The Netflix Problem

APPM: 4120
Jamie Shaw, Denis Kazakov

Abstract: We will learn how to construct an accurate content recommendation system by finding trends in the ratings that users provided for movies that they watched.

Keywords: Unsupervised learning, Funk SVD, Machine Learning, Matrix Factorization

1 Introduction

In our project, we investigated existing recommendation algorithms, learned the basics of Machine Learning necessary to complete the project and implemented the Funk SVD algorithm which is one of the most accurate ones as of today. The main challenge in this project was gaining an intuition for the implemented algorithm since we were unable to find any papers that could effectively explain it in a general sense.

2 Background

2.1 Netflix Grand Challenge

In the year 2006, Netflix announced the *Netflix Grand Challenge* in which Netflix offered a million dollars to the team who could decrease the error of the Netflix recommendation system by ten percent. The Netflix Grand Challenge sparked much interest in research in the fields of computer science and applied math. After a multitude of approaches were tested against given datasets, a team named "BellKor's Pragmatic Chaos" succeeded by reducing error by 10.06%, winning them the Netflix prize.

Netflix, however, is not the only company that can benefit from an accurate recommendation system. Recommendation systems can be applied to a wide range of disciplines so

long as the data is varied enough to realize trends. For example, Amazon uses a recommendation system to suggest products they believe their costumers will be interested in buying. Spotify, a music provider similar to Pandora, uses a recommendation system to suggest songs to their users. And finally, Quora, a Q & A website, uses a recommendation system to identify topics you may be interested in learning about. They claim this recommendation system fosters 75% of their user activity.

In general, the algorithms underlying these recommendation systems, known as suggestion algorithms, have been implemented as a ubiquitous tool in today's world for providing users' with recommendations to match their taste and preference.

2.2 Application to Students' Education

Beyond recommendations that increase user enjoyment of a product, such as Spotify, Netflix, and Quora, these algorithms have a much broader potential. For example, one could imagine using suggestion algorithms to aid student learning in a classroom setting. This may be accomplished by providing students with professor recommendations based on students' past professor evaluations. This would enable universities to appropriately match professors and students with similar teaching-learning styles to maximize students learning and experience. However, for such system to function, it would require removing the element of complete anonymity in the answer forms, Faculty Course Questionaires (FCQs). This could easily be achieved by providing each student with a unique ID which does not contain students' names and is only used to keeping track of classes and reviews that were assigned to that ID.

3 Recommender Systems

3.1 Netflix Algorithm Overview

At the heart of the Netflix Problem is a data matrix that must be deciphered for pre-existing trends. For Netflix in particular, this data represents the movie ratings given by Netflix users given to specific movies. Those reviews can be represented in an m × n matrix of movie ratings, as in Figure 1, where m = "users" and n = "movie titles".

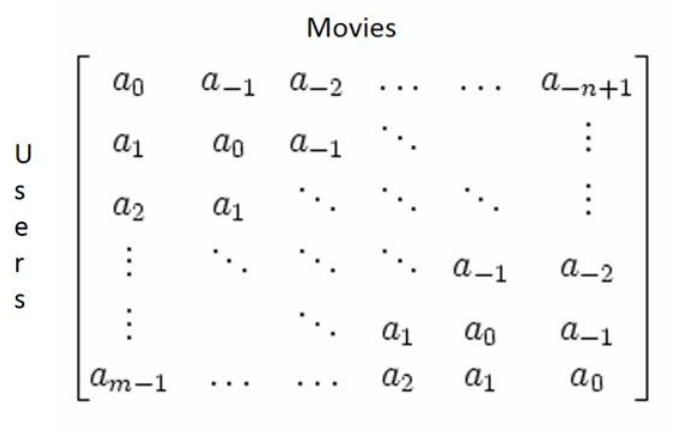

Figure 1

Netflix's current system uses a combination of Restricted Boltzmann Machines (RBM), which are generative stochastic learning algorithms, and a form of Matrix Factorization, known as SVD++, in order to identify the patterns existing in the users' ranking matrix [1]. Those patterns are then used to predict users' response to movies they have not seen yet.

3.2 Root Mean Square Error

Once this this data matrix is collected and the SVD (discussed later in this paper) is performed, the resulting patterns are fit using a linear regression in order to predict future interests. This is accomplished by reducing the error, known as Root Mean Square

Error (RMSE), between the data and the best fit. This is also known simply as standard deviation, as shown in Eq. 1. "The use of RMSE is very common and it makes an excellent general purpose error metric for numerical predictions. Compared to the similar Mean Absolute Error, RMSE amplifies and severely punishes large errors" [2].

$$RMSE \equiv \sigma = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_i - \bar{x})^2} \tag{1}$$

The *Netflix Grand Challenge* winning algorithm achieved a RMSE = 0.8558 using a linear blend of RBM and SVD++. However, due to the engineering complexity of RBM, we will not implement the original Netflix algorithm. Instead, we will carry out the algorithm using only Funk SVD matrix decomposition and use it to predict missing entries of our dataset. Given this method, our suggestion algorithm for recommending movies based on users' previous movie ratings will require a three step process: 1) acquiring data from users, 2) decomposing data according to trends via matrix factorization (Funk SVD), and 3) predicting missing data points of our ratings data matrix by using Linear Regression and Gradient Descent.

4 Design and Implementation

4.1 SVD

Discovering trends in a given dataset is a challenging task that is impossible to accomplish without mathematical oversight. An effective method of finding such trends is matrix decomposition. The typical approach of matrix decomposition is performing Singular Value Decomposition (SVD) on the given matrix. One can think of SVD as a generalization of Eigenvalue Decomposition that can be applied to matrices of arbitrary dimensions.

To understand SVD it is useful to first look at Eigenvalue Decomposition. In Eigenvalue Decomposition, a correlated data matrix is factored into eigenvectors that represent a basis set that is most effective at describing data variability among the matrix. Each eigenvector

has a corresponding eigenvalue whose size indicates the strength of variation that is described by each of the new coordinate axes of the new basis set. Generally speaking, for an n×n matrix, A, there will be n eigenvalues and n eigenvectors, of the form $Ax_i=\lambda_i x_i$ (for i=1,...,n), which can be placed into a matrix D, composed of eigenvalues on the diagonal, and matrix W composed to eigenvectors. This gives the similarity transformation shown in Eq. 2.

$$A\left[n\times n\right]=W\left[n\times n\right]D\left[n\times n\right]\left(W\left[n\times n\right]\right)^{-1} \tag{2}$$

While eigenvalue decomposition is limited to square matrices, with SVD we can extend analysis to non-square matrices as well. We generalize eigenvalues and eigenvectors to singular value and singular vectors respectively. Similar to the similarity matrix above, SVD takes the form of Eq. 3. However, now we have a matrix, S, representing a diagonal matrix of singular values and U and V representing right and left singular vectors.

$$X\left[n\times m\right]=U\left[n\times r\right]S\left[r\times r\right]\left(V\left[m\times r\right]\right)^{T} \tag{3}$$

In matrix form, this is represented by Eq. 4,

$$\overset{X}{\underset{m\times n}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix}}} = \overset{U}{\underset{m\times r}{\begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mr} \end{pmatrix}}} \overset{S}{\underset{r\times r}{\begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \vdots \\ \vdots & \cdots & s_{rr} \end{pmatrix}}} \overset{V^T}{\underset{r\times n}{\begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{r1} & \cdots & v_{rn} \end{pmatrix}}}. \tag{4}$$

4.2 A Problem with SVD

An additional challenge associated with finding trends within a given set of data is that any real set of data will likely contain numerous missing matrix elements, making the matrix "sparse". In an example with movies, let us say an average user has watched around

70 movies from 7,000 available. If we were to represent such a dataset as a matrix with each row representing a user and each column representing a movie, then 99% of our matrix is nonexistent. Filling those entries with 0's is intuitive, but not effective if we plan to apply matrix decomposition to find user movie-watching trends. This is because SVD will treat every entry of the matrix as an actual data point and try to "fit" those data points with its decomposing vectors even though 99% of the matrix doesn't technically exists. Instead, we would like these missing matrix elements to be discarded. Unfortunately, there is no easy mathematical technique that can resolve this issue for us. Such problem is known as a "sparse matrix problem".

4.3 Avoiding the "Sparse Matrix Problem"

4.4 "Funk SVD"

Handling a sparse matrix requires the use of an alternative version of matrix factorization. This version is known as Funk SVD, which was developed by an independent freelancer under pseudo name, "Simon Funk". Below we discuss how one may think of such a decomposition.

Again since we want our trend, or found patterns, to results only from filled matrix entries, we need to somehow ignore all zero matrix elemets. The best way to do this is to "fake" our SVD decomposition with a Machine Learning technique. Our Matrix Factorization will take the form of Eq. 5:

$$\overset{DataMatrix}{\underset{X}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix}_{m\times n}}} = \overset{UserFeatures}{\underset{U}{\begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \vdots \\ \vdots & \cdots & s_{rr} \end{pmatrix}_{m\times r}}} \overset{ItemFeatures}{\underset{V^T}{\begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{r1} & \cdots & v_{rn} \end{pmatrix}_{r\times n}}} \tag{5}$$

Here we notice there is no Singular Value matrix. This is because during the Machine

Learning process singular values get distributed between the U and V matrices. Later, by normalizing U and V appropriately, we may recover our singular values.

4.5 Linear Regression and Gradient Descent

In order to find these UserFeatures and ItemFeatures, we use the technique of Gradient Descent applied to Linear Regression. While we will not go into detail of this Machine Learning technique as it is fairly standard, we will provide an intuition behind it. Figure 2 shows the idea of Linear Regression and Gradient Descent applied to a single pattern found through Funk SVD.

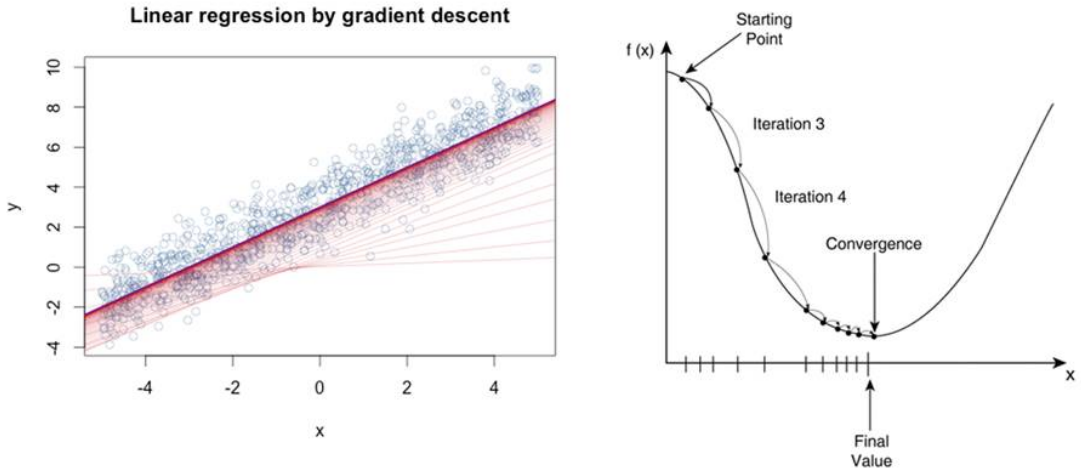

Figure 2

Gradient Descent attempts to minimize the error between the data points (in blue) and estimating function (in red). As our estimate function approaches the best fit, the steps in Gradient Descent decrease in size (image on the right). This happens because each step in Gradient Descent requires the multiplication of the gradient value and the function itself. As we approach the local minimum, the gradient value approaches zero [3]. Mathematically, this is represented by Eq. 6 below.

$$\min_{U^*,V^*} \sum_{(m,n)\in K} \left(X_{mn} - U_m^T V_n\right)^2 \tag{6}$$

In our work, we minimized the function which is the difference between real data entries (X) and predicted values (U^TV).

4.6 Algorithm

Below we have the algorithm that we have implemented for this project.

1. We divide the original dataset into a two sets, 90% learning set and 10% testing set.
2. We "guess" some values for UserFeature vector and ItemFeature vector using the "fake SVD". (aka U_i, V_j, where i,j – feature numbers)
3. P = $U_i V_j^T$ will give us a matrix of size m x n, which is a "prediction" of i'th set of features.
4. We then see how big of an error this prediction gives us...(This part allows us to ignore nonexistent entries)

   error = 0

   for i movies and j users:

   if X_{ij}=0

   error + = X_{ij}-P_{ij}
5. We update our feature matrices based on the calculated differences

   $ItemFeature_{ij}$ + = 0.001 (error x $UserFeature_{ij}$ - 0.02 x $ItemFeature_{ij}$)

   $UserFeature_{ij}$ + = 0.001 (error x $ItemFeature_{ij}$ - 0.02 x $UserFeature_{ij}$)

   (Where the 0.001 term was chosen experimentally to scale the matrices in order to make our "learning" process more smooth and gradual. And the 0.02 term regularizes the matrices to avoid "overlearning".)
6. Linear Regression + Gradient Descent: After each iteration step, we check if the derivative of our error function is leveling out to zero (because one set of feature

matrices can only be so good in estimating our data). At this point, if Gradient Descent has not yet converged, we repeat steps 3-6. Otherwise, we proceed to step 7.

7. X' = X - $U_i \times V_j^T$; i = i+1, j=j+1

   We can consider found feature vectors to be done, so we can move on to next pair of feature vectors. Therefore we update our data matrix to X', which is amount we still need to "fit" in our upcoming feature vectors.

5 Results

5.1 Results Format and Evaluation

For this project, our data was taken from movielens [4]. Small dataset had 1,000,000 ratings that had about 4,000 movies and 7,000 users. We coded in *Julia* for implementation. Julia is a high performance language recently developed by MIT. In Julia, Funk SVD decomposition only took 3 minutes to complete for training 25 features. After we were finished with the learning step of our algorithm, we had two matrices to work with:

$$X = \underset{6040\times 25}{U} \times \underset{3706\times 25}{V^T}. \tag{7}$$

We then used the two matrices to predict the data entries that we "saved" for our testing set. This means the model does not know anything about the actual values, but it predicts them based on the trends it has observed so far.

The RMSE value for this run was 0.9228, which is rather fair when compared to 0.8558 of the actual Neftlix Algorithm. RMSE "heavily punishes errors" that are bigger than one, therefore, for a majority of cases the difference between our predicted value and real rating is smaller than one. Ratings have a 5-point scale, so we can say that the algorithm is at least 80% accurate on average.

5.2 Implementing the Results

It is important to remember that this is not exactly a suggestion algorithm but instead a system that finds trends within given a dataset in order to make predictions. This means we can do more than just predict movie ratings. Below are a few things that we can do with the discovered trends:

1. Find Similar Movies

   Every movie can be described through a certain amount of features that it contains. For example, one movie can have 0.9 "action", 0.1 "romance", 0.3 "indie" etc. Another movie can have 0.8 "action", 0.2 "romance", 0.4 "indie". We can see that movies have a similar "portrait" of features. Thus, it is fair to assume that users perceive those two movies similarly.

   We can qualitatively measure the similarity between movies by the projection of one movie onto another. This is accomplished using the normalized inner product between any two movies (also known as the cosine, or dot product). Smaller angles indicate greater similarities:

$$Similarity = \cos(\theta) = \frac{A \cdot B}{||A||\ ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (8)$$

   Below in Figure 3 is an example of such an approach. We have to remember that our algorithm does not know anything about the movies other than the features it has found to describe them. Although it may makes sense to us that the predicted movies are in fact similar, the computer has no way of knowing what these movies are about. Herein lies the beauty of unsupervised learning; movie content is not necessary in predicting movie similarity, all that is required is a sufficient number of movie ratings from users. The fact that this approach

works as well as it does is quite fascinating!

```
similar_items(model, "Die Hard (1988)")
```

```
10-element Array{String,1}:
 "Die Hard (1988)"
 "Lethal Weapon (1987)"
 "Terminator 2: Judgment Day (1991)"
 "Total Recall (1990)"
 "Terminator, The (1984)"
 "Jurassic Park (1993)"
 "Aliens (1986)"
 "Lethal Weapon 2 (1989)"
 "Rocky (1976)"
 "Planet of the Apes (1968)"
```

Figure 3

2. Find Similar Users

   Applying the same cosine similarity formula to UserFeatures as opposed to ItemFeatures, we can get most similar users. Even though it is not as useful in movies, one might think of how useful would this feature be on social media platforms. For example, you are really into sci-fi books on Goodreads. What if you just want to chat about the books you read with someone who has similar taste. The easy answer lies in suggesting a new friend to you who likes similar books.

3. Rating Predictions

   We can not forget the main purpose of this algorithm which is to predict user response to movies. To do this we dot product users' features with movies' features. It's just like knowing how much the user is into specific feature (“action”, “comedy”, “drama” etc.) and how much of these features does a given movie have.

$$r_{ui} = p_u^T q_i \tag{9}$$

6 Conclusions

Today's recommender algorithms are a giant step-up from what they used to be before Neflix motivated research with the Netflix prize. They have a wide range of applications that could be explored, but we focused on a standard movie-recommendation application. Even though we can't always explain the results we get from Machine Learning techniques, they remain incredibly efficient. In choosing the Funk SVD technique, we sacrifice knowledge of how the algorithm arrived at the results it did. However, in term we benefit from increasedefficiency and accuracy of our predictions in comparison to more straightforward techniques.

References

[1] Quora: "How does the Netflix movie recommendation algorithm work?". `http://www.quora.com/How-does-the-Netflix-movie-recommendation-algorithm-work`. Accessed: 2015-04-20.

[2] Kaggle: "RMSE". `https://www.kaggle.com/wiki/RootMeanSquaredError`. Accessed: 2015-04-20.

[3] Youtube: "2.5 - Gradient Descent - [Machine Learning] By Andrew Ng.

[4] Grouplens: "MOVIELENS". `http://grouplens.org/datasets/movielens/`. Accessed: 2015-04-20.

[5] Research.yahoo. `http://research.yahoo.com/files/kdd08koren.pdf`. Accessed: 2015-04-20.

[6] Sifter. `http://sifter.org/~simon/journal/20061211.html`. Accessed: 2015-04-20.

[7] github. https://github.com/aaw/IncrementalSVD.jl.