Recommendation Algorithms

Advanced Data Analytics Final Project

Deepa Borkar

Due: December 1, 2021

Contents

1 Brief Description **3**

2 Abstract **3**

3 Introduction **3**

4 Methods **4**
4.1 Singular Value Decomposition 4
4.2 Non-Negative Matrix Factorization 5
4.3 Stochastic Gradient Descent 5

5 Results **7**
5.1 Writing Methods from Scratch 7
5.2 Using Methods from Python Packages 8

6 Pitfalls **11**

7 Future Work **11**

8 Lessons Learned **12**

9 Conclusion **12**

10 GitHub Repository **12**
10.1 Links 12
10.2 Python Files 13
10.2.1 read_data.py 13
10.2.2 final_from_scratch.py 13
10.2.3 final_from_pkg.py 13

11 Bibliography **13**

1 Brief Description

Over the years, Netflix has become one of the main streaming services for entertainment content. Even with many new competitors, Netflix still continues to remain relevant. This begs the question as to how Netflix continues to keep new and existing viewers interested in their platform, and the answer is their use of big data and machine learning techniques, like recommendation algorithms. This project will explore these recommendation algorithms in further detail.

2 Abstract

There are two main paradigms for recommendation systems: collaborative filtering and content based methods. In terms of Netflix and many popular recommendation system applications, collaborative filtering seems to be the most commonly used method for prediction. Collaborative filtering uses primarily the user-item interaction data for providing recommendations. The most common collaborative filtering method is matrix factorization or dimensionality reduction. This report will explore two common matrix factorization methods: singular value decomposition (SVD) and non-negative matrix factorization (NMF).

3 Introduction

As mentioned previously, the collaborative filtering paradigm uses primarily the user-item interaction data, sometimes called the utility matrix, in order to provide recommendations. An example of a utility matrix is shown in Figure 1.

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Figure 1: Utility matrix representing ratings on a scale of 1-5 (Source: MMDS)

The known ratings are shown in the utility matrix with the users representing the rows and the movies representing the columns. The blank entries are the ratings that need to be predicted in order to determine which new movies should be recommended to the users. This is where matrix factorization or dimensionality reduction becomes important. Singular value decomposition (SVD) and non-negative matrix factorization (NMF) are ways to factorize the sparse utility matrix into smaller dense matrices that can help provide rating estimates for movies that users have not yet rated.

4 Methods

The two methods that are explored in this report are SVD and NMF, and the algorithm that is used to solve the matrix factorization problem for both methods is stochastic gradient descent. These methods and algorithm are explained in the following subsections.

4.1 Singular Value Decomposition

The main idea of singular value decomposition (SVD) is that a sparse matrix, like the utility matrix, can be decomposed into 3 smaller dense matrices.

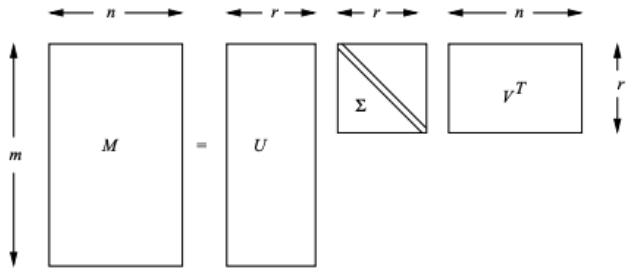

Figure 2: SVD Matrices (Source: MMDS)

These 3 matrices that make up the utility matrix can be thought of as representing the concepts or latent features that are hidden in the utility matrix. In Figure 2, the matrices U and $V.T$ are unitary matrices, meaning if the matrices are multiplied by the transpose of themselves, the product is the identity matrix. Matrix U connects the users to the concepts or latent features, and matrix V connects the movies or items to the concepts. Lastly, the matrix Σ is a diagonal matrix that defines the strength of each of the concepts. Oftentimes, the number of concepts or latent features of the utility matrix is tied to the rank of the utility matrix.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix} \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}$$

$$M \qquad U \qquad \Sigma \qquad V^{\mathrm{T}}$$

Figure 3: Decomposition for Utility Matrix with Rank 2 (Source: MMDS)

An example of a simple utility matrix with rank 2 is shown in Figure 3. In this example, there are only 2 concepts or latent features hidden within the utility matrix. These concepts or latent features could represent genres, like science fiction or romance, or groups of movies that have similar cast members. With a real dataset, like the rating data that Netflix has, there will be many more concepts and latent features that are hidden within the utility matrix.
To further simplify this problem, the matrix factorization can be rewritten in the following way:

$$\begin{aligned} M &= U\Sigma V^T && (1) \\ &= (U\Sigma^{\frac{1}{2}})(\Sigma^{\frac{1}{2}}V^T) && (2) \\ &= \tilde{U}\tilde{V}^T && (3) \end{aligned}$$

The utility matrix is then decomposed into two matrices, which can be solved for using stochastic gradient descent (SGD). The algorithm used in this report will be further described in the SGD section.

4.2 Non-Negative Matrix Factorization

Non-negative matrix factorization (NMF) is very similar to SVD, except the non-negative utility matrix is decomposed into two non-negative matrices. The matrix decomposition is similar to the decomposition used for SVD; however, there is a constraint that the entries for the matrix factors are non-negative values. Specifically, NMF finds two non-negative matrices $W \epsilon R^{mxr}$ and $H \epsilon R^{rxn}$, such that the following is true about the utility matrix M:

$$M \approx WH \quad (4)$$

This matrix factorization is found using stochastic gradient descent, similar to SVD, and with the constraint that W and H must be non-negative matrices.

4.3 Stochastic Gradient Descent

The SVD and NMF methods were solved using the stochastic gradient descent algorithm. For both methods, the matrix factors need to be calculated for the utility matrix. In order to do this, the Frobenius norm can be used, which means minimizing the difference between the actual rating values (utility matrix M) and the product of the matrix factors ($\tilde{U}\tilde{V}^T$) as shown in the following.

$$\min f(\tilde{U}, \tilde{V}^T) \equiv ||M - \tilde{U}\tilde{V}^T||_F^2 \quad (5)$$

In order to find the matrices $\tilde{U}$ and $\tilde{V}^T$ that results in the minimum of the

function, gradient descent can be used. The partial derivatives of the function are calculated below:

$$\frac{\partial f}{\partial \tilde{U}} = -2\tilde{V}^T(M - \tilde{U}\tilde{V}^T) \quad (6)$$

$$\frac{\partial f}{\partial \tilde{V}^T} = -2\tilde{U}(M - \tilde{U}\tilde{V}^T) \quad (7)$$

The partial derivatives are shown in matrix forms above; however, in practice for this report, each known rating was looked at separately through iterations, which will be shown in more detail in the code and the Results section. An example of how the ratings are calculated is shown in Figure 4.

$$\begin{pmatrix} & r_{ui} & \end{pmatrix} = \begin{pmatrix} — & p_u & — \end{pmatrix} \begin{pmatrix} | \\ q_i \\ | \end{pmatrix}$$

Figure 4: Value of each rating is the dot product of two vectors (Source: PyParis 2017)

Therefore, the updates for the matrix factors will be the following, where α is the learning rate:

$$p_u \leftarrow p_u + 2\alpha q_i(r_{ui} - p_u \cdot q_i) \quad (8)$$

$$q_i \leftarrow q_i + 2\alpha p_u(r_{ui} - p_u \cdot q_i) \quad (9)$$

The algorithm I used for stochastic gradient is shown in the following Python code, which is slightly modified from the project code shown in my GitHub repo for this project.

```
def sgd(M, latent_components, learning_rate, epochs):
    m = M.shape[0]
    n = M.shape[1]
    r = latent_components

    # randomly initialize the matrices u and i
    U = np.random.uniform(0, 1, (m, r))
    V = np.random.uniform(0, 1, (n, r))

    # go through epochs
    for e in range(epochs):
        # go through all the ratings in M
        for row, col in np.ndindex(M.shape):
            # only look at the ratings that exist (not equal to zero)
            if M[row, col] != 0:
                rating = M[row, col]
                # updating matrices with partial derivatives
                U[row] += 2 * learning_rate * (rating - U[row].dot(V[col])) * V[col]
                V[col] += 2 * learning_rate * (rating - U[row].dot(V[col])) * U[row]

    return U, V.T
```

The gradient descent process will be similar for both SVD and NMF, except NMF has the constraint that the matrix factors must be non-negative.

5 Results

The dataset used is from the Netflix Prize Data Kaggle Competition. The dataset has rating information (from 1-5) for 480,189 users and 17,770 movies. I used a much smaller sample size to test out my methods that I wrote from scratch and also used methods from the python package sklearn for comparison purposes.

In order to evaluate the results, I used the root mean square error (RMSE) to compare known ratings and estimated ratings. In the Future Work section, I will discuss how this technique should be further used to more correctly evaluate the work by calculating the RMSE for test data after training on the known data instead of just calculating the RMSE for just the known data as shown in this Results section.

5.1 Writing Methods from Scratch

Because of performance and time, I used a very small dataset in order to test out the SVD and NMF methods that I wrote from scratch. The sample data set only included about 300 users and 10 movies and I did only pick users that

rated multiple movies (excluded users that only rated 1 movie) in order to have a slightly more dense utility matrix. Because of this sampling, there is most likely some bias in the results and this should be factored in when reviewing the results.

Figure 5 shows the root mean square error (RMSE) of the small sample dataset for 3000 epoch runs and a learning rate of 0.001 in order to compare the SVD and NMF methods. At the end of the training, the RMSE for the SVD method is 0.004997 and the RMSE for the NMF method is 0.007293. This shows how similar these two methods are and to evaluate the differences further, more work needs to be done with a larger dataset and a comparison of the RMSE for a qualifying dataset.

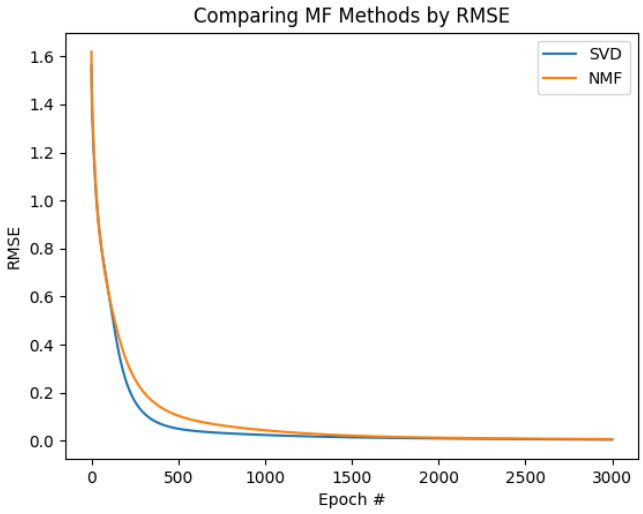

Figure 5: Comparing RMSE for Small Sample Data

5.2 Using Methods from Python Packages

In order to continue exploring the differences between the SVD and NMF methods, I also used the methods from the Python package sklearn. I wanted to vary the sample size in order to get a better idea of the RMSE results. For this dataset, I still sampled from the Netflix Prize dataset but I did not only pick users that rated multiple movies. I just varied the amount of users and movies picked from the dataset from 100 users and 100 movies to 5000 users and 5000 movies. I then used the Python packages to obtain the rating estimates and then calculated the RMSE for the known ratings. Again, to better evaluate these training models, it would be good to compare the RMSE of the qualifying or test dataset that was not included in the original training set.

Figure 6 displays the RMSE for the SVD and NMF methods for varying sample sizes and it seems in general SVD has a slightly lower RMSE than the RMSE for the NMF method but these RMSE are from the training set and are only for a small portion of the large Netflix dataset, so more differences might be found when applying the methods for the entire dataset and looking at the RMSE of a validation dataset. For the largest sample size of 5000 users and 5000 movies, the RMSE for the SVD method was $2.416e-15$ and the RMSE for the NMF method was $3.217e-10$.

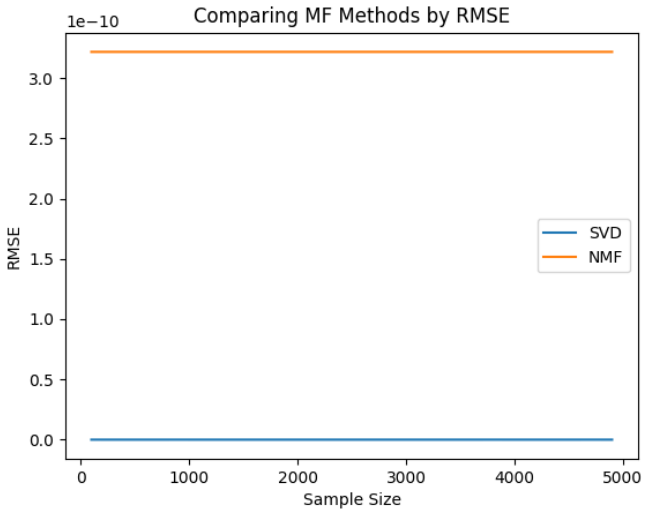

Figure 6: Comparing RMSE for Varying Sample Sizes

Because of the scale it is a little difficult to see these lines clearly so I plotted them individually to show the slight fluctuations in RMSE. Both lines are still very low and close to 0 because these are the RMSEs for the training set. An RMSE for a test or probe set may help to evaluate these methods further.

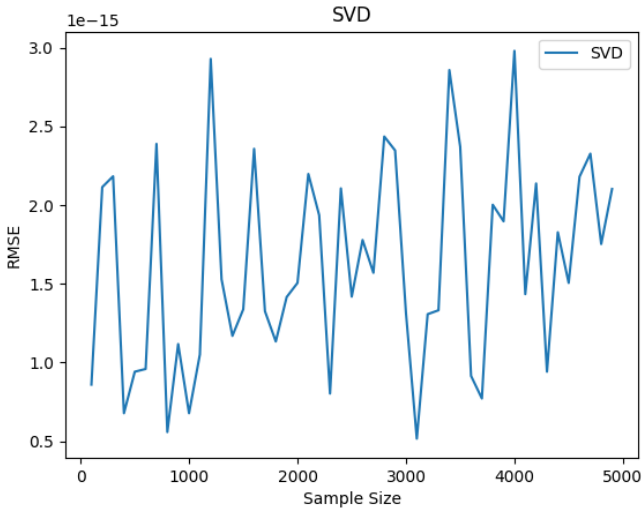

Figure 7: RMSE for SVD for Varying Sample Sizes

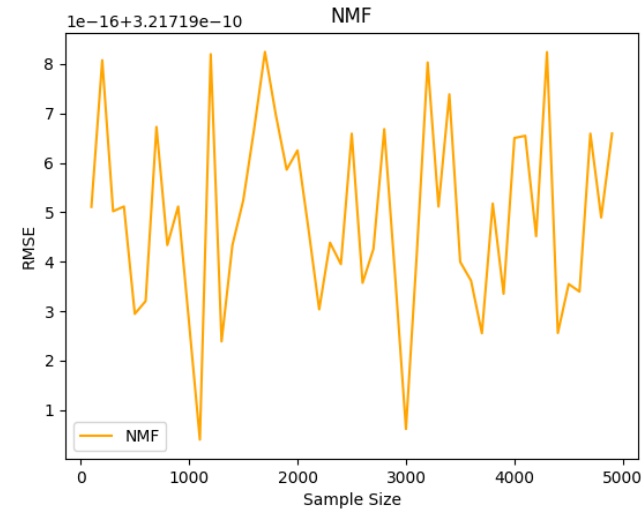

Figure 8: RMSE for NMF for Varying Sample Sizes

6 Pitfalls

After writing the methods for SVD and NMF from scratch, the main challenge was trying to run these methods on the entire Netflix Prize Dataset. The program kept crashing with a memory issue because my computer resources did not have the space and computing power to train the entire dataset. This is mainly why I needed to use a smaller dataset. I also tried using Python packages for the SVD and NMF methods because I thought the issue was due to performance, but the same memory issues were happening for the packages but I was able to train larger sample sizes a little quicker with the packages.

Another issue is that my way of evaluating the methods was not completely unbiased because I am calculating the RMSE off the training set. In order to have more accurate evaluation methods, it would be better to have a validation or qualification set. The qualification data ratings were not released with the Netflix Price Dataset, but it might have been good to sample some training data and some test data for validation, but the training and test data would need to have similar users for a good comparison, which is not necessarily a trivial problem.

7 Future Work

As mentioned in the Pitfalls section, there were some challenges with this project. With more time, it would be good to explore a few different areas.

To address the performance and time pitfalls mentioned previously, it would be good to run the program potentially on AWS or another service that provides more space and computing power to evaluate a utility matrix that includes all of the Netflix Prize dataset.

Adding regularization to the stochastic gradient descent process would also help to tune the learning process. Regularization can help avoid overfitting the data. L1 regularization is particularly useful for sparse data.

Another thing to address with this project is improving the evaluation process of the methods. It would be good to set up a way to sample the data for a training set and test set that had similar users so the RMSE of the training set can be used to predict the ratings and the RMSE of the test set can be used to evaluate the accuracy of the predicted ratings. Adding more randomization to the experiments would be useful as well. Currently just the first 1000-5000 users and movies are selected because it is simpler for matrix indexing purposes, but randomly sampling the dataset and then creating the utility matrix would improve the evaluation method.

8 Lessons Learned

I really enjoyed learning more about recommendation systems and would like to continue learning more about these systems. These experiments for this report were very interesting and I can think of more things to do to improve this report as mentioned in the Future Work section. In a previous class, we learned more about gradient descent, so it was interesting to apply that algorithm to this application. In addition to programming the SVD and NMF methods from scratch, I also learned how to use the SVD and NMF methods from the sklearn package.

9 Conclusion

From this report, it seems that the method SVD leads to a slightly lower RMSE than the method NMF. However, more evaluation would need to be done on larger datasets and with validation sets to confirm the differences between the matrix factorization methods. From the findings from this report though, it seems to make sense that SVD would have a slightly lower RMSE than NMF because the NMF method requires a constraint that the matrix factors are all non-negative matrices. However, for more accuracy, it seems that allowing negative values in the matrix factors may lead to a more accurate estimate. Nevertheless, both methods SVD and NMF seem to be very popular in predicting recommendations in the world of streaming services, so both should be considered when looking into recommendation systems.

10 GitHub Repository

Because the data files were very large, even compressed, I could not push them to the github repo due to size limitations. In order to get the data to run the read_data.py file, the data file combined_data_0.txt from the Netflix Prize Kaggle Competition link should be downloaded into the data directory of the github repo. The other combined data files can also be downloaded into the data directory as well if more data is needed.

There is a README.txt file in the github repo explaining all of the python files and how to run the files but I have included descriptions here as well for further convenience.

10.1 Links

Link to GitHub Repo: `https://github.com/deepapborkar/ADAfinalproject`

Link to Netflix Prize Kaggle Competition: `https://www.kaggle.com/netflix-inc/netflix-prize-data`

10.2 Python Files

The README.txt file in the github repo has instructions on how to the run the files, but I have provided a short description of the Python files here for more information.

10.2.1 read_data.py

This python program reads in data from combined_data_*.txt, sample_data.txt, and test_data.txt files and writes the data in an organized way to data.csv, sample_data.csv, and test_data.csv files respectively. These csv files are then used in the following python programs: final_from_scratch.py and final_from_pkg.py.

```
python3 read_data.py
```

10.2.2 final_from_scratch.py

This python program has the functions for SGD, SVD, NMF, and RMSE. These functions are then used for the test_data.csv and sample_data.csv files to perform analysis. The data.csv file is a little too large for this program and may cause a crash because of memory issues.

```
python3 final_from_scratch.py <path to data file>
Example:
python3 final_from_scratch.py data/sample_data.csv
```

10.2.3 final_from_pkg.py

This python program includes analysis using the sklearn packages for TruncatedSVD and NMF. The input for this program can be data.csv.

```
python3 final_from_pkg.py <path to data file>
Example:
python3 final_from_pkg.py data/data.csv
```

11 Bibliography

"Find Open Datasets and Machine Learning Projects." Kaggle, https://www.kaggle.com/datasets.

Hug, Nicolas. "Understanding Matrix Factorization for Recommendation." 15 June 2017, http://nicolas-hug.com/blog/matrix_facto_2.

Leskovec, Jure, et al. Mining of Massive Datasets. Cambridge University Press, 2020.

Lin, Chih-Jen. "Projected Gradient Methods for Nonnegative Matrix Factorization." Neural Computation, vol. 19, no. 10, 2007, pp. 2756–2779.,

https://doi.org/10.1162/neco.2007.19.10.2756.

"The Netflix Prize and Singular Value Decomposition."
https://pantelis.github.io/cs301/docs/common/lectures/recommenders/netflix/the-netflix-prize-and-singular-value-decomposition.

Raghuwanshi, Sandeep Kumar, and Rajesh Kumar Pateriya. "Accelerated Singular Value Decomposition (ASVD) Using Momentum Based Gradient Descent Optimization." Journal of King Saud University - Computer and Information Sciences, vol. 33, no. 4, 2021, pp. 447–452., https://doi.org/10.1016/j.jksuci.2018.03.012.

Silva, Thalles. "Understanding Linear Regression Using the Singular Value Decomposition." Thalles' Blog, https://sthalles.github.io/svd-for-regression/.

Zadeh, Reza. "Singular Value Decomposition." ICME at Stanford.
https://stanford.edu/ rezab/dao/notes/svd.pdf. Accessed 1 Dec. 2021.