An Analysis of Algorithms Used for Solving the Movie Recommendation Problem

Jeffrey Yeung

***Abstract*— Recommender systems are as important as ever given the abundance of data collected and the need to accurately generate suggestions to end-users in real-time. The Netflix Challenge has led to the creation of many algorithms that are still used to this day to generate recommendations for users. The purpose of this paper is to analyze how some of these commonly used algorithms work and assess their performance on a subset of the Netflix Dataset.**

I. Introduction

The purpose of the Netflix Challenge was to recreate user trends from a massive dataset and generate accurate predictions of what a user might rate other movies. From this, it could then be extrapolated that a user that could favorably rate something would be more likely to watch it over something that would be less likely to be favorably reviewed. In this paper, we discuss a few algorithms that generated some of the best solutions, including the baseline implementation, which attempts to predict trends using generated averages of users' general feelings and the general sentiments toward specific movies deviation from the average rating.

II. Dataset

The dataset used in this paper is the official Netflix Prize data provided by Netflix from its Netflix Challenge. The dataset is divided by each movie's ID with subcategories of information, namely the list of all the user IDs who submitted a rating, the actual rating given, and the date of submission. All unique users have different user IDs to differentiate and anonymize their identities, and their ratings are all integer values from 1 to 5, with 1 being the lowest satisfaction and 5 the highest. The dataset that we are using has 4,499 different movie titles spanning from the year 1915 to 2005; within the dataset are 2,405,357 rating submissions from 470,758 unique users.

III. Algorithms

A. General Notation

Here we formulate the general problem of estimating the rating matrix R that is composed of r_{ui}, which is the rating associated with user u and movie i. It's quite obvious that most users will not review every movie in existence, so R is naturally a sparse matrix. How sparse depends on the source of the data, but R is typically much more sparse than dense. Given this, we would like to estimate the remaining r_{ui*}; that is, we would like to generate what user u might possibly rate movie $i*$, a rating that is not present in the original dataset. For this paper, the estimates we try to achieve will be called $\hat{r}_{ui}$, where, for samples present in the dataset, $\hat{r}_{ui} \approx r_{ui}$. This means that averages across users or movies will become important. We can denote these using the common bar notation, such as $\bar{r}_u$ and $\bar{r}_i$.

Furthermore, given that the number of users m is usually much larger than the number of movies n, we would expect formulations to hopefully not depend on generating the entire $n \times m$ matrix in order to generate solutions. As a result, most of the following algorithms will depend on algorithms that iterate across the dataset. As we will see, these algorithms will usually include some form of randomness in order to ensure that the solution does not rest on suboptimal local minima/maxima, which is an issue for most of these non-convex problems. Finally, at some point there gets to be too many variables to optimize. In order to prevent cluttered notation, we describe the group of parameters to be estimated using Ψ.

B. Baseline Estimation

For the baseline estimator, we want the bare minimum needed to make a prediction of a user's rating. The most straightforward way to do this would be to measure the sorts of biases present in a user's choice and in the general rating of specific movies. More specifically, we seek to fit each rating to the following model:

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i \tag{1}$$

Eq. 1 states that the rating user u gives to movie i can be approximated using the overall mean rating, the biases present in the user's ratings, and the movie's state [or the overall bias, b_{ui}]. By the movie's state, we mean things that cause certain events such as "Movie x is **better than the average** movie" to be provably true using simple analytics. In general, this can simply mean that a movie is more favorable to many viewers so the bias inherent to the film must be larger than normal.

Since the movie dataset is very sparse, a simple way to generate estimates for these values is to use a method of optimization called stochastic gradient descent (SGD, for short). Assuming that the metric we seek to optimize on is the L2 norm of the error, the optimization problem can be rewritten as the following regularized least squares problem:

$$\min_{b_i, b_u} \sum_{(u,i) \in K} (r_{ui} - \mu - b_i - b_u)^2 + \lambda_1 \left(\sum_u b_u^2 + \sum_i b_i^2 \right) \tag{2}$$

Solving the optimization problem in Eq 2 can be done by taking the gradient with respect to the parameters to

[2]c1yeung (at) eng.ucsd.edu

optimize and solving for them. This would define the update conditions for b_*. In general, the updates are of the form $b_*^k \leftarrow b_*^k + \alpha \nabla_{b_*} Q(b_*^k)$ where Q is the objective function given above and α are the learning parameters and scale the change in the direction of the gradient. Note that in Eq. 2, K is the entire dataset (that is, user/rating pairs for a given movie) that is being optimized on, and λ_* act as regularizing terms. This is clear from the fact that larger values of b_* will generate larger values for higher λ_*, thereby "penalizing" their growth.

C. Singular Value Decomposition

Singular Value Decomposition, or SVD, is perhaps the most commonly used algorithm for estimating the actual form of R. The reason for this lies with the ability of SVD to act as a dimensional reduction along both dimensions of R in order to generate three matrices $R = U\Sigma R^*$. Given U and R^*, we can describe the space of the matrix R entirely. This would be great if it could be applied directly to R, but since R is sparse, the SVD decomposition of R does not actually exist. As a result, we attempt to formulate the estimate of something similar to the SVD decomposition algorithm using SGD as the optimization technique.

In this case, the objective is to fit all data points to the following model:

$$\hat{r}_{ui} = b_{ui} + q_i^T p_u \tag{3}$$

In Eq. 3, $p_u, q_i \in \mathbb{R}^f$, where f is a factor that determines the dimensionality of the feature space of the users and movies. Simply put, a smaller f could be said to perform a process similar to dimensionality reduction. The objective is to choose such an f that generates the best latent features that describe the overall distribution of R with respect to its users and movies.

Like the baseline case, we try minimizing on the L2 norm of the error in order to generate an optimization case as follows:

$$\min_{\Psi} \sum_{(u,i)\in K} (r_{ui} - \hat{r}_{ui})^2 + \lambda_1 (b_i^2 + b_u^2 + ||q_i||^2 + ||p_i||^2) \tag{4}$$

Like in the baseline case, the optimization problem in Eq. 4 is still easily solvable using SGD. Using SGD on the above leads to updates of exactly the same form as the baseline case for the biases b_* with the inclusion of two new updates of the following form:

$$p_u \leftarrow p_u + \alpha \left((r_{ui} - \hat{r}_{ui}) \cdot q_i - \lambda p_u \right) \tag{5}$$

$$q_i \leftarrow q_i + \alpha \left((r_{ui} - \hat{r}_{ui}) \cdot p_u - \lambda q_i \right) \tag{6}$$

D. SVD++

SVD++ is an influential algorithm that came about from the winners of the Netflix Challenge. In this algorithm, we attempt to better the results from vanilla SVD+SGD by attempting to collect more information that would potentially affect the overall ratings through implicit feedback. Implicit feedback is unobserved info that could be used to influence how a user ranks specific films, such as their preferences or their rental locations. One form of implicit feedback inherent to the original matrix R comes from the fact that the user actually chose to review movie i. We can denote this implicit feedback matrix as $N(u)$. To that end, the model for the user now becomes $p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$.

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) \tag{7}$$

$$\begin{aligned} \min_{\Psi} \sum_{(u,i)\in K} (r_{ui} - \hat{r}_{ui})^2 + \lambda (b_i^2 + b_u^2 + \\ ||q_i||^2 + ||p_i||^2 + \sum_{j \in N(u)} ||y_j||^2) \end{aligned} \tag{8}$$

As a result, the model for SVD++ is described by Eq. 7 and the optimization problem is shown by Eq. 8. To gain some intuition on this formulation before jumping directly into the SGD optimization technique, it seems reasonable to believe that given the fact that the user has deliberately chosen to rate a certain movie, that there would be some implicit bias as a result of this action [which would be quantified by the feature vector y_j]. As a result, we now seek to aim to optimize for this implicit feature vector y_j along with the explicit feature vectors q_i and p_u. Given that the formulation is exactly the same as the SVD case when $\forall j, y_j = \mathbf{0}$ [that is, in the absence of implicit information], we can see that the update equations for all parameters will remain exactly the same as in the SVD case with the inclusion of one more update equation for the latent feature vectors, y_j, as shown by Eq. 9.

$$y_j \leftarrow y_j + \alpha \left((r_{ui} - \hat{r}_{ui}) \cdot |N(u)|^{-\frac{1}{2}} q_i - \lambda y_j \right), \forall j \in N(u) \tag{9}$$

E. KNN using Pearson Correlation

The Pearson R Correlation is defined as follows:

$$s_{uv} = \frac{\sum_{(u,v)\in K} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sum_{(u,v)\in K} (r_{ui} - \bar{r}_u)^2 (r_{vi} - \bar{r}_v)^2} \tag{10}$$

The K-Nearest Neighbor algorithm depends on some way of clustering users together. In this case, we group users together given their similarities and denote the neighborhood of n users N_k. Given these n users in N_k we can shift the centroids and begin re-estimating the new neighborhoods given the similarity between two users until convergence occurs.

Initialization is generally based off of perturbations of the mean in the user space, but this can cause certain clusters to become optimized out and take a large computation time. Furthermore, to generate the similarity matrix S, notice that a $m \times m$ matrix must be generated in every step of the optimization (where m is the number of users and is generally much greater than n, the number of movies) in order to re-evaluate the new neighborhoods. This is prohibitively expensive in terms of computational cost for the Netflix dataset; so, a smaller subset will be used to evaluate its performance.

To evaluate ratings after optimization is complete, we can see that the ratings will simply be based off the weighted sum of the similarities using Eq. 10 around a given neighborhood of user u.

$$\hat{r}_{ui} = \frac{\sum_{v \in N_k} s_{uv} \cdot r_{vi}}{\sum_{v \in N_k} s_{uv}} \tag{11}$$

Notice that instead of clustering around users and evaluating with Eq. 11, we could have performed KNN along the much smaller movies space. This is indeed another possibility, and would likely be better for generating suggestions to new users, but naturally the movie-movie similarity space would favor more biases along the movie space. Note that removing the biases actually produces another algorithm (that will not be discussed here). The only change would be to the weights of the similarities, which would now be weighted with respect to the distance from the mean.

F. Slope-One

The slope-one scheme relies on a similar technique to KNN. That is, it depends on users who rated the same item and on other items that the user rated. The novelty of this algorithm, however, comes from the fact that it also takes into account ratings that are completely outside of r_u and r_i [That is, free from the user and the items chosen by the user.]. The derivation is dependent on the fact that, given two vectors v_i and w_i, the best predictor for w_i based on the model $\sum_i (v_i + b - w_i)^2$ leads to the solution $b = \frac{\sum_i w_i - v_i}{n}$. That is, b is simply the average difference between v_i and w_i.

The algorithm follows. Given our training set K, and two items with ranking r_{ui} and r_{uj} given by some user u, we can produce estimates $\hat{r}_{ui}$ by taking the set of items j rated by u that have at least one common user with i and averaging the difference between the ratings of i and j. This is more succinctly stated by Eq. 12 and Eq. 13

$$\hat{r}_{ui} = \bar{r}_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} \text{dev}(i,j) \tag{12}$$

$$\text{dev}(i,j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{ui} - ruj \tag{13}$$

Note that Eq. 13 represents the deviation between the ratings of user i and j for only values in U_{ij}, or the set of users who have both rated movies i and j. This is summed over the set of items j rated by u that have at least one common user with i. It's easy to see that if there is not a single common user, then the rating is just the mean. Such a simplistic rating would clearly not perform very well for extremely sparse matrices.

G. Metrics

All of the methods described were evaluated using RMSE (Eq. 14) as the main metric to watch during k-fold cross-validation. Note that the set S_T is the testing dataset or validation dataset, which k-fold cross-validation handles for us. The reason for choosing this as the main metric to watch is due to the fact that user ratings do not generally vary greatly from the averages generated by these estimators. That is, we do not expect user u to suddenly rate every future movie 5 given that they only review movies that they generally disliked.

$$RMSE = \sqrt{\frac{1}{|S_T|} \sum_{(u,i) \in S_T} (r_{ui} - \hat{r}_{ui})^2} \tag{14}$$

One should be careful when considering RMSE, however. While the above is generally assumed to be true (hence why most algorithms depend on such a metric), it is possible that ratings will be skewed as a result. More observations would be needed in order to form a conclusion about using the set RMSE as the main metric, but that would not be the purpose of this paper.

IV. Application

All of the SGD based algorithms were trained under 5-fold validation for at least 20 epochs per session with standard learning rates (that is, the same learning rates used by the Koren paper). These tests were performed with a smaller subset of the data and used on the entire dataset, if possible within the time constraints. We noticed that the Pearson KNN demanded unreasonable amounts of memory that possibly scaled up to 100GB, which was not feasible to acquire. Furthermore, SGD++ had been reported to take up to two hours for a dataset of one million entries with much smaller user/movie dimension. It would have taken several days to train a single SGD++ implementation, so this algorithm was also only trained on the small dataset. The results are tabulated in Table I. If possible, the algorithms were ran a few times to check if they were relatively stable during optimization. Using the k-fold cross-validation technique made it clear that the optimizations were fairly stable and usually converged to the same location.

V. Results

	Baseline	KNN	SVD	SVD++	Slope-One
Small $\mathcal{D}$	1.27	1.29	1.27	1.30	1.31
Entire $\mathcal{D}$	0.93	N/A	0.90	0.93	0.93

TABLE I: The lowest RMSE found for each model from a couple training sessions [Lower is better]

The results of the algorithms are tabulated in Table I. Since the metric used was RMSE, the lower the final RMSE, the better the algorithm performed. This means that the algorithm was better able to generalize to the known values. One thing to point out is that even though the algorithm could generalize better to the known values and unknown values from the validation set, this doesn't mean that the algorithm will perform better at suggesting ratings for all users. As with all machine learning algorithms, it's very likely that it will be better at predicting ratings for users with a lot of data but lack the collaborative information needed for some new user at some other point.

In general, the execution time for SGD++ and KNN were the largest on the order of days for the entire dataset. This was the exact reason that the experiment could not be performed on the entire dataset (so even if the similarity matrix from KNN could fit, it would take far too long to produce results). This was evidenced from the situation Koren experienced when training their models. SGD produced results at a moderately quick pace, while the Slope-One algorithms and baseline algorithms were far too simplistic to capture the necessary variations in the dataset.

VI. Analysis and Observations

Given the results tabulated above, it seems like one would be tempted to say that SVD performed better on average than SVD++. However, this would be a mistake as the implicit information matrix in this problem, $N(u)$ contains more information as the number of users and reviews grows. In fact, the larger the problem (in terms of users/movies), the more likely that $N(u)$ influences user features. To show this, we used the full dataset but only trained on 10% of the data for SGD++ for 20 epochs. After this, we evaluated the RMSE of the test set for all algorithms trained on the entire dataset with the exception of the SGD++ case (which was trained on 10% of the data points). The testing RMSE of 0.9382 in the SGD++ case compared to the SGD, Baseline, and Slope-One cases [0.788, 0.9222, and 0.9074, respectively] is shocking! The fact that certain users reviewed specific movies allowed the SGD++ case to generalize far better to a completely unknown dataset when compared to the other methods. [Note that the accuracy of SGD is an outlier here as one would expect it to be near the mean; this particular instance holds less weight despite the fact that some of the samples from the testing set were included in the training set for the SGD case.]

In addition to the Netflix recommendation system challenge, we also performed data analysis on the dataset and observed several trends that we will showcase and discuss.

1) Observation 1: A vast majority of Netflix users only submit a very low number of ratings, but there is a strong majority group of users who submit a very high number of ratings. The mean and standard deviation of the rating submissions per user are 51.1 and 74.4, respectively, and the median is much lower at 24.0. The fact that the median is much lower than the mean and the standard deviation is higher than the mean indicates that the majority of Netflix users do not care much about leaving ratings, or that they simply do not watch many movies. This may prove to be beneficial to Netflix since its subscription model is a strict one-time monthly fee, so users who do not frequently access their services must pay the same amount.

2) Observation 2: The movie industry has existed for centuries, but it is only in the recent decades that viewership and production volume significantly increased due to the advancement of technology enabling quicker and superior production schedules. From the Netflix movie catalog listing the movies in stock from 1915 to 2005, we can clearly see how its expanding volume is consistent with the fact that more movies are produced each year. The number of movies in the catalog are typically below 100 for all the years until the 1990s, when the movie industry really started booming and production vastly increased. Since the 1990s, there is a sharp exponential growth in the Netflix catalog.

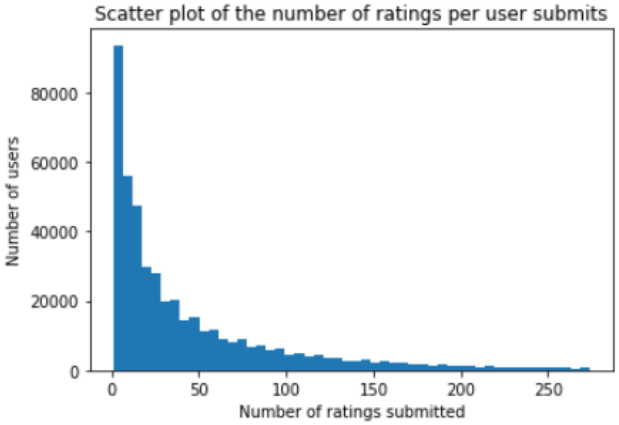

Fig. 1: Histogram plot of observation 1

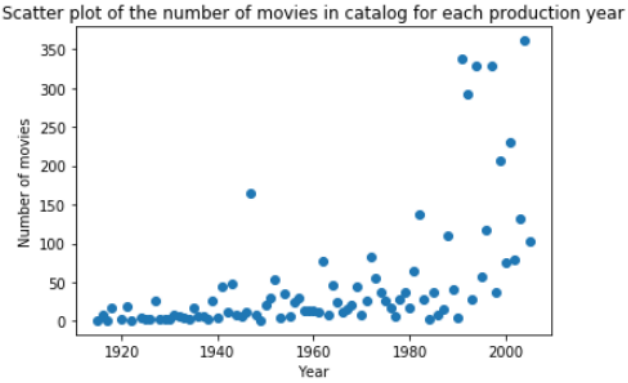

Fig. 2: Scatter plot of observation 2

3) Observation 3: Further in line with the previous observation, the advancement of technology in the recent decades lead to higher quality movies produced, and it would be reasonable to assume that movie watchers generally rate more favorably for movies produced in the later years due to their higher production qualities. However, from the dataset we discovered that this assumption is only partially correct. Although it is true that the ratings are lowest for the earliest years (around 1915 to 1940), the ratings are the highest around the 1940s to the 1980s, and slowly drops off approaching the 2000s. There are several possible explanations to this phenomenon: user may rate more favorably for older movies due to nostalgia and legacy standings, or they feel judge the newer movies more critically due to the vastly increased production volume, so there are many more movies to judge and compare against. Finally, it may also just simply be that the newer movies are not as enjoyable as the old films.

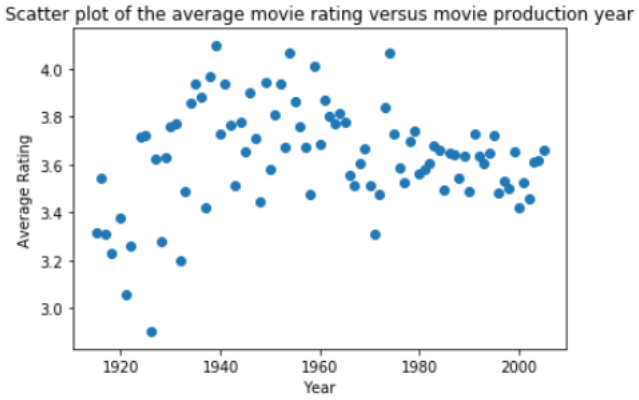

Fig. 3: Scatter plot of observation 3

4) Observation 4: In general, the number of ratings submitted by users per movie increases as the movie production year increases. In the earliest movies (1915-1920), only about 170 to 200 ratings are submitted per movie, and this number steadily rises over the years, leading to about 6,000 ratings per movie for movies made in the 2000s. This is an interesting observation because it implies that movie watchers are more likely to submit a rating for newer movies than older movies. This fact shows that users are generally more engaged and affected by their experiences from the newer movies, and feel more apathy and indifference to older movies. We believe this is due to the fact that newer movies tend to be much more sophisticated so there are more elements to dissect. We also think that the rapid spread of social media over the last decade leads to more people talking about these newer movies, so they are more likely to leave a rating.

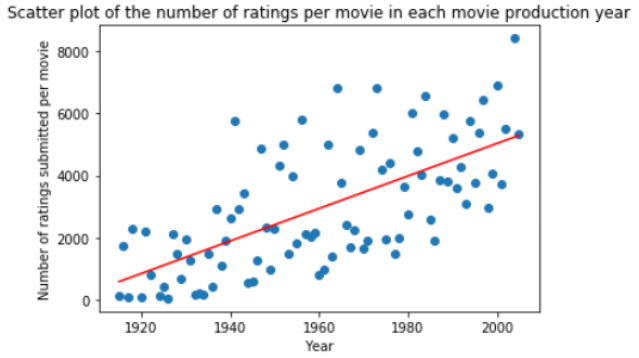

Fig. 4: Scatter plot of observation 4 with best fit line

VII. Conclusion

In general, it is easy to see that the usefulness of SGD and SGD++ are shown when used on a dataset as large as the Netflix dataset. The presence of implicit information has a clear effect on how well a dataset generalizes to items that are not reviewed by a given user. While it was not possible to set up the predictor given SGD++, it's quite clear from the performance when trained on 10% of the data and evaluated with 30%, the algorithm is quite promising when compared to the other methods generalization capability.

On a more real world applicable sense, the observations made in the later parts of the paper show that Netflix users tend to be more engaged with the newer movies, yet rate more favorably for the older movies. This is most likely due to the abundance of movies produced each year, as well as rising expectations from the advancement of film technology. Though it may be an obvious conclusion based on the observations, we believe that Netflix stands to profit more if it focused more on the newer movies, since that is what the general audience has shown to prefer.

References

[1] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. 2007. URL: http://arxiv.org/abs/cs/0702144.

[2] R. Bell, Y. Koren and C. Volinsky. The BellKor Solution to the Netflix Prize. 2009. URL: https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. 2009.

[4] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Recommender Systems Handbook. 1st edition, 2010.

[5] Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. Learning from incomplete ratings using non-negative matrix factorization. 1996. URL: http://www.siam.org/meetings/sdm06/proceedings/059zhangs2.pdf.

[6] Jonathon Shlens. A Tutorial on Principal Component Analysis. 2005. URL: https://www.cs.cmu.edu/ elaw/papers/pca.pdf

[7] Nicolas Hug. Surprise, a Python library for recommender systems. 2017. URL: http://surpriselib.com