User Taste Prediction

Annies Abduljaffar
Student
Stanford University
Stanford, CA 94305
anniesab@stanford.edu

Matt Vail
Student
Stanford University
Stanford, CA 94305
vailm@stanford.edu

Abstract

Recommendation engines are ubiquitous in today's online world. Recommendation systems provide users with personalized suggestions for products or services. There are two basic types of recommendation engines: content based and collaborative filters. These approaches can also be combined to form hybrid systems. Collaborative filtering methods can be further categorized into neighborhood models, also known as memory-based models, that rely on user-user or item-item similarity and matrix factorization models that directly profile users and items according to a set of learned latent factors [1]. In this implementation project, two generic models, seven neighborhood models, and three matrix factorization models were explored with several methods for reducing training time and improving performance on sparse matrices including optimizing parameters and preconditioning data. In addition, two new models were developed on top of the SVD and SVDpp matrix factorization models using momentum methods for optimization of model weights. In total performance against two datasets was compared across 14 models with 1,584 different combinations of input parameters and 4 different preconditioning procedures. Implementation details and further documentation can be found in the following GitHub repository: https://github.com/polymathnexus5/rec-engine-CS205L-W19

1 Introduction

Recommender systems are increasingly important for predicting users' preferences on a variety of content including movies, books, games, products, and more. These recommender systems are a specialized subset of information filtering systems, which predict a user's preference for a given item. While these systems have become ubiquitous with the rapid collection of massive data, they are still far from optimized. There are three main approaches to current recommender systems:

1. Collaborative Filtering - predictions based on the preferences of similar users (user-based) or attributes of similar items (item-based)
2. Content Based Filtering - predictions based on the preferences of the same user on similar content in the past
3. Hybrid Recommender Systems - a combination of collaborative and content based

Collaborative filtering systems can be further categorized into neighborhood, or "memory-based" systems and matrix factorization, or "model" based systems.

The neighborhood models compute an item to item or user to user similarity score in order to select a list of candidate recommendation items. User-based neighborhood models identify the k users that are most similar to you and recommend each of their favorite items while item-based neighborhood models identify the k items that are most similar to the user's favorite item and recommend those. This has the added benefit of being highly explainable as users are comfortable with the idea that, "if I liked Die Hard 1, it makes sense that I will also like Die Hard 2." Furthermore, it can help a user to make a decision within their recommendations

because you might like Die Hard but not be in the mood for Die Hard 2 and thus you can easily choose the film recommended to you because of your 5-star rating of Wedding Crashers. The model-based methods attempt to model a user preference matrix by learning a set of latent factors and model the item aspects matrix by learning a set of the same latent factors, forming two components: a user to factor preference matrix and a factor to item association matrix. In the case of a movie recommendation you can think of factors like genre, actors, and year of release as various latent factors. During the modeling process these factors are not known and sometimes researchers attempt to name these factors as a way to add visibility or interpretability into what was learned or modeled using the matrix factorization algorithm.

Some recent advances in modeling and deep learning have made new types of models possible as well as improving model parameterization and training using neural networks [2].

1.1 Motivation

The motivation for choosing this project was to gain an in-depth understanding of a real-world application of some of the core concepts learned in the CS205L. While evaluating various potential project ideas, Matrix Factorization and SVD stood out as favorite concepts that the team wanted to explore further, given the broad application and continued impact of these concepts in various fields of research. The aim of this project is to understand the performance of existing algorithms in recommendation systems and use that understanding to suggest and implement ideas to improve performance of these algorithms.

Interesting learning opportunities in this space are driven by the magnitude and the sparsity of the dataset. The complete matrix of possible user-movie ratings would be nearly 100GB, but less than 0.2% of all possible ratings were complete in the latest MovieLens dataset. This creates two very obvious challenges, first the complete user-matrix is far too large to be computed and held in memory, so iterative and stochastic approaches are required, and second the sparsity of the matrix results in a poorly conditioned problem with ample opportunity for truncation and roundoff error. While a movie ratings database is significantly large, there are other recommendation system examples that likely have even larger, sparser datasets. For instance, the number of items that are available for sale in an eCommerce site such as eBay, approximately 300M by some recent counts, will lead to an extremely sparse matrix.

These far reaching applications and direct application of continuous mathematical methods learned in class together with the availability of existing data, software libraries, and research are the primary motivations for exploring recommendation systems in this project.

1.2 Intended Outcome

To understand opportunities for improvement in recommendation systems, we designed experiments that search the hyperparameter space for each of the chosen candidate algorithms. In addition, we designed 4 pre-conditioning methods with respect to movies and users, and lastly, we implement momentum methods in 2 existing recommendation algorithms to develop new models. We expected to find a large effect on model performance with varying model parameters, especially training iterations and learning rate as those are often the most widely discussed and debated. In addition, we expected the preconditioning methods to eliminate some of the above-mentioned challenges relating to huge sparse data sets. Lastly, we anticipated that implementing the momentum methods would decrease the required training iterations and improve model performance given the same number of training iterations.

2 Related Work

Implementing and optimizing performance of recommendation systems caught the fancy of the academic and research community when Netflix announced a contest in 2006 [3]. Multiple teams participated to work on the competition and after nearly three years the prize money of $1M was awarded to the team “Belkor Pragmatic Chaos” [4]. The solution implemented by the winning team is similar to the work discussed in this paper in many respects and different in others. Similar to the Belkor team’s method, our implementation of Matrix factorization uses the baseline predictors, which are biases introduced by some users having a tendency to rate movies generously and some other users having a tendency to rate movies critically. Also, there are strong item biases introduced by how popular a given movie is and based on this perceived popularity users are likely to review certain movies higher than what their original

rating would have been. However, unlike the Belkor team's method we do not model these baseline predictors as a function of time (Matrix Factorization with Temporal Dynamics). This project uses the work done by Nathan Hug in implementing the Surprise Python package as a foundation layer for how to implement the popular "SVD" model and others in an efficient manner using Alternating Least Squares (ALS) and Stochastic Gradient Descent (SGD). In addition to traditional machine learning techniques, other groups are approaching recommendation systems from a deep learning. eBay researchers are implementing a deep learning-based method to predict the user to item preference as seen in the recent publication [2]. Combining traditional collaborative filtering and content-based recommendation systems with deep learning methods for optimizing model parameters including number of latent factors is a natural extension of this work but beyond the scope of this project. Since the close of the Netflix Price Competition there has been significant additional work in the field of recommendation systems including the combination of neighborhood and matrix factorization models [1] improvements to the time aspects of the winning model [5] applications of conjugate gradient and other optimization alternatives to SGD [6] approaches with significant preconditioning methods [7] and more.

3 Data and Software Libraries

3.1 Data

The data analyzed in this project are the MovieLens datasets from the GroupLens research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems [4]. This project explores the MovieLens latest datasets, both "Small," which contains 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users and was last updated in September of 2008 and the "Full," dataset, which contains 27m ratings and 1.1m tag applications applied to 58,000 movies by 280,000 users along with tag genome data with 14m relevance scores across 1,100 tags. The "Full" dataset was also last updated in September of 2008. Analysis of both datasets focuses on the 'ratings.csv' file, which provides 1 rating per row as a comma separated list of userId (1 – 280,000), movieId (1 – 58,000), rating (1 – 5), and timestamp. The timestamp was dropped for this analysis. The "Small" dataset is 3.3MB in total while the "Full" dataset is 1.23GB, the 'ratings.csv' file alone is 759.2MB.

3.2 Software Libraries

This project was completed in Python 3.7.2_1 and Cython 0.29.6 using standard libraries:

1. Os – miscellaneous operating system interfaces
2. SciPy – user friendly and efficient numerical routines including linear algebra methods such as dot product, vector norms, minimize, optimize, etc.
3. Numpy – scientific computing package
4. Pandas – easy-to-use data structures and analysis package
5. Matplotlib – visualization library
6. Seaborn – visualization library built on top of Matplotlib
7. Surprise – modeling frameworks including a Cython implementation of the "SVD" and "SVDpp" algorithms, parameter search, and cross validation.

The following modules were written from scratch for this project:

1. 'load_data.py' – provides methods to load MovieLens data and filter based on number of ratings per user and per movie
2. 'MF_SGD_momentum.py' – extends the Surprise SVD and SVDpp algorithms to include momentum methods for optimization of model parameters
3. 'benchmark.py' – main project module; produces all of the relevant experimental results
4. 'benchmark_latest.py' – the same as 'benchmark.py' but for the latest, 27m rating dataset, produces all of the relevant experimental results

Code was produced and executed in the open-source Atom text editor on a MacBook Pro.

4 Methods

4.1 Architecture

The experimental investigation for this project was completed using the open source Python and Cython languages. The work leveraged several standard Python libraries, with a particular dependence on Numpy for the linear algebra and scientific computing and the Surprise library for implementations of standard recommendation system algorithms including both neighborhood and matrix factorization models. The core of the Surprise library are the prediction algorithms, with supporting Dataset, BaseSearchCV, and Reader classes:

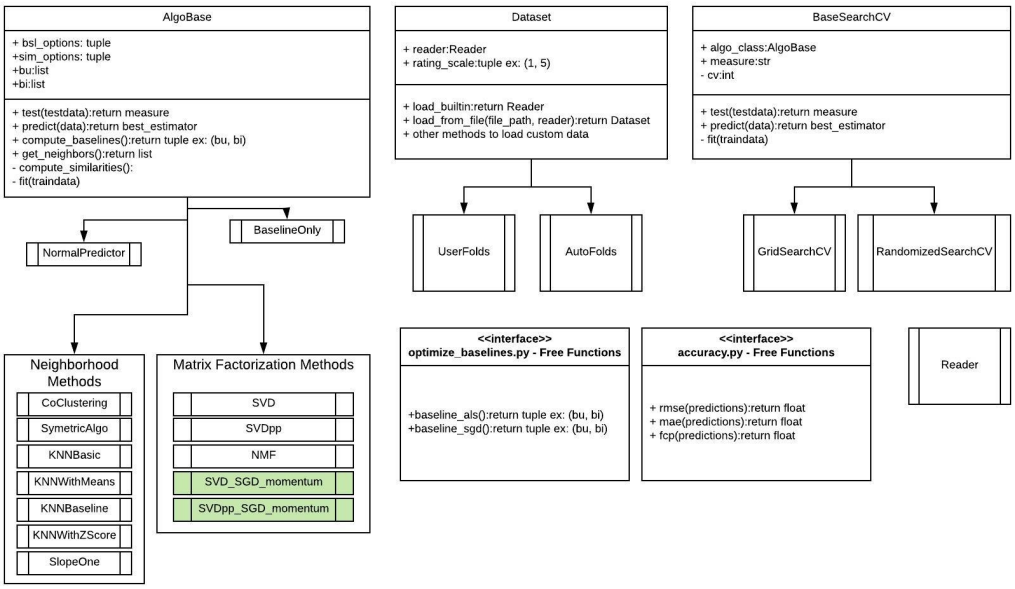

Figure 1: Surprise Library partial class diagram. Custom written classes in green.

In order to implement the SVD_SGD_momentum and SVDpp_SGD_momentum algorithms new classes were built for each. Lastly, data loading and benchmarking modules were also built from scratch in order to pipeline and process the data and save the output.

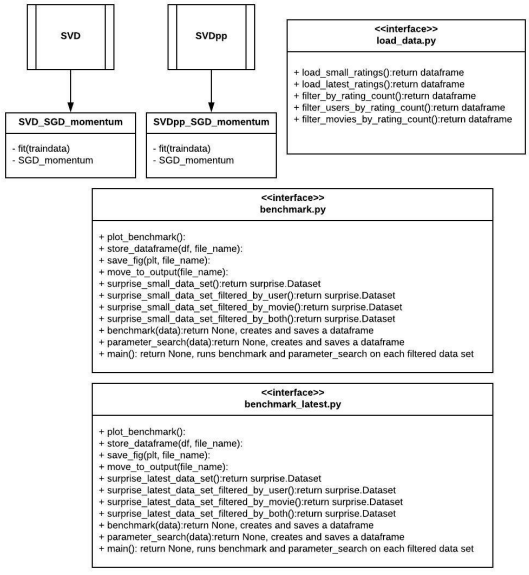

Figure 2: Custom software modules

4.2 Data Format

The data consists of ratings from m = 280,000 users and n = 58,000 movies:

$$UserId = \mathrm{U} \in \mathbb{R}^{280{,}000}, \qquad MovieId = M \in \mathbb{R}^{58{,}000}$$

Each user can be represented by a row vector of ratings of length equal to the total number of movies where each entry is a rating:

$$U_{1,\ 1} = user\ 1's\ rating\ on\ movie\ 1$$

$$U_{7,\ 7{,}328} = user\ 7's\ rating\ on\ movie\ 7{,}328$$

$$User_u = \vec{U}_1 = [R_{1,\ 1} \quad R_{1,\ 2} \quad \cdots \quad R_{1,\ 58{,}000}]$$

Each movie can similarly be represented by a column vector of ratings of length equal to the total number of users where each entry is a of rating:

$$M_{1,\ 1} = user\ 1's\ rating\ on\ movie\ 1$$

$$M_{7{,}328,\ 7} = user\ 7{,}328's\ rating\ on\ movie\ 7$$

$$Movie_i = \vec{M}_1 = [R_{1,\ 1} \quad R_{1,\ 2} \quad \cdots \quad R_{1,\ 280{,}000}]^T$$

The complete ratings matrix could be constructed represented as a matrix with each column representing a movie and each row representing a user.

$$\begin{aligned} Complete\ Ratings\ Matrix, R = rating\ and\ timestamp\ for\ each\ of\ User\ Ids\ x\ Movie\ Ids \\ = \mathbb{R}^{280{,}000 \mathrm{x} 58{,}000 x 2} \\ = 280{,}000 * (4\ \text{bytes}) + 58{,}000 * (2\ \text{bytes}) + 280{,}000 * 58{,}000 \\ * (2 + 4)\ bytes = 97GB \end{aligned}$$

$$R = \begin{bmatrix} \vec{U}_1 \\ \vdots \\ \vec{U}_{138{,}493} \end{bmatrix} = \begin{bmatrix} \vec{M}_1 & \cdots & \vec{M}_{27{,}278} \end{bmatrix} = \begin{bmatrix} R_{1,\ 1} & \cdots & R_{1,\ 27{,}278} \\ \vdots & \ddots & \vdots \\ R_{138{,}493,\ 1} & \cdots & R_{138{,}493,\ 27{,}278} \end{bmatrix}$$

However, the complete ratings matrix is much too large to hold in local machine memory (~6x typical local machine memory). Luckily, the actual ratings data is in the form of a 27m x 4 matrix:

$$\begin{aligned} Actual\ Ratings\ \in\ & \mathbb{R}^{27{,}000{,}000 x 4}(UserId, MovieId, Rating, Timestamp) \\ & = 27{,}000{,}000 * (4 + 2 + 2 + 4)\ bytes = 324\ MB(actualy = 759.2) \end{aligned}$$

The complete ratings matrix is extremely sparse; only .166% of all entries are non-zero.

$$\frac{Actual\ Ratings}{Complete\ Ratings\ Matrix} = \frac{27{,}000{,}000}{280{,}000 * 58{,}000} = .166\%$$

Therefore, the complete ratings matrix should not be created and instead calculations should be done stochastically from individual ratings.

4.3 Algorithms

4.3.1 Generic Models

This project investigated 2 generic models, the normal predictor model and the baseline only model. The normal predictor model generates predicted ratings from a normal distribution with mean and standard deviation calculated from the training data using a Maximum Likelihood Estimate [5]:

Equation 1: Normal Predictor Model

$$\hat{r}_{u,m} \epsilon \mathrm{N}(\hat{\mu}, \hat{\sigma}), where: \hat{\mu} = \frac{1}{|R_{train}|} \sum_{r_{ui} \epsilon R_{train}} r_{ui}\,, \qquad \hat{\sigma} = \sqrt{\sum_{r_{ui} \epsilon R_{train}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{train}|}}$$

The baseline only model first establishes baselines, or offsets, for each user and each item, and then generates predicted ratings as the sum of the global mean and those baselines:

Equation 2: Baseline Only Model

$$\hat{r}_{um} = \mu + b_u + b_m$$

Baselines can be estimated either using the Alternating Least Squares (ALS) method or Stochastic Gradient Descent (SGD). In both cases the objective function to be minimized is:

Equation 3: Baseline Only Model Objective Function

$$objective\ function = \sum_{(u,m)} \left((r_{um} - \mu + b_u + b_m)^2 + \lambda (b_u^2 + b_m^2) \right)$$

Where λ is a regularization parameter which can be customized for each baseline. In this project, baselines were estimated using ALS with a single regularization parameter unless otherwise noted. Ratings data tends to have very large user and item effects because there are systematic tendencies for some users to be more generous or critical than others and for some items to receive higher ratings than others [6]. For this reason, baselines are used as subcomponents in several of the other algorithms and all algorithms perform better when baselines are taken into account than when they are neglected.

4.3.2 Neighborhood Models

This project investigated 7 neighborhood models including 4 variations of the K-Nearest Neighbor Algorithm. All of the neighborhood models utilize some version of a similarity measure, which measures how similar (correlated) two users or two items are. In the case where the similarity measure is between users, those models are called, "user-based" and in the case where the similarity measure is between movies, those models are called, "item-based." In either case, the similarity measures can be in 4 forms: cosine, mean squared difference (MSD), Pearson correlation, and Pearson correlation with baseline. In this project, MSD was used as the similarity measure and models were "user-based" unless otherwise noted:

Equation 4: Mean Squared Difference (MSD) similarity measure between two users u and v

$$Mean\ Squared\ Difference\ (MSD)\ between\ users\ u\ and\ v = \frac{1}{|I_{uv}|} \sum_{i \in I_{u,v}} \left(r_{u,i} - r_{v,i} \right)^2$$

Each neighborhood is based on this basic concept of similarity and utilizes the similarity measure to generate predicted ratings.

Equation 5: KNN Basic Model using user-based MSD as the similarity measure

$$\hat{r}_{um} = \mu + \frac{\sum_{v \in N_i^k} MSD(u,v) \cdot r_{v,i}}{\sum_{v \in N_i^k} MSD(u,v)}$$

For neighborhood models, excluding there are no model parameters to be learned as the prediction is based solely on vector math (excluding learning the baseline estimates for those models that utilize a baseline).

4.3.3 Matrix Factorization Models

This project investigated 3 matrix factorization models: the "SVD" algorithm (no actual singular value decomposition is done but the algorithm was inspired by the concept of SVD), "SVDpp," which is an extension of the SVD algorithm taking into account implicit feedback, and "NMF" (for non-negative matrix factorization). In addition, two new models were developed on top of the SVD and SVDpp matrix factorization models using momentum methods for optimization of model weights.

Matrix factorization models are also known as latent factor models as they decompose the user-movie matrix into two matrices of "latent factors," which are then used for prediction an analysis [7].

4.3.3.1 SVD Model

The SVD algorithm is an extension of Probabilistic Matrix Factorization [8] popularized by Simon Funk during the Netflix Prize Competition in 2006. As previously mentioned, the user-movie matrix is very sparse (i.e. lots of missing values), and conventional SVD is undefined for incomplete matrices. One possible solution is to impute the missing values based on some distribution, but this is extremely computationally expensive and prone to overfitting; recall that if the complete user-movie matrix were to be completed, less than 0.2% or 2 in 1,000 entries would be non-zero.

Decomposing the user-movie matrix into a product of two matrices maps users and movies both to a f-dimensional space where f is the number of factors chosen. The "latent space" explains movies in terms of aspects inferred from the user-movie matrix. Each user can thus be represented by a f-dimensional vector where each component is that user's propensity or liking for a particular factor. For example, with a 3-factor decomposition, the factors might be "Action," "Drama," "Comedy" and for a particular user, Matt, who prefers comedies, their user-factor vector could be [2, 1, 10]. Similarly, each movie can be represented by a f-dimensional vector where each component is how much that movie can be described by that aspect. With more factors, the particular aspects are more and more unpredictable until they are aspects of the movie that we cannot imagine. This is helpful as it provides ease of explainability for why a particular movie is represented to a given user [7].

The dot product of the user-factor and factor-item vectors gives a scalar score for how much the item has what the user likes or vice-versa how much the user likes what the item has, and the SVD algorithm generates predicted ratings using baseline estimates and that dot product:

Equation 6: SVD Model

$$\hat{r}_{um} = \mu + b_u + b_m + p_u^T q_m$$

The model parameters are learned by minimizing the following objective function:

Equation 7: SVD Model Objective Function

$$objective\ function = \sum_{(u,m)} \left((r_{um} - \mu + b_u + b_m + p_u^T q_m)^2 + \lambda(b_u^2 + b_m^2 + \|p_u\|^2 + \|q_m\|^2)\right)$$

Where λ is again a regularization parameter which can be customized for each baseline and latent factor matrix. Again these parameters can be learned using ALS or SGD, and in this project SGD was used to learn model parameters for both the SVD model and the SVDpp model.

4.3.3.2 SVDpp Model

The SVDpp algorithm is an extension of the SVD, which takes into account implicit user feedback. Specifically, SVDpp accounts for movies that a user has rated explicitly and generates a predicted rating as follows:

Equation 8: SVDpp Model

$$\hat{r}_{um} = \mu + b_u + b_m + q_m^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right)$$

Where the set $|I_u|$ is the set of all movies that have been rated by the user (regardless of rating) and y_j is an addition factor vector that characterizes only those movies that were rated by the user essentially augmenting the "user profile" with implicit (rated or not rated) feedback. This implicit feedback concept can be directly extended to include other forms of implicit feedback such as any data one might gain from user cookies, home address, occupation, gender, etc. Implicit user feedback is often tightly coupled with content-based models and together they attempt to create a more holistic understanding of the user and therefore more accurate predictions.

Again, the parameters of this model are learned by minimizing the following objective function using ALS or SGS or any other appropriate optimization method:

Equation 9: SVDpp Model Objective Function

$$objective\ function = \sum_{(u,m)} \left(\left(r_{um} - \mu + b_u + b_m + q_m^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right) \right)^2 + \lambda (b_u^2 + b_m^2 + \|p_u\|^2 + \|q_m\|^2 + \|I_u\|^2) \right)$$

4.3.3.4 Non-Negative Matrix Factorization (NMF) Model

The NMF algorithm is nearly identical to the SVD algorithm except that during the SGD process, the NMF algorithm uses a different update rule for the user and movie factor matrices in order to force the factors to be always positive:

Equation 10: NMF SGD Update Rule, Ensures That Factors Are Always Positive

$$p_{uf} \leftarrow p_{uf} \cdot \frac{\sum_{i \epsilon I_u} q_{if} \cdot r_{ui}}{\sum_{i \epsilon I_u} q_{if} \cdot \hat{r}_{ui} + \lambda_u |I_u| p_{uf}}$$

4.3.4 Gradient Descent

Gradient descent is a popular first-order iterative optimization algorithm for finding the minimum of a function [9]. It is popular for its simplicity and ease of implementation. In order to find a local minimum of a given function, one simply takes a step proportional to the "learning rate" in the direction opposite the gradient:

Equation 11: Generic Gradient Descent Update Rule

$$\vec{a}_{k+1} = \vec{a}_k - \eta \nabla F(\vec{a})$$

Where η is the learning rate (scalar ~0.05 is generally a good starting spot), $\vec{a}$ is a vector of model parameters (or weights) being optimized, $F(\vec{a})$ is the objective function, and $\nabla F(\vec{a})$ is the gradient of the objective function.

In "standard" or "batch" gradient descent, all model parameters are updated at one time. In stochastic gradient descent, the update rule stays the same except that each parameter is updated individually. This is particularly useful given the format of the ratings data because we can implement stochastic gradient descent without creating any complete user-movie matrix and can update our model parameters with each new rating.

Both gradient descent and stochastic gradient descent are slow near the minimum of poorly conditioned problems.

4.3.5 Matrix Factorization Models Utilizing Momentum Methods for Optimization

Stochastic gradient descent with momentum "remembers" the prior update for each parameter and calculates the current update as a linear combination of the current gradient and the prior update:

$$v_0 = 0; \quad v_k = \gamma v_{k-1} + (1 - \gamma) \nabla F(a); \quad \vec{a}_{k+1} = \vec{a}_k + \eta v_k$$

Where γ is the momentum parameter. Note that setting $\gamma = 0$ results in standard SGD.

Momentum methods like this one help dampen the zig-zag behavior of SGD and typically result in faster convergence.

This momentum method was implemented for both the SVD and SVDpp models resulting in the SVD_SGD_momentum and SVDpp_SGD_momentum models with the expectation that it would decrease training time without sacrificing any performance.

4.4 Experimental Trials

Performance was measured in terms of Root Mean Squared Error (RMSE), Mean Average Error (MAE) and Fraction of Concordant Pairs (FCP) for all 14 models on two datasets ("Small" and "Full") with 4 different preconditioning procedures. In addition, 1,584 different combinations of input parameters were tested on the SVD, SVD_SGD_momentum and SVDpp_SGD_momentum models each with the same 4 preconditioning procedures.

Equation 12: Root Mean Squared Error (RMSE) was used to measure performance in all experiments

$$RMSE = \sqrt{\frac{1}{|\hat{R}|}\sum_{\hat{r}_{ui}\in\hat{R}}(r_{ui}-\hat{r}_{ui})^2}$$

4.4.1 Preconditioning Procedures

4 different preconditioning procedures were applied:

1. No filtering
2. Keep the top 1% of users by number of ratings
3. Keep the top 1% of movies by number of ratings
4. Keep only the top 1% of users and the top 1% of movies

4.4.2 Input Parameters

Input parameters were varied across 5 different inputs:

1. Iterations – [1, 2, 3, 4, 5, 10, 20, 40, 60, 80, 100]
2. Learning Rate – [.002, .005, 0.01, 0.02]
3. Regularization Parameter – [0.2, 0.6, 1.0]
4. Initial Mean – [0.0, 0.5, 1.0]
5. Initial Standard Deviation – [0.0, 0.1, 0.5, 1.0]

Note that a single learning rate and a single regularization parameter were used for all model parameters within a single trial.

The total parameter search space then is 11*4*3*3*4 = 1,584 across 3 algorithms, 2 data sets, and 4 preconditioning methods resulting in over 25,000 combinations.

5 Experimental Results and Analysis

It was our expectation that the matrix factorization models would perform better in terms of RMSE based upon prior research [10] and we also anticipated that the addition of the momentum methods would reduce training time by minimizing the "zig zag" behavior of the SGD algorithm, achieving a similar RMSE performance in fewer training iterations and better performance over the same number of training steps.

Note that unless otherwise mentioned, RMSE shown below is the average across the RMSE values holding the parameter of interest constant for each data point. For example, the RMSE values presented in Figure 5 are averages across all values of learning rate, regularization parameter, initial mean, and initial standard deviation while holding the training iterations constant for each data point. Figures 3 below shows algorithm performance in raw RMSE across both data sets and all 4 preconditioning methods. As expected, the Normal (random) predictor performs the worst, with all the other algorithms performing in a range from 0.75 to 1.05. Relative performance within algorithms is very consistent across algorithms, that is, all algorithms performed best on the unfiltered data and worst on either movie filtered or user filtered data. Overall the SVDpp and SVDpp momentum algorithms performed best, as was expected from the prior research. The NMF algorithm performed poorly, which was also to be

expected given that it does not take into account baselines.

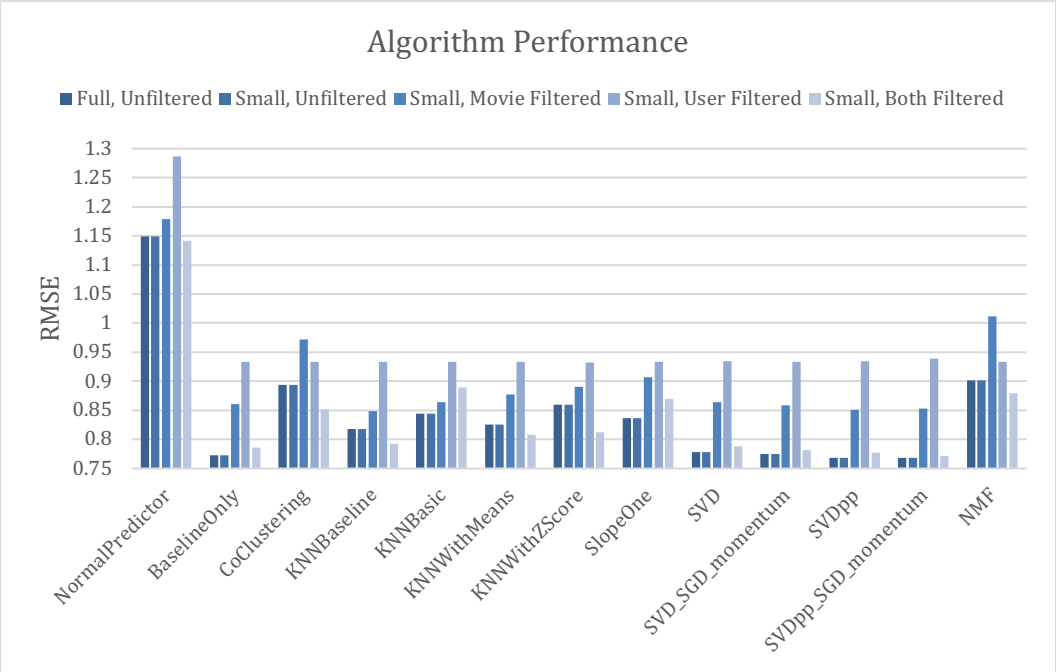

Figure 3: Algorithm Performance on Both Datasets and Four Preconditioning Methods

Figure 4 shows how well each algorithm outperformed the simple random predictor.

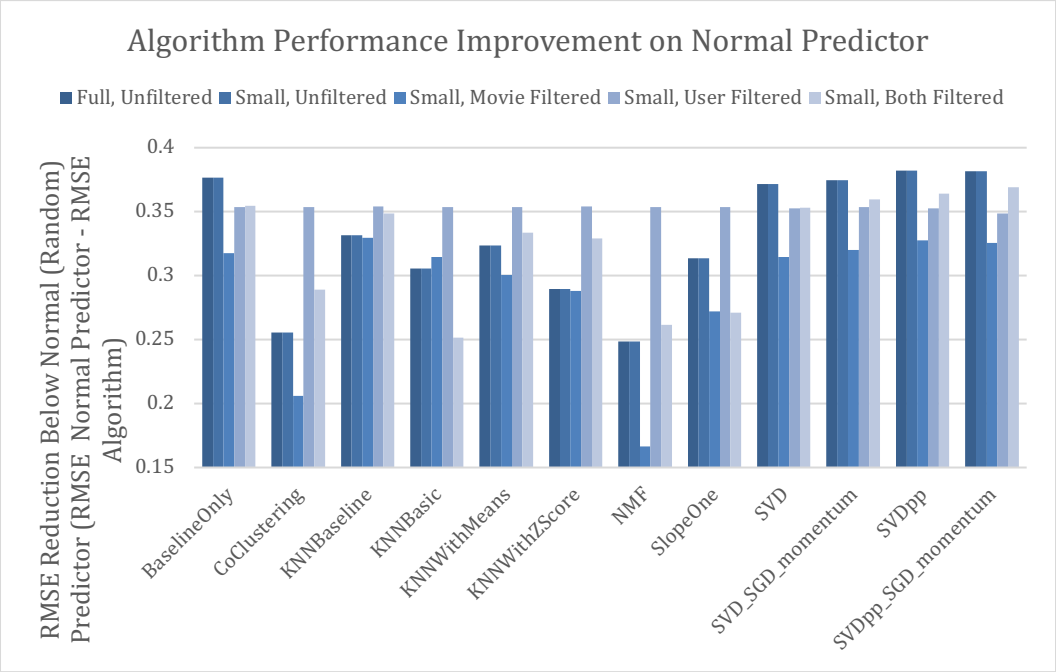

Figure 4: Algorithm Performance Improvement on Normal Predictor Model on Both Datasets and Four Preconditioning Methods

In Figure 4, a higher value means reduced error below the simple random predictor and illustrates how tightly clumped performance is with the SVD, SVDpp, and their momentum counterparts narrowly outperforming the other models. It is interesting to note that the Baseline Only model outperforms every model that does not take into account baselines (CoClustering, KNNBasic, KNNWithMeans, KNNWithZScore and SlopeOne). As noted

earlier, baselines are very important for consumer ratings data, so we would expect baselines to be very important, but it is still telling that even with the significantly more sophisticated models, if they do not factor in baselines, they fail to perform. We note also that the performance of SVDpp over the simple SVD is at most a 0.02 reduction in RMSE. Given the importance of implicit feedback in the research dialogue, it's surprising that including it barely affects performance.

In addition to the performance, we were interested in training and test time across algorithms. Figure 5 and 6 show training time for each algorithm.

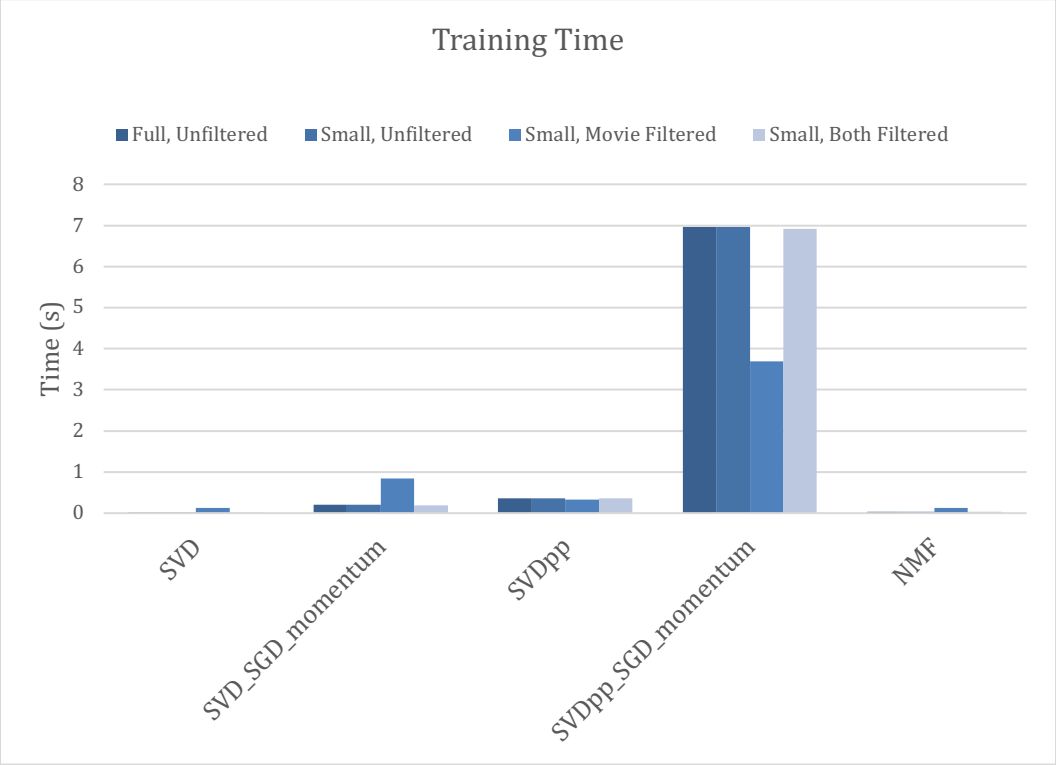

Figure 5: Training Time by Algorithm

Note that all algorithms were trained using the default 20 iterations of SGD or SGD_momentum optimization. This chart then represents the approximate time to run the optimization algorithm for any given model. Models without SGD optimization components were omitted for clarity (they all have Training Time = 0). Training time across the algorithms was relatively constant. Interestingly, training time on the SVDpp momentum version was roughly 10x longer than the next longest training time. This is because of the added computation from the momentum component within the implicit user factor matrix.

In Figure 6 below it is clear that the test time for the neighborhood models is significantly longer than for the matrix factorization methods. The reason for this is that the neighborhood models must calculate user-user similarity against every other user for every test case, which is O(n*m) where n is the number of users in the test set and m is the number of users in the entire data set. The matrix factorization models only need to do one simple predictor calculation for each test case, which runs in O(n). This is a major advantage of matrix factorization models for new users of the system.

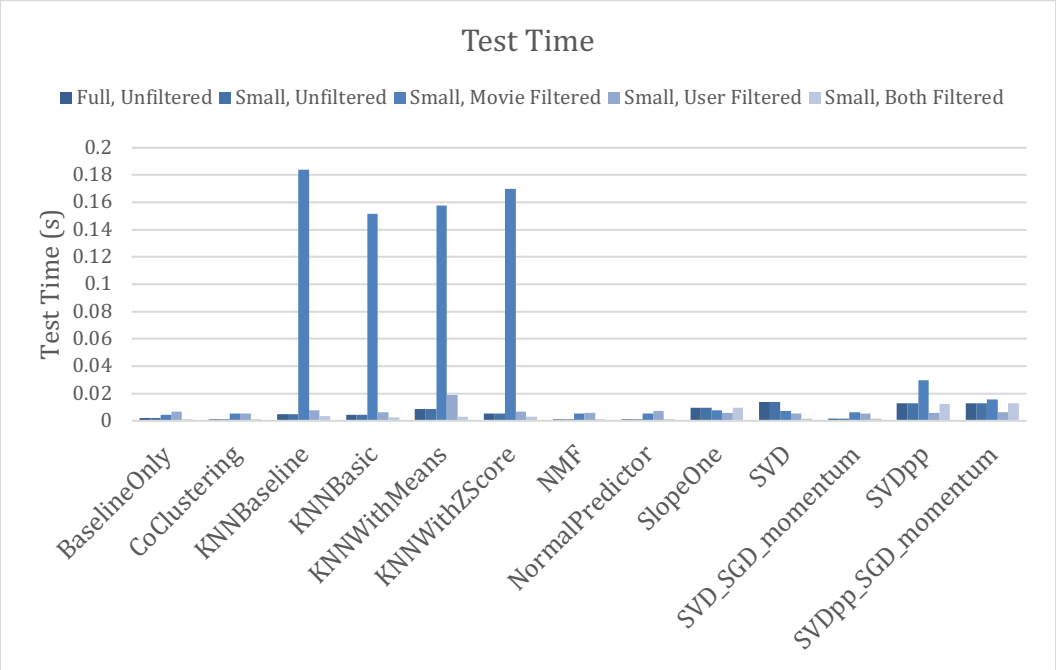

Figure 6: Test Time by Algorithm

Figure 7 below shows RMSE as a function of training iterations for the SVD, SVD_SGD_momentum, and SVDpp_SGD_momentum algorithms.

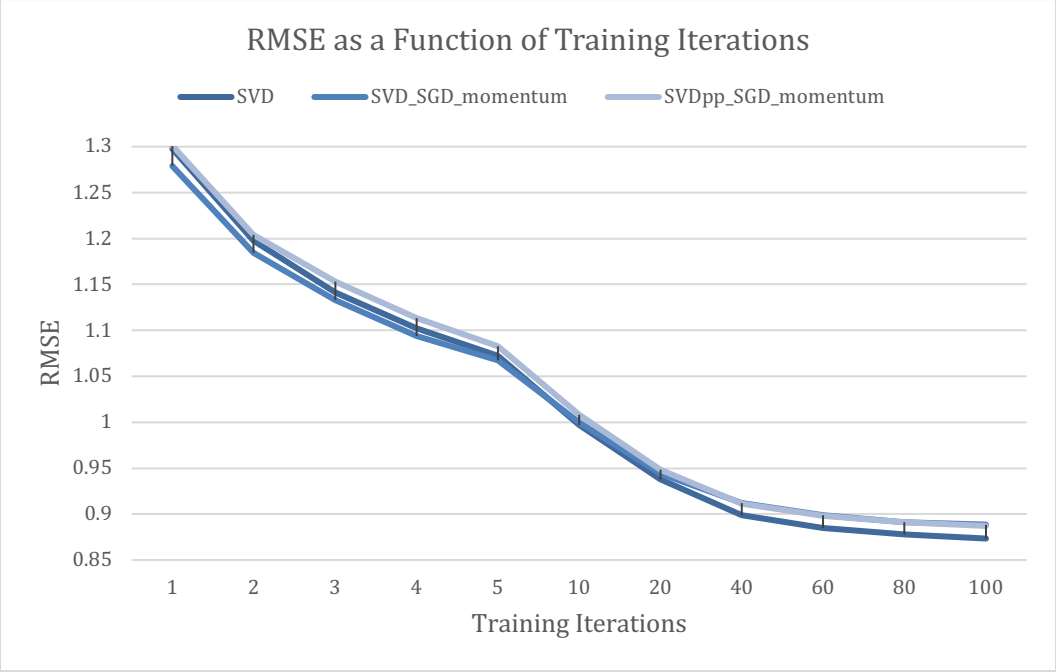

Figure 7: RMSE as a Function of Training Iterations

We can see that the error rapidly reduces from 1 to 40 training iterations at which point error begins to level out. All three models perform very similarly over the training iterations, which was not anticipated. We expected the two momentum methods to train more quickly that the standard SGD method, but all 3 algorithms see similar error reduction over each iteration.

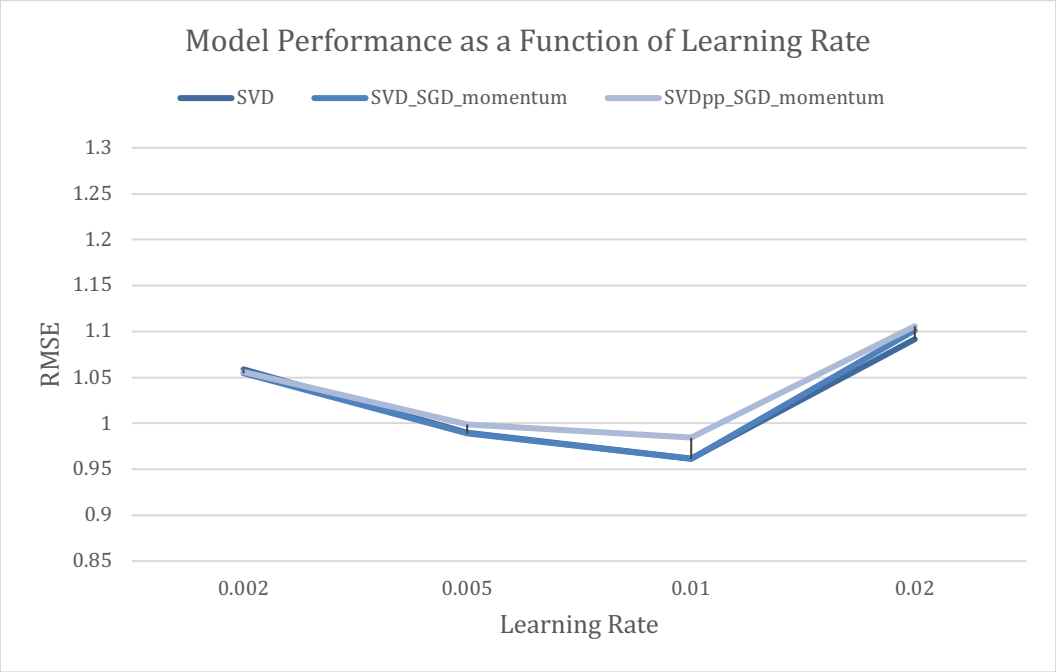

Figure 8: Model Performance as a Function of Learning Rate

Figure 8 shows model performance as a function of learning rate and seems to suggest that learning rate does not have a large impact on model performance. However, further investigation into this parameter, and all parameters, is necessary in order to truly understand the interactions between learning rate, training iterations, and other parameters.

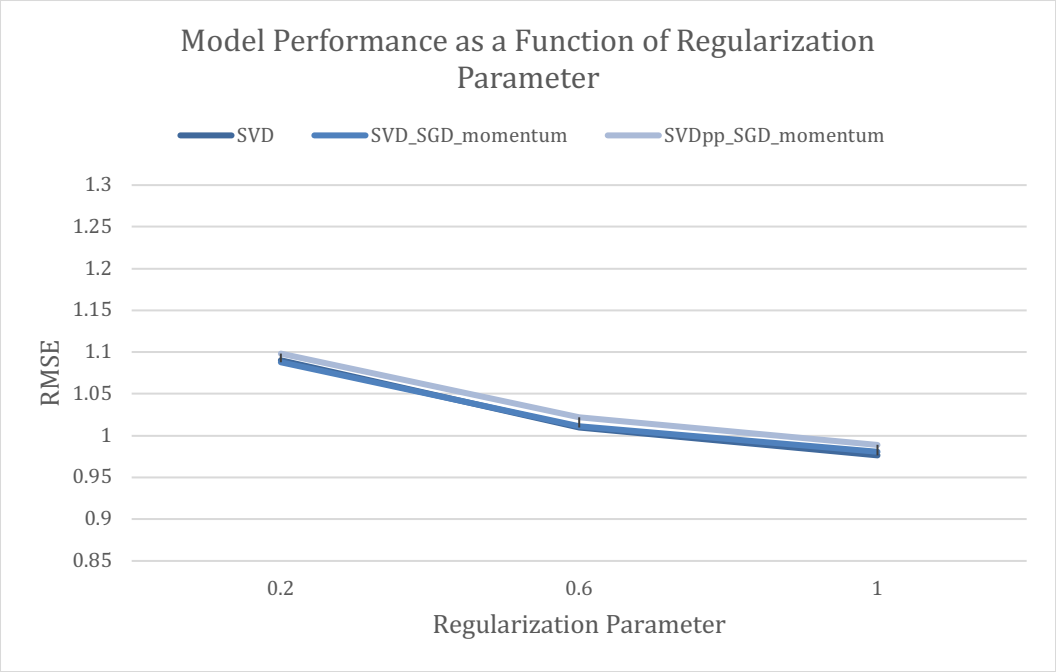

Figure 9: Model Performance as a Function of Regularization Parameter

Model performance appears to improve as the regularization parameter increases, suggesting that we may be over fitting in most of our model scenarios.

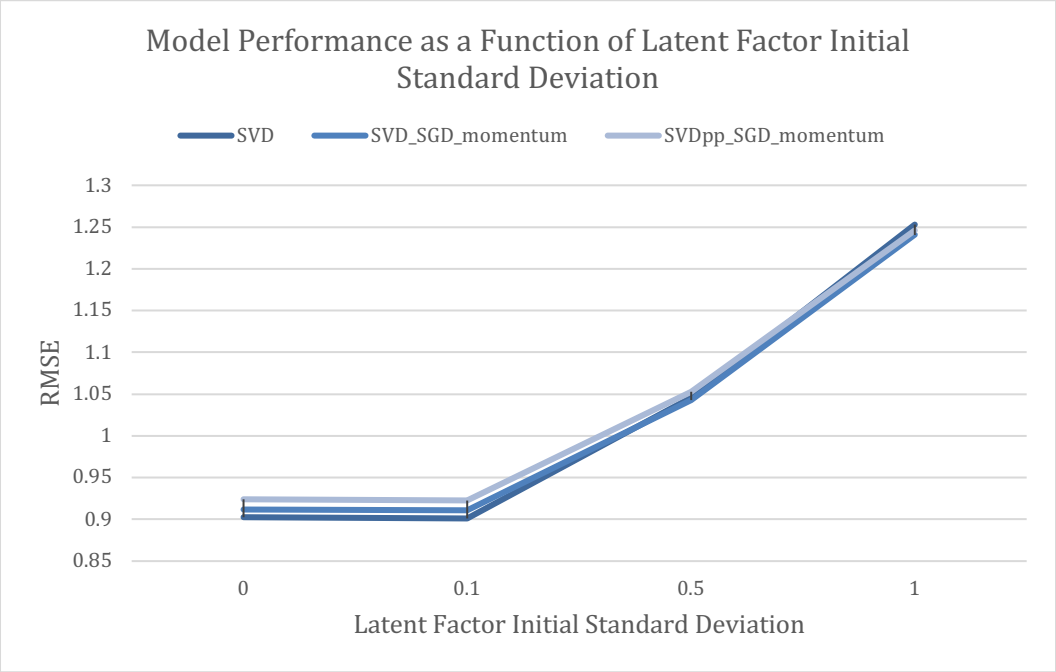

Figure 10: Model Performance as a Function of Latent Factor Initial Standard Deviation

Intuitively initiating the model with a significant bias towards negative ratings (initial mean = 0) seems like it would decrease performance and given that we can rapidly determine the mean and standard deviation of the training data, initializing the latent factors to the parameters of the actual ratings seems like a reasonable choice, however Figure 10 clearly shows that performance worsens as the initial standard deviation is increased and Figure 11 supports the same conclusion for the initial mean.

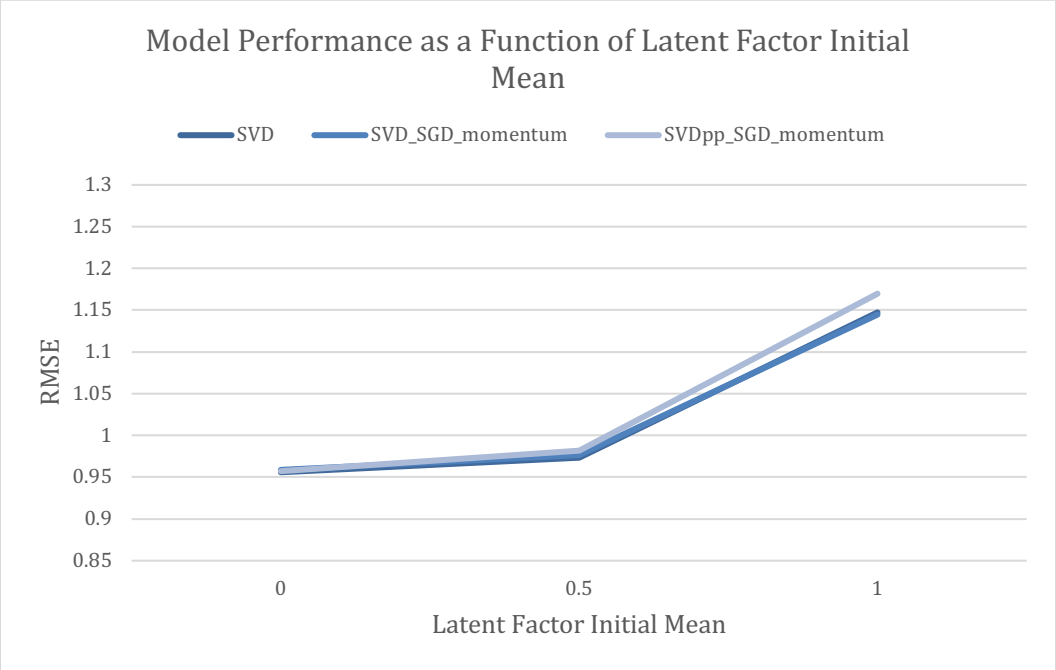

Figure 11: Model Performance as a Function of Latent Factor Initial Mean

Of the 14 models and the 25,345 algorithm-dataset-preconditioning-parameter combinations

that were completed, the top performance on every data set and preconditioning method was one of the four SVD based models. The SVDpp_SGD_momentum method with a learning rate of 0.02, regularization of 0.2, initial mean of 0, and initial standard deviation of 0.5 trained on 80 iterations was the number one performer overall, with an RMSE of .7437 on both the Small and Full unfiltered datasets.

6 Conclusion

This work evaluated 14 different recommendation models including neighborhood and matrix factorization models. We looked at 25,345 unique parameter combinations across 5 different model parameters. We found that model performance as measured by RMSE levels off after approximately 40 training iterations and the improvement between 60 and 100 iterations is negligible. The optimal learning rate appeared to be 0.01 but it was highly dependent on the other model parameters and performance tended to increase as the regularization parameter was increased, suggesting that most of the models are overfitting the data to some degree. We also found that matrix factorization models are very sensitive to initial latent factor mean and standard deviation values. Most of the outcomes were consistent regardless of the size of the dataset (Small or Full) or the preconditioning (Filtered or Unfiltered)

In addition, we implement a more sophisticated second order stochastic gradient descent algorithm with a momentum component to establish 2 new recommendation algorithms. We expected these algorithms to train faster in fewer iterations and have a lower overall error given the same training iterations. Instead we saw that model performance was almost identical, although training time was increased for the SVDpp_SGD_momentum model.

Even though there has been a great deal of research and commercialization of recommender systems, there are still myriad opportunities for future work in recommendation systems especially with a strong foundation in continuous mathematics and linear algebra. One interesting future direction is implementing other optimization methods including variations of Conjugate Gradient Descent on subsets of the user-movie matrix, other momentum methods such as Nesterov's method, Adagrad, Adadelta, RMSprop, AdaMax, etc. The challenge will be adapting each of these to a stochastic approach that avoids computing the entire user-movie matrix.

Another direction for future work is a more in-depth exploration and development of time-based methods that take into account the timestamp and the release date. Koren et al. have done some work on this [10] [11] and analysis is available in the "Recommender System Handbook" [7], but more exploration is needed to understand the full potential of models that incorporate time.

Additional work on combined methods such as those developed by Koren [1] is also interesting. Combined neighborhood-factorization models as well as combined collaborative filter-time based models and combinations of all of the above with increasingly available content and content models and more extensive utilization of deep neural networks [14] is the ultimate goal for recommender systems. In a world where our every digital move is tracked and recorded, our social networks and preferences are meticulously catalogued, and more and more our physical state, attention, location, and activity is logged it is reasonable to expect that a recommendation system could be built to provide suggestions not just for movies and restaurants or more generally for products and services but in fact for all aspects of our daily lives. With the vast troves of data collected on our lives, our DNA, and our relationships, we should expect ultra-high-quality suggestions for where to work, which degrees to pursue, what hobbies to pursue and how, and even more intimate decisions.

7 Contributions

This project was completed entirely by Annies Abduljaffar and Matt Vail.

Annies contributed to the identification of the goal and conducted preliminary work needed to identify the list of experiments. She setup the execution environment needed to execute the experiments over the big data set. She wrote the code to analyze the experimental results and create the preliminary charts (https://github.com/polymathnexus5/rec-engine-CS205L-

W19/blob/master/reports/ResultPlots.ipynb). She also wrote portions of the Introduction and Related Work sections of the report.

Matt prepared the entire GitHub repo, produced all of the experiment and supporting code aside from the code to produce preliminary charts, ran the experiments on the small data set, produced the final figures in Excel, wrote portions of the Introduction and Related Work sections, wrote all of the Abstract, Data and Software, Methods, Experimental Results and Analysis, and Conclusion sections, prepared the Bilbliography, Table of Figures, and Table of Equations, formatted and revised the entire report for presentation, and submitted.

References

[1] Y. Koren, "Factorization Meets the Neighborhood a Multifaceted Collaborative Filtering Model," 2008.

[2] Y. M. B. J. C. Daniel A. Galron, "Deep Item Based Collaborative Filtering for Sparse Implicit Feedback," 2018.

[3] "Netflix Prize," [Online]. Available: https://netflixprize.com.

[4] "Grouplens," [Online]. Available: https://grouplens.org/.

[5] "Surprise Documentation," [Online]. Available: https://surprise.readthedocs.io/en/stable/.

[6] Y. Koren, "Factor in the Neighbors Scalable and Accurate Collaborative Filtering, Koren 2010," vol. 4, 2010.

[7] L. R. B. S. P. B. K. Francesco Ricci, Recommender Systems Handbook, 2010.

[8] A. M. Ruslan Salakhutdinov, "Probabilistic Matrix Factorization".

[9] "Wikipedia - Gradient Descent," [Online]. Available: https://en.wikipedia.org/wiki/Gradient_descent.

[10] Y. Koren, "The BellKor Solution to the Netflix Grand Prize," 2009.

[11] Y. Koren, "Matrix Factorization Techniques for Recommender-Systems," 2009.

[12] Aggrawal, Recommender Systems, Yorktown Heights, New York: Springer, 2016.

[13] H. S. S. Daniel D. Lee, "Algorithms for Non-Negative Matrix Factorization".

[14] I. P. D. T. Gabor Takacs, "Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering," 2011.

[15] S. Postmus, *Recommender System Techniques Applied to Netflix Movie Data, 2018,* Amsterdam, 2018.

[16] Y. S. P. A. Prateek Sappadla, *Movie Recommender System - Project Report,* New York, New York, 2017.

[17] W. W. J. F. Sheng Wang, "Learning from Incomplete Ratings Using Non-negative Matrix Factorization".

[18] J. R. Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain," 1994.

[19] V.-T. Tran, "Dimensionality Reduction: SVD and its applications".

[20] S. M. Thomas George, "Scalable Collaborative Filtering Framework based on Co-clustering".

[21] A. M. Daniel Lemire, "Slope One Predictors for Online Rating-Based Collaborative Filtering," *arXiv,* 2008.

[22] H.-C. C. F. M. L. Robert H. Sues, "Stochastic Pre Conditioned Conjugate Gradient Method," *Probabilistic Engineering Mechanics,* pp. 175-182, 1992.

[23] M. J. Andres Toscher, *The BigChaos Solution to the Netflix Grand Prize,* 2009.

[24] M. C. Martin Piotte, *The Pragmatic Theory Solution to the Netflix Grand Prize, 2009.pdf,* 2009.

[25] "Sifter," [Online]. Available: https://sifter.org/~simon/journal/20061211.html.

Table of Figures

Table of Equations