Netflix Analytics - Movie Recommendation: P-32 Final Report

Ami Bhaskar Tanna
abtanna@ncsu.edu

Bhavesh Kasliwal
bkasliw@ncsu.edu

Rishi Jain
rjain9@ncsu.edu

1 Background

1.1 Introduction

Recommendation systems are becoming increasingly important as they give out personalized suggestions to users for movies, videos, songs and a whole bunch of other e-commerce products. Various techniques to implement such a recommendation system have been proposed and are already in use. In this project we develop a recommendation system for Netflix Prize dataset, based on movies liked by a user, we try to recommend other movies the user may like. The dataset at hand consists of movie ratings given by some 480189 distinct users to approximately 17770 different movies. This can be imagined as a huge matrix M with 17770 rows, one for each movie and 480189 columns to represent all users. It is filled with rating values given by each user for the corresponding movies. We call this user-movie matrix M and refer to it by the same denotation throughout. It is important to note here that not all users have rated all movies, in fact the entire matrix is mostly empty (sparse) considering not all users have rated all movies. This makes the matrix M a sparse matrix since the number of users and movies is much higher than the number of rating values at hand.
One of the biggest challenges while designing an algorithm for such a dataset would be sparsity or lack of available data. Due to sparsity there is insufficient ground to make any legit comparisons, analyze the neighborhood and identify peculiar rating patterns, choices and more importantly underlying movie-user interactions. As a part of our problem it is of prime importance address this issue while simultaneously identifying user choices and conforming movies.

1.2 Related Work

TThe Netflix Prize competition saw a wide variety of algorithms being implemented to recommend most befitting movies to users. There are mainly two approaches to this [4]: 1.Latent factor models that identify certain underlying factors to determine the peculiarities of a users choice and features of a movie [4]. 2. Neighborhood based approach which finds movies similar to each other locally and recommends them assuming a user would like all the movies close to each other in terms of their rating value [4]. This section of our report talks briefly about some of these methods.

1.2.1 Latent Factor Models

Generally carried out using matrix factorization, these models transform both movies and users to the same latent factor space, thus making them directly comparable[4]. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user ratings[4]. Additionally, this decomposition of users and movies into a set of latent factors helps evaluate missing ratings [4]. Typically, this is by performing singular value decomposition (SVD) on the user-movie matrix M where each of the user and movie is represented through a unique feature vector and missing rating of a particular user for a particular movie is predicted by inner product of corresponding feature vectors. Alternatively, one may perform Asymmetric SVD where users instead of having their own feature vectors are represented as a bag of items they have rated [8]. Each user is represented as a weighted sum of feature vectors of all the items they have rated [8]. This entire thing is then multiplied with the feature vector of a specific movie to obtain its rating by the user [8].

1.2.2 Neighborhood based collaborative filtering

These algorithms work on the collective choice of all users, filtering out information for a certain user based on likes and dislikes of other similar users[3]. They are based on the fact that similar users display similar choices and hence analogous rating patterns and items with similar ratings are similar in general [3]. Combining input given by all the users in terms of their choice, this algorithm learns the training data better as more and more users provide ratings and thus all these users collaborate towards a finer recommendation algorithm. There are two primary types of neighborhood-based algorithms [3]: (1)User based: It finds users similar to the given user in terms of likes and dislikes of movies and recommends the highest rated movie by the user that is most similar to the given user[3]. (2) Item based: It finds out items that are similar to each other and recommends based on the movies liked by a user[3]. In order to gauge the numerical value of similarity between two movies (or users) various methods such as cosine similarity, centered cosine similarity (which is nothing but Pearsons correlation coefficient), Spearman Rank Correlation or Euclidean distance can be used.[3]

1.2.3 Baseline Predictor model

It helps predict missing ratings by establishing a generic baseline and adding to it personalized rating patterns and movie specific rating patterns to enhance the prediction. [1] Generally baseline predictors are of the following form

$$b_{u,m} = \ + b_u + b_m[1] \tag{1}$$

u: user, m: movie, = overall average rating (general baseline), bu, bm : user and movie specific baseline predictors respectively. [1] However this model may not capture all the interaction effects between a user and movie while also missing out on generating movie and user specific feature vectors.

1.2.4 Restricted Boltzman Machines (RBM)

It is a latent factor model that deploys stochastic neural network to identify latent factors and recommend movies to users. [8]

1.3 Scope and Motivation

Neighborhood based methods are most commonly used for recommendation systems. However they only help us detect localized relationships and superficial rating trends. [4] They often ignore vast majority of ratings by a user relying only on a few significant neighborhood relations[4]. Consequently, these methods are unable to capture the totality of weak signals encompassed in all of a users ratings. [4] Latent factor models are generally effective at estimating overall structure that relates simultaneously to most or all items. However, these models are poor at detecting strong associations among a small set of closely related items, precisely where neighborhood models do best [4]. In this project we implement a combined model to improve prediction accuracy by capitalizing on the advantages of both neighborhood and latent factor approaches [4]. First we perform matrix factorization (MF) through SVD. SVD offers a wider platform to identify underlying peculiarities of user choices and movie genres in terms of individual feature vectors. Additionally it also fixes the problem of matrix sparsity. Arguably, baseline predictors also help us predict missing ratings and user idiosyncrasies but they do not yield feature vectors the way SVD does. It is interesting to see how we play around with these feature vectors while implementing neighborhood based models post SVD. In the next step, we proceed with neighborhood based similarity measures such as Pearsons Correlation Coefficient and Euclidean distance measure to identify nearest neighbors to a given movie. Notably we calculate correlation coefficient between movie ratings and for distance measurement we deploy feature vectors of individual movies to determine movies closest to the given movie. These movies form our recommendations. Correlation between ratings only give a shallow idea of movies correlated in terms of rating, it is the feature vectors that aid us evaluate similar movies in a much deeper sense. Thus the scope of this project is limited to SVD followed by item based collaborative filtering where based on movies liked by a user, we find other similar movies and recommend them.

2 Method

2.1 Singular Value Decomposition

Singular Value Decomposition or SVD as it is popularly known is a matrix factorization method that helps fix sparsity and identify underlying factors influencing the ratings of a user [11]. Here, we identify a few underlying or latent factors based on which each of these movies and users can be categorized [11]. For example, in this case we may imagine these latent factors as movie genres such as romance, thriller, comedy and action - assuming they are mutually exclusive; completely uncorrelated among themselves. Note that it is of prime importance to figure out exactly how many latent factors would yield most accurate rating predictions and hence the least error. By trying different number of latent factors ranging from 2 to 10 in our code, we discovered that 4 latent factors yield the most optimal result. Now each movie is weighed in terms of these factors and a feature vector is created for each movie using these weights. [5] Similarly each user is evaluated for his/her liking towards these latent factors and a feature vector is created for each user based on this. Finally, we have a feature vector for each user and each movie. The original matrix M can be decomposed into two separate matrices containing these feature vectors[5]: (1) One for users with the order 4xu (2) One for movies with the order mx4. The inner product of these two matrices would eventually yield our original matrix that contains the ratings for each movie by each user. The SVD factorization of a matrix M would be:

$$M = U \sum V^T [5] \tag{2}$$

Where M is ratings matrix is a diagonal matrix of singular values [5] U and V are the eigenvectors of MM^T and $M^T M$ respectively [5] Intuitively the singular values in $\sum$ are weighting factors for U and V [5]. The singular values are the square roots of eigenvalues of MM^T corresponding to the eigenvectors in U and V[5]. In our context the above equation can be written as$M = QP^T$ where matrix Q (mx4) represents feature vectors for each movie and PT (4xn) contains feature vectors for each user. The inner product of the two feature vectors corresponding to movie m and user u approximates the rating as:

$$r_{mu} = q_m p_u^* \tag{3}$$

[11] Where r_{mu} is the approximated rating by user u on movie m

2.2 Stochastic Gradient Descent The Learning Algorithm

The stochastic gradient descent algorithm actually helps us determine these feature vectors. To start with, it assumes random close to zero values and evaluates the squared error between the actual and predicted ratings summed across all known ratings. [11] This error function is minimized using gradient descent where the gradient of error function is calculated at each step and a step further is taken in the direction of steepest descent i.e. the feature vectors are updated accordingly [11]. This process is continued till we reach the global minima of error function [11]. Now we have an entire matrix filled with ratings and also the factors of this matrix containing feature vectors of each user and movie. The regularized squared error on the set of known ratings is:

$$\sum_{overallknownratings} [(r_{mu} - q_m p_u^*)^2 + \lambda({q_m}^2 + {p_u}^2)][11] \tag{4}$$

This is nothing but the sum of squares of errors between actual and predicted ratings. This is minimized to obtain the best possible prediction of missing ratings. The second term in that equation is the regularization term which prevents overfitting by penalizing the magnitude of the parameters. [4]

2.3 Pearsons Correlation Coefficient R

Pearsons correlation coefficient R can be used to find correlation in terms of rating between two movies for item based collaborative filtering or between two users for user based collaborative filtering[2]. Two movies are more similar if they have a higher correlation coefficient and hence can be recommended to a user who has liked (given a high rating) either of them[2]. To calculate R, firstly all the rating values are standardized to mean 0 and standard deviation 1 to get rid of scalability issues [2]. Since we are doing only item based collaborative filtering in this project we evaluate correlation between movie ratings of a given user [2]. The correlation coefficient between two movies m1 and m2 is given by:

$$R = 1/(n-1)\sum_{u=1}^{n} z_{um1} z_{um2}[2] \tag{5}$$

Where n = total number of users denoted by u z_{um1} , z_{um2} = normalized rating given by user u to movie m_1 and $m2$ respectively [2] In this way the top ten most correlated movies are displayed as recommended movies for a given movie.

2.4 K-NN method

The K-NN method we implemented involves calculating of Euclidean distance between movies or users in order to find their nearest neighbors. Since we perform only item-based collaborative filtering in this project, we evaluate only the distance between two movies for a particular user. It is instinctive to calculate this distance based on their rating values but the incompleteness of this measure is indicated when we recommend a movie that is close in rating but belonging to a completely different genre and the user does not particularly like it. Movies closely related to a given movie in terms of rating may not necessarily pander to the choice of a user. Besides, finding closely related movies in terms of rating is something we did previously using Pearsons correlation coefficient which determined movies most correlated in terms of rating. Measuring distance instead of correlation coefficient would only change the measure while still performing the same function at the core level and hence end up recommending similar movies. To cut down on redundancy and give better recommendations it is imperative to deploy the feature vectors at hand since they are a clearer representation of a movies genre and a users choice. The K-NN algorithm we implemented here calculates the distance between feature vectors of two movies for a particular user to determine how closely related they are. Recommendations are made based on these nearest neighbors.

3 Plan and Experiment

3.1 Description of the Dataset

This dataset used is the Netflix Prize dataset. Details about it can be found at http://www.netflixprize.com. On exploring the dataset we find that the movie rating files contain over 100 million ratings from 480 thousand Netflix customers on over 17 thousand movie titles. The ratings are on a scale from 1 to 5 stars. The dataset also includes date of each rating and the title and year of release for each movie id. The following are the specific details:

- MovieIDs range from 1 to 17,770 sequentially.
- CustomerIDs range from 1 to 2649429, with gaps with 480189 unique users.
- Ratings are on a five star scale from 1 to 5.
- Dates have the format YYYY-MM-DD.
- YearOfRelease ranges from 1890 to 2005.

3.2 Data Preprocessing

The data is preprocessed in two steps. It is first cleaned to remove all movies that have not been rated by even a single user. Besides the rating data has customer id and rating for each movie id separately. This has been combined into a huge matrix of movies and users filled with corresponding rating values. This forms our matrix M as mentioned in the introduction. In the next step of preprocessing, all movies with their review count lying in the bottom 20 percent have been eliminated. This helps us eliminate infrequent movies and users thereby ensuring the algorithm inclined towards popular choice movies and hard core movie viewers (raters). Since the dataset at hand is incredibly huge to be handled efficiently by the system we have, we chose to work only on a segment of the entire dataset. Out of a total of 17,770 movies and 4,80,189 unique users, we choose a sample corresponding to 100 random movies and all the users who rated those 100 movies, these were 41837 users in number. After preprocessing and removing the bottom 20

3.3 Experimental Setup

Since we do not have a prior idea of the characteristics of dataset and there is no known delineating criteria between high quality and low quality samples we choose our sub-sample randomly. The features at hand are movie id, user id, ratings and year of release of the movies. Since our algorithm does not focus on the year of release, we do not include it in our training set. We did however deploy it in the initial stages of our project to calculate the Euclidean distance of ratings and year between two movies, but the results obtained were not particularly interesting; like any neighborhood model it did not capture the underlying trends. So we decided against having it in our final model.

The other parameter that needs to be tuned as a part of our experiment is k for determining k-nearest neighbors of a particular movie. If we decide to make 10 recommendations with respect to a particular movie, k needs to at least be 10. However, mathematically speaking k=10 may not be the most optimal k for deciding which movies are nearest to the given movie. Out of 10, a few movies may be closely clustered around the given movie while some may be far away. But if we look at the actual list of recommendations we receive in YouTube or Spotify or even Netflix for that matter, not all of them are close enough, only a few are extremely close, few are distantly related and few are completely random. Assuming our model also to be on similar lines, we ignore the fact that not all recommendations would be accurate but just to maintain a standard number of recommendations every time we take k =10 and determine 10 movies nearest to the given movie and recommend all of them.

3.4 Training and Testing

The recommendation algorithm is supposed to recommend movies to a user based on movies already liked by the user. The best judge of these recommendations is the user itself and it is only from user feedback that we can gauge the accuracy of our predictions. However, at an intermediate step the algorithm strives to predict missing ratings in the dataset and it would be fair to conclude that an accurate prediction of these rating values would in turn imply an accurate approximation of a users actual choice, thus yielding us precise feature vectors representing the contents of each movie and choice of each user. It is these feature vectors that form our basis for recommending movies to a user. Hence, evaluating the accuracy of predicted rating values would be a reliable testing parameter for our algorithm. In order to execute this, we split the data into training and testing sets. We do not include a validation set since there is no explicit hyper-parameter that needs to be tuned. The feature vectors get updated using the gradient descent algorithm and the appropriate number of latent factors (in SVD) was something we figured out on training set itself. Hence to keep things simpler, we omit the validation part and split the sub-sample in a ratio of 8:2 for training (80For a stronger testing procedure, we implement 5-fold cross validation since it yields a more robust algorithm due to the fact that it trains over the entire dataset in one or other iteration. Needless to say 10-fold cross validation would yield a stronger algorithm, but 5-fold fits in correctly with our train-test split since we have allotted 20

3.4.1 Top-k precision in testing

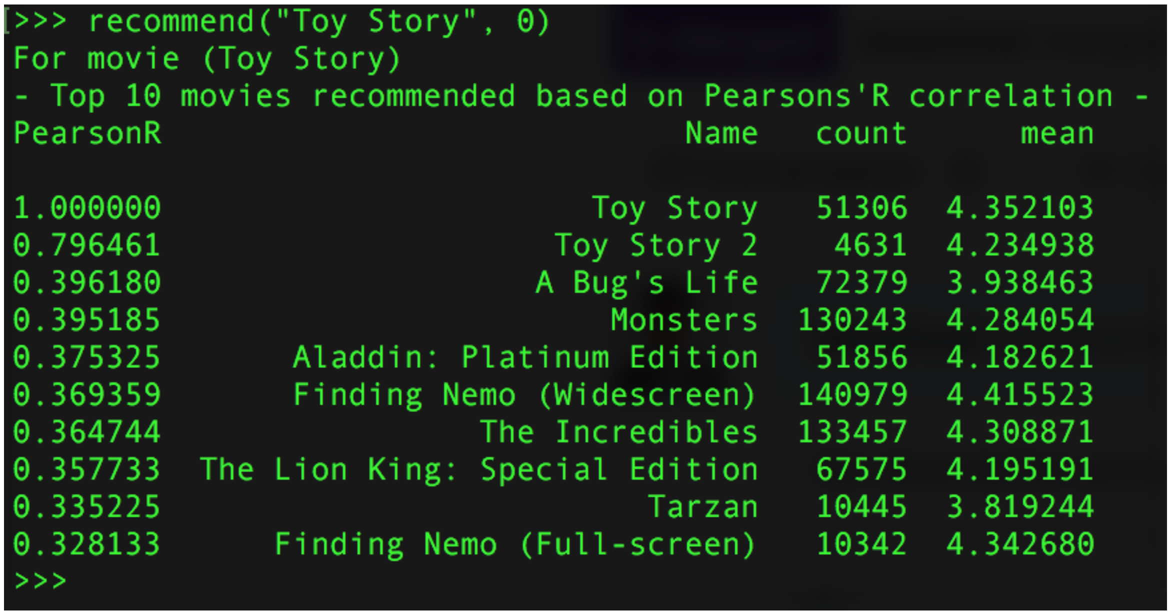
It is important to note that in a recommendation algorithm, we do not care about movies with low ratings i.e. if a user gives poor rating to a movie, we do not make recommendations based on that movie. Simultaneously, while finding k-nearest neighbors of a movie, we do not recommend a movie if it has a low rating however close it may be to the given movie. Conclusively, a recommendation algorithm is never interested in movies with low ratings or even predicting those low ratings. This brings us to conclude that it is not important to predict low ratings correctly, it is only the high ratings that matter. In our case we are predicting the missing ratings by SVD of the user-movie matrix and calculating the RMSE to evaluate the error of predicted ratings. However in a scenario where this SVD algorithm does not do very well on predicting low ratings but predicts high ratings very precisely, the RMSE will still show a higher and a rather misleading value due to poor performance on low-rated movies. In such a case it is important to eliminate those movies while evaluating RMSE since they are anyway not significant. This is known as top-k precision i.e. out of the testing set we deploy only the top k movie ratings to calculate the RMSE and evaluate the algorithm[12]. Now arises the question of deciding what this k is going to be. Fixing any value of top 20 or 25 percent of ratings would be questionable since the number of highly rated movies in the test set would actually vary and cannot be predicted randomly. However it is important to define a threshold value for categorizing high and low rated movies. Luckily, this would also solve the problem of top k-precision, where we choose all the movies that are highly rated instead of just top-k. Our aim is anyway satisfied since we only wish to disregard low-rating movies. Based upon our general knowledge of movies and the implicit meaning of their respective rating values, we decided the threshold to be 2.5. Any movie with a rating value ¿2.5 is regarded as high-rated and rest are low rated movies.

3.4.2 Additional Verification Method

As an additional non-mathematical verification method, we hope to find movie A in movie Bs recommendation list if B was in As recommendation list. To elucidate this further, we first get the recommended movies for the input movie. Next, for each of the recommended movies, get a list of recommendations. Finally, we count the number of times the input movie appeared in the recommended movie list of the initial recommended movies obtained in step 2. This count helps us estimate how accurate our original recommendations were.

4 Results

Here are the results we obtained at the end of two-step process: SVD followed by Pearsons R in Figure 1 and SVD followed by K-NN in Figure 2.



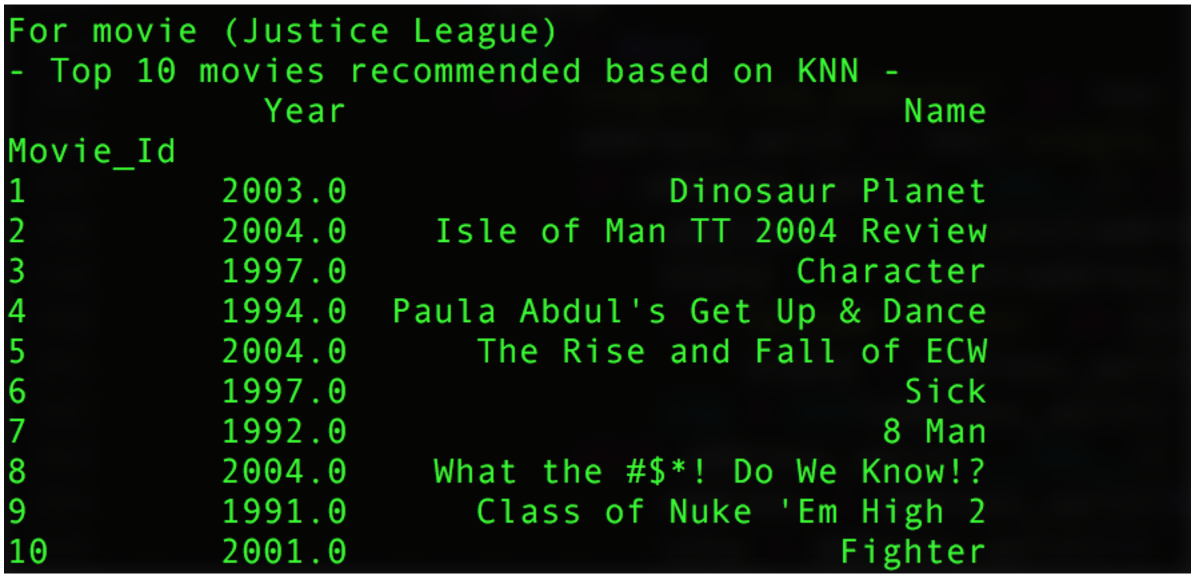
```
>>> recommend("Toy Story", 0)
For movie (Toy Story)
- Top 10 movies recommended based on Pearsons'R correlation -
PearsonR                               Name    count      mean

1.000000                          Toy Story    51306  4.352103
0.796461                        Toy Story 2     4631  4.234938
0.396180                       A Bug's Life    72379  3.938463
0.395185                           Monsters   130243  4.284054
0.375325           Aladdin: Platinum Edition    51856  4.182621
0.369359          Finding Nemo (Widescreen)   140979  4.415523
0.364744                    The Incredibles   133457  4.308871
0.357733      The Lion King: Special Edition    67575  4.195191
0.335225                             Tarzan    10445  3.819244
0.328133         Finding Nemo (Full-screen)    10342  4.342680
>>>
```

Figure 1: Result of Pearson's Correlation

As such we found no testing procedure to evaluate the recommendations obtained by Pearsons R, however for K-NN we obtained RMSE = 1.5 evaluated for rating values ¿= 2.5 (as per top-k precision). For additional verification method we obtained a score of 9/10 on average both by using Pearsons Correlation and K-Nearest Neighbor approach.

Thus it can be seen that both Pearsons R and K-NN give reasonably good recommendations but as mentioned above correlation through ratings captures only superficial trends while missing out on underlying reality. This can be illustrated through the results obtained as well. Observe that the recommendations to Toy Story contain Toy Story 2 and similarly recommendations for Harry Potter Part I would be its subsequent parts. This is exactly what we would not like in our results, since the algorithm is recommending obvious and close by results without actually realizing the users taste. Even on assuming the users choice is taken care of, such a method adds up to lack of



```
For movie (Justice League)
- Top 10 movies recommended based on KNN -
            Year                               Name
Movie_Id
1          2003.0                   Dinosaur Planet
2          2004.0        Isle of Man TT 2004 Review
3          1997.0                         Character
4          1994.0   Paula Abdul's Get Up & Dance
5          2004.0          The Rise and Fall of ECW
6          1997.0                              Sick
7          1992.0                             8 Man
8          2004.0         What the #$*! Do We Know!?
9          1991.0          Class of Nuke 'Em High 2
10         2001.0                           Fighter
```

Figure 2: Result of KNN

prediction diversity where all the movies recommended are too similar to each other. K-NN on the other hand deals with feature vectors of movies which capture the underlying trends better.

The code for the project can be found here: https://github.ncsu.edu/bkasliw/ALDA-Fall-2017.

5 Conclusion

It was important to understand this problem of recommendation systems not just to give better movie recommendations but also to aid the vast e-commerce industry where better product recommendations directly impact sales and revenue. It was interesting to see how diverse algorithms got integrated in this process of recommending movies. Conceptually everything seemed simple enough but when it came to implementing we stumbled upon a lot of experimental and interpretational pitfalls where deciding parameters and interpreting results was crucial to decide the path of algorithm. Certain parameters had to be estimated by observation of general trends since it was difficult to assess them experimentally. We confined ourselves to item-based collaborative filtering in this project but it would be interesting to see the results yielded by user-based collaborative filtering.

References

[1] Ekstrand, Michael D., John T. Riedl, and Joseph A. Konstan. ”Collaborative filtering recommender systems.” Foundations and Trends in HumanComputer Interaction 4.2 (2011): 81-173.

[2] Shimodaira, Hiroshi. ”Similarity and recommender systems.” School of Informatics, The University of Eidenburgh 21 (2014).

[3] Aggarwal, Charu C. ”Neighborhood-Based Collaborative Filtering.” Recommender Systems. Springer International Publishing, 2016. 29-70.

[4] Koren, Yehuda. ”Factorization meets the neighborhood: a multifaceted collaborative filtering model.” Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2008.

[5] https://classes.soe.ucsc.edu/cmps242/Fall09/proj/mpercy_svd_talk.pdf

[6] http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html

[7] https://beckernick.github.io/matrix-factorization-recommender/

[8] http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

[9] https://github.com/vdyashin/SVD/blob/master/example.py

[10] https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html

[11] Gower, Stephen. ”Netflix Prize and SVD.” (2014): 1-10.

[12] https://www.youtube.com/watch?v=VZKMyTaLI00