Content Recommendation System

Bhumika Singhal
School of Engineering and Applied Science
University of Pennsylvania
bhsingha@upenn.edu

Namita Shukla
School of Engineering and Applied Science
University of Pennsylvania
nashukla@upenn.edu

Riya Dholakia
School of Engineering and Applied Science
University of Pennsylvania
driya@upenn.edu

Abstract

Recommender system, an upcoming content filtering technology is widely used on websites, tamed according to the interest of users, and has varied applications[1] in media services such as movies, music, venue, books, research articles, and social media in general. One of the potent personalization technologies powering the adaptive web is collaborative filtering (CF). In our work, we perform a comparison between different CF algorithms to assess their performance. We use item-based mean imputation as our baseline and further, we evaluated K-Nearest Neighbor (KNN), Soft KNN, Item Baseline KNN, Matrix Factorization, Non-negative Matrix Factorization (NMF), and Neural-based Recommender Systems. In our experiments, we used a popular Jester Joke dataset and clustered the items based on three similarity metrics - Manhattan, Cosine, and Pearson Correlation similarities. Our results reveal that the Ensemble of NMF and Soft KNN with Item Baseline (using Pearson Similarity) for item-based CF outperformed all other algorithms examined in this paper.
Keywords— Imputation, Collaborative filtering; KNN; NMF; Cosine similarity; Pearson Correlation;

1 Motivation

Content service providers need an effective and efficient way to manage their content and provide quality recommendations to their users according to their preferences. Therefore there is an increasing market need for a good recommendation system. Currently, there are many media services, like Twitter, Amazon, Netflix, etc. which are working on building high-precision commercial recommendation systems (RS).
RS can be defined as a software technology that helps the target customer in decision-making processes, such as what products to buy, what tweets to recommend, and what movies to watch. Furthermore, these systems help the companies to raise their revenues. Amazon, Netflix, and Twitter are examples of organizations that strongly rely on RS to increase their sales profits. It can be generalized into mainly 3 categories - Content-based Filtering, Collaborative Filtering, and Hybrid Filtering techniques[2].

Content-Based Filtering is one of the simplest approaches in RS. It recommends the users a list of items that are similar to the items they liked in the past. However, the challenging part here is to define similarity metric according to the content i.e using abstract textual information such as movie genres, title keywords, tweet hashtags, etc. as our labels. This is where collaborative filtering-based approaches come into the picture. It recommends items based on the user's past behavior (items liked in the past) as well as similar decisions made by other users.

For our particular collaborative recommender system, we are considering jokes as our content items. However, the recommender system we aim to build would be a generalization of a system that learns from considering several users' past rating history and recommends items which they would probably like to see in the future. A caveat here is the problem of missing data i.e sparsity.

The methods we use to combat sparsity in our system and recommend quality items are - a) Memory-Based Approaches - KNN, Soft KNN, KNN with item baseline, and b) Model-Based Approaches - Matrix Factorization, Non-negative

Matrix Factorization (NMF), and Neural-based Recommender Systems[3]. The memory-based method evaluates the recommendations by directly accessing the database, while the model-based method uses the transaction data to generate a model that can suggest recommendations. Further, we use 3 similarity metrics - Manhattan Distance, Cosine Similarity, and Pearson Correlation.

2 Related Work

This section discusses the research papers that compared different RS algorithms.

According to Collaborative Filtering Systems, the first comparison we wanted to analyze was the mutual point of similarity, that is, the establishment of similarity between users or items. The aim of Taner Arsan et al's paper[4] was to compare User-based and Item-based Collaborative Filtering Algorithms with many different similarity indexes with their accuracy and performance. They discussed an approach to determine the best algorithm, which gave the most accurate recommendation using statistical accuracy metrics. Their results showed that item-based similarities outperformed user-based because of a variety of reasons, the main being computational advantages. Users being more in number, memory-based models don't perform well due to a very large number of similarities to compute and store. However, this paper focused more on statistical inferences and lacked interpretability in the real world.

Therefore, to learn about the methods used in industry settings, we explored the well-recognized Netflix Recommendation System prize-winning paper[5]. While their implementation techniques had a pivotal significance in our work, one key problem with their approach pointed out by Edwin Chen in his blog[6] was that the neighbors cannot be assumed to be independent, so using a standard similarity metric to define a weighted mean overcounts information. For example, suppose you ask 5 of your colleagues for restaurant recommendations. Three of them went to Mexico last week and are sick of burritos, so they strongly recommend against a taqueria. Thus, the neighbors can have inherent biases which were not taken into account in their approach.

Another key point we wanted to explore was the up-and-coming field of neural-based RS[7]. Evidently, the field of deep learning in RS is flourishing and is used by all modern systems. The key methods used in deep learning are Multilayer Perceptron (MLP), Autoencoders, and Attention Models as discussed in the survey paper by research faculty at Nanyang University[7]. They highlighted the ubiquity of deep learning and how the number of research publications on deep learning-based recommendation methods has increased exponentially in these years, providing strong evidence of the inevitable pervasiveness of deep learning in RS research. In order to explore this aspect, we will try to use a simple fully connected neural network as one of our model-based approaches.

3 Dataset

3.1 Introduction

The dataset[8] we will use consists of 100 jokes and 73,421 user ratings of these 100 jokes provided by Jester Research project under the flagship of UC Berkeley Laboratory for Automation Science and Engineering. The range of ratings are from -10 to +10. The data was divided into three excel files with each excel file having about 25,000 users. For the purpose of this project, we have limited our data to the jester-data-1 which contains 24983 users. Joke indices numbered 5, 7, 8, 13, 15, 16, 17, 18, 19, 20 are dense i.e almost all users have rated those jokes. The cells having 99 as their value are NaNs i.e not rated.

3.2 Visualizations

For the purpose of exploratory data analysis, we have dropped the ratings with value of 99 (they represent those ratings which are not rated by the user).

- Figure 1 is the plot of the user ids w.r.t to the number of jokes rated by the user. The intent behind the plot was to visually analyse the sparsity in the dataset and to observe for any evident bias/anomaly in the user ratings, if present. The plot shows no such character and the number of ratings do not seem to have any anomaly.
- Figure 2 is the plot of the joke ids w.r.t to the number of ratings received. The intent behind the plot was to visually analyse the sparsity

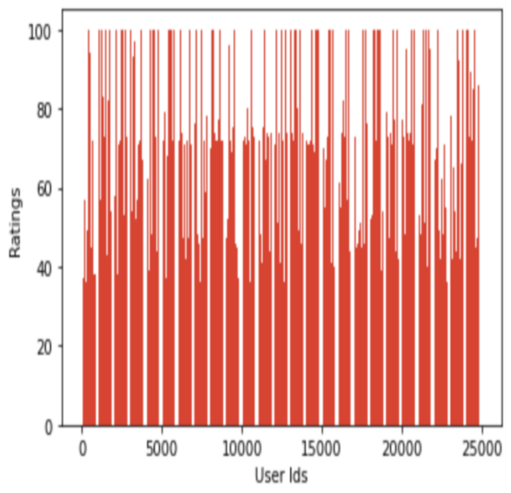


Figure 1: User IDs vs Ratings

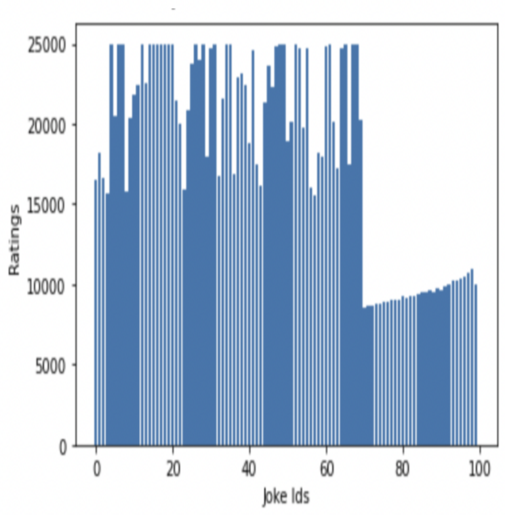


Figure 2: JokesIDs vs Ratings

in the dataset and to observe for any evident bias/anomaly in the user ratings, if present. The plot shows no such character and the number of ratings do not seem to have any anomaly. In addition, this plot revealed important character of the data. We can clearly see that the jokes beyond the 68th joke are more densely rated.

- In order to get some concrete numbers to observe the density of the data, we plotted the number of ratings received threshold vs number of jokes. Figure 3 shows that 30% of the users have rated all 100 jokes. These will form our dense matrix ratings for the calculations of similarities. Also, we observe that 50% of users have rated about 70 jokes.

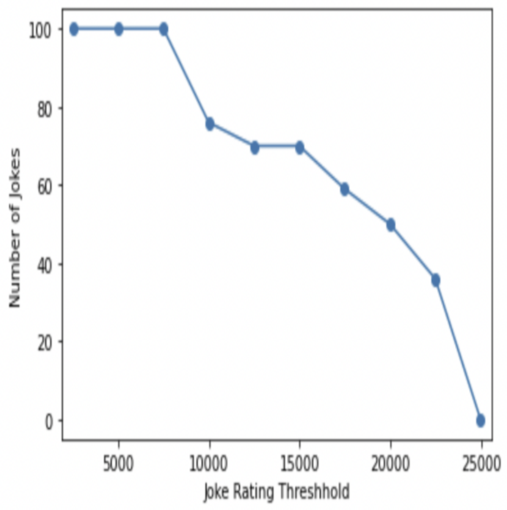


Figure 3: Number of Jokes vs Joke Rating Threshold

Since this is not extremely sparse data, we could study the effects of adding sparsity to the dataset.

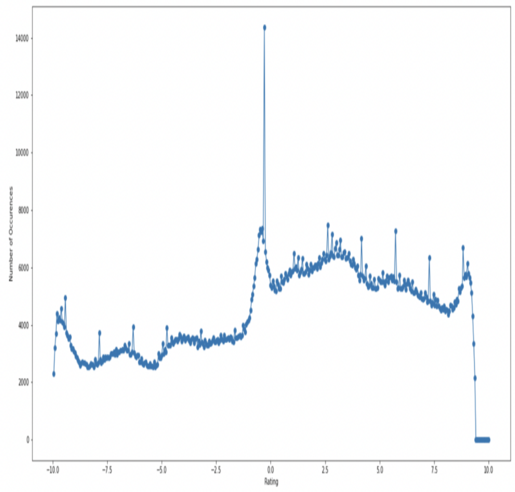


Figure 4: Number of Occurrences vs Ratings

- Figure 4 gives the frequency of unique ratings. It can be seen that ratings from 0 to -10 are comparatively less frequent than ratings from 0 to +10. The rating of -0.29 was the most frequent among all ratings. It can also be seen that there were very few ratings with high absolute values. This suggests that not many users preferred giving really high or extremely ratings. Also, this suggests that the data is not skewed.

3.3 Data Pre-processing

There were 687845 ratings in the matrix whose values were 99, representing the missing ratings which were to be imputed. Along with these ratings, we further created a sparse matrix by replacing 12.5%,25%,30% of the ratings with a value of 99. These deliberately replaced values were then used to compute losses to evaluate the performance of our models. For computing the item based similarities, we created a dense matrix that contained ratings of only those users who had rated all jokes.

4 Problem Formulation

The task of recommending content to users can be formulated as an unsupervised machine learning problem where we have to impute the missing data. However, because of the sparse nature of the data, we cannot plug the data directly into SkLearn library functions. Here, in this task we use various methods like KNN and their variants, Matrix Factorization, NMF, and Neural Networks to predict the ratings not present in our dataset using item based similarities. We have computed two loss functions L1 loss and Frobenius norm (L2 loss). We used L1 loss to compare the performance of our KNN based methods and used Frobenius norm (L2 loss) for Matrix Factorization, NMF, Neural Networks and Ensemble. We also performed hyperparameter tuning for our parameter of number of neighbors for KNN and number of components for NMF. We used similarity metrics of Manhattan distance, Cosine Similarity and Pearson Coefficient for the KNN based methods which are popular and efficient ways to compute similarities.

5 Methods

To fill in the missing ratings of the items(in our case: jokes) for a particular user, we followed the following Collaborative Filtering[9][10] methods:

- Imputation using the average rating of item across all users
- Imputation using the different flavors of KNN i.e using the ratings of K most similar items. The different versions of KNN used are :
    1. Average rating of K most similar items
    2. Soft KNN using K most similar items
    3. Soft KNN (item baseline adjusted) using K most similar items
- Matrix Factorization and Non-Negative Matrix Factorization [NMF]
- Neural Networks
- Ensemble

To assess the similarity of items based on their ratings across all other users, we have used 3 similarity methods which are as follows:

- Similarity-based on the average of the Manhattan(L1) distance. The lesser the calculated Manhattan(L1) distance, the more similar the items are.
- Similarity-based on the values of Cosine Similarity between various items. The higher the calculated Cosine Similarity, the more similar the items are.
- Similarity-based on the values of Pearson Correlation between various items. The higher the calculated Pearson Correlation, the more similar the items are.

To assess the performance of the methods, the losses we have calculated are L1 losses across the data values which we withheld from the original dataset before running the above imputation methods on the dataset for KNN based methods. Frobenius norm (L2 loss) has been calculated to assess the performance of Matrix Factorization, NMF, Neural Networks and Ensemble over the complete reconstructed data set.
In order to find the K similar items, we initially created a dense matrix from the data set which is a full rating matrix across all users and items i.e., it is those users' data who have rated all the items. It is on this dense matrix that we calculate the similarity metrics for the item to be imputed with the remaining items. Once we have the items sorted according to the similarity metrics, we impute the values for the user and item in question using the ratings of the user over the K most similar items.

5.1 Baseline Method

We have chosen the "Imputation using the average rating of item across all users" as our baseline method. This is the simplest and most non-intelligent imputation that can be performed to fill in the ratings of the items. In this method, we simply take the average rating across all the user ratings for an item and use that to fill in the missing values of the item. Since this does not take into account the similarities across users/items, it forms a good baseline to test our other methods against.

5.2 KNN [Memory Based]

This type of imputation does not take into account all the item ratings. Instead, it only takes those items into account that have been found to be similarly rated as the item in question by the other users. Since we are here considering only the best subset of items and their ratings hence, KNN is bound to provide better results than the Baseline Method. In this method, K is our hyperparameter and we have calculated the loss across multiple K values to evaluate the best K value per method and similarity metric.
Different flavors of KNN used:

5.2.1 Average rating of K most similar items

In this method, once we find the similar items on the other users from the dense dataset, we use the average rating value of max K similar items that the user in question has also rated to impute the data.

5.2.2 Soft KNN using K most similar items

In this method, once we find the similar items on the other users from the dense dataset, we use the weighted average of the rating value of max K similar items that the user in question has also rated to impute the data. In this method, the user rating is weighted by the similarity value that we calculated for that joke over the dense matrix for other users.

5.2.3 Soft KNN (item baseline adjusted) using K most similar items

This method is an improvement over the "Soft KNN using K most similar items". In this, we adjust the ratings calculated by subtracting the baseline/average rating value of a similar joke from the rating value of the user in question for that item and then adding the baseline/average rating value user in question over all the items rated. This is an improved version as here we adjust the ratings for any systematically hard/soft rated items/users.

5.3 Matrix Factorization and Non-Negative Matrix Factorization [NMF] [Model Based]

The idea behind Matrix Factorization[11] and Non-Negative Matrix Factorization [NMF] (flavor of Matrix Factorization with non-negativity constraint for which we had to rescale our data) is to represent ratings of users and items in a lower-dimensional latent space i.e we create embeddings for users and items. These lower-dimensional latent space matrices are then used to reconstruct the rating matrix which is our predicted ratings. Using the predicted ratings for the items for which we possess the original ratings, we are able to calculate the Frobenius Norm via which are able to assess the performance of the model.

Error Function for Matrix Factorization:
$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{kj})^2$

Update Rules for item and user embedding:
$p'_{ik} = p_{ik} + \alpha\frac{\partial}{\partial p_{ik}}e_{ij}^2 = p_{ik} + 2\alpha e_{ij}q_{kj}$

5.4 Neural Networks

For the Neural Networks based model, we create two embeddings for every rating i.e. a user based embedding and a joke based embedding depending on the user who rated it and for the corresponding joke. The ratings were initially scaled from -1 to 1 and those values with 99 were replaced by the average of row/column.The user embeddings for every user was a vector of the user's 100 joke ratings. While for creating joke embeddings we created a 100*100 joke similarity matrix based on cosine similarity. So, the joke embeddings for every joke was a vector of 100 joke similarity values in the order of increasing similarity to other jokes. For every rating, the user embedding and joke embedding were concatenated horizontally to form a resultant vector of 200 values. Thus, the input consisted of 25K*100 vectors each with a dimension of 200. This was passed to a 2 layer neural network with 120 neurons and 60 neurons while the output layer consisted of one neuron which was the rating itself. The data was split into a train set and a validation set with a split of 0.125. We used a learning rate of 0.01 and stochastic gradient descent. We then predicted ratings for every user and joke and calculated the L2 loss (for those values which were not originally 99).

5.5 Ensemble Method: NMF + tuned Soft KNN with Item Baseline (Pearson Similarity)

5.5.1 Average Based Ensemble

We took our fully predicted rating matrices of both the methods and averaged (giving equal weights to each) to get the final rating. We used L1 and L2 loss metrics to analyse the results.

5.5.2 Regression-Based Ensemble

We took our fully predicted rating matrices of both the methods and assigned weights to each of the methods. To predict our weights, we split our data into training(80%) and testing(20%) and ran a standard linear regression model.

5.6 Effects of Sparsity

Once we identified Soft KNN (item baseline adjusted) as the best method for imputation of the missing values, we wanted to study the effects of sparsity of the dataset on the performance of the method across all the similarity methods used. We varied the sparsity of the dataset willingly to 12.5%, 25% and 30% and compared the L1 loss calculated for the predicted values. Following are the results.

Percentage Sparsity	L1 Loss	Optimum K
12.5%	3.223	10
25%	3.2374	10
30%	4.1955	10

This proves that as the sparsity of the dataset increases, the performance of the models decreases as the L1 loss calculated can be shown to increase.

5.7 Recommendations

At the end, we picked the best method: weighted ensemble of NMF and Soft KNN with item baseline adjustment and Pearson Correlation Similarity to predict jokes for each user.

Packages Used:

- We used the "scipy" package to calculate the Cosine Similarity (scipy.spatial.distance.cosine) and the Pearson Correlation (scipy.stats.pearson)
- We used the "sklearn.impute" package to perform baseline imputation (sklearn.impute.SimpleImputer) using the average ratings.
- We used the "sklearn.decomposition" package to perform NMF (sklearn.decomposition.NMF)
- We used the " matplotlib.pyplot" package for plotting the graphs
- We used keras for neural networks

6 Experiments and Results

The following sections are related to the hyperparameter tuning of the various models based on the losses.

6.1 Memory Based Methods:

6.1.1 KNN

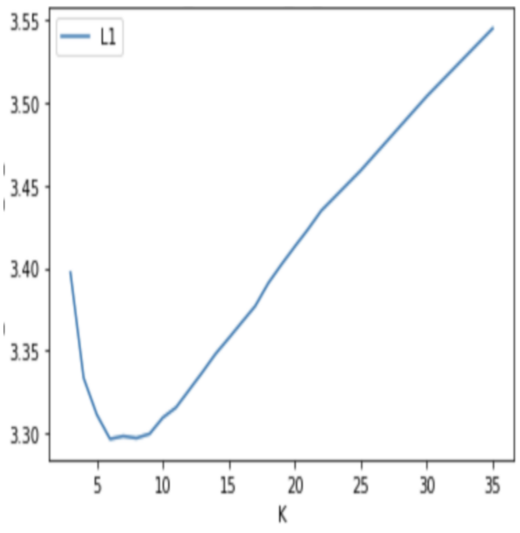


Figure 5: KNN(Manhattan) L1 Loss vs k

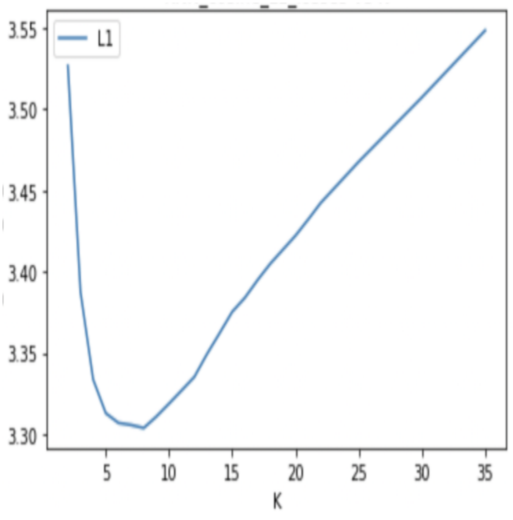


Figure 6: KNN(Cosine) L1 Loss vs k

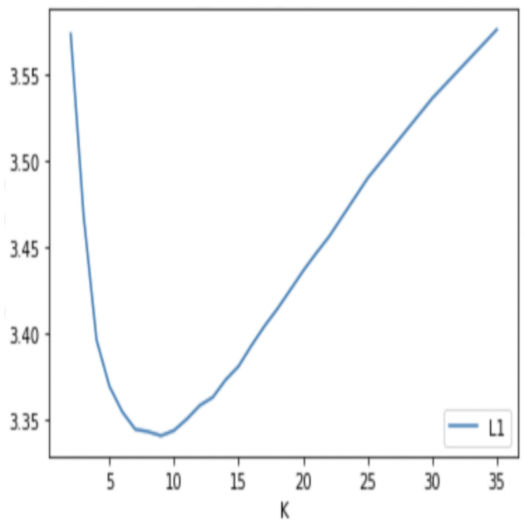


Figure 7: KNN(Pearson) L1 Loss vs k

KNN Results:

	Manhattan Similarity	Cosine Similarity	Pearson Similarity
Optimal value of K	6	8	9
L1 loss	3.2965	3.3037	3.3407
L2 loss	18.7263	18.7295	18.7304

6.1.2 Soft KNN

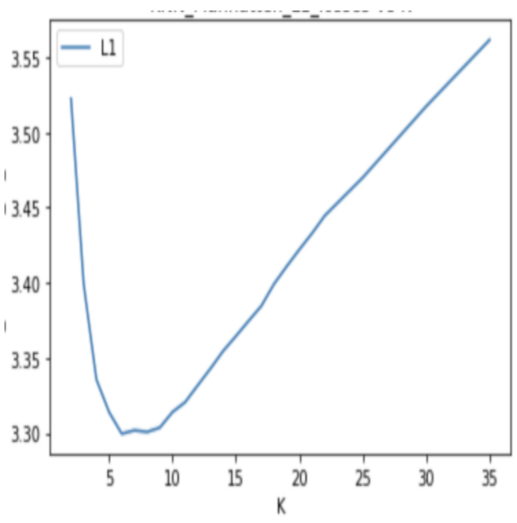


Figure 8: Soft KNN(Manhattan) L1 Loss vs k

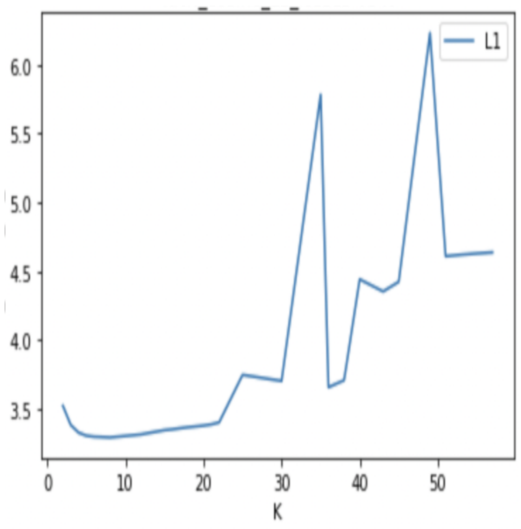


Figure 9: Soft KNN(Cosine) L1 Loss vs k

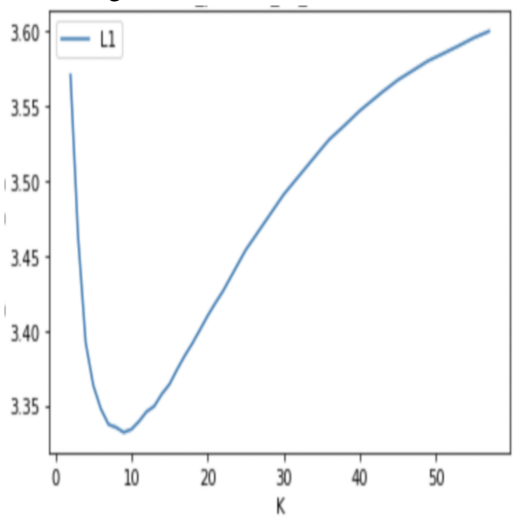


Figure 10: Soft KNN(Pearson) L1 Loss vs k

Soft KNN Results:

	Manhattan Similarity	Cosine Similarity	Pearson Similarity
Optimal value of K	6	8	9
L1 loss	3.2995	3.2939	3.3320
L2 loss	18.7506	18.6458	18.6595

6.1.3 Soft KNN with item baseline Results

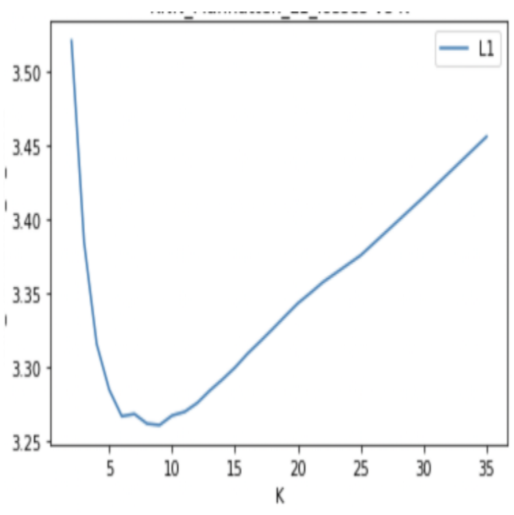


Figure 11: Soft KNN (Item Baseline)(Manhattan) L1 Loss vs k

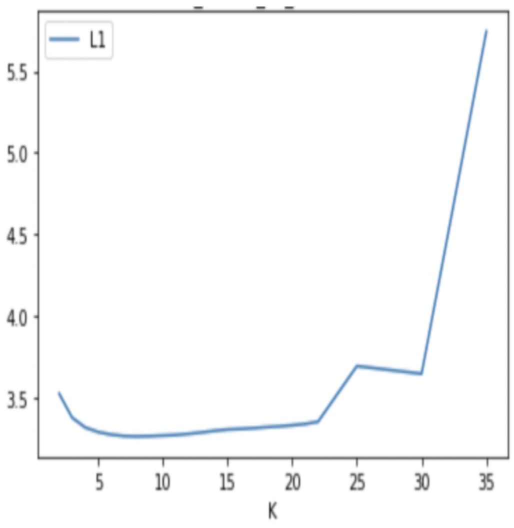


Figure 12: Soft KNN (Item Baseline)(Cosine) L1 Loss vs k

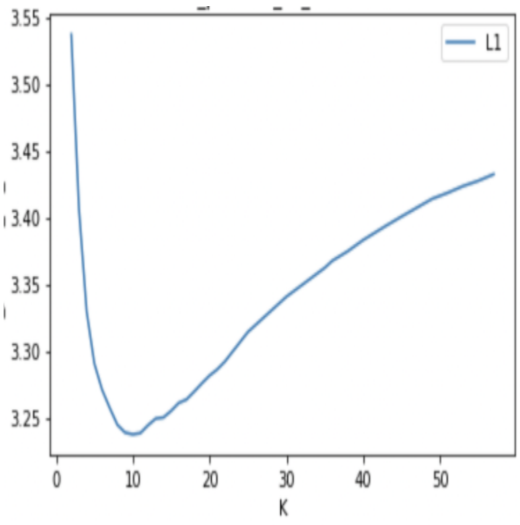


Figure 13: Soft KNN (Item Baseline)(Pearson) L1 Loss vs k

Soft KNN with item baseline Results:

	Manhattan Similarity	Cosine Similarity	Pearson Similarity
Optimal value of K	9	8	10
L1 loss	3.2605	3.2582	3.2374
L2 loss	17.9768	18.0198	17.7002

6.2 Memory Based Methods

6.2.1 Matrix Factorization

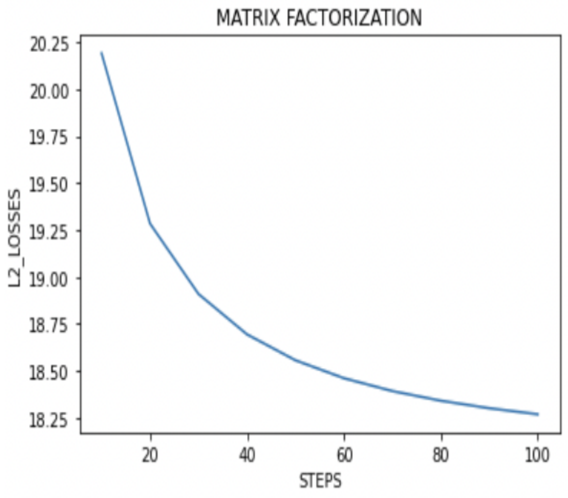


Figure 14: Matrix Factorization L2 Loss vs Iteration Size

6.2.2 Non Negative Matrix Factorization:

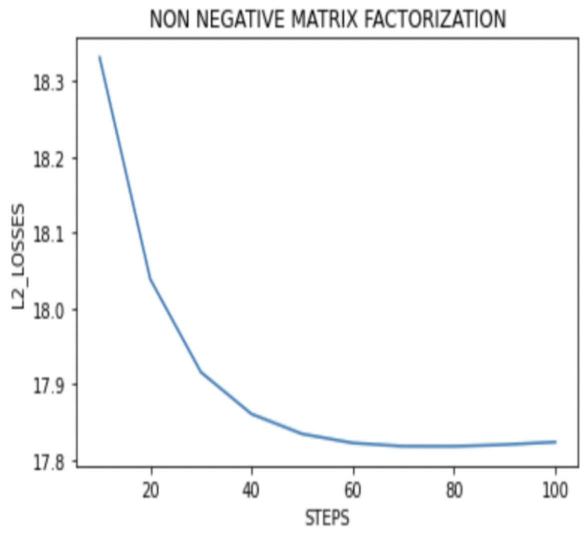


Figure 15: NMF L2 Loss vs Iteration Size

6.2.3 Neural Network Based Model

Training and Validation Loss:

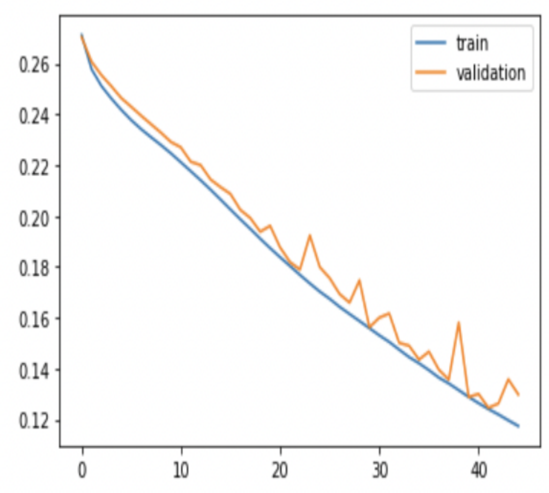
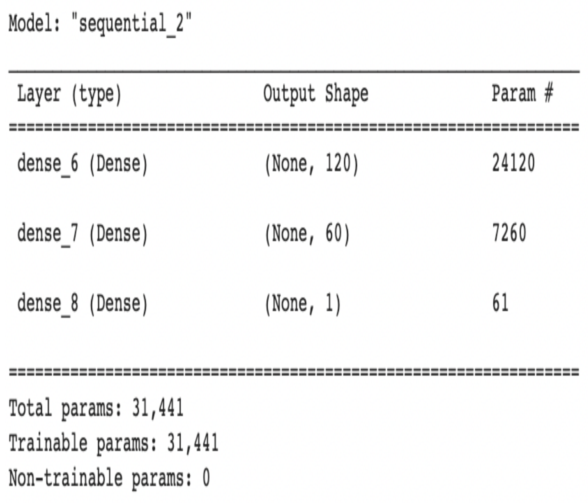


Figure 16: Mean Absolute Train Test Validation for NNs

Architecture:



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 120)               24120

dense_7 (Dense)              (None, 60)                7260

dense_8 (Dense)              (None, 1)                 61

=================================================================
Total params: 31,441
Trainable params: 31,441
Non-trainable params: 0
```

Figure 17: NN-Architecture

Results for Model based methods with no. of components = 2

Method	Reconstruction Loss
Matrix Factorization	20.23855
Non Negative Matrix Factorization	17.90958
Recommendation using Neural Based Embeddings	17.88526

6.3 Results for Ensemble Method (NMF + Soft KNN with item baseline, Pearson Similarity)

6.3.1 Average Based

	NMF	Soft KNN with item baseline, Pearson Similarity	Ensemble
L2 Loss	17.90958	17.7002	17.8112

6.3.2 Regression-Based

	NMF	Soft KNN with item baseline, Pearson Similarity	Ensemble
L2 Loss	17.90958	17.7002	17.0566

Evaluation of Metrics

- For memory-based methods, it's sensible to consider the L1 loss because we are interested in the difference between the original ratings and our predicted ratings i.e how far away is the predicted rating from the original rating (on a scale of -10 to 10).
- For model-based methods, we aim to minimize the Frobenius norm i.e the sum of L2 squared loss.

7 Conclusion and Discussion

7.1 Best Similarity Metric for each of the methods

- KNN: Manhattan for K = 6
- Soft KNN: Cosine for K = 8
- Soft KNN with item baseline: Pearson for K = 10

7.2 Memory Based

- Best method: Soft KNN with item baseline for Pearson Similarity (K =10)

7.3 Model Based

- Best Method: Neural Based Recommender System

Since neural best recommender systems required more computing power, we used NMF for our ensemble method:

7.4 Ensemble

- Best Method: Soft KNN with item baseline + NMF: Regression-based Ensemble

7.5 Qualitative results

- Ensembles perform best when done correctly over the models of choice.
- As the sparsity of the dataset increases, the performance of the models decreases.

7.6 Insights

The major part of the problem is to correctly predict the ratings which are missing. Such a problem space of ratings is scale invariant when all the ratings are scaled with a constant factor.

The similarity metrics used do not depend on the sign of the ratings (whether the ratings are positive and/or negative).

7.7 Lessons learnt

When adjusting the values for Soft KNN based on a baseline, we found that using the item baseline is not a good idea as that value gets adjusted in the overall computation leading to the model being the same as Soft KNN.

Ensembling of methods needs to be done correctly. When we assembled the results for Soft KNN (item baseline adjusted) and NMF giving equal weights to the predictions made by both, the performance of the ensemble was the average of that of Soft KNN (item baseline adjusted) and NMF. Using regression on the predictions of Soft KNN (item baseline adjusted) and NMF resulted in a more robust model.

SKlearn.impute.SimpleImputer will drop the columns which do not have any value, at the time of fit, which can be used to impute the data. This can lead to incorrect results and loss calculated. We can fill such columns with a constant value which can be specified when defining the imputer.

7.8 Novelty

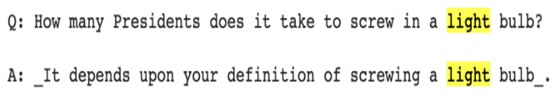
- Ensemble
- Neural based embeddings

7.9 Future Work

- Content-Based Recommender Systems using LDA

Interesting Observations

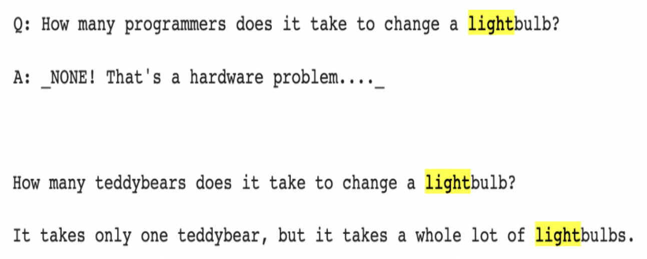
For user 7889, a highly rated joke:



```
Q: How many Presidents does it take to screw in a light bulb?

A: _It depends upon your definition of screwing a light bulb_.
```

What our system recommended:



```
Q: How many programmers does it take to change a lightbulb?

A: _NONE! That's a hardware problem...._


How many teddybears does it take to change a lightbulb?

It takes only one teddybear, but it takes a whole lot of lightbulbs.
```

This user has a particular predilection for jokes about light-bulbs.

Another intriguing thing to note was the users who liked longer jokes were recommended some longer and elaborate jokes.

Acknowledgments

The authors especially thank the University of Pennsylvania CIS520 Machine Learning instructors, Dr. Ungar and the TAs, for their advice and support throughout this project.

References

1. Recommender System Application Developments: A Survey by Jie Lu et al., April 2015, DOI:10.1016/j.dss.2015.03.008

2. A Hybrid Approach using Collaborative filtering and Content based Filtering for RecommenderSystem: G Geetha et al 2018 J. Phys.: Conf. Ser. 1000 012101

3. Collaborative Filtering with Neural Networks, https://medium.com/analytics-vidhya/collaborative-filtering-with-neural-networks-e9691683701f

4. Comparison of Collaborative Filtering Algorithms with Various Similarity Measures for Movie Recommendation, Taner Arsan et al,(IJCSEA) Vol.6, No.3, June 2016

5. Hallinan B, Striphas T. Recommended for you: The Netflix Prize and the production of algorithmic culture. New

Media Society. 2016;18(1):117-137. doi:10.1177/1461444814538646

6. Winning the Netflix Prize: A Summary, Edwin Chen, https://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/
7. Deep Learning based Recommender System: A Survey and New Perspectives, Shuai Zhang et al
8. Anonymous Ratings Data from the Jester Online Joke Recommender System: https://goldberg.berkeley.edu/jester-data/
9. Medium Article - https://towardsdatascience.com/user-user-collaborative-filtering-for-jokes-recommendation-b6b1e4ec864 2
10. Music Recommender System - https://medium.com/@brendanngo/muse-a-music-recommendation-system-f11fcb2baf63
11. Matrix Factorization:Collaborative Filtering, https://albertauyeung.github.io/2017/04/23/python-matrix-factorization.html/