An Empirical Study on Performance Measures of Collaborative Filtering Recommendation Algorithms

Alex Lu

maodou1258@gmail.com

**Abstract**

In recent times, recommendation engines have become increasingly popular within many industries. The focus of such an engine is to implement an algorithm to successfully make recommendations based on user preferences. Because of the utility provided by such tools, many industries heavily rely on such engines to provide accurate product recommendations to customers. Some popular algorithms implemented include clustering, matrix factorization, and deep learning implementations. Such algorithms are utilized within Collaborative Filtering methods and are simulated using pre-made datasets containing user and item information. Performance metrics are then applied to the algorithm to test the accuracy of the model. Results show that deep learning algorithms provide greater

performance when compared to other applications.

**Keywords -** Collaborative filtering, RMSE, Deep Learning, MAE, matrix factorization, Performance Measure

**1. Introduction**

Latterly, recommendation engines serve a great purpose in many online services and enhance the consumer experience by providing lists of recommended items to users. Many big corporations such as Netflix, Amazon, Youtube, and others provide item recommendations through such methods to promote the sales and usage of their goods. The general idea of a recommender system is to return a list of items that the user would find interesting. Implementing various different types of engines provides insight towards which algorithm is the most suitable for certain scenarios.

Different approaches by different algorithms could have varying performances depending on the type of recommendation required, and the core nature of the dataset used. This research is primarily conducted on collaborative-filtering techniques, but similar research could be conducted on content-based implementations. Examples of potential implementations could be clustering and matrix factorization approaches. The main accuracy metrics utilized in the research are the RMSE and MAE metrics which are further explained later in the paper.

Two main research questions would serve as the focus of the research. More conclusions could be drawn from the data collected, but the main aim of the research are as follows:

**I)** What CF implementation provides the accuracy measure for a standard data set containing user, item and rating information? **II)** How does data sparsity affect the prediction accuracy of the models implemented?

**2. Background**

Recommendation engines are programs used to provide item recommendations to users based on filtering each item and returning a

possible predicted rating for the user. The two main implementations of recommendation engines are content-based filtering and collaborative filtering.

**2.1 Content-based Filtering**

A content-based filtering engine takes into account the user’s own item history and focuses on keywords in items rather than similarity between users [1]. However, such implementation can lead to a scenario where providing a broad range of recommendations would become impossible. Because of the nature of content-based filtering engines, items that were utilized by similar users wouldn’t be recommended to the main user purely because of the lack of a keyword or phrase. A dataset for such an engine would incorporate items along with a detailed profile for each item.

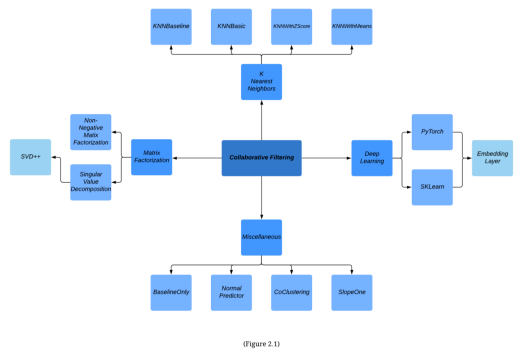
Despite its narrower scope when providing recommendations, content-based filtering systems often reduce the amount of data needed to make accurate predictions. For collaborative filtering, a greater range of data is required for a proper calculation of an item. To put it simply, the more data the engine has, the more

accurate the prediction is. However, with the nature of content-based filtering, any amount of data would suffice, providing that there are items that match the recommendation requirements.

**2.2 Collaborative Filtering**

A collaborative filtering engine takes into account other user’s ratings, and returns

algorithms. This paper would primarily focus on model-based algorithms and approaches. The similarity between users could be found in various methods. For this study, a total of twelve algorithms were implemented. The main focus was placed on nearest neighbors, matrix factorization, and deep learning algorithms. Figure 2.1 maps out the various



recommendations based on similarity. Collaborative filtering is split into two main approaches, model-based and memory based

algorithms implemented in this research. **2.3 Nearest Neighbor**

Nearest neighbor, or more specifically, k nearest neighbors is a collaborative filtering

algorithm used to find similarities between items and users based on the total distance between two neighbors [2]. A weight system could also be applied so that a neighbor closer to the main user would have more weight on the recommendations than one who is further away.

Four of the twelve algorithms used in the research were from this category. KNNBasic, KNNWithZScore, KNNWithMeans, and KNNBaseline were such implementations of this category of algorithms.

**2.4 Matrix Factorization**

Matrix Factorization is yet again another implementation of collaborative filtering. Simply put, a matrix factorization relates two separate values together under a specific value to create a grid or matrix of the data [3]. However, as data is not evenly distributed, some cells in the matrix are left empty and would need to be filled in to provide recommendations. Such values are known as “latent features”.

Some matrix factorizations implemented in the research were Negative Matrix Factorization, Singular Value Decomposition, and SVD++.

**2.5 Deep Learning**

A deep learning model utilizes a neural network to process and calculate information. Much like a human brain, deep learning attempts to make predictions based on data much like a human brain.

The deep learning model implemented in this experiment utilizes an embedding layer structure [4]. Using such a structure could organize data into a vector of discrete values which could then produce similarity results and tests through the distance between vectors. These types of embedding layers could be generated through frameworks such as “Pytorch” or “Tensorflow”, however, this research would be implemented in Pytorch. The model could then be trained through a series of epochs and eventually provide predictions.

**2.6 Miscellaneous surprise algorithms**

The miscellaneous category comprises the leftover algorithms in the surprise package that do not fit into any of the other categories in the research. Such algorithms were as listed: Co

Clustering, Normal Predictor, Baseline-Only, and Slope-One.

According to Sahar Karat Co-Clustering is a collaborative filtering algorithm used to provide recommendations by “A simultaneous clustering of the rows and columns of a matrix” [5]. Classical clustering algorithms only focus on one specific type of data, while co

clustering could be used to accommodate two simultaneously.

Normal predictor is an algorithm provided by the surprise package that provides random user recommendations based on the data distribution in the dataset. The “Maximum Likelihood Estimation” method is utilized in this calculation.

Baseline-Only is yet another surprise algorithm that makes baseline predictions based on the data provided. There are two main ways to implement this algorithm. The first implementation method is “Stochastic Gradient

Descent” (SGD), which calculates a gradient of the dataset by a random selection of data. The other implementation is to use “Alternating Least Squares” (ALS) which is a matrix factorization algorithm that works well with sparser sets of data.

Slope-One is an example of an item-based collaborative filtering recommendation algorithm. Predictions made with this model are generally based on personal ratings as well as similar community ratings.

Further information and implementations on the methods listed could be found on the surprise documentations [6].

**2.7 Accuracy Metrics**

Two accuracy metrics were used in the process of this research. Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) [11].

RMSE is a quadratic model that assigns much larger weights to larger errors observed. Accuracy is calculated through RMSE by taking the square root of the mean of the sum of the squared difference of each predicted

value and its corresponding actual value. The formula is displayed in Figure 2.2



(Figure 2.2)

MAE is a linear metric model that linearly scales error. This model is typically used when higher errors are not important to the observation. Accuracy is calculated through MAE by taking the mean of the sum of the absolute values of the differences between the predicted and actual values. The formula is displayed in Figure 2.3



(Figure 2.3)

**3. Related Work**

Various research topics have already been analyzed in the field of collaborative filtering. A few similar pieces of research provide similar methods and implementations. A plethora of CF models has been implemented and researched.

**3.1 Similar Research**

Similar research was conducted by Mojdeh Saadati et al. [7] on the implementations of various models on movie recommender systems. The two models used in the experiment were the matrix factorization model SVD and a deep learning implementation of the Restricted Boltzmann Machine. The performance of the implementations was measured in the RMSE metric. However, in this research, we go over a broader spectrum of algorithms from each group of implementations and produce results for a larger picture between the different implementations.

**3.2 Further Research**

He et al. [8] Conducted Research on how Matrix Factorization could be implemented with Deep Learning to create a better performing model for collaborative filtering. The proposed idea was to use two main models, the Generalized Matrix Factorization (GMF), and the Multi-Layer Perceptron (MLP) to create a hybrid between deep learning and matrix factorization implementations to create a new neural matrix factorization model

dubbed “NeuMF”. NeuMf was then compared to other well-known models such as KNN and ALS implementations and proved to have better accuracy and less training loss.

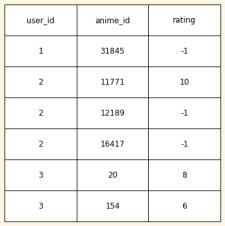
Yedder et al. [9] Researched the performance of the restricted Boltzmann machine. Various hidden units, learning rates, and other factors were incorporated into the research. Problems such as data sparsity were also encountered in the research and required other methods of implementation. This research also utilized the RMSE accuracy metric to rate the performance of the model. The research indicated an excellent RMSE measure of 0.46.

**4. Approach**

This study was broken up into three parts, cleaning, implementation, and evaluation. We will first check for invalid values throughout the dataset, then begin to implement algorithms, and finally calculate performance values.

**4.1 Dataset**

The data that would be primarily used in this study is the Kaggle dataset “Anime

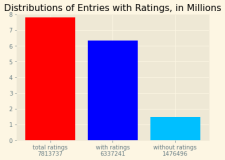
Recommendations Database” [10] based on the data collected from myanimelist.net and is included in the reference section of the study. The two files contained within the database are the “ratings.csv” and “anime.csv”. The research would mostly work with the ratings file, but the anime dataset could be further implemented in future research incorporating variables such as show genre and overall ratings. The anime dataset is separated into three separate columns. The user\_id, anime\_id, and ratings columns are provided for analyzing similarities between users. The format and example of the file are illustrated in Figure 4.1. 

(Figure 4.1)

The python module “pandas” was used to import the data into a data frame which could

then be cleaned and analyzed. Making a copy of the dataset to work on is recommended. Maintaining two different data frames would help compare the results on the original to the results in the cleaned dataset.

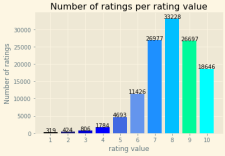
The first thing to take note of while cleaning is to remove all rows with a rating of “-1”. Such rows represent user data that does not have a specified rating for the specific show that they watched. This step is crucial as having such data in the project would result in a sparse dataset, which directly decreases performance as shown later in the results. A total of 7.8 million rows of data were in the dataset, after cleaning, about 6.3 million rows remained. The distribution of this data is shown in figure 4.2.



(Figure 4.2)

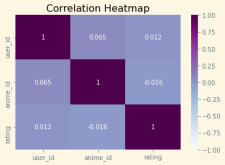
Another issue arose again while trying to apply the dataset to our engine. Certain cells in

our dataset possibly had ‘NaN’ values, which would in turn cause an error in actual predictions where a value of ‘NaN’ would be returned instead of an integer of the predicted rating. To resolve this issue, we first applied the ‘to\_numeric’ method from the pandas module to convert each value into workable integer or float values. After, we could then finally lower the runtime of our experiment by decreasing the size of our dataset from 6 million down to 125,000. This number was picked for easier splitting while implementing other algorithms and deep learning which require testing and training datasets. From there, we then calculated the distribution of the ratings by counting the number of each rating in our dataset. After clearing, it’s paramount to re index the rows in the data frame. Removing the invalid A distribution of the ratings is demonstrated in Figure 4.3.



(Figure 4.3)

A correlation heatmap could also be used to detect any possible correlations between the values given, however, as of now, there is no apparent relation.



(Figure 4.4)

**4.2 Implementation**

The python package scikit-surprise contains most of the algorithms implemented in this research. First and foremost, we must declare a rating scale for our dataset by utilizing the reader class, and also initializing our dataset with the Dataset class. We set a new

“ratings“ object over the range of one to ten, which is represented in our data. Using the three columns from our pandas data frame, we could then load the data into surprise using the *Dataset.load\_from\_df()* method.

After the data is loaded, we could then specify a “param\_list” dictionary. In the dictionary, the name of the specific parameters would be stored as the keys and the possible values as a list in the values. Further explanation of what specific params do for each algorithm is also provided in the surprise documentation.

The param\_list dictionary could then be imputed into the GridSearchCV() method along with the algorithm name, performance measures, and a cross-validation iterator of 3. There are other arguments as well, but for the sake of our research, these four should be enough. The final step is to then fit our premade data with our GridSearchCV method with the .fit() function. Once everything is set up, the accuracy metrics could then just be retrieved via the best\_score iterator.

The deep learning algorithm specifically was generated using an embedding layer with

dropout layers. The entire network consists of 4 total layers. After setting up the net, we can begin to train our model and loop over varying amounts of epochs to find the accuracy that is desired.

To determine the RMSE and the MAE values of this approach, we could separate our dataset into predictions and truth arrays. We can then apply the formulas for both RMSE and MAE as shown in Figures 2.2 and 2.3. We can implement these metrics in two ways. The first method is to use a NumPy array to subject each vector of values from each other and then to apply the formula to our newly generated values. Our second approach is more basic and rudimentary. We could subtract each truth value from each prediction value in our function by declaring a function and then apply the formula to our sum.

**5. Results**

In this research, two sets of results were collected. The data collected on the cleaned dataset would serve to be our solution to the first question. However, the data collected on the original dataset would be utilized to answer

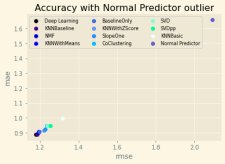
the second question concerning the performance measures on sparse datasets. **5.1 Cleaned Dataset**

We have gathered both RMSE and MAE values for each algorithm used on our cleaned dataset from our implementations. In the first part of this experiment, the data set has already been cleaned of any invalid values and has reduced sparsity. The majority of the algorithms implemented all showed similar results except a couple of outliers and certain points. A table of the data collected is shown in Figure 5.1



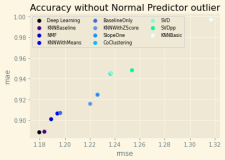
(Figure 5.1)

Using the RMSE as the x-axis and MAE for the y-axis, we can plot a scatter plot of our data (Figure 5.2).



(Figure 5.2)

Removing the Normal Predictor outlier value provides us with a clearer image of the differences in the lower valued points (Figure 5.3).



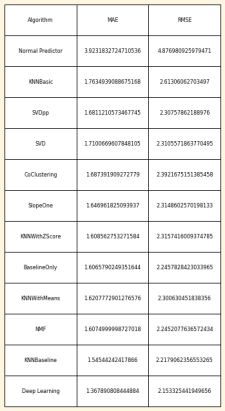
(Figure 5.3)

Our data shows that the worst-performing algorithm that we had implemented was the Normal Predictor algorithm from the surprise package, while the best-performing implementation was the Deep Learning algorithm based on neural networks.

**5.1 Pre-cleaned Dataset**

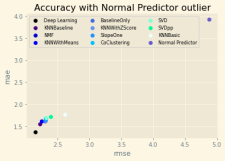
Loading up a new dataset, we can effectively run all our algorithms again while maintaining the sparsity of the original dataset. The only cleaning that had to be done was to remove all NaN values and also convert all data to numeric values. The results of running the

implementations on the sparser dataset are recorded in the figure below (Figure 4.5)



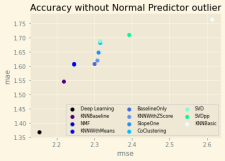
(Figure 5.4)

The results of the various algorithms on this sparse dataset were also plotted on a scatter plot as shown below in Figure 5.5.



(Figure 5.5)

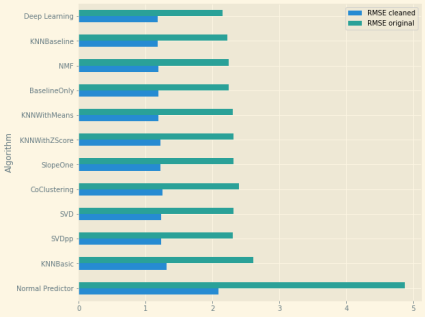
Once again, removing the normal predictor outlier gives a clearer representation of our other data points (Figure 5.6).

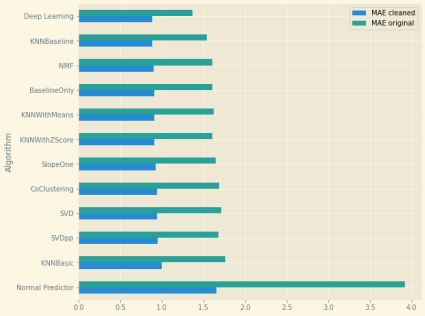


(Figure 5.6)

Although the values have increased by a considerable amount in the sparse dataset, the common trend remains between the various data points, and no changes are observed in the best and worst algorithms. Comparing the

results of the cleaned and original data sets, a clear difference could be observed in the RMSE and MAE measures (Figures 5.7-8)

(Figure 5.7)

(Figure 5.8)

**6. Discussion**

After analyzing the data collected from the research, the deep learning algorithm and the KNNBaseline implementations were observed to be the best performing with the least error observed with both RMSE and MAE metrics.

A possible reason for the results could be the usage of randomness in the calculation of the Normal Predictor algorithm. The deep learning algorithm may have performed the best because of the various training cycles allocated to it which helped to create a more accurate model after each iteration. Vice versa, the opposite could also be applied to the two worst performing algorithms KNNBasic and Normal Predictor. Such implementations

had basic calculations and weren’t able to take into account outliers and other potential biases in the data.

There was an attempt in the research to implement a Restricted Boltzmann Machine (RBM) model, however, the implementation gave varying results and was difficult to judge the extra ratings of the implementation. This

research could be further pursued in the future with the addition of more deep learning implementations and a narrower focus on the subject of deep learning as a whole. From the results acquired, deep learning has been shown to have improved results compared to other algorithms. Focused research on deep learning implementations would provide the reasoning behind deep learning accuracy.

**7. Conclusion**

In this research, we explored various machine learning algorithms, K-Nearest neighbors, Matrix Factorization, Deep learning, etc. Several approaches were implemented from the categories mentioned then tested for accuracy measures.

In both the cleaned and original datasets, the deep learning implementation was shown to the least margin of error when making recommendations. From this, it could be deduced that deep learning is a viable method for collaborative filtering engines working with user, item and rating data. Although different sets of data have varying optimal algorithms,

deep learning was still shown to be extremely accurate compared to other tested algorithms. Analyzing the data collected from the sparse dataset, we can conclude that a sparser set of data would result in less accurate recommendations, sometimes up to double the margin of error observed. Because of this observation, it can be concluded that collaborative filtering best performs with dense datasets.

**References**

[1] Kirzhner, Elena. “Machine Learning. Explanation of Collaborative

Filtering vs Content Based

Filtering.” Medium, Codeburst, 11

May 2018, codeburst.io/explanation of-recommender-systems-in

information-retrieval-13077e1d916c. [2] Harrison, Onel. “Machine Learning Basics with the K-Nearest Neighbors Algorithm.” Medium, Towards Data Science, 14 July 2019,

towardsdatascience.com/machine

learning-basics-with-the-k-nearest

neighbors-algorithm-6a6e71d01761.

[3] Chen, Denise. “Recommendation System - Matrix Factorization.”

*Medium*, Towards Data Science, 9

July 2020,

towardsdatascience.com/recommend ation-system-matrix-factorization

d61978660b4b.

[4] Sivanantham, Balavivek.

“Recommendation System Implementation With Deep Learning and PyTorch.” *Medium*, The Startup, 18 Aug. 2020,

medium.com/swlh/recommendation system-implementation-with-deep learning-and-pytorch-a03ee84a96f4.

[5] Karat, Sahar. “Co

Clustering.” *Data Science*

*Made Simpler*, 5 Mar. 2016, datasciencemadesimpler.wor dpress.com/tag/co

clustering/.

[6] Hug, Nicholas. “Welcome to Surprise' Documentation.” Welcome to Surprise'

Documentation! - Surprise 1 Documentation, 2015,

surprise.readthedocs.io/en/sta ble/.

[7] Mojdeh Saadati, Syed Shihab, Mohammed Shaiqur Rahman “Movie Recommender Systems: Implementation and Performance

Evaluation.” Semantic Scholar, 2019,

www.semanticscholar.org/paper/Mo vie-Recommender-Systems%3A Implementation-and-Saadati Shihab/01470f39285213e53f365ce0 1417b18d12467563#citing-papers.

[8] Xiangnan, He, et al. “Neural Collaborative Filtering.”

International World Wide Web Conference Committee, 3 Apr. 2017.

[9] Yedder, Hanene Ben, et al. “Modeling Prediction in

Recommender Systems Using Restricted Boltzmann Machine.”

*IEEE Explore*, IEEE, 5 Oct. 2017, ieeexplore.ieee.org/abstract/documen t/8122923.

[10] CooperUnion. “Anime

Recommendations Database.”

Kaggle, Kaggle, 21 Dec. 2016,

www.kaggle.com/CooperUnion/ani me-recommendations-database.

[11] Kampakis, Stylianos. “Performance Measures: RMSE and MAE.” *The Data Scientist*, The Data Scientist, 26 Nov. 2020,

thedatascientist.com/performance measures-rmse-mae/.