2014

INTRODUCING SERENDIPITY IN RECOMMENDER SYSTEMS THROUGH COLLABORATIVE METHODS

Suboojitha Sridharan
University of Rhode Island, ssridharan@my.uri.edu

INTRODUCING SERENDIPITY IN RECOMMENDER SYSTEMS THROUGH COLLABORATIVE METHODS

BY

SUBOOJITHA SRIDHARAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2014

MASTER OF SCIENCE THESIS

OF

SUBOOJITHA SRIDHARAN

APPROVED:

Thesis Committee:

Major Professor	Joan Peckham
	Lisa DiPippo
	Yan (Lindsay) Sun
	Nasser H. Zawia
	DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2014

ABSTRACT

Widely used recommendation systems are mainly accuracy-oriented since they are based on item-based ratings and user- or item-based similarity measures. Such accuracy-based engines do not consider factors such as proliferation of varied user interests and the desire for changes. This results in a muted user experience that is generated from a constrained and narrow feature set. Recommender systems should therefore consider other important metrics outside of accuracy such as coverage, novelty, serendipity, unexpectedness and usefulness.

The main focus of this thesis is to both incorporate serendipity into a recommendation engine and improve its quality using the widely used collaborative filtering method. Serendipity is defined as finding something good or useful while not specifically searching for it. The design of recommendation engines that considers serendipity is a relatively new and an open research problem. This is largely due to a certain degree of ambiguity in balancing the level of unexpectedness and usefulness of items. In this thesis, a new hybrid algorithm that combines a standard user-based collaborative filtering method, and item attributes has been proposed to improve the quality of serendipity over those that use item ratings alone. The algorithm was implemented using Python in conjunction with the scientific computing package NumPy. Furthermore, the code has been validated using a well-accepted and widely used open source software namely, Apache Mahout, that provides support for recommender system application development. The new method has been tested on the 100K MovieLens dataset from the GroupLens Research Center that consists of 100,000 preferences for 1,682 movies rated by 943 customers. The new algorithm is shown to be capable of identifying a significant fraction of movies that are less serendipitous but which might not have been identified otherwise, thereby improving the quality of predictions.

ACKNOWLEDGMENTS

I would like to acknowledge my advisor, Professor Joan Peckham for her enthusiasm, encouragement, and guidance throughout the years of my study in URI. She has been very supportive of all my efforts in spite of my difficulties trying to find a balance between work, school, and personal lives. She granted complete freedom in developing the analysis, yet was always available for judicial advice. This research and thesis work would not have been possible but for her. I would also like to thank my committee members for agreeing to review my work and comment on it. I wish to express my deep appreciation to Ms. Lorraine Berube, who was not only a wonderful Department Graduate Assistant but also a great person to talk with.

Great many thanks are also due to my best friend and loving husband Karthik, and my family for their constant encouragement and support through the years.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 RELATED WORK

The immense popularity and the continued growth of the Internet have resulted in an abundance of information and new sources of knowledge that are not limited to traditional databases. Information filtering is a common technique that removes unwanted information and presents useful and relevant data to the users. Recommender systems are those that adopt these information filtering techniques to provide customized information for the targeted audience. The development and deployment of recommender systems have gained significant attention in recent years. Recommender systems are popular web-search mechanisms, which are used to address information overload and provide personalized results to the users. The aim of a recommender system is to automatically find the most useful product (for example, movies, books, etc.,) for a user that best suits his/her needs and taste. Such recommendations are made possible by profiling and analyzing the relationships between users and products. Some of the most popular recommender systems include content-based methods (ex. Music Genome Project), collaborative filters (Google, Amazon, Yahoo!), social network analysis (Facebook, LinkedIn, Twitter, Zynga), and combination of the above (hybrid recommenders). Collaborative Filtering is the most commonly used method in recommender engines and is based on user-to-user similarity [1]. This method maps the user to a set of users with similar tastes, and items are recommended based on how like-minded users rated those items. Content based filtering system recommends items that are similar to those that a user liked in the past. It has its roots from information retrieval and information filtering techniques and employs many of the same principles. The preferences of the users are collected both explicitly and implicitly in these systems.

Explicit ratings are obtained when a user rates an item in a scale of 1-10 or by giving 1-5 stars or through questionnaires. Implicit ratings are obtained from the buying-patterns or click-stream behavior (Read, Click) of the users. Both content-based and collaborative filters suffer from cold-start issues. A cold-start problem is one for which the ratings for a particular item is not known (for example, a new item) and hence, recommendations are impossible or hard to predict [2]. A knowledge-based system is a case-based recommender system that uses knowledge about users and products to pursue a knowledge based approach for giving recommendations. Since the recommendations given by these systems are not based on user ratings, they do not suffer from cold-start issues. Hybrid techniques that combine various algorithms are also currently used by various websites to provide accurate recommendation to users.

However, currently many recommendation systems in use are mainly accuracy-oriented since they are mainly based on item similarity measures. Such systems are designed to predict items that are similar (accurate or close enough) to the existing list of items (for example, if a user has listened to an album of Mozart, a recommender system recommends only other albums by Mozart or at least, other classical composers). The most common metrics to evaluate the accuracy measures are the Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE). There are significant disadvantages for such accuracy-based recommender engines [3]. Accuracy-based engines do not consider factors such as proliferation of varied user interests and desire for changes. This results in a muted user experience that is generated from a constrained and narrow feature set. There is no room for user's personal growth and experience. Thus, recommender systems should also consider other important metrics outside of accuracy such as coverage, novelty, serendipity, unexpectedness and usefulness. Briefly, serendipity is defined as the accident of finding something good or useful while not specifically searching for it. Serendipity is thus closely related to unexpectedness and involves a positive emotional response of the user about a previously unknown item. It measures how surprising the unexpected recommendations are [4]. In other words, serendipity is concerned with the novelty of recommendations and in how far recommendations may positively surprise users [5]. Adamopoulos [6] has proposed a method to improve user satisfaction by generating unexpected recommendations based on the utility theory of economics. A discovery-oriented collaborative filtering algorithm for deriving novel recommendations has been proposed in Hijikata et al. [7]. Andre et al.[8] examine the potential for serendipity in Web search and suggest that information about personal interests and behavior may be used to support serendipity. Their algorithms, in addition to building a User preference, build another profile of Users Known and Unknown Items. Novel recommendations are then given based on them. Chhavi Rana [9] has proposed a methodology based on temporal parameters to include novelty and serendipity in recommender systems. Ziegler et al.[10] assume that diversifying recommendation lists improves user satisfaction. They proposed topic diversification, which diversifies recommendation lists, based on an intra-list similarity metric. In [11], it has been proposed to recommend items whose description is semantically far from users profiles. Kawamae [12] suggests a recommendation algorithm based on the assumption that users follow earlier adopters who have demonstrated similar preferences.

Serendipity is becoming a popular topic of research in the current recommender systems to enhance user experiences. This thesis will specifically deal with developing a recommender system that incorporates serendipity as a factor for enriching the predictions. Furthermore, the quality of the predicted serendipitous items will be improved using the contents or attributes of the items.

List of References

[1] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of recommendation algorithms for e-commerce," in *Proceedings of the 2nd ACM conference on Electronic commerce*. ACM, 2000, pp. 158–167.

[2] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations," in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2002, pp. 253–260.

[3] S. M. McNee, J. Riedl, and J. A. Konstan, "Being accurate is not enough: how accuracy metrics have hurt recommender systems," in *CHI'06 extended abstracts on Human factors in computing systems*. ACM, 2006, pp. 1097–1101.

[4] G. Shani and A. Gunawardana, "Evaluating recommendation systems," in *Recommender systems handbook*. Springer, 2011, pp. 257–297.

[5] M. Ge, C. Delgado-Battenfeld, and D. Jannach, "Beyond accuracy: evaluating recommender systems by coverage and serendipity," in *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 2010, pp. 257–260.

[6] P. Adamopoulos and A. Tuzhilin, "On unexpectedness in recommender systems: Or how to better expect the unexpected," *ACM Transactions on Intelligent Systems and Technology*, vol. 1, no. 1, pp. 1–51, 2013.

[7] Y. Hijikata, T. Shimizu, and S. Nishida, "Discovery-oriented collaborative filtering for improving user satisfaction," in *Proceedings of the 14th international conference on Intelligent user interfaces*. ACM, 2009, pp. 67–76.

[8] P. André, J. Teevan, and S. T. Dumais, "From x-rays to silly putty via uranus: serendipity and its role in web search," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 2033–2036.

[9] C. Rana, "New dimensions of temporal serendipity and temporal novelty in recommender system," *Advances in Applied Science Research*, vol. 4, no. 1, pp. 151–157, 2013.

[10] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *Proceedings of the 14th international conference on World Wide Web*. ACM, 2005, pp. 22–32.

[11] L. Iaquinta, M. de Gemmis, P. Lops, G. Semeraro, and P. Molino, "Can a recommender system induce serendipitous encounters," *E-Commerce*, pp. 227–243, 2010.

[12] N. Kawamae, “Serendipitous recommendations via innovators,” in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval.* ACM, 2010, pp. 218–225.

CHAPTER 2
COLLABORATIVE FILTERING METHODS

Most recommender systems take either of two basic approaches: collaborative filtering or content-based filtering. Collaborative filtering (CF) is one of the most successful approaches to building recommender systems [1]. In order to establish recommendations, CF systems need to relate two fundamentally different entities: items and users.There are two primary approaches to facilitate such a comparison, which constitute the two main techniques of CF: the neighborhood approach and the model-based approach.

2.1 NEIGHBORHOOD APPROACH

Neighborhood methods focus on relationships between users called user-user CF or between items called item-item CF. Neighborhood-based methods are also commonly referred to as memory-based approaches.

2.1.1 USER-USER CF METHOD

A user-user neighborhood approach models the preference of a user to an item based on ratings of similar users for the same item. User-user CF is a straightforward algorithmic interpretation of the core premise of CF: find other users whose past rating behavior is similar to that of the current user and use their ratings on other items to predict what the current user will like [2]. The fundamental ingredients for CF are: (i) rating matrix R that specifies the *item, user, rating/preference* tuple, (ii) a similarity function $sim(u, v)$ between user u and v, and (iii) a method for using similarities and ratings to generate predictions. The ratings matrix R is a user input and an example is shown in Table.2.1.1.

Commonly used similarity functions include cosine-based and Pearson-

Items $\rightarrow$

Users $\downarrow$

	1	2	$\ldots$	i	$\ldots$	N_{items}	Average rating
1	3	1					$\bar{r}_1$
2	2				3	4	$\bar{r}_2$
v	5	2		$r_{v,i}$	5	1	$\bar{r}_v$
$\vdots$							$\vdots$
u	3			$r_{u,i}$	5	1	$\bar{r}_u$
$\vdots$							$\vdots$
N_{users}	4			3	1	5	$\bar{r}_{N_{users}}$

Table 1. Ratings matrix that consists of $N_{users} \times N_{items}$ entries of ratings. The blank entries denote unrated items. $r_{u,i}$ is the rating of item i by user u and $r_{v,i}$ is the rating of the same item i by another user v. $\bar{r}_u$, $\bar{r}_v$ are the average ratings of users u and v.

correlation [3] based measures. Other similarity measures used in the literature also include, Spearman rank correlation, Kendalls τ correlation, mean squared differences, entropy, and adjusted cosine similarity [4]. The Pearson-correlation based similarity function is given by:

$$sim(u,v) = \frac{(\vec{r}_u - \bar{r}_u) \cdot (\vec{r}_v - \bar{r}_v)}{(\sqrt{\vec{r}_u - \bar{r}_u) \cdot (\vec{r}_u - \bar{r}_u)}\sqrt{(\vec{r}_v - \bar{r}_v) \cdot (\vec{r}_v - \bar{r}_v)}}. \tag{1}$$

The cosine-based similarity function is given by:

$$sim(u,v) = \frac{\vec{r}_u \cdot \vec{r}_v}{\sqrt{\vec{r}_u \cdot \vec{r}_u}\sqrt{\vec{r}_v \cdot \vec{r}_v}}. \tag{2}$$

In Eqns.1 and 2, $\vec{r}_u$ is the vector of items rated by user u that are also rated by user v, and $\vec{r}_v$ is similarly defined for user v. Scalars $\bar{r}_u$ and $\bar{r}_v$ denote the average ratings of users u and v.

Pearson-correlation and cosine-based similarity functions are both invariant to scaling. This implies that multiplying the ratings of users by a constant does not change the similarities between users. Pearson-correlation unlike cosine-based similarity is also invariant to adding a constant to the users' ratings. For example, if $\vec{r}_u^{\,mod} = a\ \vec{r}_u + b$, where a and b are constants, then the Pearson-correlation function value remains invariant. This is given by the following equation.

$$\begin{aligned} sim(u,v) &= \frac{(\vec{r}_u - \bar{r}_u)\cdot(\vec{r}_v - \bar{r}_v)}{(\sqrt{\vec{r}_u - \bar{r}_u)\cdot(\vec{r}_u - \bar{r}_u)}\sqrt{(\vec{r}_v - \bar{r}_v)\cdot(\vec{r}_v - \bar{r}_v)}} \\ &= \frac{(\vec{r}_u^{\,mod} - \bar{r}_u^{\,mod})\cdot(\vec{r}_v - \bar{r}_v)}{(\sqrt{\vec{r}_u^{\,mod} - \bar{r}_u^{\,mod})\cdot(\vec{r}_u^{\,mod} - \bar{r}_u^{\,mod})}\sqrt{(\vec{r}_v - \bar{r}_v)\cdot(\vec{r}_v - \bar{r}_v)}}. \end{aligned} \tag{3}$$

This is an important property because it implies that the Pearson-correlation based similarities between users do not depend on the absolute values of their ratings but only on the way they vary. This is one of the main reasons for the wide popularity of Pearson-correlation coefficient.

Finally, to generate predictions or recommendations for a user u the user-user CF first uses $sim(u,v)$ to compute a neighborhood $N \subseteq N_{users}$ of neighbors of u. N is usually taken as the top-N users based on similarity scores (based on $sim(u,v)$ sorted from highest to lowest scores). Alternately, N can also be based on a prescribed threshold for the similarity score $sim(u,v)$. For example, if the threshold is specified as 0.8, N consists of only those users for whom $sim(u,v) \geq 0.8$. Once N has been computed, the ratings of users are combined in N to generate predictions p for user u preference for an item i. This is typically done by computing the weighted average of the neighboring users ratings of i using similarity as the weights:

$$p_{u,i} = \frac{\sum\limits_{k\in N} r_{k,i}\ sim(u,k)}{\sum\limits_{k\in N} sim(u,k)}. \tag{4}$$

2.1.2 ITEM-ITEM CF METHOD

Although user-user CF filtering techniques are popular, they suffer from scalability issues associated with the frequent computation of similarity between users. When a user changes the rating of items frequently, the rating vector of such a user changes which modifies the similarity with others. Hence, the user neighborhood N for a given user cannot be pre-computed but has to be evaluated whenever recommendations are needed. This can be a big computational bottleneck for large datasets. This effect is amplified when there are more users than items that are typical of many e-commerce websites.

To eliminate this scalability issue, an item-item based CF technique was proposed by Linden et al. [5] (see, [6], [7]). These algorithms analyze the similarity between items instead of predicting the ratings based on the similarity between users. If two items tend to have the preferences from the same users, then they are similar and users are expected to have similar preferences for similar items. In systems with a high user to item ratio, frequent changing of ratings of an item by a user is unlikely to change the similarities between items since each item has far more ratings from many users that do not change. Hence, change of ratings by a very small set of users will only slightly change alter the similarity between items [2] and the users will still get good recommendations. The item-item CF method is similar to user-user CF except that the item similarity is deduced from user preference patterns rather than extracted from item data. The Pearson correlation similarity and the adjusted cosine similarity are examples of the common similarity metrics that are used to predict the similarity between items in such systems.

2.2 MODEL-BASED APPROACH

Model-based methods, fit a parametric model to the training data that can later be used to predict unseen ratings and issue recommendations. Latent factor and matrix factorization models have emerged as a state of the art methodology in this class of techniques. In its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from item rating patterns. High correspondence between item and user factors leads to a recommendation. These methods have become popular in recent years by combining good scalability with predictive accuracy. In addition, they offer much flexibility for modeling various real-life situations [8]. Other methods include cluster-based CF [9], Bayesian classifiers [10], and regression-based methods [11]. The slope-one method [12] fits a linear model to the rating matrix, achieving fast computation and reasonable accuracy.

List of References

[1] X. Su and T. M. Khoshgoftaar, “A survey of collaborative filtering techniques,” *Advances in artificial intelligence*, vol. 2009, p. 4, 2009.

[2] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan, “Collaborative filtering recommender systems,” *Foundations and Trends in Human-Computer Interaction*, vol. 4, no. 2, pp. 81–173, 2011.

[3] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, “Evaluating collaborative filtering recommender systems,” *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

[4] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, “An algorithmic framework for performing collaborative filtering,” in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval.* ACM, 1999, pp. 230–237.

[5] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: Item-to-item collaborative filtering,” *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76–80, 2003.

[6] G. Karypis, “Evaluation of item-based top-n recommendation algorithms,” in *Proceedings of the tenth international conference on Information and knowledge management.* ACM, 2001, pp. 247–254.

[7] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the 10th international conference on World Wide Web.* ACM, 2001, pp. 285–295.

[8] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[9] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering,” in *Proceedings of the fifth international conference on computer and information technology*, vol. 1. Citeseer, 2002.

[10] K. Miyahara and M. J. Pazzani, “Collaborative filtering with the simple Bayesian classifier,” in *PRICAI 2000 Topics in Artificial Intelligence*. Springer, 2000, pp. 679–689.

[11] S. Vucetic and Z. Obradovic, “Collaborative filtering using a regression-based approach,” *Knowledge and Information Systems*, vol. 7, no. 1, pp. 1–22, 2005.

[12] D. Lemire and A. Maclachlan, “Slope-one predictors for online rating-based collaborative filtering.” in *SDM*, vol. 5. SIAM, 2005, pp. 1–5.

CHAPTER 3

INTRODUCING SERENDIPITY TO RECOMMENDER SYSTEMS

Serendipity is defined as the accident of finding something good or useful while not specifically searching for it. In other words, serendipity is concerned with the novelty of recommendations and in how far recommendations may positively surprise users [1]. In recommender systems, it is defined as a measure that indicates how the recommender system can find unexpected and useful items for users. In this chapter, we propose and implement a new algorithm for generating a list of serendipity items using collaborative filtering techniques. The novelty of the current work is the proposed improvement of the quality of the serendipity list by incorporating the items' contents. It is hypothesized that the degree of ambiguity in defining serendipity is reduced by incorporating the items' contents together with the ratings.

The algorithm and its implementation are explained in detail in the following sections.

3.1 SERENDIPITY RECOMMENDER SYSTEM

The proposed serendipity algorithm is given in Algorithm 3.1.

Algorithm 1 Proposed algorithm for generating serendipity items

```
1: Build a recommendation list of items using a primitive method
2: Build a set of recommendation lists using collaborative filtering techniques
3: Predict a list of “unexpected” items using the results of first two steps
4: Generate a list of serendipity items from the unexpected items
5: Improve the quality of the serendipity list by incorporating items’ contents
```

3.1.1 PRIMITIVE RECOMMENDER METHOD

The first step in the generation of serendipity items is predicting a list of items using a primitive recommendation method which is based on the highest average ratings and highest number of users (popularity). Consequently, the level of expectedness for the user is generally high for all the predicted items using this method. Let the predicted list be denoted by PPM. The PPM model used in this study will be based on the top-N items with the highest average rating and the top-N items with the largest number of ratings. These two top-N rating will be combined as a union to produce a list of top-K items (where K is a user-specified number) of the PPM recommendation list [2].

3.1.2 EXPECTED LIST OF RECOMMENDATIONS

In this study, an expected set of recommendation items is generated using collaborative filtering methods, particularly using the user-user method as described in Section.2.1.1. The basic algorithm for user-based collaborative method is given below:

Algorithm 2 User-based recommendation method

```
1: procedure UserBasedRecommender(User $u$)
2:     for each item $i$ not rated by user $u$ do
3:         for each user $v$ that has rated $i$ do
4:             Compute $sim$ = Similarity($u$,$v$)
5:             Compute weighted moving average using $v$'s rating of $i$ and $sim$
6:         end for
7:     end for
8:     Return recommended list of items $RS$ and their scores for $u$
9: end procedure
```

In Algorithm 2, “Similarity” refers to a user-based similarity measure. Specifically, the measures employed in the present study include distance-based, cosine-based, and Pearson-correlation based metrics.

3.1.3 UNEXPECTED LIST OF RECOMMENDATIONS

One of the key ingredients of serendipitous items is a high degree of unexpectedness. As pointed earlier, the PPM list consists of items with a high degree of expectedness. The RS list, on the other hand, contains items with a varied spectrum of expectedness. The unexpected list of items is generated using the method specified in [3, 1, 2].

$$UNEXP = \{x \mid x \in RS \text{ and } x \notin PPM\}. \tag{5}$$

$UNEXP$ is the list of items that are in RS list but not in PPM.

3.1.4 PREDICTED SERENDIPITY LIST

Given the $UNEXP$ list of items, a predicted list of serendipity items $SERENDIP_P$ is generated by filtering the items in terms of an "usefulness" metric. All items whose ratings are greater than or equal to a chosen value are considered "useful" items to be recommended as serendipitous. The items are sorted from highest to lowest ratings in $SERENDIP_P$.

3.1.5 IMPROVED SERENDIPITY LIST

The quality of the list is further improved by rating the items in $SERENDIP_P$ based on the items' contents or attributes. For example, in the MovieLens database, an important content is the movie genre. The items in $SERENDIP_P$ are once again rated based on their genre. Let this list be denoted by $SERENDIP_P^t$, where t represents the attribute of interest.

Central Hypothesis:

Those movies that are in the predicted serendipity list and which are ranked lower in the genre-based ratings are considered to better satisfy serendipity. For example, let us assume that an item in the predicted serendipity list is rated 4.5 (on a 5 scale) using a user-based recommender system without taking into account the genre. If the same item is rated lower but above the usefulness threshold using a genre based recommender, it is considered more serendipitous because it is both useful and more unexpected.

The genre-based recommendation list is generated using a modified procedure as outlined in [4]. The key steps in the genre-based evaluation are the following:

Algorithm 3 Algorithm for evaluating recommendation based on genre

```
1: Compute average rating of each genre for a given user (Eqn.6)
2: Compute average rating of each item based on average rating for each genre
   obtained from Step 1 (Eqn.7)
3: Compute genre-based recommendation based on user-similarity and ratings
   computed from Step 2. (Eqn.8)
```

Following [4], let the attribute vector of a given item *item* be denoted by $Attrib(item) \subset (1, 2, 3, \ldots, Nattrib)$. Let $A^u_{item,gnr}$ be the value of attribute gnr for item *item* and for user u. For the MovieLens database, 19 genres for each movie *item* are represented by a 19×1 vector with value 1 in position gnr if it belongs to genre gnr and 0 if it does not. Then, $A^u_{item,gnr} \in (0, 1)$.

$$\bar{r}^{gnr}_u = \frac{\sum\limits_{item=0}^{Nitems} r_{u,item}\ A^u_{item,gnr}}{M_{u,gnr}} \tag{6}$$

$$r_{u,item} = \frac{\sum\limits_{gnr=0}^{Nattrib} \bar{r}^{gnr}_u\ A^u_{item,gnr}}{N_{u,item}} \tag{7}$$

$$p_{u,item} = \frac{\sum\limits_{k=0}^{Nusers} sim(u,k) r_{k,item}}{\sum\limits_{k=0}^{Nusers} sim(u,k)} \tag{8}$$

Here, $\bar{r}^{gnr}_u$ is the average rating of a genre gnr for a given user u. If no movies belong to gnr, such a genre is not taken into account. Also, $p_{u,item}$ is the genre-based predicted rating of user u for item $item$, $r_{k,item}$ is the genre-based rating of user k ($\neq u$) for item $item$, $N_{u,item}$ is the number of non-zero attributes for item i, and $M_{u,gnr}$ is the number of valid items for user u that belongs to attribute gnr.

Graphically, Algorithm 3.1.5 is represented as in Fig.1.

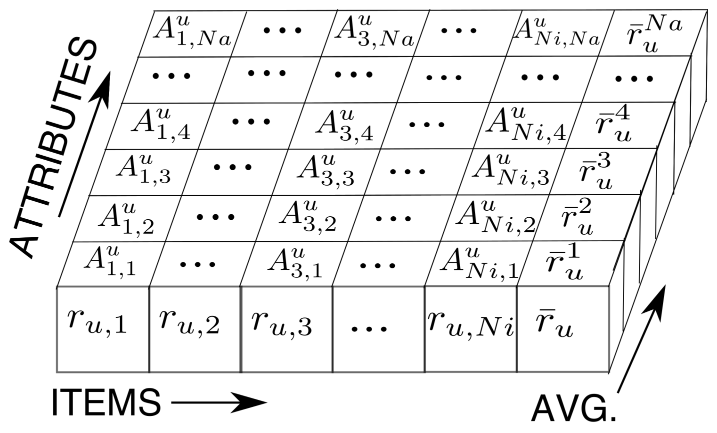

Figure 1. Graphical representation of a genre-based recommendation evaluation. The various terms appearing in the figure are evaluated using Eqns.6 - 8. The figure is mostly reproduced from the schematic provided in Fig. 1 of reference [4].

3.2 PYTHON AND NUMPY

3.2.1 REASONS FOR CHOOSING PYTHON

Python is an interpreted, high-level language that has easy-to-read syntax. The native ability to interact with data structures and objects with a wide range of built-in functionality makes it easier to write scientific programs. Moreover, it abstracts most of the memory management layers from the end users. This facilitates researchers and scientists to spend more time exploring various ideas than how to code them. The fast development time of Python scripts makes it much easier to test new ideas with prototypes.

An example provided in [5] for a “Hello World” program illustrates the difference in readability between Python and C++.

```
/*
   A C++ program to print "Hello World"
*/
#include <iostream.h>
void main()
{
    cout << "Hello World" << endl;
}
```

In Python, the above code reads as

```
print "Hello World"
```

Because of the above reasons, Python was chosen as the language to implement the serendipity recommender system for the present work. Also, Python is free and open source.

NumPy is a Python extension module that provides efficient operation on arrays of homogeneous data. It allows python to serve as a high-level language for manipulating numerical data.

3.2.2 EXAMPLE SCRIPT TO ILLUSTRATE READABILITY

An example for computing a sample-based Pearson correlation coefficient is given in the following code snippet implemented in the present study. Lines beginning with # are comments in a Python script. Here, "np" denotes the NumPy utility. Recall, sample-based Pearson similarity between two users is given by Eqn.1 which is repeated here for convenience.

$$sim(u,v) = \frac{(\vec{r_u} - \bar{r}_u)\cdot(\vec{r_v} - \bar{r}_v)}{(\sqrt{\vec{r_u} - \bar{r}_u)\cdot(\vec{r_u} - \bar{r}_u)}\sqrt{(\vec{r_v} - \bar{r}_v)\cdot(\vec{r_v} - \bar{r}_v)}}. \quad (9)$$

where, $\vec{r_u}$ is the vector of items rated by user u that are also rated by user v, and $\vec{r_v}$ is similarly defined for user v. Scalars $\bar{r}_u$ and $\bar{r}_v$ denote the average ratings of users u and v (see, Table 2.1.1).

```
#========================================================
# Given a (num_users x num_items) matrix of ratings
# and average ratings of each user:
# Compute a sample-based Pearson correlation coefficient
# between users "i" and "j"
#========================================================
def pearson_similarity_sample(ratings_matrix,
                              avg_ratings,
                           i, j):
#=====================================================
# Mask all those items for each user that have not
# been rated by either of them.
# This mask is a logical vector of booleans.
# true  -- if both users have rated an item
# false -- otherwise
# It’s size is the same as rating vector of i (or j).
# Note:
#      Unrated items are marked by 0.
#=====================================================
    mask = np.logical_and((ratings_matrix[i, :] > 0),
                          (ratings_matrix[j, :] > 0))
# If user has not rated any item return a 0 value
    if np.sum(mask) == 0:
        return 0
    r_i = ratings_matrix[i,:] - avg_ratings_of_users[i]
    r_j = ratings_matrix[j,:] - avg_ratings_of_users[j]
#=====================================================
# Compute dot products of only unmasked elements of
# the vectors r_i and r_j
#=====================================================
    numerator = np.dot(r_i[mask], r_j[mask])
    norm_ri   = np.sqrt(np.dot(r_i[mask], r_i[mask]))
    norm_rj   = np.sqrt(np.dot(r_j[mask], r_j[mask]))
    denom     = norm_ri * norm_rj
    if denom == 0 :
        return 1
    return numerator/denom
```

List of References

[1] M. Ge, C. Delgado-Battenfeld, and D. Jannach, "Beyond accuracy: evaluating recommender systems by coverage and serendipity," in *Proceedings of the fourth ACM conference on Recommender systems.* ACM, 2010, pp. 257–260.

[2] P. Adamopoulos and A. Tuzhilin, "On unexpectedness in recommender systems: Or how to expect the unexpected," in *Workshop on Novelty and Diversity in Recommender Systems (DiveRS 2011), at the 5th ACM International Conference on Recommender Systems (RecSys' 11).* Chicago, Illinois, USA: ACM, 2011, pp. 11–18.

[3] T. Murakami, K. Mori, and R. Orihara, "Metrics for evaluating the serendipity of recommendation lists," in *New frontiers in artificial intelligence.* Springer, 2008, pp. 40–46.

[4] T.-H. Kim and S.-B. Yang, "Using attributes to improve prediction quality in collaborative filtering," in *E-Commerce and Web Technologies.* Springer, 2004, pp. 1–10.

[5] A. B. Downey, J. Elkner, and C. Meyers, *Think Python: How to think like a computer scientist.* Wellesley, Massachusetts: Green Tea Press, 2008.

CHAPTER 4
RESULTS AND DISCUSSION

The serendipity recommender algorithm developed in the previous chapter is tested on the widely used MovieLens dataset from the GroupLens Research Center [1]. Representative results are presented and discussed in this chapter.

4.1 DESCRIPTION OF INPUT DATA FILES

The MovieLens dataset consists of 100,000 preferences for 1,682 movies rated by 943 customers. The customer preferences are represented as integer values from 1 to 5. A higher value means higher preference. The database from GroupLens consist of various files and the description of the files used in the present study are provided in Table.4.1.

Filename	Description
u.data	Tab separated list of 100000 ratings by 943 users on 1682 items that are randomly ordered. Users and items are numbered starting from 1. The columns are user id, item id, rating, and timestamp.
u.info	The number of unique users, unique items, and total ratings in u.data
u.item	Information about the items (movies); this is a tab separated list of: movie id, movie title, release date, video release date, IMDB URL, unknown (genre), Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western. The last 19 fields are the genres. A value of 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once. The movie ids are the ones used in the u.data.

Table 2. Description of the input data for the 100K ratings database from GroupLens Research Center[1]

4.1.1 PREPARATION OF RATINGS MATRIX

Given the set of input files listed in Table.4.1, a ratings matrix of size $Nusers \times Nitems$ is prepared first. Here, $Nusers$ is the number of unique users and $Nitems$ is the number of unique items. The python function to generate the ratings matrix is shown below.

```
def parse_and_prep_movielens_data():
#=====================================================
# It was easier to read some of the data files into
# MS-Excel and export them as CSV files.
# The CSV files were parsed using "numpy.genfromtxt"
# utility.
#=====================================================
# Parse u.data
    data_ratings = np.genfromtxt('ml-100k/u.data',
                                 delimiter='\t',
                                 dtype=[('userID',int),
                                        ('itemID',int),
                                        ('rating',float),
                                        ('timestamp',int)])
# Parse u.info
    data_user  = np.genfromtxt('ml-100k/u_user.csv', delimiter=',',
                               dtype=[('userID',int), ('age',int),
                                      ('gender', 'S1'),
                                      ('occupation', 'S20'),
                                      ('zipcode', int)])
# Parse u.item
```

```
    data_items_genre = np.genfromtxt('ml-100k/u_item_rev2.csv',
                                     delimiter=',',
                                     dtype=[('itemID',int),
                                            ('genre','19float')])
    nitems = data_items_genre.shape[0]
    nusers = data_user.shape[0]
# Prepare the ratings matrix of size Nusers x Nitems.
    ratings_matrix = np.zeros((nusers, nitems))
    for i in xrange(data_ratings.shape[0]):
        userid = data_ratings[i]['userID']
        itemid = data_ratings[i]['itemID']
        rating = data_ratings[i]['rating']
        ratings_matrix[userid-1, itemid-1] = rating
```

4.2 VALIDATION

The in-house code is first validated against Apache Mahout [2]. We selected Mahout because it is a workbench platform that provides many of the desired characteristics required for a recommender engine development. Mahout is also a production-level, open-source software and consists of a wide range of commonly used collaborative filtering algorithms that are easy to use for validation purposes. Some of the recent studies have used Mahout as their preferred platform include [3, 4, 5, 6, 7] which indicates its popularity.

4.2.1 DESCRIPTION OF TESTCASES

As a first test case, we consider the recommendation of top 20 movies for a random set of users using a user-based recommendation method and Pearson correlation coefficient. The recommended set of movies (IDs and scores) are then compared with the in-house developed code. The user IDs chosen are $(5, 121, 269, 842)$. While a random number generator can be used to pick the set of users, we have not done so for the present study. These users were, however, selected without any bias. The following methods and conditions have been used to validate the code.

1. A user-based recommendation is used because of its popularity.
2. Similarity metric based on Pearson correlation is used.
3. A similarity threshold value of 0.8 is used to select a small set of neighbors for a given user. Such a threshold is used reduce the computation time and is a necessary input in Mahout.

In Fig.2, the recommended item IDs for the chosen set of users are plotted. Excellent agreement is seen between the present and Mahout's results. There is however a small discrepancy in one of the items recommended for user 842 as can be seen from Table.4.2.1. It is observed that one of the movies is different between the two implementations. However, the movie ratings are identical for both, and hence different movies with same ratings are considered acceptable.

Comparison plots of the ratings of the recommended movies for different users are shown in Fig.3. Very good agreement is once again seen between the present results and Apache Mahout. The difference pointed out in Table.4.2.1 is apparent here as well. As can be seen, the ratings for these different movies are identical.

4.2.2 IMPORTANT OBSERVATIONS

During the validation phase, two important observations were made in Apache Mahout that was not apparent from the Java documentation of their methods.

First, the Pearson correlation metric used is applied to a population and not to a sample. Mathematically, the sample based Pearson correlation for two vectors

User ID	Recommended Movies
842 (Present)	[653 898 875 169 359 516 474 483 513 654 657 1084 285 493 524 57 318 498 774 **1159**]
842 (Mahout)	[653 898 875 169 359 516 474 483 513 654 657 1084 285 493 524 57 318 498 774 **862**]

Table 3. Recommended movies for user 842 using a user-based recommendation and Pearson correlation coefficient. The differing movie is highlighted in bold.

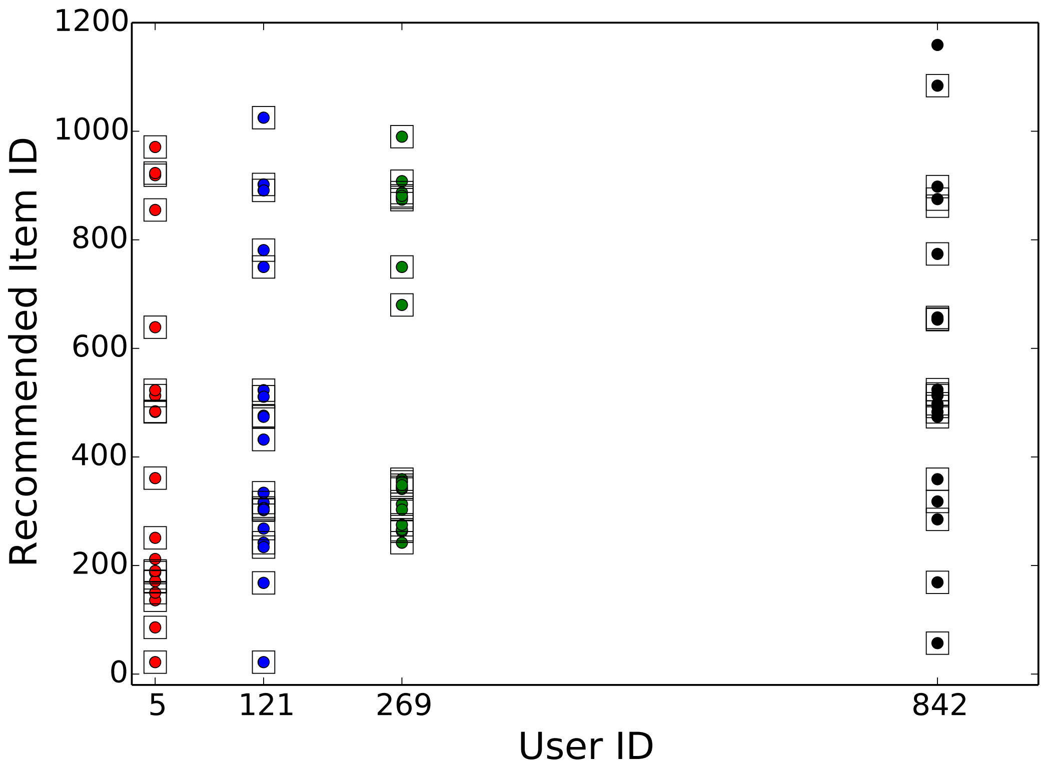

Figure 2. Comparison of recommendation results for a random set of users obtained from the current code with Apache Mahout. Here, □ denotes the predicted items from Mahout and • are the results from the present code. Color codes for the filled circles represent results for different users from the present code.

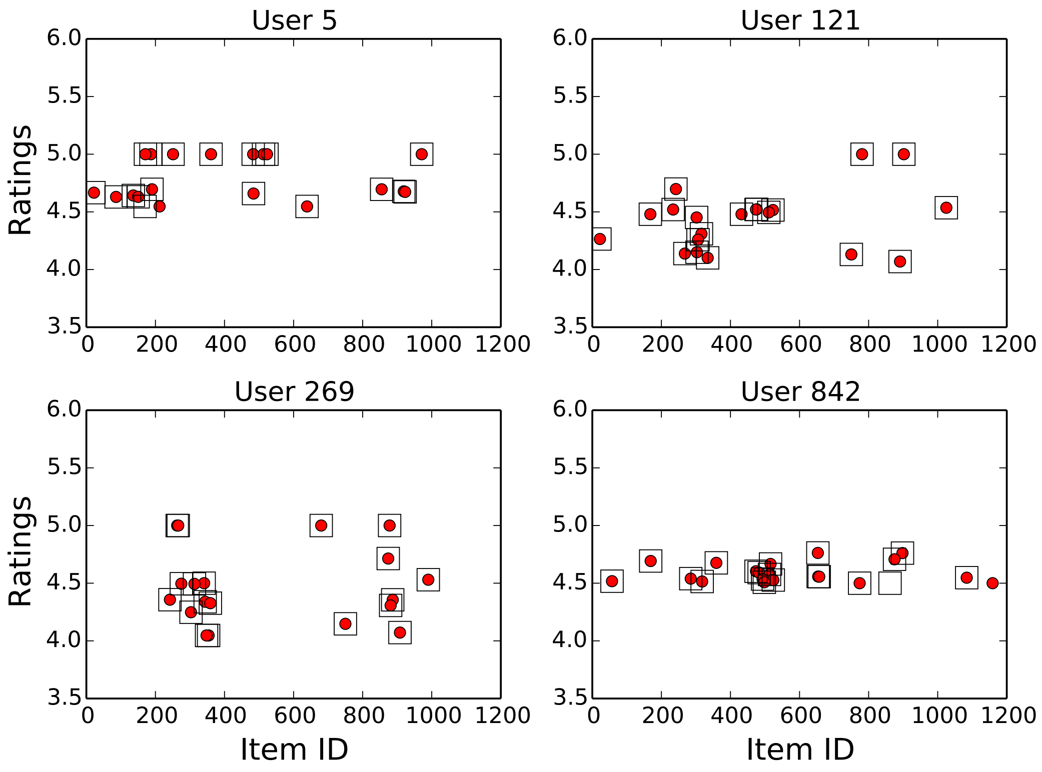

Figure 3. Comparison of recommendation movies vs. ratings for a random set of users obtained from the current code with Apache Mahout. Here, □ denotes the predicted items from Mahout and ● are the results from the present code.

$\vec{r_i}$ and $\vec{r_j}$ is expressed using Eqn.9. The population based Pearson correlation for the same vectors is expressed as:

$$sim(i,j) = \frac{\text{Cov}(\vec{r_i}, \vec{r_j})}{\sigma_{\vec{r_i}}\ \sigma_{\vec{r_j}}} \tag{10}$$

Here, $\text{Cov}(\vec{r_i}, \vec{r_j})$ is the covariance and σ is the standard deviation. It should be noted that the standard deviation is the square root of the variance. The python function to evaluate Eqn.10 is provided below.

```
def pearson_similarity_population(ratings_matrix,
                                  avg_ratings_of_users,
                                  i, j):
# Mask items where one of the users hasn’t rated the item
```

```
    mask = np.logical_and((ratings_matrix[i, :] > 0),
                          (ratings_matrix[j, :] > 0))
# if user has not rated any item or only 1 item
# return with a 0 value.
    if np.logical_or((np.sum(mask) == 0),
                     (np.sum(mask) == 1)):
        return 0
    r_i = ratings_matrix[i,:] - avg_ratings_of_users[i]
    r_j = ratings_matrix[j,:] - avg_ratings_of_users[j]
#=====================================================
# Numpy utility "cov" provides the variance
# and covariance values in a 2x2 matrix
# [0,1] entry is the covariance
# [0,0] is the variance of the first vector
# [1,1] is the variance of the second vector
# Sqrt of variance gives the standard deviation
#=====================================================
    var_covar_2x2_array = np.cov(r_i[mask], r_j[mask])
    numerator = var_covar_2x2_array[0,1]
    var0     = np.sqrt(var_covar_2x2_array[0,0])
    var1     = np.sqrt(var_covar_2x2_array[1,1])
    denom     = var0 * var1
    if denom < 1e-6 : # a small value
        return 0
    return numerator/denom
```

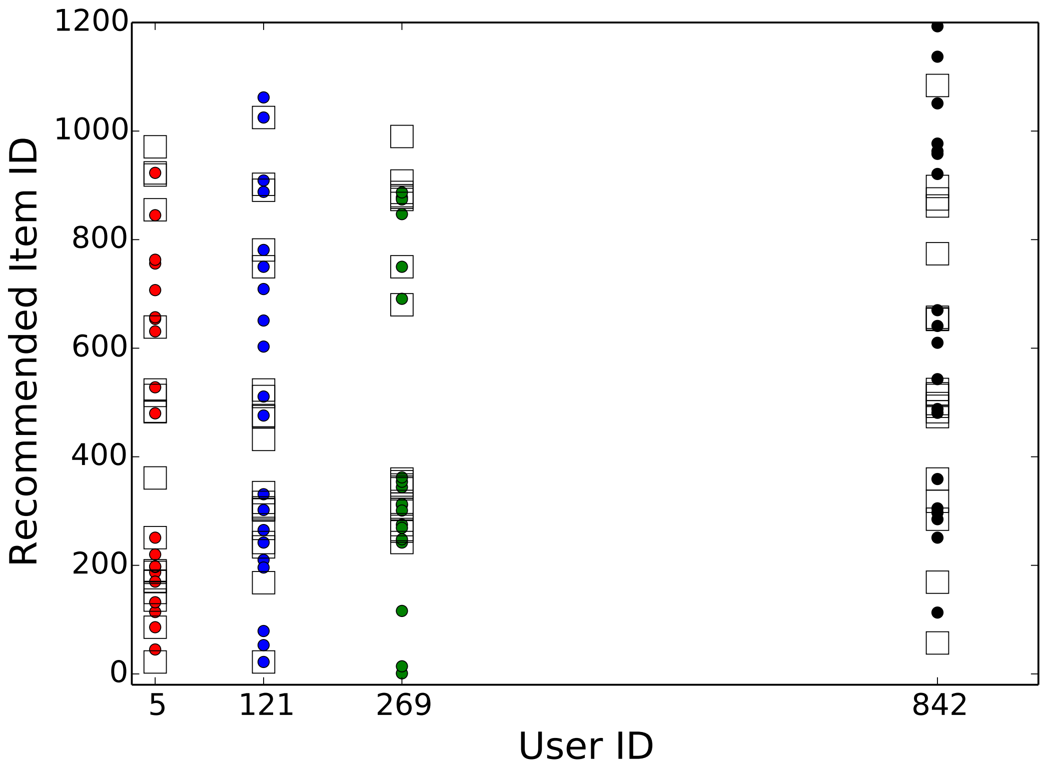

Figure 4. Comparison of recommendation results for a random set of users obtained from the current code using sample-based Pearson correlation with Apache Mahout. Here, □ denotes the predicted items from Mahout and • are the results from the present code. Color codes for the filled circles represent results for different users from the present code.

The results obtained from the sample based Pearson correlation are shown in Figs.4 and 5 that correspond to Figs.2 and 3. As can be seen, the results are dramatically different. This is because, the sample based technique given in Eqn.9 uses estimates of the covariance and variances of the vectors and therefore is not accurate.

The second important observation is that, in computing the ratings using a similarity weighted moving average, the total sum of the similarities must exceed a threshold that was found to be 1.0 by trial and error (this was not documented in Apache Mahout). This condition may be to avoid rating a movie that is rated

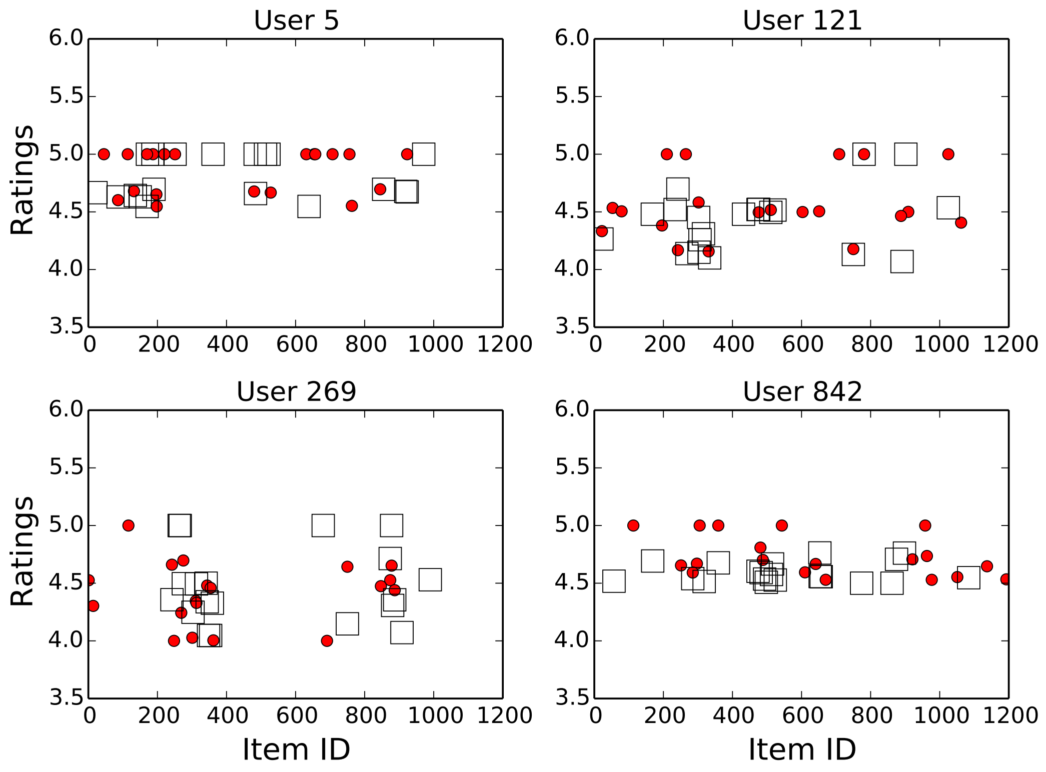

Figure 5. Comparison of recommendation movies vs. ratings for a random set of users obtained from the current code using sample-based Pearson correlation with Apache Mahout. Here, □ denotes the predicted items from Mahout and • are the results from the present code.

by only very few users.

4.3 RESULTS FOR SERENDIPITY

Based on the results and observation from the previous section, the serendipitous recommendation engine developed for the present study uses the following methods.

1. User-based recommendation.

2. Pearson correlation based on population.

3. A similarity-based threshold of 0.8 is used to identify nearest neighbors.

4. The PPM list is generated for top-100 movies.

5. The recommendation list RS is 100 movies.

Although many of the above assumptions can all be relaxed (such as using k-nearest neighbor selection, choosing larger values for RS and PPM), the above set has been chosen for demonstration purposes. It is straightforward to implement an item-based similarity metric as well but this has not been attempted in the present study. Incorporating various other methods for building recommendation engines will form a part of the future work.

A recommender system built using a standard collaborative filtering method predicts preferences based only on the ratings of the items. While this is useful and has proven to be largely successful, it may not consistently give high quality recommendations to the customers. This is because such recommendation systems do not consider other attributes of the items being recommended [8]. Another interpretation of this phenomenon is that such recommendation systems built on ratings alone have a higher probability of predicting items that are unexpected. Thus, there is an implicit unexpectedness built in recommendation systems that do not consider item attributes. Hence, in the present study, a list of unexpected items are first generated using the standard collaborative filtering method as has been done in the works of [9, 10]. Then a serendipity list is generated based on a "usefulness" metric as described in detail under Section 3.1.4. Let us denote this ratings of the predicted serendipity list by $SRDP$. In order to improve the level of serendipity, the predicted serendipity list is further rated using a genre-based collaborative technique as explained in Section 3.1.5. Let us denote the newly obtained ratings using the genres as $SRDP_{genre}$. The difference in these ratings are then normalized and then converted to percentage as given by Eqn. 11 and denoted by Δ_n. For the movies, the normalizing factor is taken as 5.0 because of the 1-5 rating scale.

$$\Delta_n = \frac{SRDP - SRDP_{genre}}{5.0} \times 100.0 \tag{11}$$

If Δ_n is negative ($SRDP_{genre} > SRDP$), it implies that the movie is more expected than unexpected because $SRDP_{genre}$ is generally considered more accurate than $SRDP$ [8]. Movies whose Δ_n are below a certain threshold can be removed from the serendipity list. For example, with $\Delta_n = -5\%$, a plot of the top 100 serendipity movies are shown for random users against the Δ_n values in Figs. 6 and 7. We see that a significant number of movies are less serendipitous based on the above threshold.

User ID	Number of less serendipitous movies
5	3
15	16
47	0
121	11
269	23
300	0
912	2

Table 4. The number of movies that are considered less serendipitous for different users in the predicted list of serendipity movies using a standard collaborative filtering method.

Table.4.3 lists the number of movies that are considered less serendipitous for different users using Eqn.11 and a threshold of -5%. We see that for some users, the numbers are 0. For such users, the standard method of predicting the serendipity list is sufficient. However, for other users such as 15, 121, and 269, a significant fraction is less serendipitous. Filtering out this fraction of movies will lead to better list of serendipitous movie prediction.

Thus, the newly suggested metric is shown to very useful in increasing the quality of serendipity by filtering out those movies that are rated higher by the genre based recommender compared to the standard ones. Therefore, our methodology

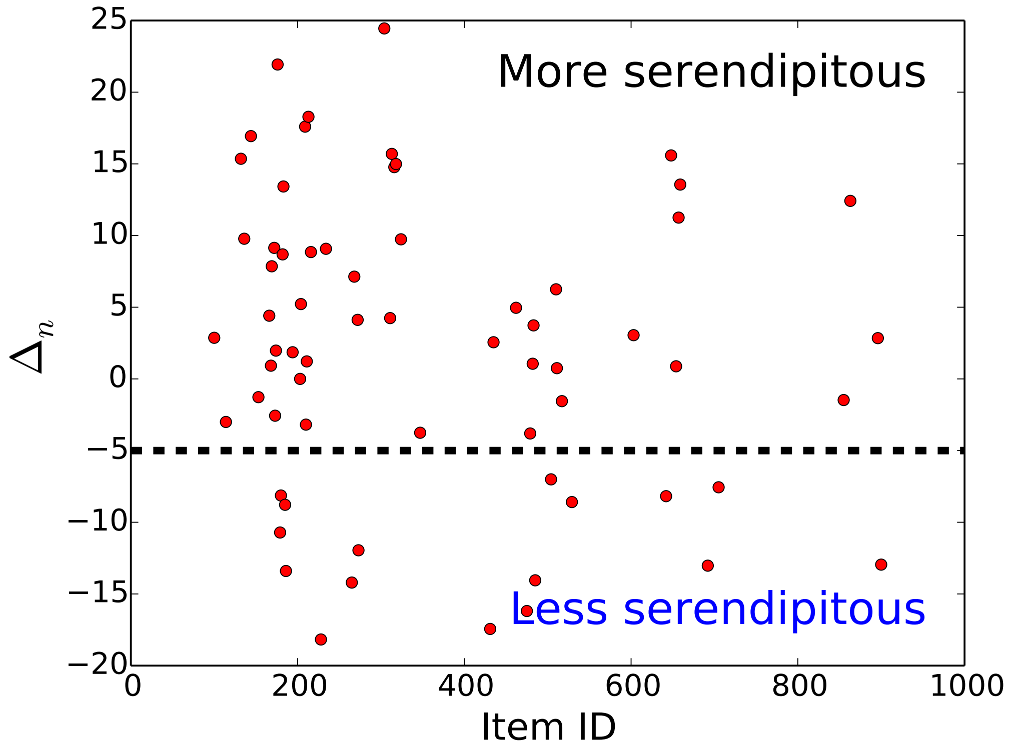

Figure 6. Plot of Δ_n from Eqn.11 for the serendipity list of movies generated by a standard collaborative filtering method for user 15. Δ_n below -5% are considered as less serendipitous. A significant fraction of movies has been identified as less serendipitous items using the present algorithm.

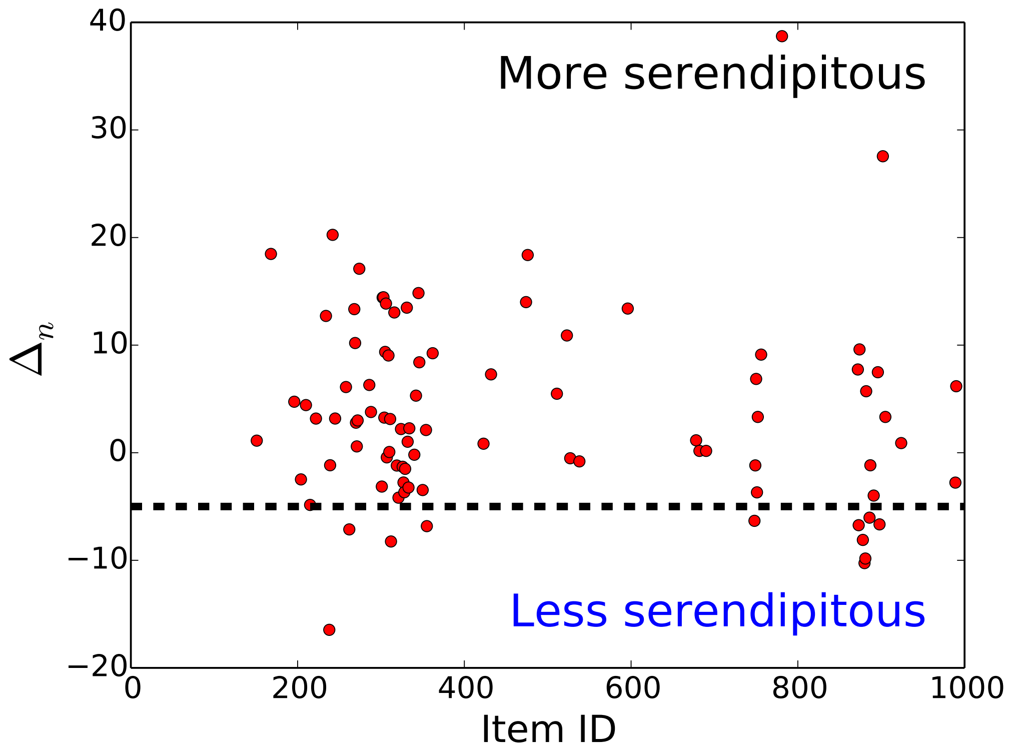

Figure 7. Plot of Δ_n from Eqn.11 for the serendipity list of movies generated by a standard collaborative filtering method for user 121. Δ_n below -5% are considered as less serendipitous. A significant fraction of movies has been identified as less serendipitous items using the present algorithm.

can aid in much better prediction of serendipitous movies and thereby improve user experience. A main advantage of our proposed algorithm is that for those items where attributes or contents cannot be explicitly defined, the predicted serendipity list (see, Section 3.1.4) can be used as the final list.

4.4 LIMITATIONS

Since our recommender engine is primarily based on collaborative filtering techniques, it suffers from the following problems:

1. **New User Problem:** When a new user is introduced, recommendations cannot be produced for them since the person may not have rated any movies.

This limitation could however be addressed using content-based filtering using parameters such as age, occupation, gender etc.,

2. **New Item Problem:** If an new item has not been rated, it will not participate in the recommendation list generation.

3. **Very large datasets:** The ratings matrix is a dense $N_{user} \times N_{items}$ matrix. This does not scale for massively large (big) datasets. This can be overcome by using sparse matrix representations of the ratings. The SciPy module for Python supports a variety of sparse data representations. Such representations will be incorporated in future work.

4. **Homogeneity of contents:** The in-house code is based on a single-content analysis which for the MovieLens dataset is its genre. Because of the homogeneous nature of the contents, the present code is not amenable to using multiple contents.

5. **Lack of experimental validation:** A rigorous validation of the proposed algorithm is possible only through real-user experiments and the quality of the ratings obtained from these experiments. Such a validation will further improve the quality of the serendipitous items by identifying the bounds on the difference in the ratings to be used in our computational algorithm (see, Eqn. 11).

List of References

[1] "Movielens Dataset, Grouplens Research Center."

[2] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in action.* Manning, 2011.

[3] R. M. Esteves and C. Rong, "Using mahout for clustering wikipedia's latest articles: a comparison between k-means and fuzzy c-means in the cloud,"

in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on.* IEEE, 2011, pp. 565–569.

[4] S. Guo *et al.*, "Analysis and evaluation of similarity metrics in collaborative filtering recommender system," 2014.

[5] S. Schelter and S. Owen, "Collaborative filtering with apache mahout," *Proc. of ACM RecSys Challenge*, 2012.

[6] C. E. Seminario and D. C. Wilson, "Case study evaluation of mahout as a recommender platform," in *Proceedings of the 6th ACM conference on recommender engines (RecSys 2012)*, 2012.

[7] S. G. Walunj and K. Sadafale, "An online recommendation system for e-commerce based on apache mahout framework," in *Proceedings of the 2013 annual conference on Computers and people research.* ACM, 2013, pp. 153–158.

[8] T.-H. Kim and S.-B. Yang, "Using attributes to improve prediction quality in collaborative filtering," in *E-Commerce and Web Technologies.* Springer, 2004, pp. 1–10.

[9] M. Ge, C. Delgado-Battenfeld, and D. Jannach, "Beyond accuracy: evaluating recommender systems by coverage and serendipity," in *Proceedings of the fourth ACM conference on Recommender systems.* ACM, 2010, pp. 257–260.

[10] P. Adamopoulos and A. Tuzhilin, "On unexpectedness in recommender systems: Or how to expect the unexpected," in *Workshop on Novelty and Diversity in Recommender Systems (DiveRS 2011), at the 5th ACM International Conference on Recommender Systems (RecSys' 11).* Chicago, Illinois, USA: ACM, 2011, pp. 11–18.

CHAPTER 5
CONCLUSION AND FUTURE WORK

In this work, a new method to improve the quality of the serendipitous list of items has been proposed. Previous works include the use of standard collaborative filtering methods and ratings to incorporate serendipity into recommender systems. In the present work, these methods have been extended to include the items' contents for enriching the quality of serendipitous items. Specifically, a standard user-based collaborative filtering method has been combined with attributes-based filtering in order to identify items with the potential of being less serendipitous. The new algorithm has been implemented using Python and NumPy. The code has been validated using a widely used open-source platform, namely, Apache Mahout. Excellent agreement has been shown. The new algorithm has been tested on the 100K MovieLens Dataset from GroupLens. It has been demonstrated that the new algorithm identifies a significant fraction of movies that are less serendipitous and which might not have been otherwise identified. Such identification may significantly improve the quality of recommending serendipitous items and thereby, improve user experience.

The future work on this project will address the following issues. The newly developed code can easily handle small data sets such the 100K MovieLens Dataset. With the advent of Big Data analytics, the current code is inefficient in handling very large data both due to the restrictions of the language and the type of data structures employed. Both these restrictions have to be addressed. In terms of the algorithms used, the present work has been built on collaborative filtering methods because of the popularity they have enjoyed thus far. Incorporating new and scalable methods such as clustering-based techniques for handling Big Data need to be addressed as well.

CHAPTER 6

APPENDIX

6.1 APPENDIX A: PYTHON CODE

```
"""CF_Serendipity.py


Project Goal: 1. Create a simple user-user CF recommendation engine
               using thr 100K dataset from movielens.org.
             2. Incorporate serendipity
             3. Improve serendipity using item contents


Creator: Puja Sridharan, URI
Advisor: Prof. Joan Peckham
"""
import numpy as np
from collections import defaultdict
def parse_and_prep_movielens_data():
#=====================================================
# It was easier to read some of the data files into
# MS-Excel and export them as CSV files.
# The CSV files were parsed using "numpy.genfromtxt"
#=====================================================
# Parse u.data
   data_ratings = np.genfromtxt('u.data',
                              delimiter='\t',
                              dtype=[('userID',int),
                                     ('itemID',int),
                                     ('rating',float),
                                     ('timestamp',int)])
# Parse u.info
    data_user = np.genfromtxt('u_user.csv', delimiter=',',
                              dtype=[('userID',int), ('age',int),
                                     ('gender', 'S1'),
                                     ('occupation', 'S20'),
                                     ('zipcode', int)])
# Parse u.item
    data_items_genre = np.genfromtxt('u_item_rev2.csv',
                                     delimiter=',',
                                     dtype=[('itemID',int),
                                            ('genre','19float')])
    np.sort(data_items_genre, order='itemID')
    nitems = data_items_genre.shape[0]
    nusers = data_user.shape[0]
# Prepare the ratings matrix of size Nusers x Nitems.
    ratings_matrix = np.zeros((nusers, nitems))
    for i in xrange(data_ratings.shape[0]):
        userid = data_ratings[i]['userID']
        itemid = data_ratings[i]['itemID']
        rating = data_ratings[i]['rating']
        ratings_matrix[userid-1, itemid-1] = rating
    return ratings_matrix, data_items_genre
#====================================================
def cosine_similarity(ratings_matrix, avg_ratings_of_users, i, j):
    # The following condition sets the location of items to "TRUE"
    # if both users have rated.
    # If not, it's set to FALSE. The boolean matrix is named "mask".
    # Note: rating = 0 => an unrated item
    mask = np.logical_and((ratings_matrix[i, :] > 0),
                          (ratings_matrix[j, :] > 0))
    if np.sum(mask) == 0 : # if user has not rated any item
        return 0
    r_i = ratings_matrix[i]
    r_j = ratings_matrix[j]
    numerator = np.dot(r_i[mask], r_j[mask])
    denom = np.linalg.norm(r_i[mask]) * np.linalg.norm(r_j[mask])
    if denom < 1e-6 : # a small value
        return 0
    return numerator/denom
#=====================================================
def pearson_similarity_population(ratings_matrix,
                                  avg_ratings_of_users,
                                  i, j):
    # Mask items where one of the users hasn't rated the item
    mask = np.logical_and((ratings_matrix[i, :] > 0),
                          (ratings_matrix[j, :] > 0))
    # if user has not rated any item or only 1 item
    if np.logical_or((np.sum(mask) == 0),
                     (np.sum(mask) == 1)):
        return 0
    r_i = ratings_matrix[i,:] - avg_ratings_of_users[i]
    r_j = ratings_matrix[j,:] - avg_ratings_of_users[j]
    # Numpy utility "cov" provides the variance
    # and covariance values in a 2x2 matrix
    var_covar_2x2_array = np.cov(r_i[mask], r_j[mask])
    # [0,1] entry is the covariance
    numerator = var_covar_2x2_array[0,1]
    # [0,0] is the variance of the first vector
    # [1,1] is the variance of the second vector
    # Sqrt of variance gives the standard deviation
    var0    = np.sqrt(var_covar_2x2_array[0,0])
    var1    = np.sqrt(var_covar_2x2_array[1,1])
    denom    = var0 * var1
    if denom < 1e-6 : # a small value
        return 0
    return numerator/denom
#========================================================
# Given a (num_users x num_items) matrix of ratings
# and average ratings of each user:
# Compute a sample-based Pearson correlation coefficient
# between users "i" and "j"
#========================================================
def pearson_similarity_sample(ratings_matrix,
                              avg_ratings,
                              i, j):
    #====================================================
    # Mask all those items for each user that have not
    # been rated by either of them.
    # This mask is a logical vector of "true" or "false".
    # true -- if both users have rated an item
    # false -- otherwise
    # It's size is the same size as rating vector of i (or j).
    # Note:
```

```
    #     Unrated items are marked by 0.
    #=====================================================
    mask = np.logical_and((ratings_matrix[i, :] > 0),
                          (ratings_matrix[j, :] > 0))
    # if user has not rated any item or only 1 item
    if np.sum(mask) == 0:
        return 0
    r_i = ratings_matrix[i,:] - avg_ratings[i]
    r_j = ratings_matrix[j,:] - avg_ratings[j]
    numerator = np.dot(r_i[mask], r_j[mask])
    norm_ri  = np.sqrt(np.dot(r_i[mask], r_i[mask]))
    norm_rj  = np.sqrt(np.dot(r_j[mask], r_j[mask]))
    denom    = norm_ri * norm_rj
    if denom == 0 :
        return 1
    return numerator/denom
#=====================================================
def topN_similarity(ratings, avg_ratings_of_users,
                    given_id, similarity_metric, N):
    scores = np.zeros((ratings.shape[0] - 1),
                      dtype=[('sim',float),('uid',int)])
    ctr = 0
    for uid in range(ratings.shape[0]):
        # do not compare a given id with itself.
        if uid != given_id :
            sim = similarity_metric(ratings,
                                    avg_ratings_of_users,
                                    given_id, uid)
            scores[ctr] = (sim, uid)
            ctr += 1
    # sort the array using the similarity score
    scores.sort()
    # reverse the order to get top scores first
    scores[:] = scores[::-1]
    return scores[0:N]
#=====================================================
def ratings_based_on_genre(ratings,
                        items_and_their_genre,
                        uid, itemid):
    num_genre = 19
    genre_matrix = items_and_their_genre['genre']
    ratings_genre_based = 0.0
    denom = 0.0
    ratings_genre_user = 0.0
    for genre in range(num_genre):
        vec1 = ratings[uid, :]
        vec2 = genre_matrix[:,genre]
        mask = np.logical_and((vec2 != 0.0), (vec1 != 0.0))
        sum_mask = np.sum(mask)
        if sum_mask > 0 :
            ratings_genre_user = np.dot(vec1[mask],vec2[mask]) \
                                /sum_mask
        if genre_matrix[itemid, genre] > 0.0 and \
           ratings_genre_user > 0.0 :
            ratings_genre_based += genre_matrix[itemid, genre] * \
                                    ratings_genre_user
            denom += 1.0
    if denom > 0.0:
        return ratings_genre_based/denom
    else:
        return 0
#====================================================
def recommendations_genre_based(ratings, items_and_genre,
                             avg_ratings_of_users,
                             similarity_metric,
                             similarity_threshold,
                             given_id):
    totals      = defaultdict(float)
    totals_genre = defaultdict(float)
    sim_sums    = defaultdict(float)
    EPS = 1e-6
    alluserIDs = ratings.shape[0]
    allitemIDs = ratings.shape[1]
    for itemid in range(allitemIDs):
        # if item is not rated by the given user
        if ratings[given_id, itemid] < (1.0-EPS) :
            for uid in range(alluserIDs):
                if uid != given_id:
                    # if the other user has rated the item
                    if ratings[uid, itemid] > (1-EPS) :
                        sim = similarity_metric(ratings,
                                  avg_ratings_of_users,
                                        given_id, uid)
                        if sim >= similarity_threshold :
                            ratings_genre_based = \
                            ratings_based_on_genre(ratings,
                                                 items_and_genre,
                                                 uid, itemid)
                            totals[itemid] += \
                             sim * ratings[uid, itemid]
                            totals_genre[itemid] += sim * \
                                       ratings_genre_based
                            sim_sums[itemid] += sim
    rankings = np.zeros(len(totals),
                 dtype=[('itemid',int), ('rating',float)])
    ctr = 0
    for itemid, total in totals.items():
        tot_sim = sim_sums[itemid];
        recommended_rating = 0.0;
#   The following condition that sum total of similarities > 1.0
#   is necessary to make our results perfectly agree with MAHOUT
        if tot_sim > 1.0 + EPS :
            recommended_rating = total/sim_sums[itemid]
        # store item id as fortran index
        item_id_recommended = itemid + 1
        rankings[ctr] = (item_id_recommended, recommended_rating)
        ctr += 1
    # sort the array using similarity score
    rankings.sort(order='rating')
    rankings[:] = rankings[::-1] # reverse the order

    rankings_genre = np.zeros(len(totals_genre),
                           dtype=[('itemid',int),
                           ('rating',float)])
    ctr = 0
    for itemid, total in totals_genre.items():
        tot_sim = sim_sums[itemid];
        recommended_rating = 0.0;
#   The following condition that sum total of similarities > 1.0
#   that made our results perfectly agree with MAHOUT
        if tot_sim > 1.0 + EPS :
            recommended_rating = total/sim_sums[itemid]
        # store item id as fortran index
        item_id_recommended = itemid + 1
        rankings_genre[ctr] = (item_id_recommended, \
                            recommended_rating)
        ctr += 1
    # sort the array using similarity score
    rankings_genre.sort(order='rating')
    rankings_genre[:] = rankings_genre[::-1] # reverse the order
    return rankings, rankings_genre
#====================================================
def average_ratings_based_on_genre(ratings, items_and_their_genre):
    num_genre = 19
    num_users = ratings.shape[0]
    num_items = ratings.shape[1]

    genre_matrix = items_and_their_genre['genre']

    ratings_genre_user = np.zeros((num_users,num_genre))
    ratings_genre_based = np.zeros((num_users,num_items))
    for uid in range(num_users):
        for genre in range(num_genre):
            vec1 = ratings[uid, :]
            vec2 = genre_matrix[:,genre]
            mask = (vec2 != 0.0)
            sum_mask = np.sum(mask)
            if sum_mask > 0 :
                ratings_genre_user[uid, genre] = \
                np.dot(vec1[mask],vec2[mask])/sum_mask

        for itemid in range(num_items):
            for genre in range(num_genre):
                vec1 = ratings_genre_user[uid, :]
                vec2 = genre_matrix[itemid, :]
                mask = (vec2 != 0.0)
                sum_mask = np.sum(mask)
                if sum_mask > 0 :
                    ratings_genre_based[uid, itemid] = \
                    np.dot(vec1[mask],vec2[mask])/sum_mask
#=====================================================
def topN_PPM(ratings_matrix, N):
#
# This primitive recommendation is based on
# picking top movies that has got the ratings
# from largest number of users and movies that got
# the highest average ratings.
#
```

```
mask = (ratings_matrix > 0)
# Sum the rows for each column which gives
# number of users for each item
number_of_users_for_items = mask.sum(axis = 0)

# 1. Compute the movie IDs with the largest number of users.
# 2. Use numpy "argsort" to get the original indices (itemIDs)
#    of the sorted array.
# 3. Zip the sorted array and item IDs into tuples
# 4. Reverse the sorted tuple from highest to lowest.
topN_users_for_items = \
np.array(zip(np.argsort(number_of_users_for_items),
            np.sort(number_of_users_for_items)),
        dtype=[('itemid',int), ('total_users',int)])
# Reverse sort
topN_users_for_items[:] = topN_users_for_items[::-1]

# 1. Compute the movie IDs with the highest average rating.
# 2. Use numpy "argsort" to get the original indices (itemIDs)
#    of the sorted array.
# 3. Zip the sorted array and item IDs into tuples
# 4. Reverse the sorted tuple from highest to lowest.
avg_ratings_of_items = np.divide(ratings_matrix.sum(axis=0),
                              number_of_users_for_items)
topN_ratings_for_items = \
np.array(zip(np.argsort(avg_ratings_of_items),
            np.sort(avg_ratings_of_items)),
        dtype=[('itemid',int), ('avg_rating',float)])
```

```
    # Reverse sort
    topN_ratings_for_items[:] = topN_ratings_for_items[::-1]

    total_movies = np.vstack((topN_users_for_items['itemid'],
                    topN_ratings_for_items['itemid'])).ravel()

    # Use numpy unique function to get the unique movies.
    # This automatically sorts the IDs as well
    return np.unique(total_movies)[0:N]
#=====================================================
def main():
#     users, items, ratings_matrix = parse_data_old()
    ratings_matrix, items_and_their_genre = \
    parse_and_prep_movielens_data()
    avg_ratings_of_users = ratings_matrix.sum(axis=1)
    # set the position of elements to TRUE in the matrix if > 0
    mask = (ratings_matrix > 0)
    # number of items rated by each user
    mask = mask.sum(axis = 1)
    avg_ratings_of_users = np.divide(avg_ratings_of_users, mask)

#   Step 1: Get the PPM list based on a primitive
#           recommendation method. Here, top 100 movies are
#           selected based on the highest average rating and
#           largest number of users for each movie.
    PPM = topN_PPM(ratings_matrix, 100)
    # Select a random set of users
    users = [5,15,47,121,269,300,408,511,607,729,842,912]
    similarity_threshold = 0.8
    number_of_rankings_requested = 100
    oupt_file = open("plot_new_serendipity.dat", "w")
    for user in users:
        RS, RS_using_genre = recommendations_genre_based(
                                ratings_matrix,
                                items_and_their_genre,
                                avg_ratings_of_users,
                                pearson_similarity_population,
                                similarity_threshold,
                                user-1) # 0-indexing
#   Step 2.
        # Compute the list of unexpected movies from a
        # a given number of recommendation requested.
        # First, get the mask for all items that are in RS
        # but not in PPM. This is done using "in1d" function
        # and setting the "invert" flag to true.
        mask   = np.in1d(RS['itemid'][0:nrankings],
                         PPM, invert=True)
        UNEXPEC = RS[mask]
        #print "UNEXPEC.shape = " , UNEXPEC.shape

#   Step 3.
        # Compute the list of useful items from the unexpected
        # set of movies.
        # Usefulness is when the ratings of movies > 2.5
        mask = (UNEXPEC['rating'] > 2.5)
        USEFUL = UNEXPEC[mask]
        #print "USEFUL.shape = " , USEFUL.shape


#   Step 4
        # Get the genre based ratings of all the USEFUL movies
        indices = [i for i, itemid in \
                  enumerate(RS_using_genre['itemid'])
                  if itemid in USEFUL['itemid']]
        USEFUL_GENRE = np.sort(RS_using_genre[indices],
                             order = 'rating')
        #print "USEFUL_GENRE.shape = " , USEFUL_GENRE.shape


        USEFUL_sorted = np.sort(USEFUL, order='itemid')
        USEFUL_GENRE_sorted = np.sort(USEFUL_GENRE, order='itemid')
        ratings_difference = np.sort(USEFUL,
                        order='itemid')['rating'] - \
                          np.sort(USEFUL_GENRE,
                        order='itemid')['rating']
        normalized_ratings = np.divide(ratings_difference,5.0)
        mask = (normalized_ratings < -0.05)
        lesser_sdpr_movies = mask.sum()
        oupt_file.write("# user = %s \n" % user)
        oupt_file.write("# less srdpty = %s \n"
                       % lesser_sdpr_movies)
        np.savetxt(oupt_file, USEFUL_sorted,
                  fmt='%d  %1.5e', \
                  delimiter=' ', newline='\n', header='', \
                  footer='', comments='# ')
        oupt_file.write("\n")
        oupt_file.write("# USEFUL_GENRE" )
        np.savetxt(oupt_file, USEFUL_GENRE_sorted,
                  fmt='%d  %1.5e', \
                  delimiter=' ', newline='\n', header='', \
                  footer='', comments='# ')

        # Print the specified number of rankings
        print "##########################################"
        print "         user = ", user
        buffer_rankings = RS[0:nrankings]
        print np.sort(buffer_rankings['itemid'])
        oupt_file.write("# user = %s \n" % user)
        np.savetxt(oupt_file, buffer_rankings, fmt='%d %1.5e', \
                   delimiter=' ', newline='\n', header='', \
                   footer='', comments='# ')
        oupt_file.write("\n")
        print np.sort(RS_using_genre[0:nrankings]['itemid'])
        print "##########################################"
    oupt_file.close()
#=====================================================
if __name__ == "__main__":
    main()
```

6.2 APPENDIX B: APACHE MAHOUT CODE

```
package com.predictionmarketing.userrecommend;

import java.io.File;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.util.List;

import java.io.IOException;

import java.util.Arrays;

import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;

import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;

import org.apache.mahout.cf.taste.impl.neighborhood.

   NearestNUserNeighborhood;

import org.apache.mahout.cf.taste.impl.neighborhood.

   ThresholdUserNeighborhood;

import org.apache.mahout.cf.taste.impl.recommender.

   GenericUserBasedRecommender;

import org.apache.mahout.cf.taste.impl.similarity.

   PearsonCorrelationSimilarity;

import org.apache.mahout.cf.taste.model.DataModel;

import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;

import org.apache.mahout.cf.taste.recommender.RecommendedItem;

import org.apache.mahout.cf.taste.recommender.UserBasedRecommender;

import org.apache.mahout.cf.taste.similarity.UserSimilarity;

// ------------------------------------

public class UserRecommend {

       public static void main(String[] args) throws Exception {
            DataModel dm = new FileDataModel(new File("data/movies.csv
               "));
            UserSimilarity similarity = new
               PearsonCorrelationSimilarity(dm);
            UserNeighborhood neighborhood = new
               ThresholdUserNeighborhood(0.8, similarity, dm);
            UserBasedRecommender recommender = new
               GenericUserBasedRecommender(dm, neighborhood,
               similarity);
            int userId = 842;
            List<RecommendedItem> recommendations = recommender.
               recommend(userId,20000);
            BufferedWriter bw = new BufferedWriter(new FileWriter("
               data/RS842.csv"));
            for (RecommendedItem recommendation : recommendations) {
                   bw.write(recommendation.getItemID() + "," +
                      recommendation.getValue() + "\n");
                   System.out.println(recommendation);
            }
            bw.close();
      }
}
```

BIBLIOGRAPHY

"Movielens Dataset, Grouplens Research Center."

Adamopoulos, P. and Tuzhilin, A., "On unexpectedness in recommender systems: Or how to expect the unexpected," in *Workshop on Novelty and Diversity in Recommender Systems (DiveRS 2011), at the 5th ACM International Conference on Recommender Systems (RecSys' 11).* Chicago, Illinois, USA: ACM, 2011, pp. 11–18.

Adamopoulos, P. and Tuzhilin, A., "On unexpectedness in recommender systems: Or how to better expect the unexpected," *ACM Transactions on Intelligent Systems and Technology*, vol. 1, no. 1, pp. 1–51, 2013.

André, P., Teevan, J., and Dumais, S. T., "From x-rays to silly putty via uranus: serendipity and its role in web search," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, 2009, pp. 2033–2036.

Downey, A. B., Elkner, J., and Meyers, C., *Think Python: How to think like a computer scientist.* Wellesley, Massachusetts: Green Tea Press, 2008.

Ekstrand, M. D., Riedl, J. T., and Konstan, J. A., "Collaborative filtering recommender systems," *Foundations and Trends in Human-Computer Interaction*, vol. 4, no. 2, pp. 81–173, 2011.

Esteves, R. M. and Rong, C., "Using mahout for clustering wikipedia's latest articles: a comparison between k-means and fuzzy c-means in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on.* IEEE, 2011, pp. 565–569.

Ge, M., Delgado-Battenfeld, C., and Jannach, D., "Beyond accuracy: evaluating recommender systems by coverage and serendipity," in *Proceedings of the fourth ACM conference on Recommender systems.* ACM, 2010, pp. 257–260.

Guo, S. *et al.*, "Analysis and evaluation of similarity metrics in collaborative filtering recommender system," 2014.

Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J., "An algorithmic framework for performing collaborative filtering," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval.* ACM, 1999, pp. 230–237.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T., "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

Hijikata, Y., Shimizu, T., and Nishida, S., "Discovery-oriented collaborative filtering for improving user satisfaction," in *Proceedings of the 14th international conference on Intelligent user interfaces.* ACM, 2009, pp. 67–76.

Iaquinta, L., de Gemmis, M., Lops, P., Semeraro, G., and Molino, P., "Can a recommender system induce serendipitous encounters," *E-Commerce*, pp. 227–243, 2010.

Karypis, G., "Evaluation of item-based top-n recommendation algorithms," in *Proceedings of the tenth international conference on Information and knowledge management.* ACM, 2001, pp. 247–254.

Kawamae, N., "Serendipitous recommendations via innovators," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval.* ACM, 2010, pp. 218–225.

Kim, T.-H. and Yang, S.-B., "Using attributes to improve prediction quality in collaborative filtering," in *E-Commerce and Web Technologies.* Springer, 2004, pp. 1–10.

Koren, Y., Bell, R., and Volinsky, C., "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

Lemire, D. and Maclachlan, A., "Slope-one predictors for online rating-based collaborative filtering." in *SDM*, vol. 5. SIAM, 2005, pp. 1–5.

Linden, G., Smith, B., and York, J., "Amazon.com recommendations: Item-to-item collaborative filtering," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76–80, 2003.

McNee, S. M., Riedl, J., and Konstan, J. A., "Being accurate is not enough: how accuracy metrics have hurt recommender systems," in *CHI'06 extended abstracts on Human factors in computing systems.* ACM, 2006, pp. 1097–1101.

Miyahara, K. and Pazzani, M. J., "Collaborative filtering with the simple Bayesian classifier," in *PRICAI 2000 Topics in Artificial Intelligence.* Springer, 2000, pp. 679–689.

Murakami, T., Mori, K., and Orihara, R., "Metrics for evaluating the serendipity of recommendation lists," in *New frontiers in artificial intelligence.* Springer, 2008, pp. 40–46.

Owen, S., Anil, R., Dunning, T., and Friedman, E., *Mahout in action.* Manning, 2011.

Rana, C., "New dimensions of temporal serendipity and temporal novelty in recommender system," *Advances in Applied Science Research*, vol. 4, no. 1, pp. 151–157, 2013.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J., "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web.* ACM, 2001, pp. 285–295.

Sarwar, B. M., Karypis, G., Konstan, J., and Riedl, J., "Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering," in *Proceedings of the fifth international conference on computer and information technology*, vol. 1. Citeseer, 2002.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J., "Analysis of recommendation algorithms for e-commerce," in *Proceedings of the 2nd ACM conference on Electronic commerce.* ACM, 2000, pp. 158–167.

Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M., "Methods and metrics for cold-start recommendations," in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval.* ACM, 2002, pp. 253–260.

Schelter, S. and Owen, S., "Collaborative filtering with apache mahout," *Proc. of ACM RecSys Challenge*, 2012.

Seminario, C. E. and Wilson, D. C., "Case study evaluation of mahout as a recommender platform," in *Proceedings of the 6th ACM conference on recommender engines (RecSys 2012)*, 2012.

Shani, G. and Gunawardana, A., "Evaluating recommendation systems," in *Recommender systems handbook.* Springer, 2011, pp. 257–297.

Su, X. and Khoshgoftaar, T. M., "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, p. 4, 2009.

Vucetic, S. and Obradovic, Z., "Collaborative filtering using a regression-based approach," *Knowledge and Information Systems*, vol. 7, no. 1, pp. 1–22, 2005.

Walunj, S. G. and Sadafale, K., "An online recommendation system for e-commerce based on apache mahout framework," in *Proceedings of the 2013 annual conference on Computers and people research.* ACM, 2013, pp. 153–158.

Ziegler, C.-N., McNee, S. M., Konstan, J. A., and Lausen, G., "Improving recommendation lists through topic diversification," in *Proceedings of the 14th international conference on World Wide Web.* ACM, 2005, pp. 22–32.