Introduction to Recommender Systems
Project Report
Comparison of Recommendation Models On the Amazon Automotive Dataset

Abhay Mittal and Suraj Subraveti

May 2017

1 Introduction

Recommender systems have become an integral part of most online services today, be it recommending news articles, products for people. There has been a lot of research in this area and a lot of algorithms have been developed for doing this. The aim of this project is to compare the performance of some popular recommendation algorithms on an Amazon dataset, containing ratings of automotive products purchased by various users on Amazon. We measure the Mean Absolute Error(MAE) and Root Mean Squared Error(RMSE) of rating predictions made by these algorithms trained on a subset of user-item pairs.

In addition to just ratings given by users, we also experimented with reviews users gave for the products they purchased. We obtained sentiment analysis scores of these reviews and combined these with the ratings given by the users to compared the performance of these algorithms on both types of ratings.

Finally, we generated the top 10 recommended items for all users and evaluated these recommendations using Precision, Recall and F-Measure.

The next few sections will shed more light on our pipeline, the dataset we used, the algorithms we compared, and the results we observed.

2 Dataset and data preprocessing

2.1 Dataset

We used the Amazon Automotive dataset [2] for analysis. This dataset contained 20473 user-item pairs, with 2928 users and 1835 items. Clearly, the user-item matrix for this dataset is really sparse. Out of approximately 5 million entries, only 20473 exist.

2.2 Data preprocessing

We used Python's Pandas [8] library to pre-process the dataset. We got user item pairs and corresponding reviews into a Pandas dataframe, and to create the train and test splits, for each user, we stored 80% of his/her items into the train set, and the rest into the test set. All entries are shuffled once for each user before splitting them into train and test sets. After creation of the train and test splits, the entries in each of these sets are shuffled again so that the sets do not have contiguous blocks of samples from the same user. Note that we stored intermediate results as pickle dumps to increase the speed of pre-processing in the later runs.

3 Recommendation Algorithms

To perform our analysis on this dataset, we used recommendation algorithms implemented in the Python library Surprise [3]. To make the package use only our training set to create the folds for cross validation and not the test set as well, we had to create a new wrapper class which inherited the DatasetAutoFolds class of Surprise. We now explain the recommendation algorithms we experimented with briefly:

3.1 Singular Value Decomposition

Singular Value Decomposition (SVD) is a Matrix Factorization algorithm which involves decomposing a matrix into a set of factors to try and uncover latent features. It is difficult to apply SVD directly on the user item ratings matrix as traditional SVD is not defined for matrices with missing values which are prominent in the user-item matrix. So generally a regularized model is used and only the observed ratings are modeled [4].
The SVD algorithm transoforms users and items to a shared latent space. That is, every item i is mapped to a vector $q_i \in \mathbb{R}^f$ and every user u is mapped to a vector $p_u \in \mathbb{R}^f$ in the shared space. The relationship between the user and item is given by the inner product of their vectors [9]. The rating for item i by user u is predicted as:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \tag{1}$$

where μ is the global mean for ratings, b_i and b_u are the item and user baseline predictors (biases), i.e., deviation of user u and i from average ratings [4, 9].

Learning the model parameters is done by minimizing regularized square error:

$$\min_{p*,q*,b*} \sum_{(u,i)\in TrainSet} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \|p_u\|^2 + +\|q_i\|^2) \tag{2}$$

where λ is the regularization constant. Surprise uses stochastic gradient descent to perform this minimization.

3.2 Non-negative Matrix Factorization

Non-negative Matrix Factorization (NMF) like SVD, is a matrix factorization algorithm in which the user-item matrix is decomposed into user and item factors. The condition, however, is that the user-item matrix and the factors must all have non-negative values [5]. The predicted rating is given as:

$$\hat{r}_{ui} = q_i^T p_u \tag{3}$$

The user and item factors are initialized with random values and the optimization is done by stochastic gradient descent [7]. This algorithm is highly dependent on the initialized values for the factors. User and item baselines (biases) can also be incorporated but the model becomes susceptible to overfitting which however can be controlled by a good choice of the regularization parameter.

3.3 K-Nearest Neighbors With Means

In user based K-Nearest Neighbors with Means (KNNM), we take the k nearest neighbors (near is defined on the basis of a similarity function) who have rated item i and use their ratings and similarity to predict the rating for user u on item i [9, 3], i.e., the rating is predicted as:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v\in N_i^k(u)} sim(u,v)(r_{vi} - \mu_v)}{\sum_{v\in N_i^k(u)} sim(u,v)} \tag{4}$$

where,
$\hat{r}_{ui}$ - predicted rating
μ_u - average rating by user u
$N_i^k(u)$ - set of k nearest users who have given ratings to item i
$sim(u,v)$ - similarity score of u and v

An item-based version of this algorithm also exists in which we check the k nearest items rated by user u and use their ratings and similarity with item i to predict the rating for item i.

3.3.1 Similarity Measures

We used the following similarity measures [3] in cross validation:

a) **Cosine Similarity**
It is defined as:

$$cosine\_sim(u,v) = \frac{\sum_{i\in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i\in I_{uv}} r_{ui}^2}\sqrt{\sum_{i\in I_{uv}} r_{vi}^2}} \tag{5}$$

where I_{uv} is the set of items that both user u and user v have rated

b) **Mean Squared Difference similarity**
The Mean Squared Difference is defined as:

$$msd(u,v) = \frac{1}{|I_{uv}|} \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2 \tag{6}$$

where I_{uv} is the set of items rated by both user u and user v.
The similarity is then defined as:

$$msd\_sim(u,v) = \frac{1}{msd(u,v)+1} \tag{7}$$

c) **Pearson similarity**
It is defined as:

$$pearson\_sim(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)(r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}} \tag{8}$$

where, I_{uv} is the set of items rated by both user u and user v.

4 Sentiment Analysis

As stated above, we also wanted to see the effect of reviews given by users on the rating prediction.For this, we used a Python library called TextBlob[6], a natural language processing library built on top of Stanford's Natural Language Processing Toolkit(NLTK)[1].

TextBlob does sentiment analysis in two ways:

- Classifies sentences as positive or negative based on a Naive Bayes Classifier trained on movie review data
- Generates sentiment scores based on analysis of patterns in a sentence.

We chose the second option because we needed a continuous representation of the sentiment scores, so that they could be combined with user ratings.

After obtaining sentiment scores for all reviews given by users, we combined these sentiment scores with the normal ratings by scaling both these to zero mean and unit standard deviation.Mathematically, the combined rating r_c for a given user-item pair was obtained as follows:

$$r_c = r'_u + r'_s$$

where

$$r'_u = \frac{r_u - \bar{r_u}}{SD(r_u)}$$
$$r'_s = \frac{r_s - \bar{r_s}}{SD(r_s)}$$

r_u = User rating on a particular product
r_s = Sentiment score provided by TextBlob on the review of that product by the user
$\bar{r_u}, \bar{r_s}$ = Mean of the entire range of user ratings and sentiment scores in the dataset.
$SD(r_u), SD(r_s)$ = Standard deviations of user ratings and sentiment scores

We did this because the sentiment scores given by TextBlob and the user ratings are on different scales, and thus couldn't be added to each other directly.

5 Hyperparameter optimization

For selecting the best hyperparameters for each algorithm, we used GridSearch over a range for each hyperparameter. For each unique combination of hyperparameters, we measured the validation RMSE obtained on performing five-fold cross validation on the training data, and selected the combination of hyperparameters that gave rise to the least RMSE. The following hyperparameters were optimized for each algorithm:

- SVD: Number of epochs, Number of factors
- NMF: Number of epochs, Number of factors
- KNNWithMeans: Number of neighbors, similarity measure.

We did hyperparameter selection on both types of data, i.e. normal ratings, and combined ratings. Here are the plots of validation RMSE and MAE for each algorithm:

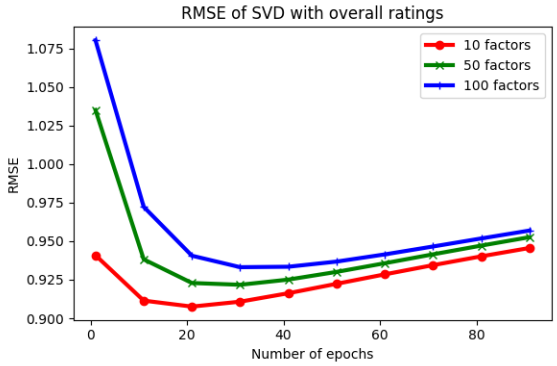

Figure 1: Validation RMSE for SVD on user ratings

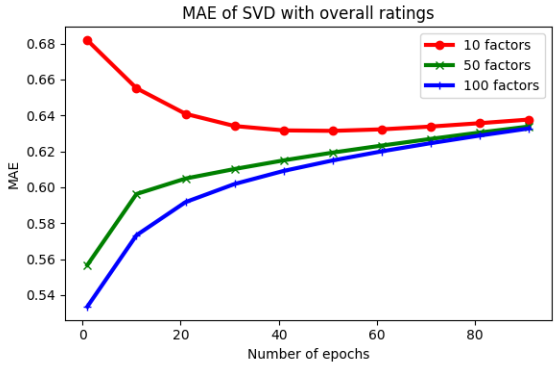

Figure 2: Validation MAE for SVD on user ratings

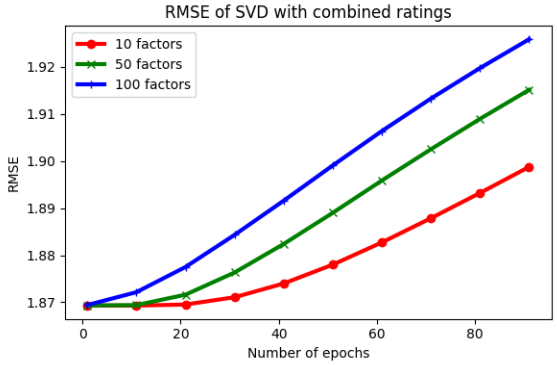

Figure 3: Validation RMSE for SVD on combined ratings

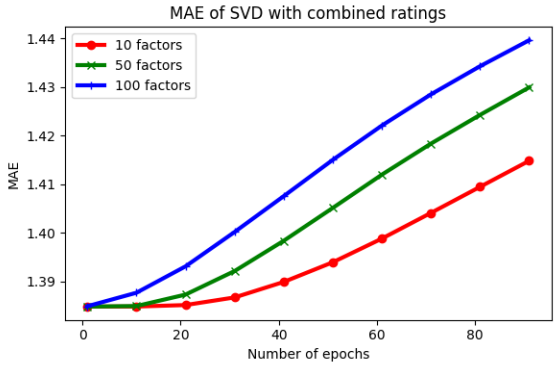

Figure 4: Validation MAE for SVD on combined ratings

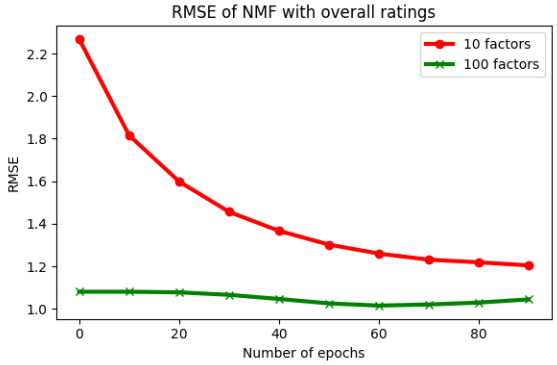

Figure 5: Validation RMSE for NMF on user ratings

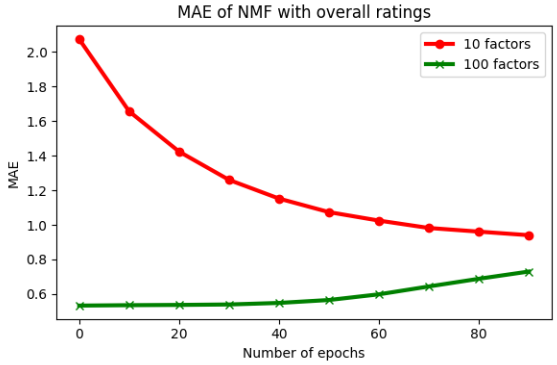

Figure 6: Validation MAE for NMF on user ratings

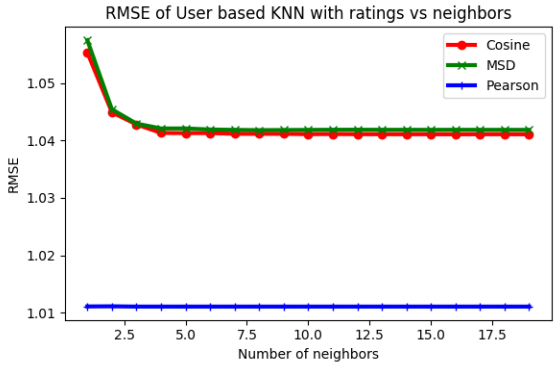

Figure 7: Validation RMSE for KNNWithMeans on user ratings

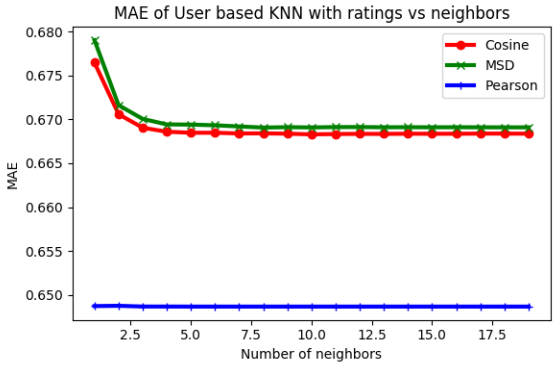

Figure 8: Validation MAE for KNNWithMeans on combined ratings

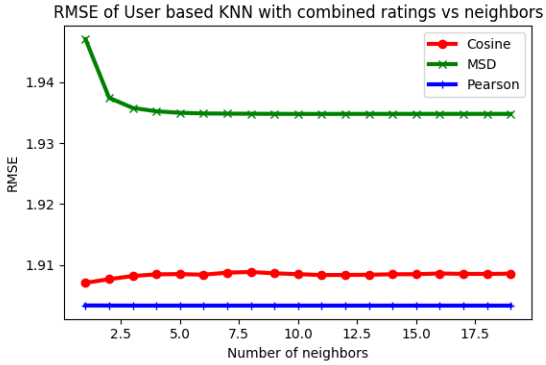

Figure 9: Validation RMSE for KNNWithMeans on combined ratings

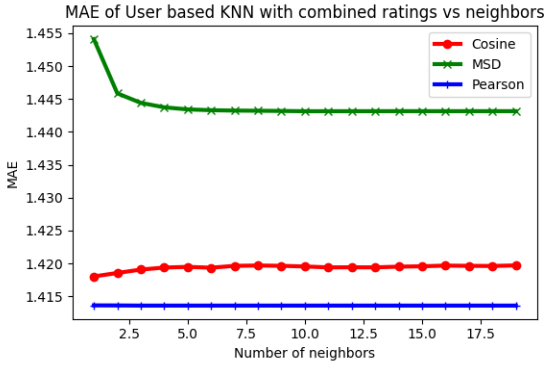

Figure 10: Validation MAE for KNNWithMeans on combined ratings

6 Top N recommendations

After testing on the test set, we decided to generate Top 10 recommendations and measure the performance of the trained models. We evaluated the models based on precision, recall and F-measure. To do this, we first took 1000 random items which the user did not buy and the items in the test set. We then predicted the ratings for all the items in this set and ranked the items according to the ratings. Finally we measured the precision, recall and F-measure of our recommendations. The results are described in section 7.2.

7 Performance Evaluation

7.1 Rating predictions

After hyperparameter selection, we tested the best models picked by GridSearch on the dataset. We performed three experiments:

- Select best model parameters for the dataset with user ratings,and train on and predict user ratings
- Select best model parameters for the dataset with combined ratingsand train on and predict these combined ratings
- Select best model parameters for the dataset with user ratings, train on and predict combined ratings

The results that we found are shown below:

Algorithm	Score	User ratings	Combined ratings
SVD	RMSE	0.8746	1.8454
	MAE	0.6084	1.3734
NMF	RMSE	0.9597	2.0372
	MAE	0.5363	1.4979
KNNM	RMSE	0.9406	1.8650
	MAE	0.6150	1.3891

Table 1: RMSE and MAE of the dataset on test data

As we can see from the above table, the performance of NMF on user ratings is good, but surprisingly bad on combined ratings.

We did perform the third experiment, but since it didn't show any promising results, we didn't record the results here.

We first didn't run NMF on combined ratings, because as mentioned before, for NMF, the entries in the user-item matrix all need to be non-negative, which is not true in case of combined ratings. But we were curious to see what would happen if we did do this, and clearly the stochastic gradient descent optimization process failed to estimate the factors correctly.

7.2 Top N recommendations

To measure the quality of our Top N recommendations, we used precision, recall and F-score as mentioned above. The results of our experiments are shown below:

Algorithm	Score	User ratings	Combined ratings
SVD	Precision	0.45%	16.88%
	Recall	2.26%	97.69%
	F-measure	0.0075	0.2879
NMF	Precision	9.88%	6.32%
	Recall	55.22%	3.73%
	F-measure	0.1675	0.0108
KNNM	Precision	16.5%	16.97%
	Recall	97.1%	98.5%
	F-measure	0.2822	0.2894

Table 2: Precision, Recall and F-measure for Top N recommendations

We can see here also that the NMF model which wasn't applicable for Combined ratings performed very bad. Furthermore, we can see that there is an improvement for SVD and KNNM when using combined ratings. We believe the reason for this is that most of the values in the combined ratings dataset are continuous values in the scale [-1,1] while the user ratings have discrete values in [0,5]. So, even a small difference in the estimated rating for combined dataset can change the rank by a lot.

8 Conclusion

From the results, it is clear that the combined ratings model didn't do well. To investigate this, we looked at the sentiment scores of the reviews, and found them anomalous.

Some reviews that had a few negative words but actually indicated that the customer was happy with the product were given lower scores, and since the user was happy with the product, the user rating for such products was high.

For example, a product with review

"Takes the dust off my car without leaving any streaking that some report with other brands. Just don't press real hard"

which indicates that the product was actually good, received a sentiment score of -0.28, whereas it had a user rating of 5.

It could also be the case that finding a better way to combine sentiment scores and user ratings might actually lead to a better result.

9 Code

The code is posted online at https://bitbucket.org/abhaymittal/recommender_project

References

[1] Edward Loper Bird, Steven and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc, New York, NY, USA, 1st edition, 2009.

[2] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*, pages 507–517. International World Wide Web Conferences Steering Committee, 2016.

[3] Nicolas Hug. Surprise, a Python library for recommender systems. http://surpriselib.com, 2017.

[4] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[5] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[6] Steven Loria. Textblob: Simplified text processing. https://textblob.readthedocs.io, 2017.

[7] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, 10(2):1273–1284, 2014.

[8] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[9] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.