Movie Recommender for Dummies

A juxtaposition of modern collaborative-filtering algorithms for recommending movies

by

Alexander Flynn and

Jarrod Jerowski

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

Contents

1. Executive Summary

Did you know that the world's largest streaming service, Netflix, reports its users spend 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 through the analysis of multiple users' behaviors and preferences 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 decomposes 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, the scope is limited to user-user KNN 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

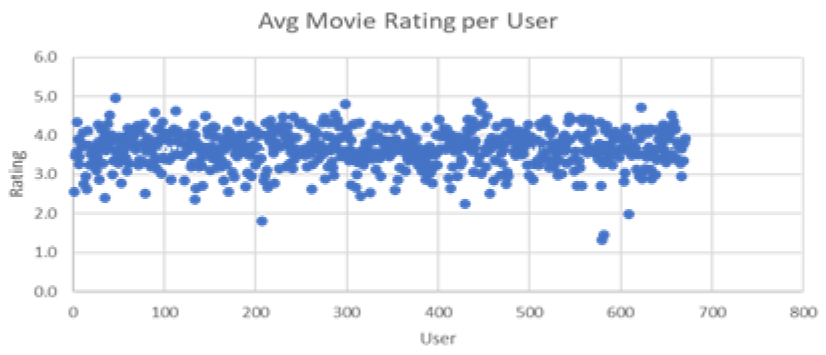
As you can see, the URL contains a multitude of files, from which there is an original version and a "_small" version of each, and in our project we utilized the "_small" versions due to running this project on our personal PCs rather than having a specialized server to run them on. Specifically, we used the ratings_small.csv, links_small.csv, and 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.

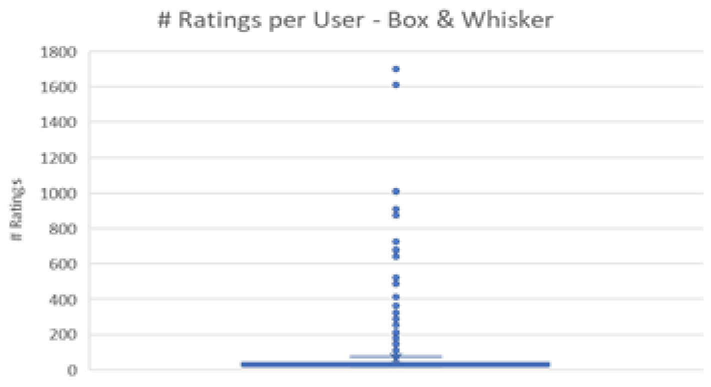
B. METRICS & OUTLIER DETECTION

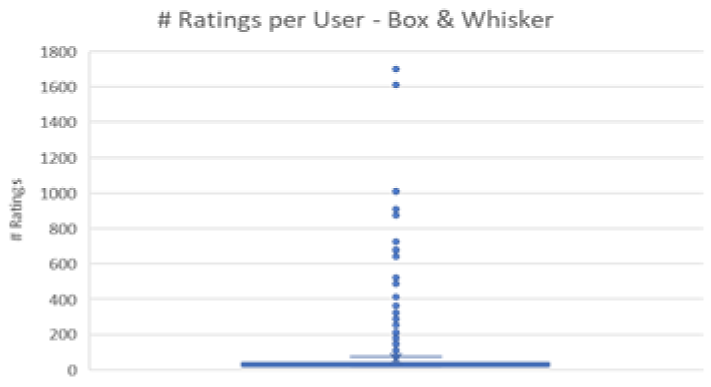
The following metrics were run to get a better understanding of the distribution of the rating scores, number of ratings per movie, and number ratings per user. The original intent of these metrics was to identify any outliers in the data and remove them from the dataset to reduce the chance of introducing 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. Regarding the 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 descriptive statistics in the determination of outliers, the movies that were filtered instead were as a result of broken, incomplete 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 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 the algorithms were provided by incorporating the scikit surprise library, and 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 i.e. a value 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 sparsity 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. Lastly, the same workflow (expanded upon in subsection F Morphisms) is used regardless of the chosen algorithm. That is, the data for which the parameters are hyper tuned on in the cross-validation stage is the entirety of the observed ratings, and then once parameters have been hyper tuned for the chosen algorithm, the model is trained on a training subset of the observed data, where the training data is built using a build_fulltrainset() function available in the surprise library. In the predictive method, the now trained model is then tested on all of the unobserved user-movie interactions using the .test() function on a test set built with the surprise function build_antitestset(), allowing the predicted set to only contain movies the target user has not seen, exactly the intended goal of the workflow. It is ensured that it is only the set of unobserved interactions since the build_antitestset() method literally creates a test set only containing unobserved user-movie interactions.

B. MATRIX FACTORIZATION

In matrix factorization, the aforementioned 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 resemble attributes of the 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

$\mu + b_u + b_i$, where the variables are global mean, user u's bias, and movie I's bias respectively, of which the latter two variables 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 $R = UM^T$. Unlike traditional SVD, the SVD performed in this experiment, through the implementation of the surprise function, has the predictive equation of

$$\tilde{r_{ui}} = \mu + b_u + b_i + q_i^T p_u$$

where $\tilde{r_{ui}}$ is the predicted user "u"'s rating for film "I", q_i is the feature vector for movie i, and lastly, p_u 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 vectors by using SGD optimization to minimize the difference between the actual ratings and the predicted rating equation as seen above or in other words:

$$\sum_{r_{ui} \in R_{train}} \left((r_{ui} - (\mu + b_u + b_i + q_i^T p_u))^2 + \lambda (b_u^2 + b_i^2 + \left| |q_i| \right|^2 + \left| |p_u| \right|^2) \right)$$

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 only studied for 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 applies them to the unobserved values 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 similarity 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 hyper tuned parameter, and taking the average of those k nearest neighbors ratings and applying them directly. I.e.

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} sim(u,v) * r_{vi}}{\sum_{v \in N_i^k(u)} sim(u,v)}$$

where sim(u,v) is, as applied to this project once again, either the Pearson or cosine similarity measurement of user u to user v, both of which the formulas can be found in the morphism table. The KNNWithMeans and KNNWithZScore are extensions 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:

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

Meanwhile, the z-score normalized algorithm similarly takes the average z-score of the neighbors and multiplies it by the target user's 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:

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

These methods are much simpler intuitively than the matrix factorization algorithms which discover latent features and learn through optimization techniques, whereas these nearest neighbors methods solely perform computations in order to generate predictions. Once again, the FCP and RMSE are tracked for these algorithms and more information is 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 normalization 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 each case of algorithm selection, 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 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 the test and measured by means of RMSE or other accuracy measurements (RMSE was used as the indicator for determining cross-validation performance in this project.) By using grid search 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 for a given hyper parameter combination 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 parameter combination to be used as the hyperparameters for the prediction algorithm selected. 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 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 lastly but 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 progression of steps performed before, during, and after the building and deployment of a machine learning model. This includes, but not limited to, data collection, data preprocessing, data transformation, hyperparameter tuning, model training, and model evaluation. In the workflow of this project, the models being evaluated are SVD, SVDpp, BaselineOnly(ALS), BaselineOnly(SGD), KNNBasic, KNNWithMeans, and KNNWithZScore. As shown in the section below titled "Machine Learning Morphism" each algorithm has a unique morphism construction, except for the two baseline algorithms, as they are operating on the same learning function and minimizing the same empirical risk function as one another, differing only in the optimization techniques to do so. Notice further that the morphisms listed in this section are of the form of $\mathbb{R}^f, \mathbb{R}$ for the input and output respectively, where the former, and the first argument in the morphism, is a target user's vector where f is either the number of latent features in the SVD cases or the number of movies in the other cases. The output, which is a scalar in the morphisms, is the individual rating for a target user and a specific movie. As shown in the learning function, or the third parameter of the morphism function, the equation is the equation for a specific predicted rating, $r_{\tilde{ui}}$, and thus necessitates the need for the output space parameter to be one-dimensional, or a scalar. If you look even further ahead to the morphism workflow section, the form that this assumes the input is the entire user-movie interaction matrix dataset, and the output also assumes the entire dataset in the case of the matrix factorization algorithms, while the nearest neighbors can produce an estimate of a specific target user. In each case, the input should be R^{uxm}, while in the case of matrix factorization the output would match the input space as it is the same dimension of the original matrix, but now has the unobserved films filled with predictions, and with the nearest neighbors the output space would be $\mathbb{R}^f$, or the user vector for the target user. Once the process progresses through the workflow to the point at which output Y is provided, a prescriptive method (not shown below) is used to filter out the predictions for the target user so that only the ten highest predicted values will be displayed as recommendations.

$$Machine\ Learning\ Morphisms: (X, Y, F(x, \Theta), P_{\Theta}(\Theta), L(y, F))$$

1) $MLM_{SVD} = (\mathbb{R}^f, \mathbb{R}, \mu + b_u + b_i + q_i^T p_u\ , P(\theta) = 1,\ \sum_{r_{ui} \in R_{train}} ((r_{ui} - (\mu + b_u + b_i + q_i^T p_u))^2 + \lambda(b_u^2 + b_i^2 + ||q_i||^2 + ||p_u||^2))$

2) $MLM_{SVDpp} = (\mathbb{R}^f, \mathbb{R}, \mu + b_u + b_i + q_i^T (p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j\ , P(\theta) = 1, \sum_{r_{ui} \in R_{train}} (\left(r_{ui} - (\mu + b_u + b_i + q_i^T (p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j)\right)^2 + \lambda\left(b_u^2 + b_i^2 + ||q_i||^2 + ||p_u||^2\right))$

3) $MLM_{Baseline(ALS/SGD)} = (\mathbb{R}^f, \mathbb{R}, \mu + b_u + b_i\ , P(\theta) = 1, \sum_{r_{ui} \in R_{train}} ((r_{ui} - (\mu + b_u + b_i))^2 + \lambda(b_u^2 + b_i^2))$

4) $MLM_{KNNZ} = (\mathbb{R}^f, \mathbb{R},\ \mu_u + \sigma_u \frac{\sum_{v \in N_{i(u)}^k} sim(u,v) * \frac{r_{vi} - \mu_v}{\sigma_v}}{\sum_{v \in N_{i(u)}^k} sim(u,v)}\ , P(\theta) = 1,\ N/A)$

5) $MLM_{KNN\ MEAN} = \left(\mathbb{R}^f, \mathbb{R}, \mu_u + \frac{\sum_{v \in N_{i(u)}^k} sim(u,v) * (r_{vi} - \mu_v)}{\sum_{v \in N_{i(u)}^k} sim(u,v)}\ , P(\theta) = 1,\ N/A\right)$

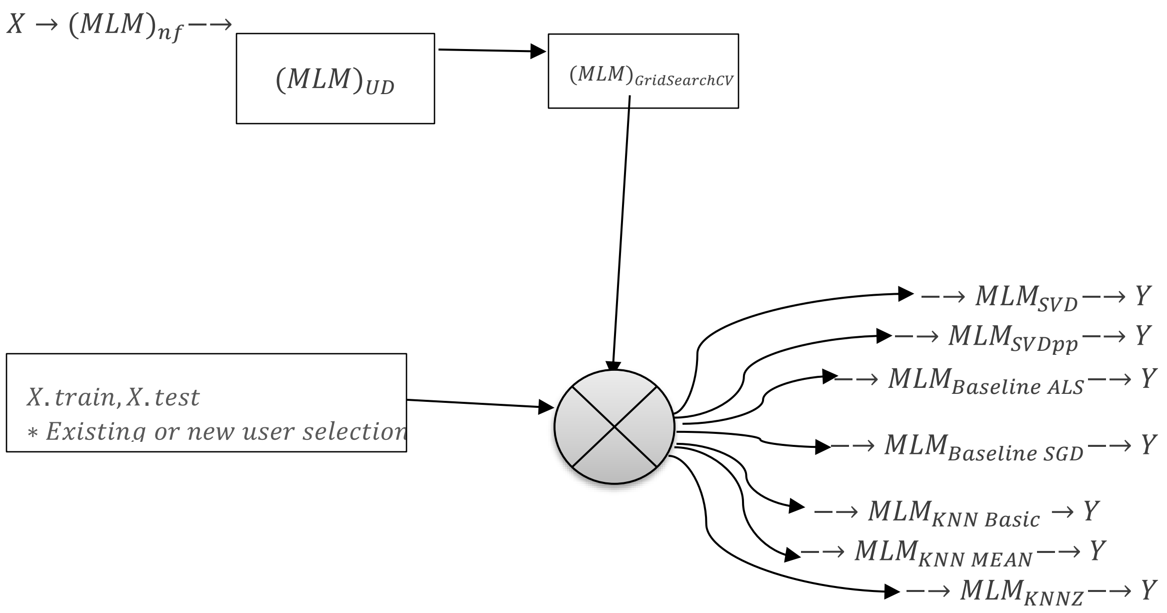
6) $MLM_{KNN\ Basic} = \left(\mathbb{R}^f, \mathbb{R}, \frac{\sum_{v \in N_{i(u)}^k} sim(u,v) * r_{vi}}{\sum_{v \in N_{i(u)}^k} sim(u,v)}\ , P(\theta) = 1,\ N/A\right)$

7) $MLM_{UD} = G(X, choice) = if\ choice == SVD: Y \rightarrow 0000001, elif\ choice == SVDpp: Y \rightarrow 0000010,\ elif\ choice == ALS: Y \rightarrow 0000100, elif\ choice == SGD: Y \rightarrow 0001000,\ elif\ choice == KNN_{Basic}: Y \rightarrow 0010000,\ elif\ choice == KNN_{Means}: Y \rightarrow 0100000, elif\ choice == KNN_{ZScore}: Y \rightarrow 1000000.\ return\ X, Y$

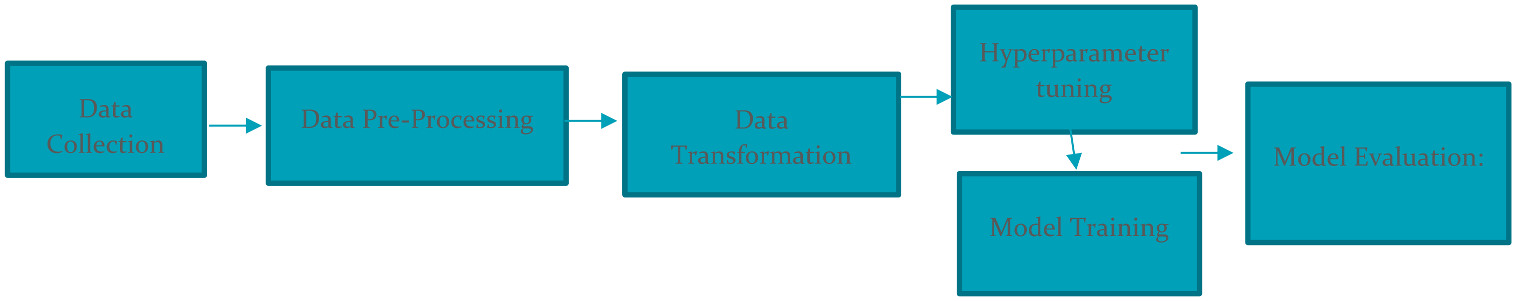
8) $MLM_{GridSearchCV} = H(X) = if\ X == 1: params \rightarrow SVD[Parameters], elif\ X == 2: params \rightarrow SVD++[Parameters], elif\ X == 4: params \rightarrow BaselineOnly_{ALS}[\text{Parameters}], elif\ X == 8: params \rightarrow BaselineOnly_{SGD}[\text{Parameters}],\ elif\ X == 16: params \rightarrow$

$$KNN_{Basic}[\text{Parameters}],\ elif\ X == 32{:}\ params \rightarrow$$
$$KNN_{Means}[Parameters], elif\ X == 64{:}\ params \rightarrow$$
$$KNN_{Zscore}[Parameters].\ \ return\ GridSearchCV(X, params).BestEstimator$$

Machine Learning Workflow:


$X \rightarrow (MLM)_{nf} \dashrightarrow$
$(MLM)_{UD}$
$(MLM)_{GridSearchCV}$
$X.train, X.test$
$*$ Existing or new user selection
$\dashrightarrow MLM_{SVD} \dashrightarrow Y$
$\dashrightarrow MLM_{SVDpp} \dashrightarrow Y$
$\dashrightarrow MLM_{Baseline\ ALS} \dashrightarrow Y$
$\dashrightarrow MLM_{Baseline\ SGD} \dashrightarrow Y$
$\dashrightarrow MLM_{KNN\ Basic} \rightarrow Y$
$\dashrightarrow MLM_{KNN\ MEAN} \dashrightarrow Y$
$\dashrightarrow MLM_{KNNZ} \dashrightarrow Y$

The project's workflow shown above was modeled based off the traditional workflow:



Morphism Tables:

Algorithm	Input Space X	Output Space	Parameter Prior
SVD			

SVD++	User-movie rating vector of R^m, 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.
BaselineOnly			
KNN			
KNNWIthMeans			
KNNWithZScore			
KNNBasic			

Models	Learning Functions: $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{\Theta})$	Loss Functions: $\boldsymbol{L}(\boldsymbol{y}, \boldsymbol{F})$	Parameters
SVD	Traditional SVD Decomposition which the model is based off of: $R = U\Sigma M^T$ SGD is used for the following parameter updates until convergence for both SVD and SVDpp: $b_u = b_u + \alpha(e_{ui} - \lambda b_u)$ $b_i = b_i + \alpha(e_{ui} - \lambda b_i)$ $p_u = p_u + \alpha(e_{ui} q_i - \lambda p_u)$ $q_i = q_i + \alpha(e_{ui} p_u - \lambda q_i)$ SVD Parameter Updates Actual SVD Learning Function: $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{\Theta}) = \mu + b_u + b_i + q_i^T p_u$ SVDpp: $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{\Theta}) = \mu + b_u + b_i + q_i^T (p_u + \lvert I_u \rvert^{-\frac{1}{2}} \sum_{j \in I_u} y_j)$ Additional Feature Vector (implicit feedback) $q_i = q_i + \alpha(e_{ui}(p_u \lvert I_u \rvert^{\frac{1}{2}} \sum_{j \in I_u} M_j + - \lambda q_i)$	Regularized Squared Error Loss: $\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \lvert\lvert q_i \rvert\rvert^2 + \lvert\lvert p_u \rvert\rvert^2)$	{'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]}

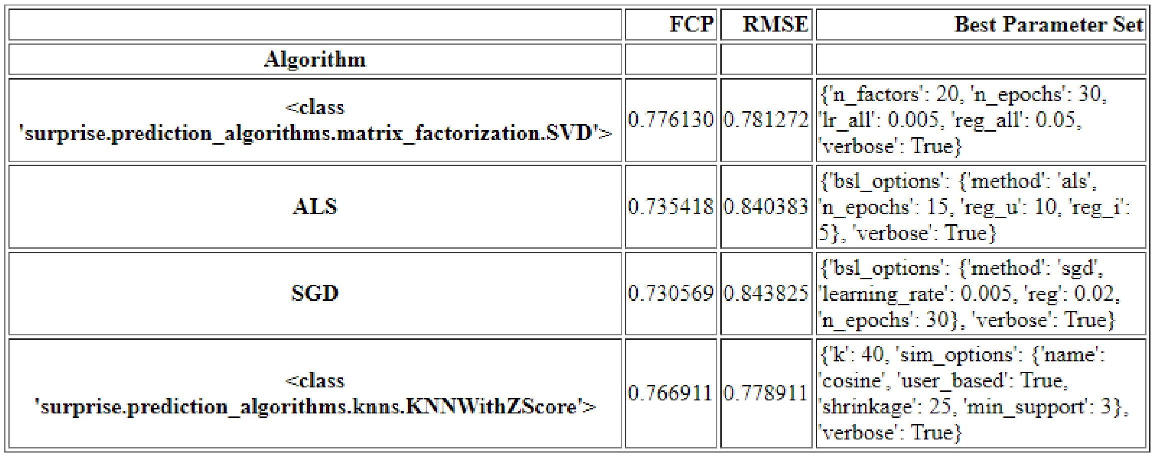
BaselineOnly	SGD and ALS methods to optimize the following: $b_u = b_u + \alpha(e_{ui} - \lambda b_u)$ $b_i = b_i + \alpha(e_{ui} - \lambda b_i)$ $\mu + b_u + b_i$	Regularized Squared Error Loss: $\sum_{r_{ui} \in R_{train}} ((r_{ui} - (\mu + b_u + b_i))^2 + \lambda(b_u^2 + b_i^2))$	ALS:{'bsl_options':{'method': ['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. $Pearson : sim(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)(r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}}, Cosine: sim(u,v) = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u} {r_{ui}}^2} \sqrt{\sum_{i \in I_v} {r_{vi}}^2}}$		
KNNWithMeans	$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} sim(u,v)(r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} sim(u,v)}$ (user-user)		{'k': [20, 40], 'sim_options': {'name': ['pearson_baseline', 'cosine'],'shrinkage':[100,75,50],'min_support': [3],'user_based': [True]}
KNNWithZScore	$\hat{r}_{ui} = \mu_u + \sigma_u \frac{\sum_{v \in N_i^k(u)} sim(u,v) \frac{(r_{vi} - \mu_v)}{\sigma_v}}{\sum_{v \in N_i^k(u)} sim(u,v)}$ (user-user)		

KNNBasic	$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} sim(u,v) * r_{vi}}{\sum_{v \in N_i^k(u)} sim(u,v)}$ (user-user)	
Model Evaluation	RMSE & FCP Metrics: $\sqrt{\frac{1}{n}\sum_{i}^{n}(\frac{r_{ui} - \hat{r}_{ui}}{\sigma})^2}, FCP = \frac{n_c}{n_c + n_d}$	Evaluation of parameters for each model on test set.
Variables	$\mathrm{R_{ui}, B_u, U, M, p_u}, q_\mathrm{i} =$ Ratings /Baseline matrices for user "u" & movie "i", user/movie feature matrices, user/movie feature vectors $r_{ui}, \hat{r}_{ui}, \mathrm{\mu_u, b_u, b_i} =$ *predicton and true ratings for user* "u" & movie "i", global average ratings, user bias, movie bias I_u = One − hot encoded vector representing movies user "u" has interacted with $\lambda_1, \lambda_2, \alpha =$ L1 and L2 regularizer terms & learning rate $sim(u,v)$ = similarity between users "u" & "v" $n_c^u = \|\{(i,j)\| \hat{r}_{ui} > \hat{r}_{uj} \,\&\, r_{ui} > r_{uj}\}, n_c = \sum_u n_c^u, n_d^u = \|\{(i,j)\| \hat{r}_{ui} > \hat{r}_{uj} \,\&\, r_{ui} > r_{uj}\}, n_d = \sum_u n_d^u$ $N_u^k(i), N_i^k(u) =$ *set of* "k" *users most similar to* "u" & *set of* "k" movies most similar to "i"	

4. Results and Insights

A. METRICS RESULTS



	FCP	RMSE	Best Parameter Set
Algorithm			
<class 'surprise.prediction_algorithms.matrix_factorization.SVD'>	0.776130	0.781272	{'n_factors': 20, 'n_epochs': 30, 'lr_all': 0.005, 'reg_all': 0.05, 'verbose': True}
ALS	0.735418	0.840383	{'bsl_options': {'method': 'als', 'n_epochs': 15, 'reg_u': 10, 'reg_i': 5}, 'verbose': True}
SGD	0.730569	0.843825	{'bsl_options': {'method': 'sgd', 'learning_rate': 0.005, 'reg': 0.02, 'n_epochs': 30}, 'verbose': True}
<class 'surprise.prediction_algorithms.knns.KNNWithZScore'>	0.766911	0.778911	{'k': 40, 'sim_options': {'name': 'cosine', 'user_based': True, 'shrinkage': 25, 'min_support': 3}, 'verbose': True}

Algorithm Metrics

As an additional option provided to the user at the index page of the flask web application used as the interface for this project, the user can select metrics rather than opting for an algorithm and proceeding to the recommendations generation path. The table above, "Algorithm Metrics", is the results gathered from running the 4 listed algorithms out of the possible 7 options. This process includes the same GridSearchCV function used in the recommendations workflow (even kept the k number of folds consistent at 3) but now with a somewhat different and expanded field of hyperparameters to test on in order to ensure each algorithm works at the highest level of optimality, and therefore, as good barometers to compare against one another. The results of this grid search cross-validation were stored in the zip file located at the link found in section 6 under citation 3a. It should be noted, however, that the testset was different than the recommendations workflow in the sense that the predictions intentionally tests on the anti testset to predict unseen movies, whereas for the sake of these metrics the models were run, after cross validation, on a test set (called by the build_testset() function in surprise) that contains only interactions of known ratings but with those ratings hidden, and hence that is how the RMSE and FCP were able to be calculated by taking actual vs predicted values

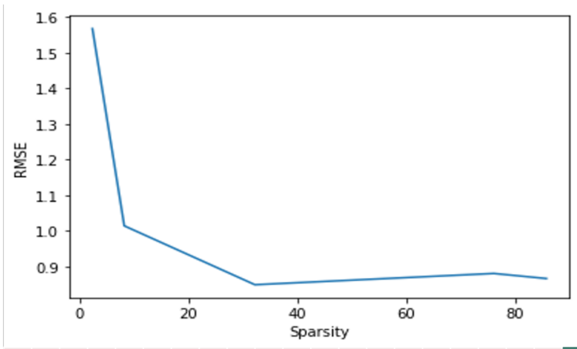
The reason for these 4 being selected is simple: SVDpp, although was seen to outperform the generic SVD surprise algorithm, cost way more in terms of time and space and resulted in upwards of a 30-minute runtime on a 4 core PC, and was therefore decided against for the purposes of this comparison. Furthermore, the z-score normalized KNN algorithm was chosen over the others as it was observed that this extra level or normalization consistently outperformed the other two KNN algorithms, and since they all three share the same hyperparameters, it was decided that no additional insights would be provided by including the less accurate versions of KNN. The baseline algorithms, one optimized through ALS and the other through SGD, were kept as exactly what they sound like, baselines to compare against. Since we know that this baseline equation is simply the global mean plus the learned user bias and item bias, a very fast and simplistic algorithm, the table shows what the added complexity of SVD, which includes the user and movie feature matrices embedded with learned latent features through SGD, provides in terms of accuracy. And although KNN may be in a different classification in terms of its learning scheme, nearest neighbors vs the other three falling under the matrix factorization category, it is more involved and takes into account information from other users in order to provide a prediction, and was therefore also included to show what the added complexity can do for the model in terms of accuracy.

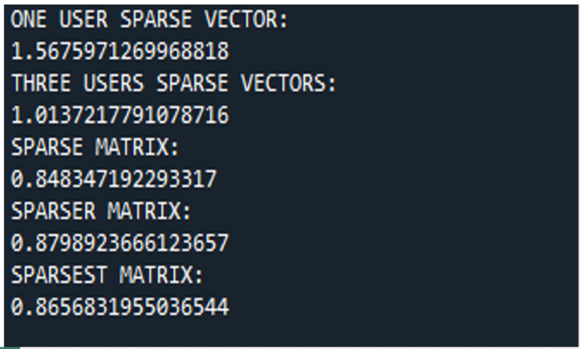
Unsurprisingly, the KNN and SVD both outperformed the baseline algorithms across both measurements of accuracy, as is displayed in the middle two columns. The last column shows the hyperparameter combination that resulted in the best grid search scores averaged across all folds of cross validation. Although nearest neighbors methods typically suffer from the cold start problem, that didn’t seem to occur in this experiment as in terms of both RMSE and FCP the scores are very comparable between SVD and KNN (interesting note is that even though they are close, one outperforms the other in one metric and vice versa for the other). This could be due to the fact that the data used in the training and test wasn’t as sparse as the full matrix, and therefore, may not generalize very well as the train and test data used in these algorithms for this metric comparison was comprised of only the observed values. Therefore, every unobserved interaction was filtered out by the surprise library’s creating of the training and test set, resulting in users only have a small subset of there actual interactions involved in the model. This would cause less sparsity as the training data would be larger than the test data, and therefore the "unobserved" (hidden in order to run the algorithm on the data points) test set is smaller than the data the model was trained on, which is dissimilar to the recommendations workflow where the model is trained on the observed values (a subset of the 1.6% of the interactions observed) and then tested on the anti testset which comprises the the 98.6% of interactions that are initially unobserved. So how then can this metric comparison be generalized to the recommendations predictions accuracy? This question is explored in the following section, where a completely filled benchmark matrix is established and tested, and then an increasing amount of sparsity is introduced into the data in order to establish a relationship between sparsity and accuracy.

B. BENCHMARK VALIDATION

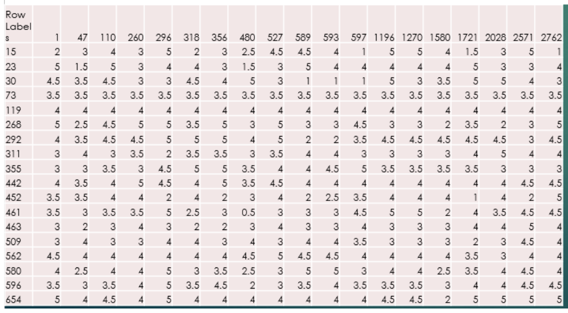
In order to generalize the results from subsection A to the recommendations and their respective accuracies in the recommendations workflow, this experiment was created. It involved the truncating of the original user-movie interaction matrix from its initial size of 671x9066, down to an eventual 18x19 matrix that is completely observed. The process to filter down to this new size was arbitrary, as movies and users were iteratively filtered in areas where there was heavy sparsity up until the point in which this new, filled matrix was established. This new matrix was coined "benchmark matrix" and can be seen in the bottom left of the following graphic.

Benchmark Matrix Experiment:

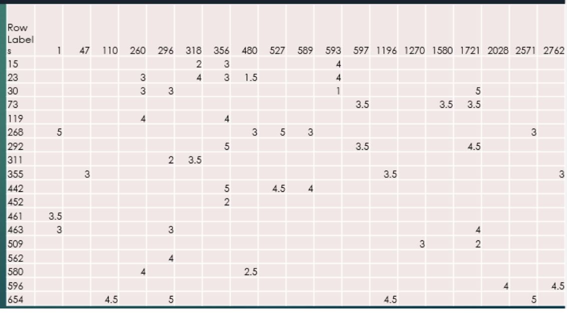




```
ONE USER SPARSE VECTOR:
1.5675971269968818
THREE USERS SPARSE VECTORS:
1.0137217791078716
SPARSE MATRIX:
0.848347192293317
SPARSER MATRIX:
0.879892366123657
SPARSEST MATRIX:
0.8656831955036544
```

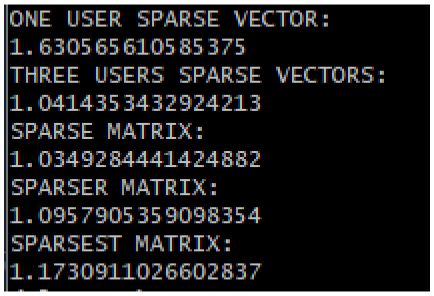


Row Labels	1	47	110	260	296	318	356	480	527	589	593	597	1196	1270	1580	1721	2028	2571	2762
15	2	3	4	3	5	2	3	2.5	4.5	4.5	4	1	5	5	4	1.5	3	5	1
23	5	1.5	5	3	4	4	3	1.5	3	5	4	4	4	4	4	5	3	3	4
30	4.5	3.5	4.5	3	3	4.5	4	5	3	1	1	1	5	3	3.5	5	5	4	3
73	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5
119	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
268	5	2.5	4.5	5	5	3.5	5	3	5	3	3	4.5	3	3	2	3.5	2	3	5
292	4	3.5	4.5	4.5	5	5	5	4	5	2	2	3.5	4.5	4.5	4.5	4.5	4.5	3	4.5
311	3	4	3	3.5	2	3.5	3.5	3	3.5	4	4	3	3	3	3	4	5	4	4
355	3	3	3.5	3	4.5	5	5	3.5	4	4	4.5	5	3.5	3.5	3.5	3.5	3	3	3
442	4	3.5	4	5	4.5	4	5	3.5	4.5	4	4	4	4	4	4	4	4	4.5	4.5
452	3.5	3.5	4	4	2	4	2	3	4	2	2.5	3.5	4	4	4	1	4	2	5
461	3.5	3	3.5	3.5	5	2.5	3	0.5	3	3	3	4.5	5	5	2	4	3.5	4.5	4.5
463	3	2	3	4	3	2	2	3	4	3	3	4	3	3	3	4	4	5	4
509	3	4	3	3	4	4	3	4	3	4	4	3.5	3	3	3	2	3	4.5	4
562	4.5	4	4	4	4	4	4	4.5	5	4.5	4.5	4	4	4	4	3.5	3	4	4
580	4	2.5	4	4	5	3	3.5	2.5	3	5	5	3	4	4	2.5	3.5	4	4.5	4
596	3.5	3	3.5	4	5	3.5	4.5	2	3	3.5	4	3.5	3.5	3.5	3	4	4	4.5	4.5
654	5	4	4.5	4	5	4	4	4	4	4	4	4	4.5	4.5	2	5	5	5	5

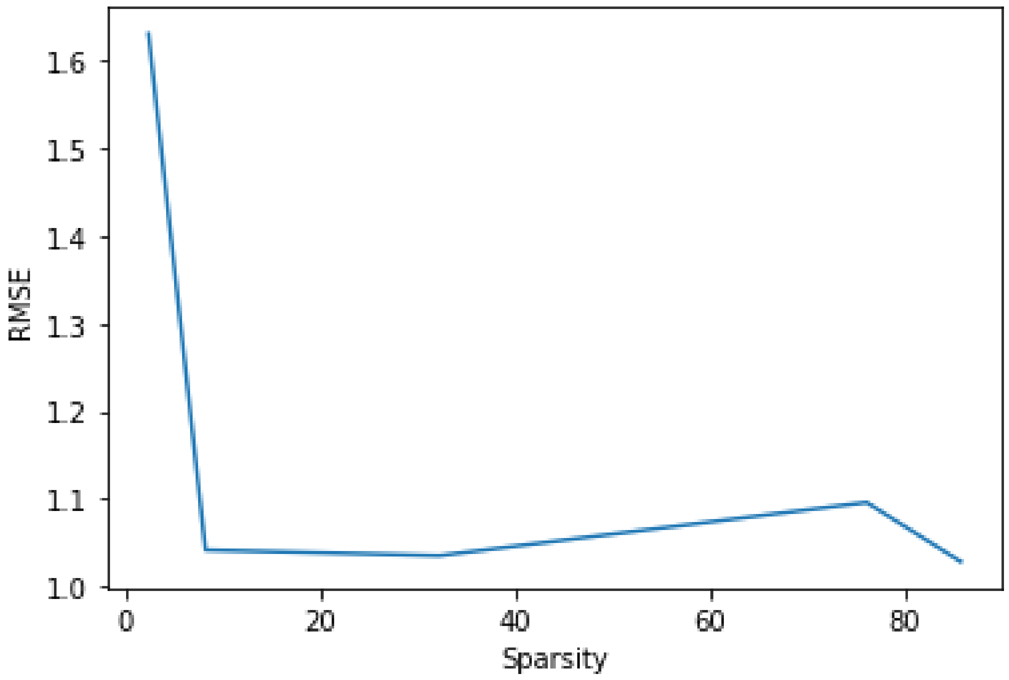


Row Labels	1	47	110	260	296	318	356	480	527	589	593	597	1196	1270	1580	1721	2028	2571	2762
15						2	3				4								
23				3		4	3	1.5			4								
30				3	3						1					5			
73												3.5			3.5	3.5			
119				4			4												
268	5							3	5	3								3	
292							5					3.5				4.5			
311					2	3.5													
355		3											3.5						3
442							5		4.5	4									
452							2												
461	3.5																		
463	3				3											4			
509														3		2			
562					4														
580				4				2.5											
596																	4		4.5
654			4.5		5								4.5					5	

After this, sparsity was gradually increased for subsequent trials up until the level of sparsity shown in the bottom right matrix of the previous graphic. The top two pictures in the graphic are the RMSE accuracy vs sparsity% (top left) and the RMSE values for each run in the right with a vague description of the amount of sparsity introduced. It should be noted that this was for the SVD run. The results for the KNNZscore trials were:



```
ONE USER SPARSE VECTOR:
1.630565610585375
THREE USERS SPARSE VECTORS:
1.0414353432924213
SPARSE MATRIX:
1.0349284441424882
SPARSER MATRIX:
1.0957905359098354
SPARSEST MATRIX:
1.1730911026602837
```



KNN Results

As the results show, the SVD outperformed the KNN algorithm in terms of RMSE for all levels of sparsity, with the difference in accuracy increasing as sparsity increased, proving the hypothesis that SVD is immune to the cold start problem that nearest neighbors methods tend to suffer from. Upon further investigation of the trends of the graph, it was hypothesis prior to running the experiment that the graph would display a parabolic behavior where there would be an optimal sparsity point somewhere in the middle and then as sparsity continued to increase it so to would the error. This hypothesis is due to little sparsity intuitively creating a training set out of a large portion of the data, causing the model to overfit and not generalize well to new data, and on the flip side, too much sparsity would not allow the model to be trained properly as it would be predicting on a test set much larger than what it was trained on, and therefore, the learning wouldn't be adequate to handle new data. The truth lies somewhere in the middle, as the derivative of the graph was downwardly steep initially, probably due to overfitting being reduced, and then the error began to increase as more sparsity was introduced but then tailed off at the end to a lower error. The workflow used in each of these trials was set up similarly to the recommendations workflow the user progresses through in the application, as the model was trained on the full observed data, then rather than being run against a test set where a subset of the observed values are hidden and then used as predictions to compare against the actuals, the anti testset (unobserved interactions) was utilized. This was intentionally done in order to predict unseen movies just as it did in the actual recommendations workflow. The errors between the anti testset predictions and the actual ratings were calculated by hand using the actual values found in the benchmark matrix, which has every interaction observed, since the surprise library models wouldn't contain the unobserved actual values in the trial matrices, and therefore, could not inherently calculate the RMSE on its own.

5. Conclusions

Recommendation systems are ubiquitous to the consumer world, and as such the research and engineering invested into them have followed suit. Therefore, this project investigates and analyzes the current state of modern algorithms deployed by these systems, particularly the algorithms using collaborative filtering methods. Collaborative filtering algorithms learn and predict recommendations based off users' previous behaviors and the behaviors of users deemed similar to them, as opposed to content based filtering, which focuses on concrete attributes of the items themselves. Although these algorithms can be harnessed in any domain where item recommendations can and are encouraged to be made, the scope of this study was limited to the film domain, mirroring systems adapted by multimedia streaming services.

As highlighted in the morphisms section, the workflow consisted of first, the descriptive stage, where initialization of the dataset occurred by removing any movies whose corresponding tmdbId was invalid. These were removed because the lack of a tmdbId mapped to the a movieId from our original user-movie interaction matrix prevents that movieId from being linked to a film title, not allowing for intuitive results for the user. Further outlier detection was investigated and reported in section 2 of this report, but it was concluded that additional filtering provided no real benefit, and rather, the amount of sparsity was encouraged to attempt to mirror what the streaming services utilizing these recommendations face in terms of incomplete data. After the removal of the 42 films with invalid tmdbIds, the matrix used as input data consisted of 671 users and 9024 movies, where the former were the rows and the latter the columns, and the user's rating for a specific film was the value at that same given user's row at that specific movie's column. The interface for the user was a webpage powered by the flask server, which at the index page, allowed the user the option to select from amongst the 7 discussed algorithms, 4 of which were categorized as matrix factorization while the other 3 were all nearest neighbors methods. Furthermore, the nearest neighbors algorithms were just different levels of normalization on the KNN algorithm.

After user's selection of the algorithm, the user then chooses whether they would like the recommendations for an existing user in the database over the movies that that user hasn't seen, or if they would like to be given recommendations themselves. In the latter case, the user is prompted with several film titles and is allowed to give ratings from 0.5-5, or skip, until the user has inserted 20 ratings (done to avoid the cold start problem). Once either of these processes is completed the previously chosen algorithm is hyper tuned through grid search cross validation, where several combinations of parameters are tested and the best one in terms of RMSE averaged across 3 folds is chosen as the hyperparameters to be used in the predictive stage. The cross-validation step is run on the entire observed interactions set in order to create an unbiased estimator. The surprise library allows for the creation of a training set for the model training that happens after hyperparameter tuning, and then the trained model is tested on an anti testset which is composed of all the unobserved, and therefore, unseen films, allowing the scope of predictions to be limited to the desired subset. After the prediction stage has occurred, a brief prescriptive method filters out all the films aside from those with one of the 10 highest predicted ratings. These are then presented to the user as the set of recommendations and are therefore, the output of the system.

In addition to the option of choosing an algorithm at the index of the webpage, there is an alternate metrics selection, which was reported in section 4. As expected, the table "Algorithm Metrics" shows that the KNN and SVD algorithms provided substantial benefit over the simplistic baseline models, where only the global mean, user bias, and item bias are taken into account for the predictions. The SVD outperformed the KNN z-score algorithm in terms of the FCP metric, but was outperformed by its counterpart in the RMSE sense. A prior hypothesis was that due to the cold start problem inherent to KNN algorithms, and due to the sparsity of our data, it was theorized that the SVD would greatly outperform the KNN algorithms in all measures of performance given that out data was 98.4% sparse. Upon further discovery, a reason for this is that in this metrics table, the results were not run on the traditional recommendation workflow we mentioned in the paragraph above, where the accuracy was calculated on the predictions for unobserved interactions. Instead, these results were gathered by use of the surprise's capability to create training and tests, which sets aside a large subset of the observed values for training, and then setting aside the remaining subset for test data by hiding the actual values during prediction, greatly reducing the overall sparsity in the prediction phase and therefore, allowing KNN to avoid its cold start pitfall. To clarify, the sparsity is drastically reduced since upon creating these training and test datasets, all of the unobserved interactions are filtered out, and the training and test set are much more comparable in size, in fact, the training data is actually larger than the test set, a luxury not afforded to the recommendations algorithm where the training data is only a maximum of 1.6% of the interactions (the percentage of observed interactions) while the remaining 98.4% are used as the test set (the anti test set) when predicting.

Due to the differences in the metrics process and the actual recommendations workflow, which is the crux of the project, a benchmark experiment was proposed and was discussed in section 4.B of this report. The benchmark experiment studied KNN vs SVD starting with the “benchmark matrix” which was a truncated version of the original user-movie down to an 18 users by 19 movies matrix. This matrix truncation was arbitrary, as it didn’t follow a specific procedure, but rather was filtered down until a fully observed matrix was formed. Sparsity was then gradually introduced into the benchmark matrix until there was 6 new matrices, each with a different level of sparsity, and the two algorithms were run on each of these matrices, and then the RMSE was calculated for the predicted values against the actual values. The accuracy vs sparsity was then plotted in hopes of establishing a pattern, however, the results turned out inconclusive. It was hypothesized prior to performing this experiment that the initial low levels of sparsity would have a relatively higher RMSE as the data would be overfit to the training data and wouldn’t generalize well to the small set of test data it was predicting on, but then it would eventually reach an optimal point somewhere in the middle before it increased back towards higher levels of error when the sparsity increased to the point where training data became inadequate. What happened instead was the initial errors for the low sparsity matrices were in fact high, and then there was a steep drop off in error towards an optimal sparsity level, at which point the error then increased again as the sparsity grew past this point, but rather than assuming a parabolic shape where the error continued to increase for further sparsity, the error once again dropped for the matrix with the highest level of sparsity’s prediction values. This led to an inconclusive, and perhaps incorrect, hypothesis. The relationship described above held true for both the KNN and SVD algorithms, and this lack of a clear relationship provokes the desire for further study into the matter on how exactly sparsity impacts recommendation algorithm’s effectiveness. Alternatively, the other hypothesis of SVD outperforming KNN when presented with sparse data rang true, as in addition to the SVD provided predictions resulting in a lower RMSE at every sparsity level than KNN, the increase in performance over KNN grew as the levels of sparsity increased.

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/

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

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

a)https://github.com/aflynn0213/MovieRecommenderForDummies/metrics_cv_results/

Hug, Nicolas. *Surprise*, Sept. 2019, https://surpriselib.com/

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.