Statistical Analysis and Recommendation System based on Amazon Product Reviews

Pavan Manjunath
Stony Brook University
Dept of Computer Science
New York, USA
109916057

Arvind Ram Anantharam
Stony Brook University
Dept of Computer Science
New York, USA
110283876

Hari Prasath Raman
Stony Brook University
Dept of Computer Science
New York, USA
110283168

ABSTRACT

Recommendation systems play an important role in driving sales and increasing financial gains for online web stores like Amazon. Building a recommendation system that considers various parameters like user's preferences, their viewing and buying patterns, product attributes etc is challenging, with the challenge being to recommend products that are very closely aligned with the specific user's likings. Stronger the recommendations, higher the probability of user buying a product. Also, Exploratory Data Analysis(EDA) provides valuable insights about user's behavior. Incorporating this data into the recommendation system yields improved results. In this paper, we apply techniques of Collaborative Filtering(CF) to obtain recommendations. These results are further refined by multiple parameters obtained from EDA and the filtered results are ranked using methods of Matrix Factorization. We are also building a tool that implements the recommendation system and also supports a dashboard. The dashboard gives a visual representation of the user's behavior via various plots and graphs. We evaluate our system on a subset of the Amazon Product Catalog.

1. RELATED WORKS

J. McAuley et al. [1] have built systems that utilize the network structure of related products to recommend substitutable and complementary products. Even though we do not use the concept of substitutes and complements, we do utilize the network structure of the product graph to hone our recommendation system. Linden et al. [2] and Sarwar et al.[3] have demonstrated the efficacy of collaborative filtering in recommendation systems, specifically for Amazon products.

2. DATA PREPERATION

2.1 Data Summary

For this project, we are working with the Amazon Customer Review Dataset[4] that contains 143.7 million reviews and is 20GB in size. As it is extremely unwieldy to work on such a huge dataset with the limited computing resources we have, we decided to work on a subset, the Movies and TV dataset. We summarize this dataset in the following table-

Description	Value
Size of dataset	4G
Number of reviews	4.6M
Number of unique users	2M
Number of unique products	201K
Number of users with reviews greater than 15	4203
Ratio of 5 star to 1 star reviews	8.13
Time Span	18 years

Table 1: Summary of Movies and TV Dataset

The data consists of two components-

1. Review Data - Contains information about users and the reviews they've written- *reviewerID, asin, reviewerName, helpful, reviewText, overall, summary, unixReviewTime, reviewTime.*

2. Product Data - Contains meta information about the product itself- *asin, title, price, imUrl, related* - {*also_bought, also_viewed, bought_together*}, *salesRank, brand, categories.*

2.2 Challenges with Handling Large Datasets

The Amazon Movie Review Dataset was in JSON format. The natural choice to store and operate on the data was *mongoDB*, a JSON based document store. After storing the entire json dump, the fields that we don't need were removed from the data store. The remaining fields were exported into a csv file, to feed into python scripts, using *mongoexport*. Handling such a large dataset resulted in the following challenges-

1. The *user-movie* matrix and the *user-user* similarity matrices are very huge to store and operate in memory. In addition, the time taken to compute the similarity between all the users was also very high and thus, it was not feasible to load and compute them during every run. To avoid this, the entire movie dump was loaded into the PostgreSQL and loaded on demand to compute the similarities between the users and stored back in the database. This DB was used as the backend for the recommendation engine. This made operating with large dataset feasible.

2. Constructing the network graph of all the movies (200K nodes) and finding the distance between the two given nodes is a complex task in terms to both time and space. To tackle this challenge, we used Neo4j [5], graph database, to form the graph and query on it for calculate the distance between two movies. This, was also linked with the python scripts using the *py2neo* client. This made the graph construction and querying with large number of nodes fairly simple.

3. DATA EXPLORATION

EDA can reveal some significant insights about user behavior and correlation between different users. We present a few of them here.

3.1 User Similarity

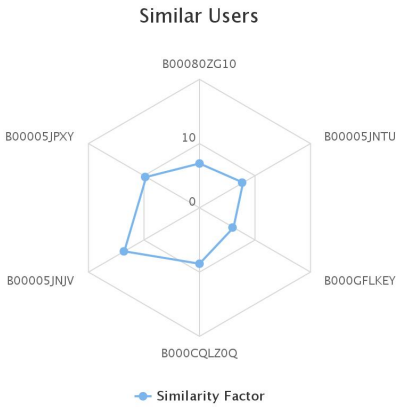

Figure 1: Spider Web Diagram showing Similarity Between Various Users and User A370Z6I5GBWU44.

In Figure 1, closer the point is to the corners of the hexagon, higher the similarity between users.

3.2 Correlation of Users' Reviewing Pattern

Figure 2 shows a circular network graph comparing the reviewing behavior of two users on various days of the week. Thicker the ribbon, higher the number of reviews. We can see that the reviewing behavior of the two users is quite different. User AV6QDP8Q0ONK4 (orange ribbon) writes reviews mainly on Fridays and Saturdays, where the user ANCOMAI0I7LVG is totally silent on Fridays and Saturdays.

3.3 Time Series Analysis

The size of the bubble in Figure 3 is directly proportional to the number of reviews written in that particular year.

4. DESIGN

Our recommendation system is based on 3 variables[6] -

1. Social Recommendation - Recommend things based on the past behavior of similar users.

2. Personalized Recommendation - Recommend things based on the individual's past behavior.

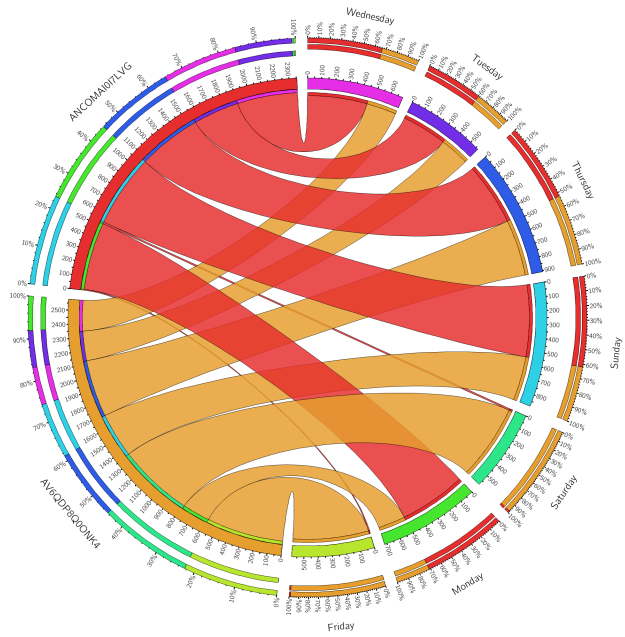

Figure 2: Circular Network Graph showing Reviewing Behavior on Various Days of Week by two users.

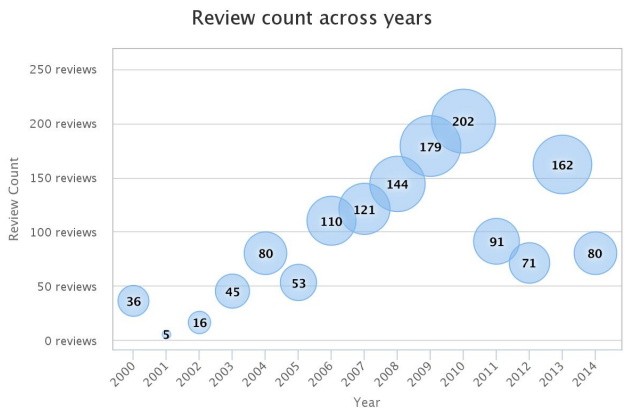

Figure 3: Bubble Graph showing number of movies reviewed over years by a single user.

3. Item recommendation - Recommend things based on the thing itself

We summarize our methodology in the following flow diagram-

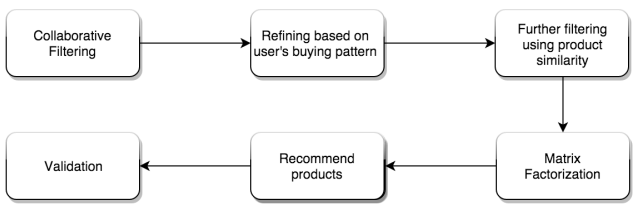

Figure 4: Block Diagram of the Recommendation System.

4.1 Social Recommendation

4.1.1 Collaborative Filtering

To identify similar users for a particular user, we construct a user-movie matrix as shown in Table 2.

-	M1	M2	M3	...	Mn
U1	0	1	0	...	1
U2	1	1	0	...	1
U3	1	1	1	...	0
U4	1	0	0	...	0

Table 2: User Movie Matrix

The value of a matrix cell *Matrix(U,M)* is 1 if the user U has reviewed the movie M, 0 otherwise. From this matrix, similarity between different users is computed using *Jaccard Similarity* and stored in a *User Similarity Matrix* as shown in Table 3.

-	U1	U2	U3	...	Un
U1	1.0	0.43	0.15	...	0.0
U2	0.43	1.0	0.65	...	0.7
U3	0.15	0.65	1.0	...	0.4
U4	0.56	0.48	0.29	...	0.16

Table 3: User Similarity Matrix

Now, for a given user, we select *Top k* similar users and group together all movies that these users have reviewed except the ones the current user has also reviewed. The resulting set of movies will form the base set of recommendations for the given user.

This method of Collaborative Filtering(CF) produces huge number of results and recommending all of them will overwhelm the user. Also, the results haven't been personalized to the user's tastes. We next describe various custom filtering techniques to further refine the recommendations.

4.2 Personalized Recommendation

4.2.1 Filtering based on User Purchase Range

Cost of a product plays a major role in influencing a user's buying behavior. The results obtained in CF may closely reflect the user's tastes but the cost of product may discourage the user from buying his favorite movie! What is the purchasing patterns of different users? Do different users buy products in different price ranges? Figure 5 shows a box plot summarizing the purchasing ranges of various users.

As you can see, both the range of prices as well as the median price is quite different for different users. We now apply this *price filter* to the results obtained from CF to filter out movies not in the user's purchasing range.

4.3 Item Recommendation

4.3.1 Filtering based on Movie and User Rating Distribution

The rating distribution of a product encapsulates information about its quality. A good movie has more number of 4 or 5 star ratings or in other words, the histogram is *skewed left* distribution. Conversely, a bad movie has more number of 1 or 2 star ratings or a histogram that is *skewed right*. Figure 6 shows the histogram of a good movie and a bad movie respectively. We are able to fit a quadratic curve to the histogram with a high goodness of fit value ($R^2 = 0.9$). Now, we can automatically detect whether a movie is good or bad by finding the slope of this quadratic curve at $x = 1$ and $x = 5$. Slope of a quadratic curve given by the equation $ax^2 + bx + c$ is simply the first derivative, $2ax + b$. A bad movie as a high negative slope at $x = 1$ and a low positive slope at $x = 5$. A good movie on the other hand, has a low positive slope at $x = 1$ and a high positive slope at $x = 5$

Now, good or bad is a very broad classification. We can further refine this idea by considering the ratio of these two slopes and then fixing thresholds for moderately good movies, very good movies and so on. Also we can consider the given user's rating distribution to make this decision. For instance, a user who consistently rates movies in 1-3 range has high level of expectations from a movie and we can recommend only those movies that have a very high slope at $x = 5$ and a very slow slope at $x = 1$. For our project, we have chosen an absolute value of 3 for the ratio threshold. This value was obtained by fixing a slope of $tan\ 60^0 = \sqrt{3}$ for the line at $x = 5$ and a slope of $tan\ 30^0 = 1/\sqrt{3}$ at $x = 1$.

4.3.2 Filtering based on Product Similarity

After the filtering process of previous stages, the recommendation list will contain only good quality products. This list of products can be further filtered based on movie similarity and the personal interests of the user. Two similar users will not have exactly same amount of interests towards all genres. For instance, let us say we have identified two similar users *U1* and *U2*. *U1* and *U2* have liked lot of movies in common. In addition to this, *U1* buys lot of action related movies and *U2* buys lot of movies related to philosophy. We should not recommend philosophical movies to *U1*. For such filtering, we make use of the network graph of all movies built using the *also bought* and *also viewed* information available in the meta data of the movies dataset. Figure 7 shows the network graph of related movies. This graph can be used to identify the distance between movies the given user has already watched and the movies that we are going to recommend. If the shortest distance between the movie to be recommended and movies the user has already watched (Figure 8) is beyond a particular threshold(4 for our project), we remove that movie from the recommendations. With this kind of filtering, we can remove movies which are far from the interests of the given user. From the above example, this technique can be used to filter out all philosophical movies, because they will be far from the movies liked by *U1*, from the recommendation list.

4.4 Matrix Factorization

Multiple levels of filtering will give us a refined list of movies to recommend to the user. This list has to be ranked before being presented to the user. We use techniques of Matrix Factorization[7] to estimate the possible ratings the given user might give the movies that we are about to recommend to him/her. To achieve this, we need to consider the given user's rating assignment to movies he/she has already

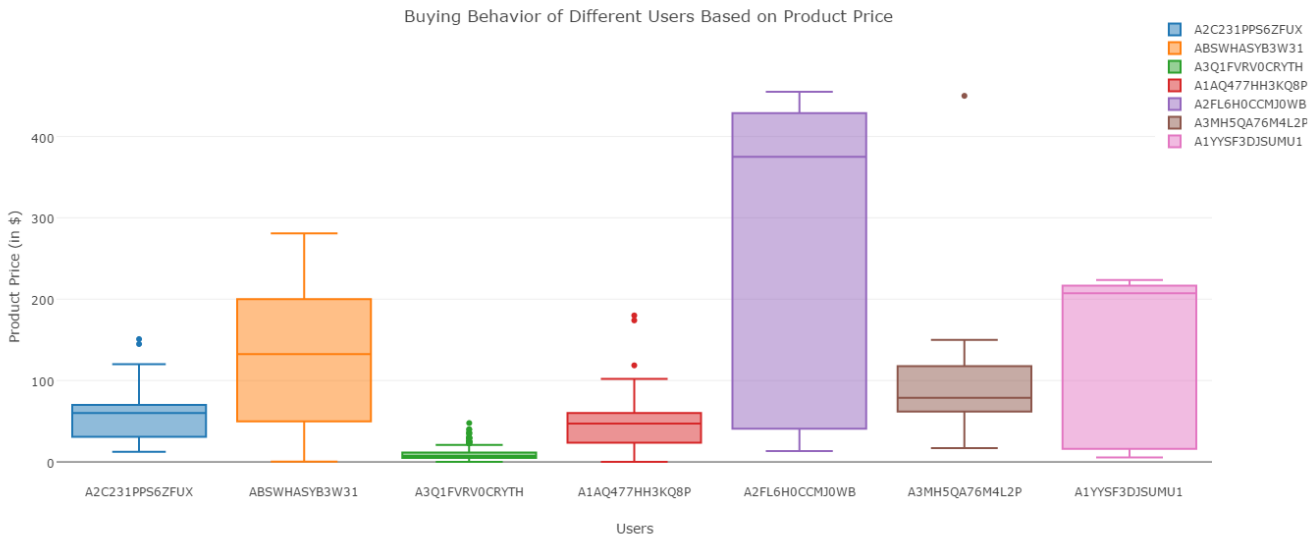

Figure 5: Box Plot Depicting Purchasing Ranges of Various Users.

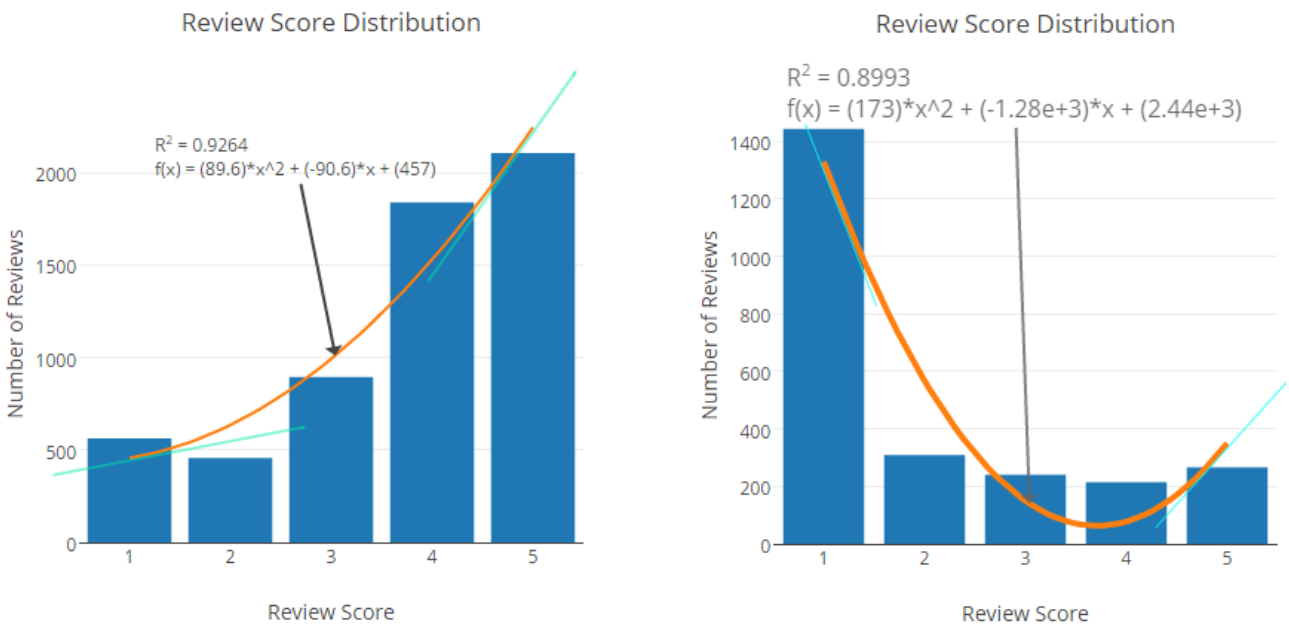

Figure 6: Rating Distribution (a) Flight starring Denzel Washington (b) Spring Breakers starring James Franco

watched along with rating behavior of other similar users. Table 4 summarizes the problem of predicting the missing ratings-

	M1	M2	M3	M4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	5	4

Table 4: Predicting the Missing Ratings via Matrix Factorization.

The basic idea behind using matrix factorization for predicting the missing ratings is that there may be some hidden features based upon which users rate movies. For example, if both users *U1* and *U2* have rated the movie *M1* high, its probably because the movie has some feature that both these users like, may be the genre or the actors in the movie. So if we can find the strength of associations between a movie's features and the user's preference for those, we can predict a user's possible rating for an unseen movie.

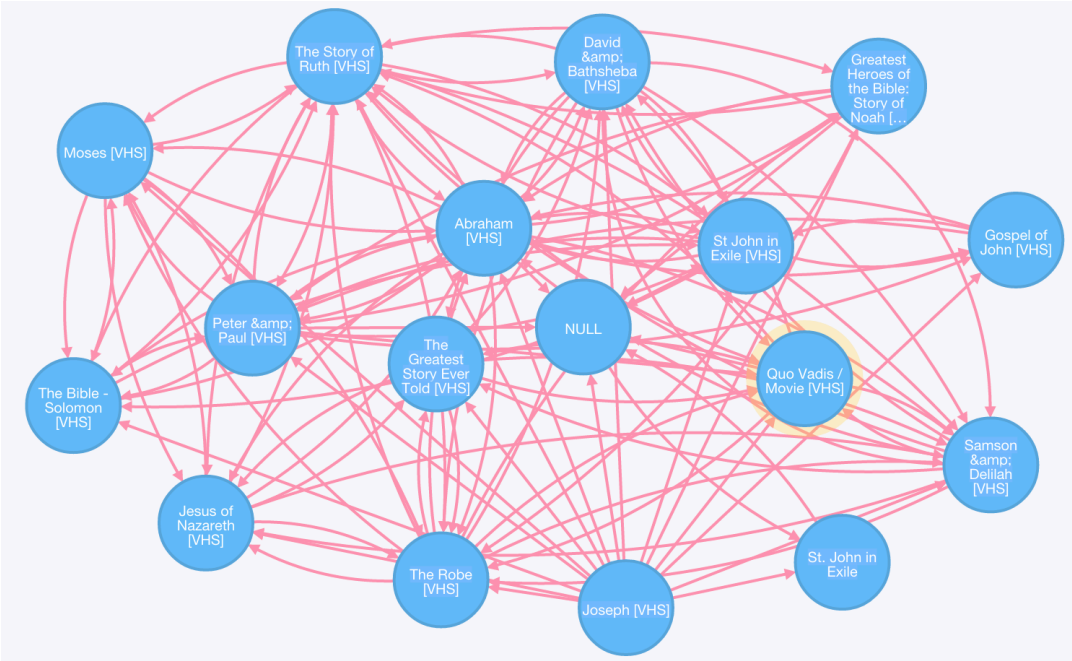

Figure 7: Network Graph Showing *also bought* and *also viewed* Movies

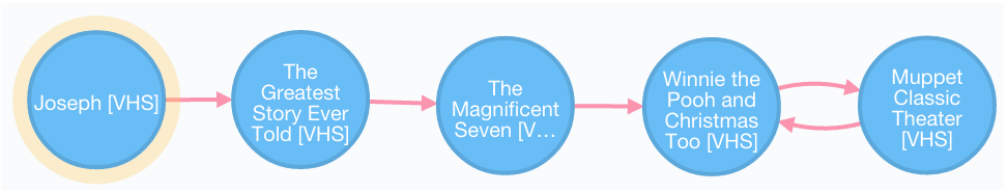

Figure 8: Shortest Path between a Movie the User has Reviwed and a Movie from the Recommendation Set

Mathematically, let U be the set of users and M be the set of movies(Note that this set contains recommended movies as well as movies already watched by the user). Let R of size $|U| \times |D|$ be the matrix that contains all the ratings assigned by the users. Also, let K be the number of latent features we intend to discover. The problem now boils down to finding two matrices $\mathbf{P}$ (a |U| $\times$ K matrix) and $\mathbf{Q}$ (a |D| $\times$ K matrix) such that their product approximates $\mathbf{R}$:

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}} \tag{1}$$

Quoting [8], each row of $\mathbf{P}$ would represent the strength of the associations between a user and the features. Similarly, each row of $\mathbf{Q}$ would represent the strength of the associations between an item and the features. To get the prediction of a rating of an item d_j by u_i, we can calculate the dot product of the two vectors corresponding to u_i and d_j:

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj} \tag{2}$$

Now, we have to find a way to obtain $\mathbf{P}$ and $\mathbf{Q}$. One way to approach this problem is to first initialize the two matrices with some values, calculate how different their product is to $\mathbf{M}$, and then try to minimize this difference iteratively. We use gradient descent to achieve this, aiming at finding a local minimum of the difference.

5. TOOL

We have built a tool that has two components-

1. Dashboard - It shows a user's profile in terms of his buying behavior, the time series plot of the number of reviews written by him/her over years, rating distribution, similarity to other users, his rating patterns on different days of the week etc

2. Recommendation Panel - This shows the top recommendations for the specific user after running our filtering mechanisms.

   Front end of the tool is built using HTML5, Javascript with BootStrap UI[9] framework. The server side is built in Python using Flask[10] framework backed by PostgreSQL database.

6. VALIDATION

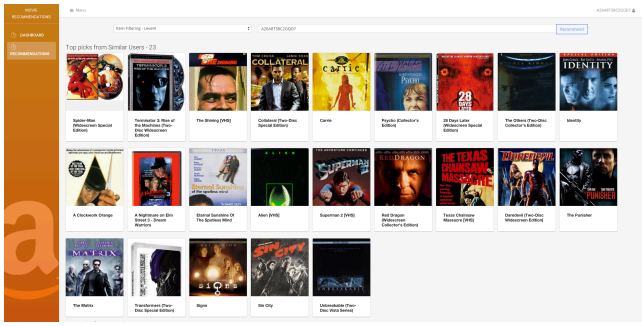

Figure 9: Screenshot of the Recommendation System.

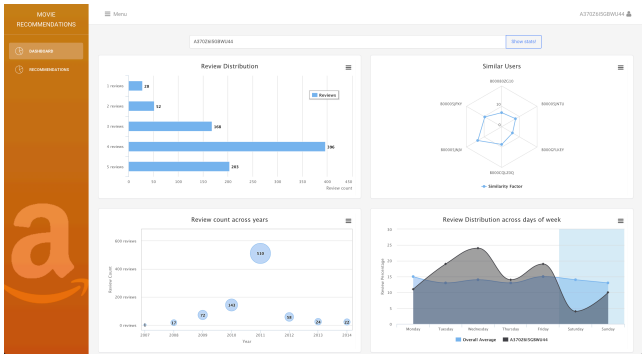

Figure 10: Screenshot of the EDA Panel in the Tool.

We split the total dataset into two parts, all reviews before and after 2010. We use all the reviews before 2010 and generate recommendations for the user. Now, to validate the recommendations we need to identify the number of movies the user has bought after 2010 which are in the recommendations we have made. For this, we need to remove the movies which released after 2010. To achieve this, we ignore all the movies which has received its first review after 2010. After removing all such movies, we measure the percentage of movies that the user bought from the list that we have recommended.

There are many complications with validation. For instance, in real life recommendation systems, recommendations influence a user's purchasing behavior but we do not have that luxury. We are kind of trying to *predict the past*. This factor has to be considered before evaluating the performance of our recommendation system.

7. RESULTS

We define success of our recommendation system as

$$Success = \frac{No.\ of\ Items\ Purchased\ from\ Recommended\ List}{Total\ Number\ of\ Recommendations} \tag{3}$$

Figure 11 shows success percentage achieved over successive filtering stages for a set of 5 users.

For user A370Z6I5GBWU44, our recommendation system performs extremely well scoring a success rate of nearly 60%. For the remaining users, the success percentage is decent but not great, averaging somewhere around 12-15 %. Many other users not shown here had an average success rate of just around 5-6 %. This is probably because of *Predicting the Past Problem* that we described in Section 7 that talks about Validating the Results.

We now apply Matrix Factorization to further rank the filtered results. Matrix factorization employs Stochastic Gradient Descent(SGD) as one of the technique to minimize the error in predicted values. In our experimentation we saw that SGD is converging extremely slowly. For 500 users and 500 movies, it was taking time of the order of 3 hours. Hence we selected only top 10 users that resulted in significant reduction of computation time. The resulting matrix for 10 users and 500 movies is too big to be shown here. So we show a part of the matrix. Table 5 shows different movies rated by different users. The missing rating values are populated by SGD and is shown in Table 6.

	M1	M2	M3	M4	M5	M6
U1	0.0	5.0	5.0	4.0	4.0	5.0
U2	0.0	4.0	0.0	0.0	5.0	0.0
U3	0.0	0.0	0.0	0.0	0.0	0.0
U4	5.0	5.0	5.0	5.0	0.0	0.0
U5	3.0	0.0	0.0	5.0	4.0	0.0
U6	4.0	4.0	4.0	0.0	0.0	3.0

Table 5: User Movie Rating Matrix

	M1	M2	M3	M4	M5	M6
U1	3.9356	4.8925	4.8482	3.9845	4.0307	4.4481
U2	4.0323	4.6834	4.7588	4.8200	4.4036	4.1097
U3	4.1648	4.8939	4.9510	4.8518	4.5014	4.3217
U4	4.1859	4.9958	5.0249	4.7038	4.4601	4.4483
U5	3.6766	4.1155	4.2410	4.7416	4.1440	3.5368
U6	3.1898	3.5766	3.6833	4.1002	3.5902	3.0767

Table 6: User Movie Rating Matrix After Matrix Factorization

Legend
M1 - GoodFellas.
M2 - Superman the Movie.
M3 - Superman 2.
M4 - The Italian Job.
M5 - Letter from Iwo Jima.
M6 - Monster's Ball.

U1 - AQZH7YTWQPOBE.
U2 - A30LMTLC9XVWKJ.
U3 - A8DI0COTCMRDV.
U4 - A2M1N8G4W4END8.
U5 - A1QQPIUXWD4G0Z.
U6 - A96JD9312DHWC.

The results are satisfactory. Consider movie M6. Two users have rated it 5 and 3 stars respectively. And the missing ratings fall somewhere in the range 3.5-4.5 which is realistic.

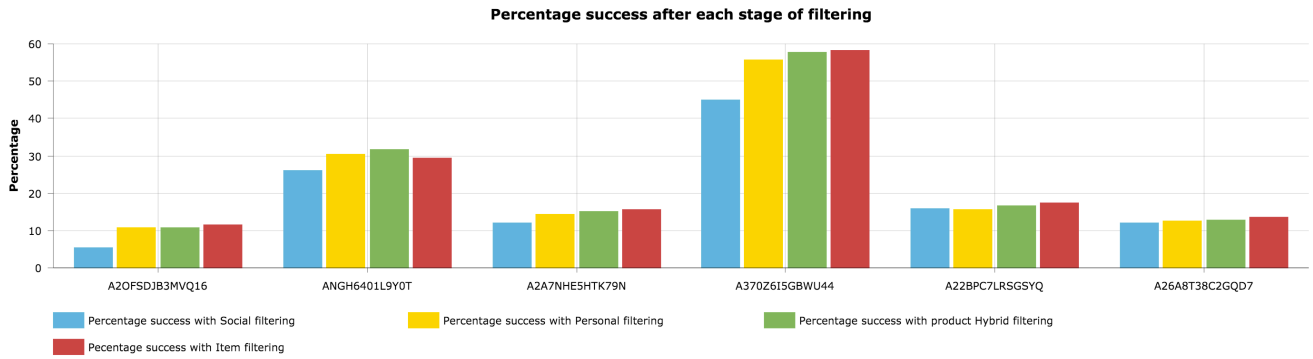

Figure 11: Success Percentage of Various Users after Successive Filtering Stages.

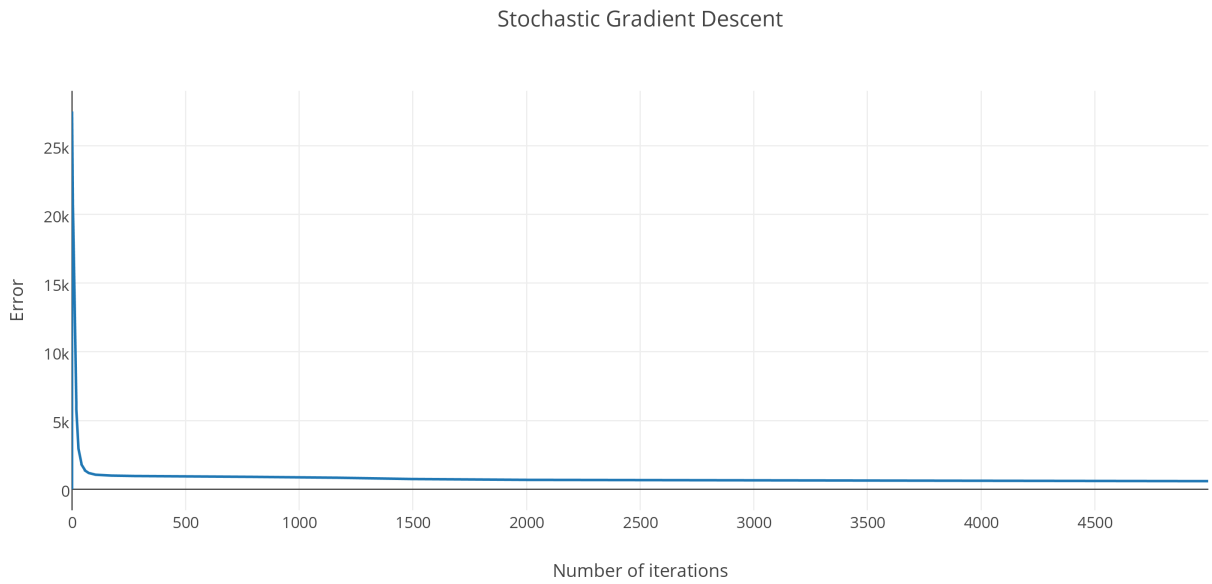

Figure 12: Convergence of Stochastic Graident Descent

We also show how error in SGD converges in Figure 12

8. REFERENCES

[1] Inferring Networks of Substitutable and Complementary Products by Julian McAuley et al. `http://cseweb.ucsd.edu/~jmcauley/pdfs/kdd15.pdf`

[2] Amazon.com recommendations: item-to-item collaborative filtering `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1167344&tag=1`

[3] Analysis of Recommendation Algorithms for E-Commerce `http://files.grouplens.org/papers/ec00.pdf`

[4] Amazon Customer Review Dataset, May 1996 - July 2014. `http://jmcauley.ucsd.edu/data/amazon`

[5] Neo4j, the World's Leading Graph Database `http://neo4j.com/`

[6] The Art, Science and Business of Recommendation Engines `http://sdl_demo.s3.amazonaws.com/89/www/24746/1/1.pdf`

[7] Matrix Factorization Techniques for Recommender Systems. `http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf`

[8] Matrix Factorization: A Simple Tutorial and Implementation in Python `http://tinyurl.com/njjqeo9`

[9] Bootstrap, most popular HTML, CSS, and JS framework. `http://getbootstrap.com/`

[10] Python Flask `http://flask.pocoo.org/`