|  |
| --- |
| **­­User Taste Prediction** |

**­­**

**Annies AbduljaffarMatt Vail**

Student Student

Stanford University Stanford University

Stanford, CA 94305 Stanford, CA 94305

[anniesab@stanford.edu](mailto:anniesab@stanford.edu) [vailm@stanford.edu](mailto: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 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.

**1 Introduction**

Recommender systems are increasingly important for predicting users’ preferences for 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:

Collaborative Filtering - predictions based on the preferences of similar users

Content Based Filtering - predictions based on the preferences of the same user on similar content in the past

Hybrid Recommender Systems - a combination of collaborative and content based

Within the class of recommender systems implemented using Collaborative Filtering, there are two subtypes namely memory-based systems and model-based systems.

The memory-based systems compute an item to item or user to item similarity to select a list of candidate recommendation items. The model-based methods try to model the user preference matrix by learning a set of latent factors and model the user to item preference matrix as two components: a user to factor preference matrix and a factor to item association matrix. In case of a movie recommendation you can think of factors like genre, actors, year of release as various latent factors. During the modeling process these factors are not known and sometimes researchers look into 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 possible new types of systems called model based systems, where deep learning is utilized to minimize any gaps in similarities between user to user or item to item by optimizing for reduction of a cost function [2].

Motivation & Intended Outcome: Our motivation behind choosing this project is 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 and use that understanding to suggest and implement ideas to improve performance of the algorithms.

Some of the interesting learning opportunities in this space are driven by the sheer magnitude of the dataset (size of the matrix, given the high cardinality of the user vector and the movie vector) and the sparsity of the dataset. In the Netflix prize dataset, only one in 85 entries in the user preference had known values and the rest of the values, 84 out of 85 had to be predicted. This example helps understand the difficulty of the problem and this is magnified in other spaces where the cardinality of items may be much higher than those of movies. For instance, the number of items that are available for sale in an eCommerce site like eBay, approximately 300M by some recent counts will lead to a extremely sparse matrix.

Another challenge is in designing the algorithms and implementing them in a way that does not require the entire matrix to be held in memory all at the same time. As detailed about due to the high cardinality of the user and item vector space, it is not feasible to expect a user’s personal computer to hold the entire matrix in memory. Based on Nathan Hug’s implementation of SVD using Stochastic Gradient Descent, this project suggests an improvement that manages memory usage efficiently by holding only two vectors in memory at a time, while iterating in the SGD method.

To understand opportunities for improvement, we designed experiments that search the hyperparameter space for each of the candidate algorithms that we chose. In addition, we designed few variants that rely on pre-conditioning of the data with respect to movies and users. More details on the experiments can be found in the Methods and Results section. We evaluated few candidates that could improve the performance of the algorithms, by reducing the RMSE on the test data compared to the existing algorithms. These were replacing a Conjugate Gradient Method for identifying the factors for the user preference matrix and adding a momentum method for the same. Though these methods did not actually result in a reduced RMSE as expected, the performance was close to some of the best performing algorithms available in the literature, as elaborated in the results section.

**ANNIES - Discuss the context of the problem, your motivation for looking at the problem, and clearly state the intended outcomes of the project (10%)**

**2 Related Work**

Implementing and optimizing performance of recommendation systems caught the fancy of the academic and research community when Netflix announced a contest [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 (add link to reference)”. The solution implemented by the winning team is similar to the work discussed in this paper in some respects and different in some other respects. 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 Singular Value Decomposition (SVD) in an efficient manner using Stochastic Gradient Descent. Another project that is tackling similar challenges is the research done by eBay research team for the improvement of item recommendations for eBay users. eBay researchers are implementing a deep learning based method to predict the user to item preference as seen in the recent publication [2]. We were interested in exploring the benefits of a deep learning method but could not pursue it due to time constraints.

**ANNIES - Discuss at least 2 pieces of related literature (or software, if you're doing an implementation project) and how your project compares to them (5%)**

**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 [2]. 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:

­­­A screenshot of a cell phone

Description automatically generated

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.

A screenshot of a cell phone

Description automatically generated

Figure 2: Custom software modules

**4.2 Data Format**

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

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:

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:

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

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:

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

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

**4.3 Algorithms** [3]

**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 Liklihood Estimate:

Equation 1: Normal Predictor Model

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

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

Where 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 [3]. 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

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

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 [5].

**4.3.3.1 SVD Model**

The SVD algorithm is an extension of Probabilistic Matrix Factorization [4] 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 [5].

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

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

Equation 7: SVD Model Objective Function

Where 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

Where the set is the set of all movies that have been rated by the user (regardless of rating) and 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

**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

**4.3.4 Gradient Descent**

Gradient descent is a popular first-order iterative optimization algorithm for finding the minimum of a function [7]. 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

Where is the learning rate (scalar ~0.05 is generally a good starting spot), is a vector of model parameters (or weights) being optimized, is the objective function, and 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:

Where is the momentum parameter. Note that setting 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, SVDpp, SVD\_SGD\_momentum and SVDpp\_SGD\_momentum models each with the same 4 preconditioning procedures.

**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 combination of parameters then is 11\*4\*3\*3\*4 = 1,584.

**5 Experimental Results and Analysis**

**5.1 Results**

**5.2 Analysis**

Show your results, and if appropriate, analyze them. Note that the weighting between the two will change depending on your project type: for visualization projects and network exploration projects we'll weight experiments more heavily, but for implementation projects and theory projects we'll weight the analysis more heavily. (40%)

**6 Conclusion**

Implementing Conjugate Gradient on subsets of the matrix.

Implementing other more sophisticated momentum methods.

Implementing time-based methods that take into account the timestamp and the release date (movie ratings go up over time because people who liked them rewatch them)

Clearly summarize your results and describe a few potential avenues for future work. (5%)

**7 Contributions**

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

Annies contributed to the identification of the goal for this project 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 contributed code to the GitHub repository, specifically the pieces that analyze the experimental results and create the preliminary charts (specifically code found [here](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 relevant 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] | "Grouplens," [Online]. Available: https://grouplens.org/. |
| [3] | Y. Koren, "Factor in the Neighbors Scalable and Accurate Collaborative Filtering, Koren 2010," vol. 4, 2010. |
| [4] | Aggrawal, Recommender Systems, Yorktown Heights, New York: Springer, 2016. |
| [5] | Y. M. B. J. C. Daniel A. Galron, "Deep Item Based Collaborative Filtering for Sparse Implicit Feedback," 2018. |
| [6] | H. S. S. Daniel D. Lee, "Algorithms for Non-Negative Matrix Factorization". |
| [7] | I. P. D. T. Gabor Takacs, "Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering," 2011. |
| [8] | S. Postmus, *Recommender System Techniques Applied to Netflix Movie Data, 2018,* Amsterdam, 2018. |
| [9] | Y. S. P. A. Prateek Sappadla, *Movie Recommender System - Project Report,* New York, New York, 2017. |
| [10] | A. M. Ruslan Salakhutdinov, "Probabilistic Matrix Factorization". |
| [11] | W. W. J. F. Sheng Wang, "Learning from Incomplete Ratings Using Non-negative Matrix Factorization". |
| [12] | J. R. Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain," 1994. |
| [13] | V.-T. Tran, "Dimensionality Reduction: SVD and its applications". |
| [14] | R. B. C. V. Yehuda Koren, "Recommender-Systems-[Netflix]," 2009. |
| [15] | S. M. Thomas George, "Scalable Collaborative Filtering Framework based on Co-clustering". |
| [16] | A. M. Daniel Lemire, "Slope One Predictors for Online Rating-Based Collaborative Filtering," *arXiv,* 2008. |
| [17] | H.-C. C. F. M. L. Robert H. Sues, "Stochastic Pre Conditioned Conjugate Gradient Method," *Probabilistic Engineering Mechanics,* pp. 175-182, 1992. |
| [18] | Y. Koren, "The BellKor Solution to the Netflix Grand Prize," 2009. |
| [19] | M. J. Andres Toscher, *The BigChaos Solution to the Netflix Grand Prize,* 2009. |
| [20] | M. C. Martin Piotte, *The Pragmatic Theory Solution to the Netflix Grand Prize, 2009.pdf,* 2009. |
| [21] | "Sifter," [Online]. Available: https://sifter.org/~simon/journal/20061211.html. |
| [22] | "Surprise Documentation," [Online]. Available: https://surprise.readthedocs.io/en/stable/. |

**Table of Figures**

[Figure 1: Surprise Library partial class diagram. Custom written classes in green. 2](#_Toc3988793)

[Figure 2: Custom software modules 3](#_Toc3988794)

**Table of Equations**

[Equation 1: Normal Predictor Model 4](#_Toc3992554)

[Equation 2: Baseline Only Model 4](#_Toc3992555)

[Equation 3: Baseline Only Model Objective Function 5](#_Toc3992556)

[Equation 4: Mean Squared Difference (MSD) similarity measure between two users u and v 5](#_Toc3992557)

[Equation 5: KNN Basic Model using user-based MSD as the similarity measure 5](#_Toc3992558)

[Equation 6: SVD Model 6](#_Toc3992559)

[Equation 7: SVD Model Objective Function 6](#_Toc3992560)

[Equation 8: SVDpp Model 6](#_Toc3992561)

[Equation 9: SVDpp Model Objective Function 7](#_Toc3992562)

[Equation 10: NMF SGD Update Rule, Ensures That Factors Are Always Positive 7](#_Toc3992563)

[Equation 11: Generic Gradient Descent Update Rule 7](#_Toc3992564)