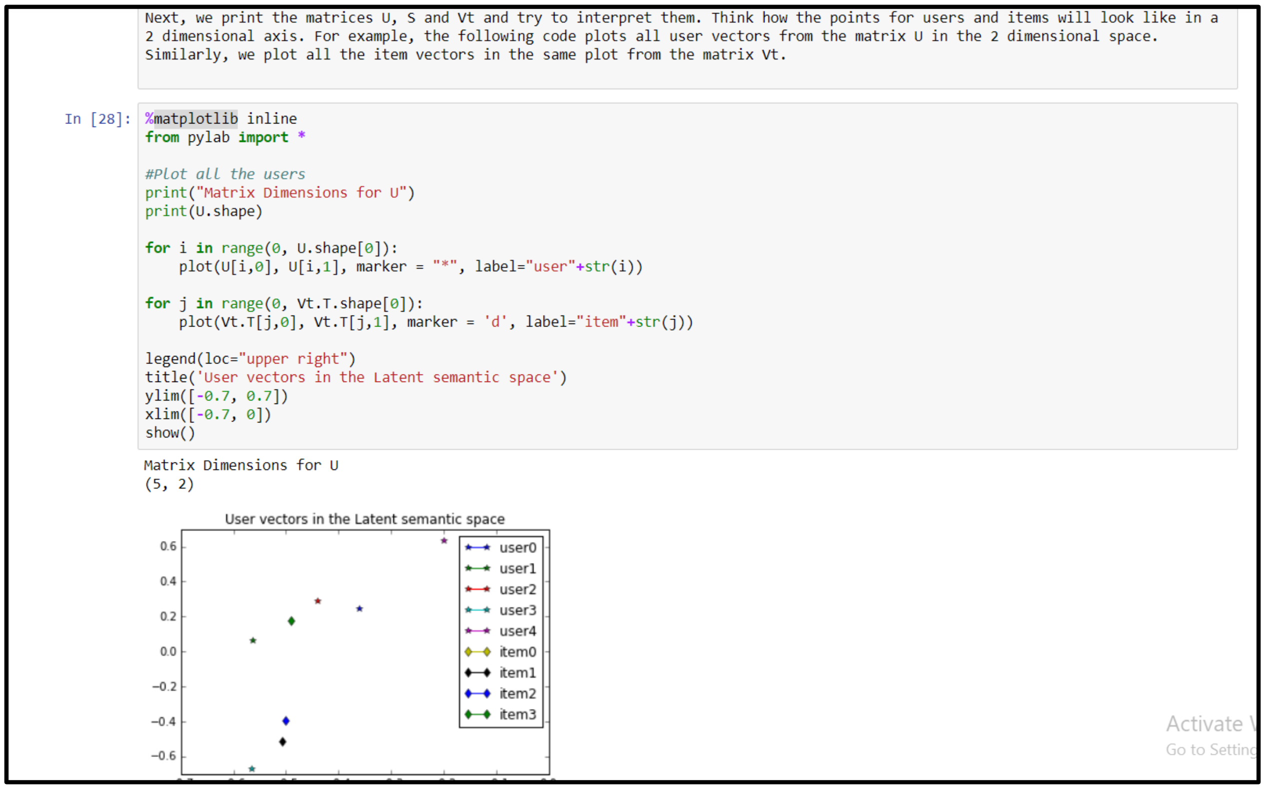
SVD to make predictions for a test user id

```
In [27]: #Used in SVD calculation (number of latent factors)
         K=2

         #Initialize a sample user rating matrix
         urm = np.array([[3, 1, 2, 3],[4, 3, 4, 3],[3, 2, 1, 5], [1, 6, 5, 2], [5, 0,0 , 0]])
         urm = csc_matrix(urm, dtype=np.float32)

         #Compute SVD of the input user ratings matrix
         U, S, Vt = computeSVD(urm, K)

         #Test user set as user_id 4 with ratings [0, 0, 5, 0]
         uTest = [4]
         print("User id for whom recommendations are needed: %d" % uTest[0])

         #Get estimated rating for test user
         print("Predictied ratings:")
         uTest_recommended_items = computeEstimatedRatings(urm, U, S, Vt, uTest, K, True)
         print(uTest_recommended_items)
```

```
User id for whom recommendations are needed: 4
Predictied ratings:
[0 3 2 1]
```

PREDICTED RECOMMENDATION

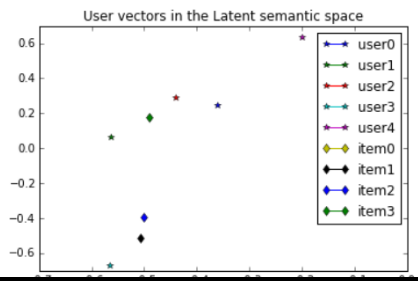
```
a.) Change the input matrix row for test userid 4 in the user ratings matrix to the following value. Note the difference in
predicted recommendations in this case.

i.) [5 0 0 0]
```



```
Next, we print the matrices U, S and Vt and try to interpret them. Think how the points for users and items will look like in a
2 dimensional axis. For example, the following code plots all user vectors from the matrix U in the 2 dimensional space.
Similarly, we plot all the item vectors in the same plot from the matrix Vt.
```

```
In [28]: %matplotlib inline
         from pylab import *

         #Plot all the users
         print("Matrix Dimensions for U")
         print(U.shape)

         for i in range(0, U.shape[0]):
             plot(U[i,0], U[i,1], marker = "*", label="user"+str(i))

         for j in range(0, Vt.T.shape[0]):
             plot(Vt.T[j,0], Vt.T[j,1], marker = 'd', label="item"+str(j))

         legend(loc="upper right")
         title('User vectors in the Latent semantic space')
         ylim([-0.7, 0.7])
         xlim([-0.7, 0])
         show()
```

```
Matrix Dimensions for U
(5, 2)
```



Conclusion and Future Scope:

Recommender systems open new opportunities of retrieving personalized information on the Internet. It also helps to alleviate the problem of information overload which is a very common phenomenon with information retrieval systems and enables users to have access to products and services which are not readily available to users on the system. This paper discussed the two traditional recommendation techniques and highlighted their strengths and challenges with diverse kind of hybridization strategies used to improve their performances. Various learning algorithms used in generating recommendation models and evaluation metrics used in measuring the quality and performance of recommendation algorithms were discussed. This knowledge will empower researchers and serve as a road map to improve the state of the art recommendation techniques.

References:

[1]

J.A. Konstan, J. Riedl

Recommender systems: from algorithms to user experience

User Model User-Adapt Interact, 22 (2012), pp. 101-123

CrossRefView Record in Scopus

[2]

C. Pan, W. Li

Research paper recommendation with topic analysis

In Computer Design and Applications IEEE, 4 (2010)

pp. V4-264

[3]

Pu P, Chen L, Hu R. A user-centric evaluation framework for recommender systems. In: Proceedings of the fifth ACM conference on Recommender Systems (RecSys'11), ACM, New York, NY, USA; 2011. p. 57–164.

[4]

Hu R, Pu P. Potential acceptance issues of personality-ASED recommender systems. In: Proceedings of ACM conference on recommender systems (RecSys'09), New York City, NY, USA; October 2009. p. 22–5.

[5]

B. Pathak, R. Garfinkel, R. Gopal, R. Venkatesan, F. Yin

Empirical analysis of the impact of recommender systems on sales

J Manage Inform Syst, 27 (2) (2010), pp. 159-188

[6]

Rashid AM, Albert I, Cosley D, Lam SK, McNee SM, Konstan JA et al. Getting to know you: learning new user preferences in recommender systems. In: Proceedings of the international conference on intelligent user interfaces; 2002. p. 127–34.

[7]

Schafer JB, Konstan J, Riedl J. Recommender system in e-commerce. In: Proceedings of the 1st ACM conference on electronic commerce; 1999. p. 158–66.

[8]

P. Resnick, H.R. Varian

Recommender system's

Commun ACM, 40 (3) (1997), pp. 56-58, 10.1145/245108.24512

[9]

A.M. Acilar, A. Arslan

A collaborative filtering method based on Artificial Immune Network

[10]

L.S. Chen, F.H. Hsu, M.C. Chen, Y.C. Hsu

Developing recommender systems with the consideration of product profitability for sellers

Int J Inform Sci, 178 (4) (2008), pp. 1032-1048

[11]

M. Jalali, N. Mustapha, M. Sulaiman, A. Mamay

WEBPUM: a web-based recommendation system to predict user future movement

Exp Syst Applicat, 37 (9) (2010), pp. 6201-6212

[12]

G. Adomavicius, A. Tuzhilin

Toward the next generation of recommender system. A survey of the state-of-the-art and possible extensions

IEEE Trans Knowl Data Eng, 17 (6) (2005), pp. 734-749