# Anime Recommender System Exploration: Final Report

Scott Freitas & Benjamin Clayton

**Abstract**—This project is an exploration of modern recommender systems—utilizing various methodologies ranging from global baseline to item-item collaborative filtering and latent factor models. Each of the aforementioned techniques are industry standards which we will use to analyze an anime dataset. We believe that this paper will be of particular interest to the anime community as there is currently a dearth of recommender systems targeting this audience. Each technique was implemented in Python and used a training-test set split of 80%-20%, respectively, on each model developed. Error for each model is reported as root mean squared error (RMSE).

**Index Terms**—Recommender systems, collaborative filtering, latent factor model, SVD, gradient descent.

## 1 Introduction

The goal of a recommender system is to make predictions about a user's tastes. Specifically, what a user will want to watch or buy in the future. In order to predict these things, large amounts of user data is needed to find patterns and associate prior tastes with future expenditures. This is a particularly interesting problem because it's often very difficult to give good recommendations to users based on the limited information available about the user. The only information that is readily available on a user is their previous ratings for some set of given items. However, these ratings are often sparse, where they have only rated a few of potentially millions of items. Therein lies the challenge—how do we create meaningful user suggestions, with very limited data? In order to do this, we propose using three industry standard techniques, including: (1) global baseline, (2) item-item collaborative filtering and (3) latent factor models. In addition, we're using an anime television show dataset that keeps a record of user/movie pairs of ratings—of which there are 73,516 unique users and 12,294 unique animes. This dataset is freely accessible to the public and is hosted on Kaggle [7].

**Areas of Impact.** The challenge of generating interesting and novel recommendations to users is a problem faced in a variety of business domains. Many companies are currently spending significant sums of money in R&D towards developing improved product recommendations. This isn't surprising given that many companys' bottom line is directly affected by the products they sell to customers. One example where recommendations is particularly important is in the Netflix business model of movie and TV show streaming. Customer attraction and retention is based on how many interesting products a user can find on their site—a large part of which is found based on their in-house recommendation system. As part of an effort to improve their in-house recommendation model, they held an open competition that concluded in 2009, with the aim of beating their baseline recommendations by 10% [5]. In this paper, we implement and discuss one of the most popular methods in the challenge, latent factor models.

Aside from Netflix, there are a large variety of business's that require improved product recommendations, ranging from items such as food and music to clothes and electronics—all of which will benefit from improved recommender systems.

• *S. Freitas and B. Clayton are with the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, 85281.*
*E-mail: see safreita@asu.edu and bsclayt1@asu.edu*

*Manuscript sent in on December 1st, 2017*

## 2 Related Works.

Recommender Systems is a thriving field of research. There are many teams producing their own innovations to this area. The 2013 literature survey on Recommender Systems covers many of the features of this project [3]. This paper indicates that collaborative filtering is the basis of many state of the art tools. Additionally, RMSE is one of the main methods to evaluate performance. This affirms the choice of having collaborative filtering as an intermediate step as well as the evaluation metric.

The 2009 paper, "Matrix Factorization Techniques for Recommender Systems", discusses latent factor model in greater detail than will be found in this paper [6].

The seminal 2001 paper, "Item-Based Collaborative Filtering Recommendation Algorithms", introduces the classic item-item collaborative filtering algorithm implemented in this project [4]. The Netflix award winning paper by BigChaos introduces a novel blend of methods and notes how the RMSE decreases when more information is incorporated into the latent factors [5].

The recent paper published in KDD, HoORaYs, proposed an improvement to the loss function used for the latent factor model [2]. This improvement is the inclusion of second-order rating distance. That is, consider the prediction error for the other items that a user has rated when updating a single item. This was aimed to reduce the variance in prediction errors and outperformed alternative methods of rating prediction on the MovieLens, Google Play, Lastfm, and Delicious datasets. The alternative methods were probabilistic matrix factorization, bayesian personalized ranking, and collaborative topic regression.

## 3 Implementation Methodology.

### 3.1 Assumptions.

It is assumed that when optimizing the latent factor model, a local optimum is good enough regarding gradient descent. This is reasonable since global optimum cannot be guaranteed under the vast majority of situations. In order to guarantee convergence to the global minimum the problem would have to be formulated as a convex optimization problem or satisfy the KarushKuhnTucker conditions. Neither of which are feasible for the problem as it currently exists.

Additional assumptions were made regarding the integrity of the code libraries that were used in this program. In particular: Numpy's linear algebra functions, Scipy's sparse matrices, and Sklearn's sparse similarity functions.

### 3.2 Problems Encountered.

Several problems were encountered during the course of the project. The foremost of which is that due to the scale of the dataset the implementation not only had to be correct but also efficient. The anime dataset has 6,337,239 non-zero ratings which we divided into 5,069,791 training samples and 1,267,448 test samples (80-20% split) for both the arbitrary and randomly sampled splits. This large training set often caused many difficulties with implementation and necessitated significant optimization of the code. The function to compute the similarity matrix used to take around eight hours before being optimized. The RMSE for collaborative filtering used to be around thirty hours. And the epochs for latent factor gradient descent remain slow.

There were additional problems encountered with latent factor. There are parameters that control the learning rate and the regularization component. Initially, these values were too large and within a few updates the matrices converged to infinity. The parameters were reduced a few orders of magnitude and tested at that scale for the rest of the project.

### 3.3 Preprocessing.

In order to facilitate the data processing, we remapped the original anime dataset to have anime ids corresponding to their row number in the csv file instead of the arbitrary number assignment they originally had. This let us avoid having anime ids with values significantly larger than the number of anime shows in the dataset. This allows us to have the sparse matrix indices in increasing order and meaningfully represent the items.

Before remapping was done, we removed all entries that were not actual ratings—this included entries that were '-1' or empty. This lead to a circumstance where a large number of both users and movies had no ratings and were therefore removed from the dataset. Originally we had 73,516 unique users and 12,294 unique animes—this number was reduced to 69,600 users and 9,926 animes after removing these entries.

After the dataset was remapped we created a training and test set split using two different methodologies. The first, naive method, was to arbitrarily split the data into training and test sets by getting the first 'x' elements that represented 80% of the dataset and making those the training data and creating the test dataset from the remaining elements. The second method we used was to randomly sample the test data without replacement until we reached 20% of the data. All elements that weren't selected for the test dataset were then made into the training dataset.

The final phase involves saving the resulting sparse matrices for both training and test datasets as .npz files. These are the files that the rest of the project depends on.

### 3.4 Baseline Estimate.

**Parameter Calculation.** This is the basic estimate, it is primarily used to determine the efficacy of the other methods by providing a baseline for performance. The process is fairly straightforward: to predict a rating simply add the global baseline with the relative user mean and relative movie mean as seen in equation (4) [1]. A more precise implementation can be seen in algorithm 1. Each of these parameters are calculated for both training datasets.

- $r_{ui}$ = Rating of user $x$ on item $i$
- $\hat{r}_{ui}$ = The predicted rating of user $x$ on item $i$
- $b_{ui}$ = Global baseline estimate on user $x$ and item $i$
- $\mu$ = Overall mean movie rating
- $b_u$ = Rating deviation of user $u$ = (avg. rating of user $u$ across all movies user $u$ has rated) - $\mu$
- $b_i$ = (avg. rating of movie $i$) - $\mu$
- $N$ = |Training set|
- $N'$ = |Test set|

The global baseline rating is calculated by (1).

$$\mu = \frac{\sum_{(i,u)\in Training} r_{iu}}{N} \tag{1}$$

The relative user mean is calculated by (2).

$$b_u = \frac{\sum_{i\in r_u} r_{iu}}{|r_u|} - \mu \tag{2}$$

The relative movie mean is calculated by (3 ).

$$b_i = \frac{\sum_{u\in r_i} r_{iu}}{|r_i|} - \mu \tag{3}$$

The predicted rating is calculated by (4).

$$\hat{r}_{iu} = b_{iu} = \mu + b_u + b_i \tag{4}$$

The root mean square error (RMSE) is calculated by (5).

$$\text{RMSE} = \sqrt{\frac{\sum_{(i,u)\epsilon Test}(r_{iu} - \hat{r}_{iu})^2}{N'}} \tag{5}$$

**Testing.** The performance of this method was calculated using the RMSE equation (5) on the test dataset for both the randomly sampled and arbitrary split of the data.

**Algorithm 1:** Baseline Estimation

```
Input: Sparse training matrix $T$, sparse test matrix $T'$
       (both COO format)
Result: RMSE on $T'$
Calculate the following values using $T$:
 $\mu = \frac{\sum_{(i,u)\in Training} r_{iu}}{|T|}$;
$T_c = COO\_To\_CSC(T)$
for column (u) in $T_c$ do
    $b_u = \frac{\sum_{i\in r_u} r_{iu}}{|r_u|} - \mu$;
end
$T_r = COO\_To\_CSR(T)$
for row (i) in $T_r$ do
    $b_i = \frac{\sum_{u\in r_i} r_{iu}}{|r_i|} - \mu$;
end
for (item, user) in $T'$ do
    summed_error = summed_error +
      $(r_{iu} - \mu + b_u + b_i)$;
end
RMSE = $\sqrt{summed\_error/|T'|}$;
```

### 3.5 Item-Item Collaborative Filter.

**Preprocessing.** The first step to this process is to center the movie ratings by subtracting each movie's mean rating from the corresponding individual ratings of each movie. Unseen ratings are not centered. This is done to more accurately capture the relevant data regarding the similarity of movies. The similarity of the movies is determined by computing cosine similarity as detailed in the following equation (6) [1]. To do this, the sparse movie rows are converted to Numpy arrays and Numpy's linear algebra library is used to compute the inner product and L2-norm of the movie vectors, $a$ and $b$. This similarity metric is otherwise known as Pearson correlation.

$$s_{ij} = \frac{a^T b}{||a||_2||b||_2} \tag{6}$$

Due to the size of the full similarity matrix, running exact KNN on it for every movie is prohibitively expensive. Instead, the vector of similarities is partitioned around a pivot point using Numpy's argpartition. All elements that are in the top $K$ go to one side of the pivot, leaving the insignificant ones on the other side. The most significant side is retained for prediction.

**Prediction.** The following equation (7) was used to determine the predicted rating for a given user on a particular movie. The similarity selection algorithm implemented looks at only movies that the user has rated and determines an arbitrary number of most similar items to the specified movie. The number of similar items selected was five for this project.

- $u$ = the specified user
- $i$ = the movie for which the rating is being predicted
- $r_{iu}$ = the predicted rating for user $u$ on movie $i$
- $j \in N(i;u)$ = movies $j$ in the neighborhood of movie $i$ for user $u$
- $b_{iu}$ = the baseline estimate for user $u$ on movie $i$
- $s_{ij}$ = the similarity between movie $i$ and $j$
- $r_{ju}$ = the true rating for user $u$ on movie $j$ from training data
- $b_{ju}$ = the baseline estimate for user $u$ on movie $j$

$$\hat{r}_{iu} = b_{iu} + \frac{\sum_{j\epsilon N(i,u)} s_{ij}(r_{ju} - b_{uj})}{\sum_{j\epsilon N(i,u)} s_{ij}} \tag{7}$$

Note that when there are no other movies that a user has rated in the training set, this prediction is equivalent to the baseline estimate. See algorithm 2, below, for implementation details.

**Algorithm 2:** Item-Item Collaborative Filtering

```
Input: Sparse Training matrix $T$ (CSR), Sparse Test
       matrix $T'$(COO)
Result: RMSE on $T'$
Calculate the Pearson correlation matrix using the
 centered $T$
S = $[|items| \times |items|]$
for $item_i$ in $T$ do
    for $item_j$ in $T$ do
        $s_{ij} = \frac{a^T b}{||a||_2||b||_2}$
    end
end
for $(item, user)$ in $T'$ do
    Determine KNN for the item for the user
    Calculate the Baseline Estimate for the (item,user)
    $b_{iu} = \mu + b_u + b_i$
    Calculate predicted rating
    $\hat{r}_{iu} = b_{iu} + \frac{\sum_{j\epsilon N(i,u)} s_{ij}(r_{ju}-b_{uj})}{\sum_{j\epsilon N(i,u)} s_{ij}}$
    summed_error = summed_error + $(r_{iu} - \hat{r}_{iu})$;
end
RMSE = $\sqrt{summed\_error/|T'|}$;
```

**Testing.** The performance of this method was calculated using RMSE as in the baseline estimate on both the randomly sampled dataset and the arbitrarily split dataset.

### 3.6 Latent Factor Model.

The idea behind latent factor model is to decompose the matrix, $R_{i\times u}$ (sparse rating matrix of $i$ items and $u$ users), using Singular Value Decomposition (SVD) into two separate components $Q_{i\times k}$ and $P_{k\times u}$, where $k$ is the number of factors used to represent the low-rank matrix—instead of the traditional three components that SVD produces. Next, the decomposed matrices, $Q$ and $P$ are optimized using the sum of squared error seen in equation (8) and stochastic gradient descent (SGD). The missing values in the training set are ignored when running SGD [1].

- u = number of users
- i = number of items
- k = number of factors = 10
- R = $R_{i\times u}$
- Q = $Q_{i\times k}$
- P = $P_{k\times u}$
- $q_i$ = row i of Q
- $p_x$ = column x of $P^T$
- $\hat{r}_{xi}$ = Estimated rating of user $x$ on item $i$

- $\lambda$ = 0.006
- $\eta$ = 0.06
- $epochs$ = till convergence

**Preprocessing.** To begin, we center the COO (Coordinate Format) training matrix based on the columns (users). To do this, we calculate the mean rating of a user and subtract it from every rating of that user. This is then repeated for every other user in the training matrix. We then convert the COO sparse matrix to CSC (Compressed Sparse Column) and CSR (Compressed Sparse Row) matrices.

**Calculate $P$ and $Q$.** In order to calculate matrices $P$ and $Q$ we first calculate $U$, $\Sigma$ and $V$ using SVD were $R = U\Sigma V^T$. Matrix $Q$ is equivalent to $U$ and $P^T$ is calculated by multiplying $\Sigma V^T$. This now lets us form a new equation where $R \approx Q \times P^T$ [1]. It should be noted that matrices $P$ and $Q$ are now represented using dense matrices since they are significantly smaller when represented by k-factors.

**Optimizing $P$ and $Q$ using SGD.** In order to minimize the error between the predicted value and the actual value, we minimized $J(P,Q)$ by optimizing $P$ and $Q$ using stochastic gradient descent. In addition, to prevent overfitting of the data we introduced a regularization parameter, $\lambda$, which we experimentally set to 0.06 [1]. In addition, we set the learning rate, $\eta$, to 0.006, the number of latent factors, $k$, to 10.

Below are the equations used to calculate the update for every non-zero entry in matrices P and Q using SGD. In addition, algorithm 3 below is what we based the code on.

$$J(P,Q) = min_{P,Q} \sum_{training} (r_{iu} - p_u^T q_i)^2 + \lambda[\sum_x ||p_u||^2 + \sum_i ||q_i||^2] \tag{8}$$

The equations below were used to implement gradient descent, optimizing matrices P and Q. Matrices P and Q are initialized using SVD as discussed above.

$$p_{ku} \leftarrow p_{ku} + \eta * \bigtriangledown p_{ku} \tag{9}$$

$$q_{ik} \leftarrow q_{ik} + \eta * \bigtriangledown q_{ik} \tag{10}$$

$\bigtriangledown Q$ is the gradient of matrix Q (partial derivative). $q_{ik}$ is entry $k$ of row $q_i$ of matrix Q. The calculation for $\bigtriangledown P$ is repeated similarly below.

$$\bigtriangledown q_{ik} = 2(r_{iu} - p_u^T q_i)p_{ku} - 2\lambda q_{ik} \tag{11}$$

$$\bigtriangledown p_{ku} = 2(r_{iu} - p_u^T q_i)q_{ik} - 2\lambda p_{ku} \tag{12}$$

**Testing.** We ran multiple simulations of latent factor model with various hyperparameter settings. Of the many simulations ran, we let three run overnight and collected progress data at each epoch. Due to the significant amount of time necessary to run a simulation, we were unable to collect more data on various hyperparameter settings. The three aforementioned trials had settings as follows: Trial (1) $\eta = 0.006$, $\lambda = 0.06$, $k = 10$, $epochs = 49$, trial (2) $\eta = 0.006$, $\lambda = 0.02$, $k = 5$, $epochs = 17$ and trial (3) $\eta = 0.009$, $\lambda = 0.06$, $k = 10$, $epochs = 38$.

**Algorithm 3:** Latent Factor Model

```
Input: Sparse Training matrix $T$, Sparse Test matrix
       $T'$, epochs, $\eta$, $\lambda$, $k$
Result: Dense matrices $P$, $Q$
$U$, $\Sigma$, $V$ = SVD($T$);
$Q = U$; $P^T = \Sigma V$;
for epoch in epochs do
    Calculate RMSE on $T'$;
    Save $P$ and $Q$;
    for (movie, user) ϵ $T$ do
        $\bigtriangledown p_{ku} = 2(r_{iu} - p_u^T q_i)q_{ik} - 2\lambda p_{ku}$;
        $p_{ku} \leftarrow p_{ku} + \eta * \bigtriangledown p_{ku}$;
        $\bigtriangledown q_{ik} = 2(r_{iu} - p_u^T q_i)p_{ku} - 2\lambda q_{ik}$;
        $q_{ik} \leftarrow q_{ik} + \eta * \bigtriangledown q_{ik}$;
    end
end
for $(item, user)$ in $T'$ do
    summed_error = summed_error + $(r_{iu} - p_u^T q_i)$;
end
RMSE = $\sqrt{summed\_error/|T'|}$;
```

Each simulation ran for a varying number of epochs (also due to limited computational resources available at the time of each simulation). At each epoch of stochastic gradient descent we measured the RMSE of the test dataset and saved the matrices $P$ and $Q$ for potential later use. It should be noted that all of this testing was done using the randomly sampled dataset.

## 4 RESULTS.

### 4.1 Baseline Estimate.

The results of Baseline Estimate are merely here to benchmark the other methods. However, it already indicates that using the arbitrary split does not lead to good results relative to the randomly sampled dataset.

| | Arbitrary | Random |
|---|---|---|
| Test RMSE | 1.42524 | 1.23407 |

### 4.2 Item-Item Collaborative Filter.

Random sampling showed a significant improvement in RMSE using collaborative filtering over the baseline estimate. The arbitrarily split dataset RMSE had no gain since there was no information in the model specifically about most of the test samples. Collaborative filtering only works when information about users and items exist in the training set.

| | Arbitrary | Random |
|---|---|---|
| Test RMSE | 1.42524 | 1.12882 |

### 4.3 Latent Factor Model.

Optimizing the matrices $P$ and $Q$ was done using the randomly sampled dataset with 80% of the data as training. Each epoch had it's RMSE calculated using the remaining 20% of the data.

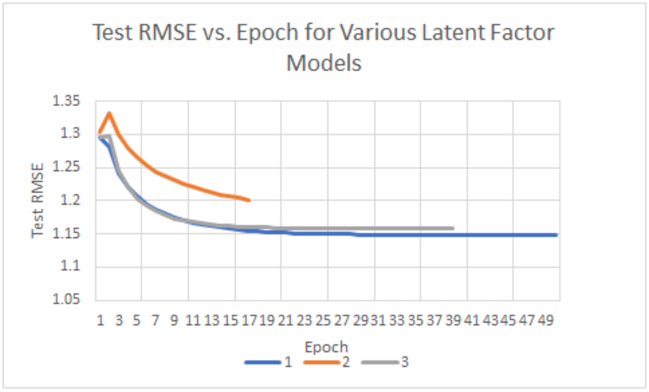

Fig. 1. Latent factor model gradient descent simulations with varying hyperparameters. Series (1) $\eta = 0.006$, $\lambda = 0.06$, $k = 10$, $epochs = 49$. Series (2) $\eta = 0.006$, $\lambda = 0.02$, $k = 5$, $epochs = 17$. Series (3) $\eta = 0.009$, $\lambda = 0.06$, $k = 10$, $epochs = 38$.

## 5 Discussion.

### 5.1 Data Prepossessing

The data was remapped from the original movie IDs to a new arbitrary ID to make the rest of the implementation simpler. Originally, the movie IDs did not seem to correlate to anything meaningful and were artificially inflating the indices of the data matrix. They also led to the inability to simply increment the index counter and properly retrieve information about the n-th movie. To remedy this, the IDs were mapped directly to the indices that reference them.

The process of remapping the data took about seven minutes on a Windows PC with 16GB RAM a 4.0GHz six-core i7e processor on an SSD. Running the random and arbitrary splits each took approximately 22 seconds to run on the same machine.

### 5.2 Baseline Estimate.

When reviewing the results from the baseline estimate, we observed some data that implied the arbitrarily split dataset has worse performance than the randomly sampled dataset. The RMSE for the arbitrary dataset split was approximately 0.2 worse than the randomly sampled one.

In terms of run time speed, this model was by far the quickest. The training and testing of each dataset takes only a few seconds, significantly faster than any of the other models tested.

### 5.3 Item-Item Collaborative Filter.

When calculating the RMSE of the item-item collaborative filtering model on the arbitrary dataset, we can see that it does not change between the baseline estimate and collaborative filtering due to the fact that collaborative filtering depends on items that a user has rated to deviate from the baseline estimate. Since the testing data is the entirety of the last 20% of the data, the only user that could have ratings in the training set is the one that gets split at the boundary between the sets. All other users in the test set have no ratings for the trained model, and thus nothing to improve the predicted rating with relative to the baseline estimate. This confirms the prior theory regarding the poor suitability of the arbitrary split dataset to the recommender problem—one of the reasons that we decided not to test the arbitrarily split dataset on the latent factor model.

Item-item collaborative filtering takes approximately two hours to calculate the test RMSE on the same machine as the one in baseline estimate. One way that this run time could be reduced is by getting rid of the similarity matrix and reformatting it to a similarity adjacency list. However, this is not feasible due to the scale of the dataset. The top twenty items would have to be stored for every user-rating pair in the test set.

### 5.4 Latent Factor Model.

In each simulation of latent factor stochastic gradient descent we used the randomly sampled dataset. This is due to two reasons: (1) Baseline estimate RMSE for randomly sampled data was significantly lower than for the arbitrarily split data and (2) we had limited computational resources and time to run simulations for both datasets. In addition, to the simulations in Fig. 1, we ran another ten or so trials for a much shorter duration to gather an intuition on appropriate hyperparameter settings. Obviously, choosing hyperparameters is a significant problem in many gradient descent based optimization problems and as such is outside the scope of this project. Therefore, our goal was only to select reasonable parameters to gather results on.

Due to the problem of optimal hyperparameter selection, we often ran into local optima very quickly in our training procedure, as can be seen in Fig 1. By epoch eleven, most of the RMSE gains began to flat-line, as one would expect when finding a local optima. The lowest RMSE was achieved on Test (1) with a value of 1.1485 on epoch 49. While there are methods to prevent and potentially resolve the local optimum issue, that is outside the scope of this paper and we leave it as future research.

The time it takes to optimize the matrices $P$ and $Q$, greatly depends on the hyperparameter selection. Depending on the choice of learning rate ($\eta$), the model would often overshoot or undershoot the optimal step, leading to a greater number of epochs needed to converge to the local optimum. In addition, the number of latent factors ($k$), had a rather extreme impact on the run time of the training procedure. It was experimentally noticed that when doubling the number of factors, the training time would roughly double as well. This led us to avoid selecting higher values of $k$, for the sake of computational time, even though a higher selection may have lead to a more optimal local minimum.

Comparing the results of latent factor model to the baseline estimate and item-item collaborative filtering we can see some interesting results. Latent factor model was marginally better than the baseline prediction on the randomly sampled dataset with an approximate improvement of 0.09 on the test RMSE. However, when comparing the latent factor model to item-item collaborative filtering it can be seen that latent factors obtained a worse RMSE by approximately 0.02. We attribute this worse performance due to the fact that we were unable to overcome the local optimum problem. In general, we note that latent factor models have been shown to have better prediction accuracy resulting in a lower RMSE than item-item collaborative filtering [1].

## 6 Conclusion.

We believe this work has provided some interesting insights into the analysis of anime recommendations through the implementation of three industry standard recommendation systems—(1) baseline estimate, (2) item-item collaborative filtering and (3) latent factor model. The results gathered show the expected improvement of item-item collaborative filtering and latent factor model over the baseline estimate, in addition to the difficulties of improving latent factor model over collaborative filtering. We believe that with improved hyperparameter and optimization parameter selection that latent factor models will perform significantly better than item-item collaborative filtering for anime recommendations. However, this is left as future work due to time constraints. Overall, we believe this work will be of significant interest to the anime community as a way improving current recommendation methods.

## 7 Future Work.

**Hyperparameter Selection.** Currently there is no guarantee that the hyperparameters for the latent factor model are optimal. There exists strategies such as grid search for the regularization parameter and line search for learning rate that can be used to improve the hyperparameter selection [9]. Both of these would provide for an interesting continuation of this work.

**Additional Datasets.** These models should be fully applicable to other data sets, such as the Movielens dataset. These may lead to future refinements in the models and patterns that aren't noticeable in the anime dataset. In addition, testing on new datasets will help determine each models relative performance on a variety of data, giving potentially elucidating information if different datasets perform better with certain models.

**Optimization.** One potential way to further improve the latent factor model predictions would be to change the optimization equation to include both biases for the user $b_u$ and the items $b_i$ in the error and regularization terms. This would allow the model to learn the appropriate parameters [1]. Another option to assist the optimization process is to modify the learning rule to include momentum, thereby speeding up the learning process and potentially avoiding local optimum.

**Multiclassifiers.** We believe that both hybrid and ensemble methods could be used to further increase performance. With the hybrid method a linear combination of the models used in this project, in addition to models not used here, could be used to generate better predictions than a single model alone. On the other hand, ensemble methods such as boosting and bagging could be applied here to further increase the performance [8].

## Appendix A
## Procedure to Run Program

1) Create three folders in the code directory named 'csv', 'matrices' and 'optimization'
2) Download the anime dataset from Kaggle: https://www.kaggle.com/CooperUnion/anime-recommendations-database
3) Add the 'anime.csv' and 'rating.csv' files to the 'csv' folder you just created
4) Run the 'RecommenderSystem.py' file and it will walk you through the process of running the program with an interactive dialogue.

## Acknowledgments

The authors would like to thank the TAs for taking the time to grade this project in addition to Dr. Li for allowing us to develop our own proposal for the class.

## References

[1] J. Leskovec, J. Ullman, A. Rakaraman, Mining Massive Datasets—Stanford University [Online] Available: https://www.youtube.com/playlist?list=PLLssT5z_DsK9JDLcT8T62VtzwyW9LNepV. Accessed: Nov. 25, 2017.

[2] J. Xu and Y. Yao and H. Tong and X. Tao and J. Lu, *HoORaYs: High-order Optimization of Rating Distance for Recommender Systems*, KDD 2017.

[3] J. Bobadilla , F. Ortega, A. Hernando, A. Gutirrez, *Recommender systems survey*, Elsevier 2013.

[4] B. Sarwar, G Karypis, J Konstan, J Riedl, *Item-Based Collaborative Filtering Recommendation Algorithms*, ACM 2001.

[5] A Toscherd, M. Jahrer, R. Bell, *The BigChaos Solution to the Netflix Grand Prize*.

[6] Y. Koren, R. Bell, C. Volinsky, *Matrix Factorization Techniques For Recommender Systems*, IEEE 2009.

[7] CooperUnion, Anime Recommendations Database—Kaggle. [online] Available at: https://www.kaggle.com/CooperUnion/anime-recommendations-database.

[8] Xristica, What is the difference between Bagging and Boosting?—Quantdare. [online] Available at: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/.

[9] R. Duda, P. Hart, D. Stork, Pattern Classification (2nd Edition), Wiley 2000.