**by**

**Alexander Flynn and**

**Jarrod Jerowski**

***Movie Recommender for Dummies***

*A juxtaposition of modern collaborative-filtering algorithms for recommending movies*

ESE 527: Practicum in Data Analytics and Statistics | May 12, 2023

Contents

[***Movie Recommender for Dummies*** 0](file:///C:\Users\aflyn\Downloads\ESE527_FinalReport.docx#_Toc134660472)

[1. Executive Summary 2](#_Toc134660473)

[2. Data Description/Preprocessing 3](#_Toc134660474)

[A. Data Assets 3](#_Toc134660475)

[B. Metrics & Outlier Detection 3](#_Toc134660476)

[3. Modeling Approach 5](#_Toc134660477)

[a. Model Implementation 5](#_Toc134660478)

[B. Matrix Factorization 6](#_Toc134660479)

[c. K-Nearest Neighbors 7](#_Toc134660480)

[D. Cross-Validation 8](#_Toc134660481)

[E. Prescriptive Methods 9](#_Toc134660482)

[F. Model Morphisms 10](#_Toc134660483)

[11](#_Toc134660484)

[11](#_Toc134660485)

[12](#_Toc134660486)

[4. Results and Insights 14](#_Toc134660487)

[A. User Run Results 14](#_Toc134660488)

[B. BenchMark Validation 15](#_Toc134660489)

[5. Conclusions 17](#_Toc134660490)

[6. References 19](#_Toc134660491)

# 1. Executive Summary

Did you know that the world’s largest streaming service, Netflix, reports its users spedn on average 18 minutes per day trying to make a selection?(1) This is despite the fact that Netflix, along with all the other major players in the streaming industry, provide a state-of-the-art recommendation system. Therefore, this study is on the efficacy of the modern algorithms used in recommendation systems, particularly collaborative filtering algorithms. The movie recommendation system built and discussed in this report acts in the same manner and with the same goal as your traditional collaborative filtering-based recommendation system: to assist the user in selecting a movie they will enjoy by, hopefully, providing more personalized suggestions by analyzing the behavior and preferences of multiple users to identify patterns and similarities in their interactions with items.

This is achieved via two distinct classes of collaborative filtering, one of which is Matrix Factorization which decompses a user-movie interaction matrix into two distinct lower dimensional matrices, one composed of users and newly discovered features, and the other composed of movies and the same newly discovered features that are implicitly learned by means of stochastic gradient descent (SGD) based off the existing user-movie ratings. Once SGD is performed, these two matrices are multiplied to reconstruct a matrix with the form of the original matrix, but now all the previously empty values for the movies in which a user hasn’t seen, are filled with the estimates for those user-movie interactions, and then used to make recommendations to a specified user. The other method deployed and analyzed in this experiment is known as K-Nearest Neighbors. For the purposes of this study, we limit our scope to user-user filtering. In this case, the algorithms work by identifying a group of similar users for a given target user and then groups the “k” most similar, as measured by a variety of similarity scores, as the target user’s neighbors. Once the neighbors are found, a weighted average of the observed values in the neighbors’ vectors is created and given as an estimate for the unobserved values for the target user. The two methods, and the algorithms under each aforementioned method, are compared and contrasted in terms of accuracy, efficiency, relevancy, and a multitude of other metrics in order to gather insights on the current state of recommendation systems ubiquitous to the streaming world. It should be noted that although this study is limited to the case of users rating movies, these algorithms are domain agnostic, and can be utilized in any application where users are providing ratings or feedback on products (ex: amazon uses similar techniques when recommending new items to prospective customers).(2)

Ideally, movie recommendation systems bring enormous business value and satisfaction to the user by making the movie selection process more efficient and accurate. This should result in higher satisfaction from the users since it recommends a list of movies that they are likely to enjoy while also saving them time searching. Satisfied customers are then likely to spread positive feedback and market the tool to others, which will ideally lead to an increased reputation and trust of the recommendation system. The tool may also expose consumers to movies, genres, actors, directors, or platforms that they may not have found or thought they would have enjoyed without access to the tool. Finally, a more well-versed and open consumer base will ultimately impact revenue for that streaming platform, certain movies, and other movie related businesses.

# 2. Data Description/Preprocessing

## A. Data Assets

The following is the link to the kaggle open source dataset used in our project:

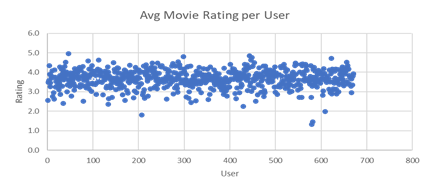
<https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset>

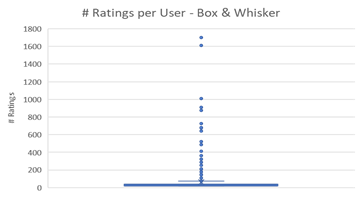
As you can see, the url contains a multitude of files, from which there is an original version and a “\_small” version of each, and in our project we utilized the “\_small” versions due to running this project on our personal PCs rather than having a specialized server or more capable devices in terms of computing power to run them on. Specifically, we used the ratings\_small.csv, links\_small.csv, and lastly the movies\_metadata.csv. The ratings files contains a list of over 100,000 ratings provided by 671 users over a span of 9,066 movies. The links\_small.csv file contains three columns of which are the unique “movieId”s found in our ratings\_small.csv file, and their corresponding “imdbId” and “tmdbId” in the proceeding columns. Our links\_small.csv file was, therefore, used to map the “movieId” column from ratings\_small to the “tmdbId” that’s also the “id” column in our movies\_metadata.csv file so that the title of the film could be fetched from movies\_metadata and provided to the user rather than providing a meaningless id number the user wouldn’t recognize. The movies\_metadata file consists of a list of 45,466 movies with features such as id, movie title, genre, a brief overview, language, release date, vote average, vote count, the cast, the crew, and keywords, etc. For our purposes, the only information that is read in is the ”id”, which as just mentioned is the tmdbId found in the links\_small.csv file, and the “title” of the film, which is used for relevancy to the user since the ids are, once again, meaningless to a user.

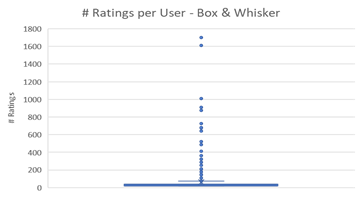
## B. Metrics & Outlier Detection

The following metrics were run to get a better understanding of the distribution of the rating scores, number of ratings per movie, and number ratings per user. The original intent of these metrics was to identify any outliers in the data and remove them from the dataset to reduce the chance of introducing unneeded bias into our models. See below for the metrics and plots that describe the dataset. Notice that the standard deviation of the ratings distribution is 1.1, indicating a moderately high variance amongst the user ratings. In regard to number of ratings per movie, we found that the average is 11, median is 3, upper quartile is 9, lower quartile value is 1, and standard deviation is 24. It is evident that the average is skewed much higher than the median, which can be attributed to a subset of movies that have a significant number of ratings while 25% of the movies only have 1 rating. We also see a similar trend with the number of ratings by user, where the average is 61, median is 20, lower quartile is 20, upper quartile is 40, and standard deviation is 136. Again, the average is skewed much higher than the median while 25% of users performed under 20 ratings and 75% of user performed under 40 ratings. With these metrics, we used the IQR equation to determine if there is a threshold in which we should delete some subset of user or movies with insufficient number of ratings. The result was a threshold less of less than 0 in both cases, so we determined it was not necessary to delete any additional data from the dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| **9066 Movies 671 Users** | **# Ratings per Movie** | **# Ratings by User** | **Rating Score** |
| **Average** | 11.0 | 60.9 | 3.5 |
| **Median** | 3.0 | 20.0 | 4.0 |
| **Standard Deviation** | 24.0 | 136.3 | 1.1 |
| **Lower Quartile** | 1.0 | 20.0 | 3.0 |
| **Upper Quartile** | 9.0 | 40.3 | 4.0 |
| **IQR** | 8.0 | 20.3 | 1.0 |
| **Lower outlier threshold** | -11.0 | -10.4 | NA |

Table 2B-1 Ratings Distribution





Due to not utilizing statistics in the determination of outliers, the outliers came in the form of broken data. Specifically, all the movies whose titles weren’t able to fetched were filtered out from our ratings\_small.csv file. This happened as a result of either the links\_small.csv file not having a corresponding tmdbId (some tmdbIds were flat out just missing from the file), or the tmdbId tied to our filtered movies didn’t have a matching id in the movies\_metadata.csv file, and therefore, a title wasn’t found for them. The end result of this validity checking was 42 films were filtered out of the ratings data.

# 3. Modeling Approach

## a. Model Implementation

In order to provide some background, the way the movie recommendation system operates is once navigating to the webpage, the user is prompted with the choice of an algorithm to be run and provide recommendations. Once that is selected, the user is rerouted to a page asking whether they want recommendations for a user already in the database, or for themselves, and in the latter’s case, are prompted with movies for them to rate until they have reached the arbitrarily selected threshold of 20 movies. At this point, after selection of an existing user or themselves, the algorithms are run, and predictions follow.

As mentioned in the introductory section of the report, the models deployed in the recommendations system can be categorized into two discrete classes: Matrix Factorization and K-Nearest Neighbors. Each are forms of collaborative filtering as they interpret and discover patterns in the user-movie interactions and then makes predictions based off the preferences learned or given by similar users. All of the algorithms were provided by incorporating the scikit surprise library. The following two subsections go into further detail for each algorithm category used throughout the study. To keep the experiment consistent, and due to the nature of the application itself, the input into each algorithm was kept constant and was the result of transforming the ratings\_small.csv, which once again contained user’s ratings for a variety of movies, into a user-movie interaction matrix where the former are the rows and the latter the columns and where the values at row “u”, column “m” is the rating given by user “u” for the movie “m”. This allowed for easier descriptive analysis of the data, as was shown through the descriptive statistics in section 2, where it was explained that the only filtering performed was due to invalid tmdbIds disallowing the ability to fetch movie titles. Furthermore, and as one can expect, this matrix is incredibly sparse, in fact, this original matrix prior to learning was seen to be 98.4% sparse. This amplifies the interesting case study between the two classes of algorithms, as typically, data sets with the descriptive statistic of heavy sparisty are better suited as input into a matrix factorization algorithm rather than nearest-neighbors, which suffers from the cold-start problem, or the problem of not containing enough information for a target user, a direct result of heavy sparsity.

## B. Matrix Factorization

In matrix factorization, the aformentioed user-movie interaction matrix gets decomposed into two lower-dimensional matrices that represent users and movies in a latent feature space. The goal of matrix factorization is to identify underlying latent features that are not explicitly represented in the data but underlie user preferences and represent actual concrete attributes of films. As an intuitive example, in the case of a one-dimensional latent feature space, the sole feature resembles the popularity of the movies, and therefore in the case of the user-feature matrix for the purpose of this example, the user’s inclination for popular movies. The decomposition works because the lower-dimensional space captures the most important information in the data, while reducing the dimensionality of the problem making the model more tractable and less prone to overfitting. These two lower dimensional embedded matrices, one representing the users in a latent feature space, and the other the movies in the same latent feature space, are learned through optimization techniques, either stochastic gradient descent (SGD), or alternating least squares (ALS). In the scope of this study, 4 different matrix factorization algorithms are options for the user to select: SVD, SVDpp, and a baseline equation of

, where the variables are global mean, user u’s bias, and movie I’s bias respectively, of which the latter two were learned via sgd or als. As seen with traditional SVD, the user-movie interaction matrix, or let’s call it R, can be decomposed into the user and movies matrices, or U and M respectively and the resulting equation is . Unlike traditional SVD, the SVD performed in this experiement, through the implementation of the surprise function, has the predictive equation of

where is the predicted user u’s rating for film I, is the feature vector for movie i, and lastly, is the feature vector for user u. It clearly mirrors traditional SVD, but includes the mean, and biases adjusting for variations in user and movie ratings, and allowing for greater model flexibility and, therefore, accuracy. It learns the biases and the embedded matrices by using SGD optimization to minimize the difference between the actual ratings and the predicted rating equation as seen above or in other words:

The specific parameter updates for this algorithm are seen below in the morphism table in the column “Learning Function” labeled “SVD Parameter updates” in the corresponding SVD row. The prediction equation can also be found in the learning function entry in both morphism #1 and in the morphism table in row SVD. Additionally, it can be seen that the SVDpp follows very similarly to the first SVD algorithm, with the additional caveat of implicit ratings, or an emphasis on the fact that a user rated a movie irrespective of what the actual rating was. The morphisms will be expanded upon in a future section. Lastly, the baseline algorithms follow the same predictive procedure as seen with SVD, minus the user and movie embedded matrices (a loss of a lot of meaningful data and were really only captured as well a “baseline performance” measure), and the two algorithms run on the baseline equation only differ in that one updated the biases with sgd while the other through als. The predictive methods were analyzed by tracking the root mean square error and the fraction of concordant pairs metric, which measures the proportion of pairs of items that are predicted to have the correct order of ratings by the model of the actual vs predicted. A pair of items is said to be concordant if the predicted rating order matches the actual rating order, and discordant otherwise. The FCP score is then calculated as the fraction of concordant pairs out of the total number of pairs.

## C. K-Nearest Neighbors

In addition to the matrix factorization models, the K-Nearest Neighbors predictive algorithms from the same surprise library were deployed and made available to the user. Nearest neighbors collaborative filtering is a technique that relies on finding the most similar users, in the case of this study, to the target user and takes a weighted average of the observed interactions for those nearest neighbors and applys them to the unobserved vaules for the target user. A descriptive method for Nearest Neighbors algorithms is the construction of the neighborhoods of the original data by means of a similary measure such as pearson correlation coefficient or cosine similarity, a choice that is made usually through the predictive means of cross-validation (more on that in section D). The nearest neighbors methods utilized in this project were the traditional nearest neighbors, or KNNBasic as labeled in the surprise library, the KNNWithMeans, and the KNNWithZScore algorithms. The KNNBasic calculation directly computes its predictions by taking the most similar “k” users, another hypertuned parameter, and taking the average of those k nearest neighbors ratings and applying them directly. I.e.

where sim(u,v) is, as applied to this project once again, either the pearson or cosine similarity measurement, both of which the formulas can be found in the morphism table. The KNNWithMeans and KNNWithZScore are extentions of this where the means variety takes the target user’s mean and then averages the difference of every neighbor’s rating from their mean, and then adds it to the target mean:

while the z-score normalized algorithm similarly takes the average z-score of the neighbors and multiplies it by the target users standard deviation and then adds that product to the target user’s mean to essentially take the predicted z-score of the target user for a specific movie and then apply that to the target mean:

These methods are much simpler intuitively than the matrix factorization algorithms which discover latent features and learn through optimization techniques as these nearest neighbors methods solely perform computations in order to generate predictions. Once again, the FCP and RMSE are tracked for these algorithms and are displayed in the metrics page of the application, which is provided in section 4: Insights/Results of this report. As expected, the more normalization performed on each algorithm, and provided by the descriptive statistics mean and/or standard deviation of the original data, the higher the accuracy of the model was in general. This is due to the normalizaiton reducing the variability in the different user’s tendency, as some user’s might have proclivity to rating movies towards one of the extremes causing a variability in the different user’s scales. Thus, normalization reduces outliers and bias inherent in the data.

## D. Cross-Validation

In order to optimize each individual algorithm individually, grid search cross-validation was performed prior to any algorithm being deployed. This was done through use of the GridSearchCV function in the surprise library. Cross-validation is a technique used to evaluate the performance of a machine learning model and also to perform hyperparameter tuning. The goal of cross-validation is to estimate how well the model is likely to perform on unseen data by splitting the available data into two or more sets: a training set, which is used to train the model, and a test set, which is used to evaluate its performance. The model is trained on the training set and then tested on test and measured by means of RMSE or other accuracy measurements (RMSE was used as the indicator for determining performance in this project.) By using gridsearch cross-validation, a grid of hyperparameter combinations is created, usually specific to the algorithm being run (except in the case of nearest neighbors methods all utilizing the same grid of combinations, and the matrix factorization algorithms sharing some overlap but inevitably having some unique hyperparameters), and then fed into the GridSearchCV function which then performs k-fold cross-validation on every parameter combination in the grid. K-fold cross validation is a method which involves randomly dividing the data into k subsets of roughly equal size where the model is trained on k-1 folds and tested on the remaining fold, and this process is repeated k times, with each fold serving as the test set once. The k value chosen was 3, purely to save time while providing an adequate amount of runs to properly test the different hyperparameter combinations thoroughly. The results from the k runs is then averaged together and given as the score for that hyperparameter combination’s cross-validation score. After running k-folds cross validation on every hyperparameter combination, the combination that resulted in the highest average score across the folds is retained as the paramter combination to be used as the hyperparameters for the prediciton algorithm in question. The parameter grids used in GridSearchCV for each algorithm can be found in the Parameters column of the morphism table.

Cross-validation can help to avoid overfitting, which occurs when a model performs well on the training data but poorly on the test data, by providing a more accurate estimate of the model's generalization performance, due to it being fit and tested on “k” different sets of training data. This is incorporated as part of the prediction method as it essentially is the model selection technique for the algorithms and aids the overall workflow by producing the models that theoretically should generalize the best to new data, thereby optimizing the chances that a specified algorithm will predict unseen movies with the highest accuracy.

## E. Prescriptive Methods

Prescriptive analytics are used to recommended actions and/or strategies based off the findings from predictive models. In this study, the most straightforward implementation of a prescriptive analytic is through the the presentation of the top 10 highest predictions for a given user, an action taken based off the model’s predictions. Other ways, which were not implemented in the scope of this project, in which collaborative filtering based systems could incorporate prescriptive analytics would be through contextual awareness information, such as time of day and/or user’s mood at time of rating, user-feedback upon receiving recommendations, and particularly relevant to the workflow of this project, which algorithm is selected with the highest frequency.

## F. Model Morphisms

A machine learning workflow consists of the steps performed building and deploying a machine learning model. In our project, this involves the data collection, data preprocessing, data transformation, hyperparameter tuning, model training, and model evaluation. In our case, the models being evaluated are SVD, SVDpp, BaselineOnly, KNNBasic, KNNWithZScore, and KNNWithMeans. For all of these algorithms, we denote the input data as part of an input space called “X” and the outcomes as part of “Y”. For the purpose of our movie recommender models, the input space would be our user-movie rating matrix, where each row represents a user, each column represents a movie, and each cell contains a rating for that user-movie pair. While the output space would be the predicted rating for each user-movie pair in the input space, which is used then used to build the personalized list of recommended movies for a specific user profile. When implementing our models, we assume that there is some relationship between our inputs contained in “X” and the output “Y” that we want to learn. To do this we want to estimate the conditional mean of “Y” given “X”, so in our case that would be the predicted list of movies given the movie, user, rating matrix data set. Yet before we get into this learning process, each of these models goes through the same preprocessing workflow where we perform some basic filtering and manipulation described in the preprocessing section. Additionally, some machine learning models may perform data transformation techniques like feature engineering. With regard to our models, the only transformations occur in the SVD decomposition which is part of the learning morphism. After preprocessing, our algorithms go through hyperparameter tuning via grid search 3-fold cross-validation to determine the best set of parameters specific to each modeling through the use of each algorithm’s learning and loss morphisms. If we take SVD and BaselineOnly via stochastic gradient descent as an example, the machine learning morphism would be the gradient descent learning process where we update the weight parameters by moving in the opposite direction of the gradient with a regularizer and learning rate parameter. Within that learning morphism is the regularized squared error loss function in which the learning (gradient updates) works by minimizing this error until it reaches our predetermined threshold. In some cases a machine learning model may also include parameter priors, where this prior information known about the data set is used to explicitly set the parameters to assist in regularizing the model. For example, in Bayesian supervised learning we model parameters with known prior information and make updates and predictions on new inputs based on this probability distribution. In our case, we are solving a semi-supervised learning problem because we have a labeled dataset where we have a sparse matrix of user-movie ratings and are attempting to predict on an unknown set. In that sense, we could attempt to estimate a prior distribution for our model based on the normalized value of the individual movie average rating divided by the overall average rating. Yet, unlike Bayesian learning, we are not explicitly modeling this probability into our model parameters, so we would say we have an improper probability since aren’t explicitly using prior distributions to build our models. Finally, the last step of the workflow is to fit these hypertuned models on our training set and use this to predict on an unknown test set where we measure performance based on RMSE and FCP metrics. See below for more details regarding the machine learning workflow and morphisms used in our project.

### 

### 

Model Training

Hyperparameter tuning

Data Pre-Processing

Data Collection

Data Transformation

Model Evaluation:

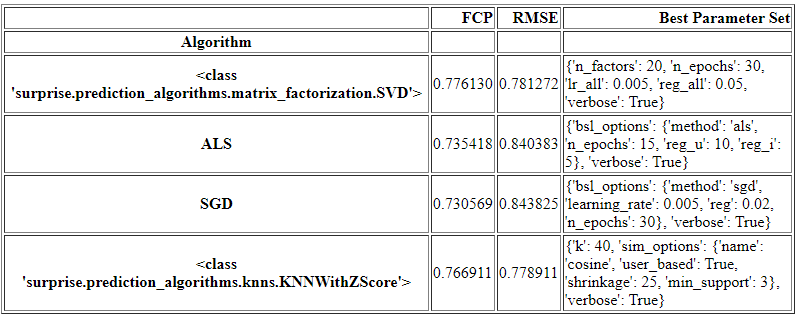
### 

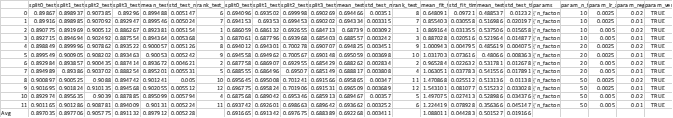
|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Input Space X | Output Space | Parameter Prior |
| SVD | User-movie rating vector of, where element j in the vector represents the specific user’s rating for movie j. | Predicted rating for each user-item pair in the input space. Each algorithm gets to this end result via different processes. | No prior assumptions were made on the data so apriori probability is set to 1 in each morphism. |
| SVD++ |
| BaselineOnly |
| KNN |
| KNNWIthMeans |
| KNNWithZScore |
| KNNBasic |

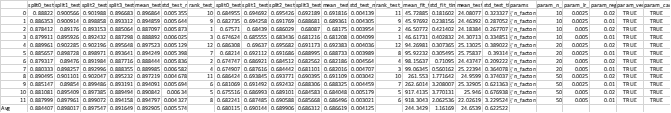
|  |  |  |  |
| --- | --- | --- | --- |
| **Models** | **Learning Functions:** | **Loss Functions:** | **Parameters** |
| **SVD** | Traditional SVD Decomposition which the model is based off of:    SGD is used for the following parameter updates until convergence for both SVD and SVDpp:  SVD Parameter Updates  Actual SVD Learning Function:  SVDpp:  Additional Feature Vector (implicit feedback) | Regularized Squared Error Loss:  ) | {'n\_factors': [10,20,50],'lr\_all':[0.0025,0.005],'reg\_all': [0.02,0.01]} |
| **SVD++** | {'n\_factors': [10,20,50], 'lr\_all':[0.0025,0.005],'reg\_all': [0.02,0.01} |
| **BaselineOnly** | SGD and ALS methods to optimize the following: | Regularized Squared Error Loss: | ALS:{'bsl\_options':{'method': ['als'],'reg\_i':[10,15],'reg\_u':[15,20],'n\_epochs':[10,20]}}  SGD:{'bsl\_options':{'method': ['sgd'],'reg':[.02,.05],'learning\_rate':[.005,.01,.02],'n\_epochs':[15,20]}} |
| **KNN** | KNN use similarity scores which aren’t learning/loss functions in the traditional sense. | | |
| **KNNWithMeans** | (user-user) | | {'k': [20, 40], 'sim\_options': {'name': ['pearson\_baseline', 'cosine'],'shrinkage':[100,75,50],'min\_support': [3],'user\_based': [True]} |
| **KNNWithZScore** | (user-user) | |
| **KNNBasic** | (user-user) | |
| **Model Evaluation** | RMSE & FCP Metrics: | | Evaluation of parameters for each model on test set. |
| **Variables** | = similarity between users “u” & “v” | | |

# 4. Results and Insights

## A. User Run Results









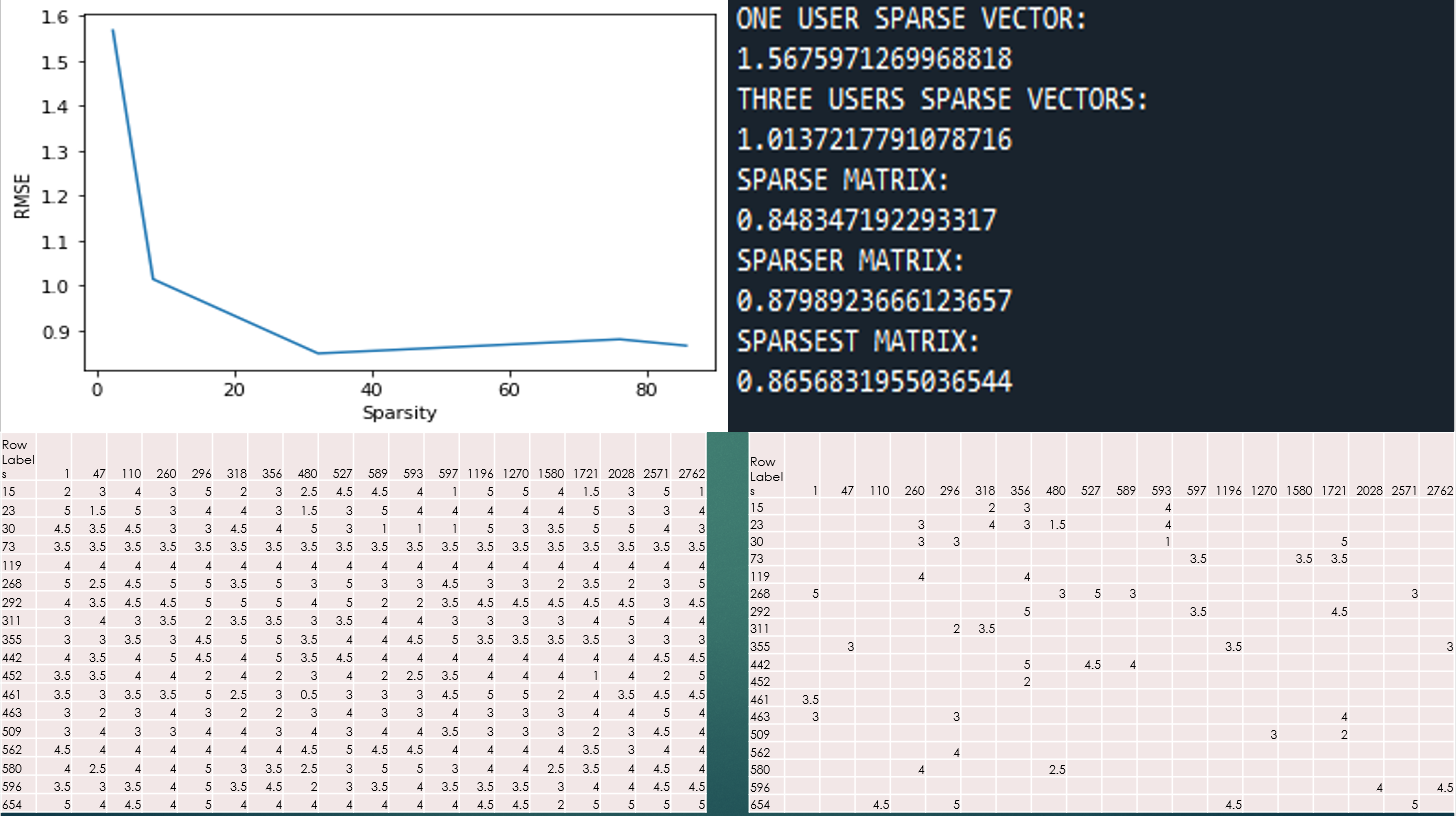


## B. BenchMark Validation

As an additional performance measure for our project, we built the benchmark matrix shown below. The intention of the benchmark matrix is to provide a subset of data that represents the overall dataset that gives a more objective measure to compare performance to the overall matrix. To construct our benchmark matrix, we began manipulating the overall user-movie matrix to sort the movies with the most ratings across the columns and the users with the most ratings down the rows. From there, we began to filter out the user movie interactions that had null values as it is ideal to have a full matrix for our benchmark. Filtering left to right, we stopped when we got to a nearly square matrix containing 18 users and 19 movies. Next, we transposed this matrix, sorted each column in descending order, and gave a 1-19 movie ranking by user where 19 denotes the highest rating. This was transposed back into its original form where the result is a full 18x19 user-movie matrix with a custom ranking system related to the 19 movie ratings by each user. From here, we will the benchmark matrix to train our algorithms and then introduce the other 653 users where the benchmark trained algorithm will fill out the rest of the primary matrix. Then to validate our benchmark we ran a sparsity analysis by running predictions after training on our benchmark matrix when one user has some deleted ratings, three users have some deleted ratings, and almost all users have at least a couple ratings deleted. To prepare each of our benchmark matrices for validation we went through the same process as described in our modeling approach. This is where we run grid search cross validation to find optimal parameters, fit to the entire benchmark matrix, build training/test/antitest sets, and make predictions on the test and antitest set where the antitest ratings are all unknown . This ensure that each model learns hyerparameters separately for that specific set of data.

Once we complete this final step, we hope to find that the algorithms fit well on both the benchmark and primary matrices. This would conclude that the algorithms do a good job of generalizing to the data and are making accurate predictions for unknown user-movie interactions.

Benchmark Matrix Experiment:



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **UserID/MovieID** | **356** | **296** | **318** | **593** | **260** | **480** | **2571** | **1** | **527** | **589** | **1196** | **110** | **1270** | **47** | **2762** | **2028** | **1580** | **1721** | **597** |
| **15** | 2 | 19 | 5 | 19 | 19 | 9 | 19 | 5 | 12 | 12 | 19 | 9 | 19 | 19 | 2 | 9 | 12 | 3 | 6 |
| **73** | 19 | 19 | 19 | 11 | 11 | 7 | 11 | 19 | 19 | 4 | 19 | 7 | 19 | 19 | 7 | 11 | 4 | 1 | 4 |
| **452** | 14 | 19 | 19 | 19 | 14 | 19 | 2 | 5 | 14 | 14 | 14 | 14 | 14 | 3 | 19 | 14 | 14 | 1 | 4 |
| **580** | 6 | 19 | 15 | 15 | 15 | 4 | 19 | 15 | 15 | 19 | 15 | 19 | 4 | 15 | 15 | 15 | 1 | 6 | 4 |
| **509** | 13 | 19 | 13 | 13 | 19 | 8 | 14 | 8 | 19 | 4 | 19 | 19 | 8 | 13 | 13 | 8 | 4 | 4 | 4 |
| **311** | 19 | 7 | 15 | 2 | 11 | 15 | 11 | 7 | 19 | 15 | 7 | 7 | 15 | 1 | 11 | 19 | 7 | 11 | 19 |
| **30** | 19 | 19 | 19 | 11 | 11 | 11 | 1 | 11 | 19 | 11 | 11 | 19 | 19 | 11 | 19 | 19 | 11 | 11 | 11 |
| **461** | 7 | 12 | 19 | 19 | 12 | 19 | 12 | 3 | 7 | 19 | 19 | 7 | 19 | 19 | 12 | 3 | 1 | 7 | 12 |
| **654** | 2 | 19 | 19 | 8 | 19 | 8 | 19 | 19 | 19 | 19 | 19 | 8 | 8 | 8 | 19 | 19 | 8 | 19 | 2 |
| **23** | 17 | 17 | 19 | 17 | 17 | 7 | 10 | 2 | 7 | 7 | 17 | 7 | 17 | 17 | 10 | 10 | 7 | 19 | 2 |
| **119** | 3 | 19 | 13 | 13 | 19 | 13 | 13 | 1 | 19 | 13 | 13 | 19 | 13 | 19 | 13 | 19 | 13 | 3 | 13 |
| **596** | 19 | 19 | 19 | 13 | 2 | 13 | 16 | 6 | 16 | 13 | 6 | 13 | 6 | 13 | 16 | 13 | 2 | 13 | 6 |
| **463** | 4 | 4 | 19 | 19 | 19 | 12 | 19 | 4 | 19 | 12 | 19 | 19 | 12 | 12 | 12 | 12 | 12 | 12 | 1 |
| **268** | 19 | 19 | 4 | 19 | 19 | 9 | 9 | 19 | 19 | 19 | 19 | 9 | 19 | 6 | 4 | 19 | 6 | 2 | 2 |
| **355** | 19 | 13 | 19 | 13 | 19 | 4 | 19 | 3 | 9 | 9 | 19 | 9 | 9 | 19 | 13 | 13 | 9 | 1 | 2 |
| **292** | 19 | 19 | 19 | 15 | 15 | 3 | 8 | 8 | 19 | 8 | 15 | 8 | 8 | 15 | 15 | 3 | 3 | 15 | 15 |
| **562** | 7 | 19 | 19 | 19 | 19 | 19 | 19 | 10 | 7 | 19 | 19 | 7 | 2 | 10 | 1 | 19 | 7 | 7 | 10 |
| **442** | 19 | 15 | 15 | 19 | 19 | 15 | 15 | 4 | 15 | 4 | 15 | 15 | 4 | 15 | 15 | 19 | 15 | 4 | 15 |

# 5. Conclusions

After hypertuning and testing our model prediction performance using RMSE and FCP, we came up with the results displayed in the tables above. To understand these scores as it relates to each model’s performance, it is important to understand the differences in how these metrics are computed and the specifics of each algorithm’s workflow. RMSE simply measures the average difference between the predicted ratings on the test set and the actual ratings from the users, which gives us an idea of how well the model can predict ratings for movies that the user has not seen yet. This means a lower score indicates that the model predicted ratings are closer to that of the actual ratings In comparison, the FCP metric measures the proportion of user-movie pairs within the test set in which the model was able to make an accurate prediction, which gives us an understanding of how well the model can predict for all of these user-movie pairs as opposed to just the ones in which the model was able to make a prediction. This means that a higher FCP score tells us that the model has good understanding of the user-movie interactions in the test set, so we can be confident that the model has the ability to make recommendations for a large set of user-movie combinations. In general, both are great measures of performance, but they differ somewhat in the fact that RMSE is more focused on measuring prediction accuracy and FCP measuring model overall coverage based on prediction accuracy.

First off, we concluded that our SVD++ algorithm performed the best on both RMSE and FCP performance metrics, which can be attributed to the latent feature discovery and the implicit feedback involved in SVD++. The dimensionality reduction factorizes our user-movie matrix to discover latent features that help describe user-movie interactions better while minimizing the impact of other less relevant features. While the implicit feedback enables the model to handle missing data like we have in our data set, by making implicit inferences between existing user ratings and similarities between movies. The result is a model that considers a larger range of user preferences when making user predictions. Yet, while SVD++ maybe be the superior method in terms of accuracy scores, it is also much longer in terms of time complexity. Therefore, since the objective of our problem is to produce an accurate and efficient system for movie recommendations, we believe the tradeoff with much faster but still highly accurate models would be best for our purposes. If we exclude SVD++ due to the extensive time complexity, we see that SVD and KNNWithZscore are the top two performers, where SVD was the best performer on the FCP metric and KNNWithZscore on the RMSE metric. When thinking about both methods, we understand that SVD looks to accurately predict ratings and KNNWithZscore looks to produce ranking of movies for users based on similarity. Now, one might ask why SVD performed slightly better on FCP and KNNWithZscore performed slightly better on RMSE. That being said, the performance of both models were very similar and performance can obviously vary based on the given data set being evaluated. Therefore, both models are viable options to be used as our best prediction model for the sake of our project.

Now, when trying to understand why these two method performed better compared to the others, we dug into each of the algorithm’s specific machine learning workflows and morphism’s as it related to their performance. First, we can attribute the success of SVD due to the similar advantages it shares with SVD++, where SVD also has the ability to handle sparse data sets and identify underlying user-movie preferences through the use of matrix decomposition and latent feature discovery (minus the implicit feedback and bias terms). On the other hand, our other top performer KNNWithZscore, works by standardizing the user’s mean ratings and dividing by the standard deviation, which makes it more accurate to compared similarity between users by providing a more robust measure of similarity between users since it takes into account both the mean and variance of the ratings. In comparison, KNNWithMeans centers the user ratings on the mean and KNNBasic simply calculates similarity based on the raw user ratings. Therefore, KNNWithMeans may find similar users that differ significantly in their rating distribution since it doesn’t take into account the variance of ratings, and KNNBasic may result in less accurate similarities based on user biases since it is only using raw ratings. Hence, the reason why we find KNNWithZscore as the superior method over KNNBasic and KNNWithMeans.

Next, when evaluating the other matrix factorization method BaselineOnly via stochastic gradient descent (SGD) and alternating least squares (ALS), we identified some potential limitations which may have attributed to their lower performance compared to SVD and KNNWithZscore. After studying their machine learning workflows, we found several areas of interest as it relates to the sparsity of the data, lack of feature engineering, regularization, and cold start issues. In comparison SVD, SGD and ALS include bias terms that implicitly describe the features of the data, but they do not perform any explicit latent feature discovery that enables SVD to identify valuable patterns in the data. Therefore, they don’t have the same mechanisms to deal with sparse data or lack of information about new users, so their effectiveness can dwindle for portions of the data that have little to no user-movie interactions. Lastly, there is potential that the regularization parameters ran through cross-validation could be improved, since insufficient regularization parameters can lead to over or under fitting on the data set.

Now that we have dug into performance as it related to each of the algorithm’s machine learning workflow, we have a few final remarks for potential improvements going forward. As it relates to our current implementations, we are confident in our SVD and KNNWithZscore models based on their RMSE and FCP scores that performed very well by normal standards. Although, given that SVD latent feature discovery performed so well, this gives us reason to believe it may be beneficial to evaluate some additional feature engineering through methods like rating normalization, incorporation of metadata, and clustering. Not only could this enhance our high performing SVD and KNNWithZscore models, but it also may improve the lower performing models due to their limitations on sparse and unknown data. Additionally, this system may also benefit from hybrid learning by combining other methods like content-based filtering and deep learning. This would enable us to explicitly incorporate the metadata features and utilize methodologies like bag-of-words, feature embeddings, and neural network attention mechanisms to better tailor our predictions to the user’s preferences. Aside from that, we can confidently say that our models performed well on the given data set, specifically SVD and KNNWithZscore which performed very well by any standard measure of these metrics. Finally, we would like to point out that as our data set is updated, the user-movie interactions and overall sparsity can significantly change as well. This means the performance of each model may differ based on the advantages and limitation contained in their machine learning workflows. Therefore, given the data set at hand, we would deploy SVD and KNNWithZscore models as our current standard for making timely accurate predictions.

# 6. References

1) Maglio, Tony. “Netflix Users Spend 18 Minutes Picking Something to Watch, Study Finds.” *TheWrap*, 21 July 2016, [www.thewrap.com/netflix-users-browse-for-programming-twice-as-long-as-cable-viewers-study-says/](http://www.thewrap.com/netflix-users-browse-for-programming-twice-as-long-as-cable-viewers-study-says/).

2) Hardesty, Larry. “The History of Amazon’s Recommendation Algorithm.” *Amazon Science*, 1 Dec. 2022, [www.amazon.science/the-history-of-amazons-recommendation-algorithm](http://www.amazon.science/the-history-of-amazons-recommendation-algorithm).

Flynn, Alex, and Jarrod Jerowski. “AFLYNN0213/MovieRecommenderForDummies: ESE 527 Project.” *GitHub*, <https://github.com/aflynn0213/MovieRecommenderForDummies>.

Hug, Nicolas. *Surprise*, Sept. 2019, [https://surpriselib.com/.](https://surpriselib.com/.%20)

Cawi, Eric, et al. “Designing Machine Learning Workflows with an Application to Topological ...” *Designing Machine Learning Workflows with an Application to Topological Data Analysis*, DGE-1745038, National Science Foundation Graduate Researc, 2 Dec. 2019, <https://www.ese.wustl.edu/~nehorai/paper/Cawi_Design_ML_TDA_PLOS_ONE_2019.pdf>.

Koren, Yehuda, and Joseph Sill. “Collaborative Filtering on Ordinal User Feedback - IJCAI.” *Collaborative Filtering on Ordinal User Feedback∗*, <https://www.ijcai.org/Proceedings/13/Papers/449.pdf>.