Data Mining Final Semester Project: Netflix Prize Dataset Recommender System

Aaron Gauthier, Pedro Uria Rodriguez, Zachary Buckley

The George Washington University
Data Science Program
DATS6103: Intro to Data Mining

Submitted
22 April 2019

1.- Introduction

1.1 Background

The problem we have selected is to predict Netflix user ratings for movies based on their previous movie ratings and those of other users. We have selected this because we are aware that Netflix provided anonymous customer ratings previously, and believe that we can expand on a competition dataset that was provided between 2006 and 2009 to potentially get good prediction results using a combination of data-mining techniques we've learned in class.

The Netflix Prize competition commenced on October 2, 2006, and would continue until the grand prize was awarded, unless Netflix decided to cancel the competition (which they could do after October 2, 2011). The grand prize would be awarded to the first participating team to improve on Netflix's prediction algorithm (called CineMatch) by at least a 10% margin based on RMSE against a test set that wasn't provided to the participants. On September 21, 2009 "The Netflix Prize" and $1 million dollars, were awarded to team "BellKor's Pragmatic Chaos". They were able to achieve a 10.6% improvement over Cinematch's score from 2006 on the test dataset by using ensemble methods to combine the results of several of the top teams algorithms.

1.2 Data

The original dataset included 100,480,507 user ratings from 480,189 users given to 17,770 movies. The user and movie fields are integer ID's while the ratings were

integer values between 1 and 5 (stars). The original contest required that the participating team's algorithm predict ratings on the entire qualifying dataset, but they were only informed of the score for half of the data (i.e. the "quiz set") of 1,408,342 ratings. The second half of the 1,408,789 were used by the judging panel to determine the potential prize winners. Only the judges knew which ratings were in the quiz set and which were in the test set. Competition results were released once every year, but didn't include the RMSE scores on the test set, until after the competitions grand prize winner was announced. This arrangement was intended to ensure no team could 'train' their model to the test set through multiple submissions, making it necessary for the algorithms being developed to strike the right balance of bias towards the training data, in order to avoid overfitting, which would lead to worse scores on the training set.

Submitted predictions were scored against the true grades in terms of Root Mean Squared Error (RMSE). The goal was to reduce this error as much as possible. Lastly, Netflix chose to identify a "probe" subject of 1,408,395 ratings within the training dataset. These values chosen had similar statistical properties compared to the training dataset. A separate file for movie title and year of movie release was provided as part of the dataset. There was no information provided about the users.

Netflix attempted to protect the privacy of customer data by deliberately changed some of the data to make it more synthetic. This means some of the ratings were deleted, modified, or entirely generated based on the original dataset from netflix's customers. The training set was arranged such that the average user had provided over 200 movie ratings. On average, each movie was rated over 5,000 times. Some of the

movies have as few as 3 reviews, and one user actually rated over 17,000 movies, so there is a lot of variability in the provided data. There was some controversy with regards to this competition. Many have argued that a 10% reduction in the RMSE would not benefit the users. Netflix claims that even a small improvement as little as 1% RMSE results in a significant difference in the rankings amongst the top 10 most recommended movies for a user.

Kaggle.com has a repackaged version of the dataset Netflix provided for the competition. (https://www.kaggle.com/netflix-inc/netflix-prize-data). The original Netflix dataset included a tar file with over 17000 files, one file per movie. The data in these files was repackaged into just 4 files, which were then compressed. These repacked and compressed files were the point we started our analysis from. The data doesn't seem to have too many feature number problem, as once the data is combined into a single table, the vast majority of the data revolves around only 4 features (movie_id, user_id, rating, and date). But, it will suffer from the Curse of Dimensionality due to the very large number of instances (100,480,507).

1.3 Predictive Models

We've used two different approaches in attempting to provide recommendations, both based on a heavily reduced dataset, from that provided by netflix for the competition:

1. Item-Item Collaborative Filtering: we compute the correlation between movies based on the user ratings, and recommend movies that are highly correlated to movies the user already likes.

2. Linear Regression: we cluster the users, and then predict their ratings on movies they haven't seen. We then recommend movies which we predicted the user will like more.

1.4 Frameworks

In our work on the netflix prize dataset, we've utilized the following tools and packages in the python ecosystem:

- Core Python Packages: os, sys
- 3rd Party Python Packages: scikit-learn, NumPy, pandas, PyQt5
- Integrated Development Environments (IDEs): pycharm, jupyterlab

1.5 Graphical User Interface (GUI)

We focused our GUI development on the data preloading and preprocessing steps. The gui allows the user to quickly downsample the data provided by netflix, and save it off for use in subsequent analysis. The user can specify the minimum number of ratings that netflix customers must have to be included in the downsampled dataset, the minimum number of ratings that movies must have to be included, and the random state value used for the simple random sampling without replacement.

The GUI was built using the pyQt5 gui framework, and the qt designer application. Each operation available to the user from the GUI is performed in a separate QThread, to avoid hanging the main gui event thread. The GUI provided cross-platform support by utilizing the os.path utilities provided by python3 to manipulate file paths, and by utilizing relative paths wherever possible. Progress bars

are used to indicate incremental progress for all operations on the dataset, though in some cases the incremental progress reported is limited, as functions from pandas and sklearn don't provide a mechanism for giving incremental progress updates during execution. The GUI operation is broken down into the following steps:

1. Decompress the netflix prize data provided by kaggle (same as original netflix dataset, though packaged slightly differently).
2. Parse and Load into memory the entirety of the 100,480,507 movie ratings, including the user_id, movie_id, and the rating.
3. Dimensionality Reduction
    a. Reduce the number of movies to be included based on a cut-off that specifies the minimum number of ratings a movie must have.
    b. Reduce the number of users to be included based on a cut-off that specifies the minimum number of ratings that a user must have made.
    c. Finally Reduce the dataset further using simple random sample on the users to be included.
4. Save to CSV
    a. This will save off the reduced dataset to a csv file located in the Data Directory. (Data/netflix-prize/downsampled-csv/few_samples.csv)

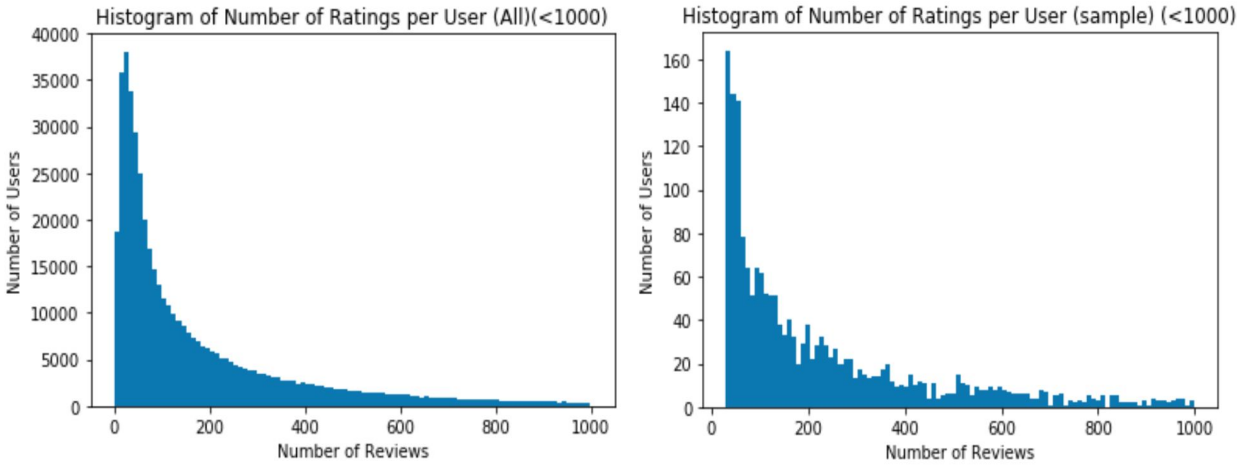
2.- EDA & Preprocessing

We had four .txt files, which we combined into one dataframe. Due to the size, our initial attempt to build the combined dataframe incrementally was a very slow operation. We managed to speed this up through many incremental adjustments to optimize the memory usage management of the function parsing the files. We eventually landed on a solution based on a stackoverflow post regarding the incremental creation of scipy sparse matrices of the Coordinate format. We've adapted that solution, and after several modifications to it, are able to utilize the python array package to construct a c-style array to collect up the movie_id, user_id and ratings information, and then directly transform it into a couple numpy arrays through numpy's from Buffer operation. We can then build the pandas DataFrame relatively quickly by using a dictionary of column names to numpy arrays (this code is located in preprocessing/downsample.py).

The number of observations in the combined DataFrame is 100,477,253, each one contains movie_id, user_id, and rating. The date that the user made a review, is also provided by netflix, but as we aren't using it for our analysis, we aren't parsing it into the dataframe (which does help reduce the load times). This was obviously much more than what our machines could handle using conventional statistical models, at least not within a reasonable amount of time without some kind of parallel processing and graphical processing units (GPUs) acceleration, which is beyond the scope of the course.

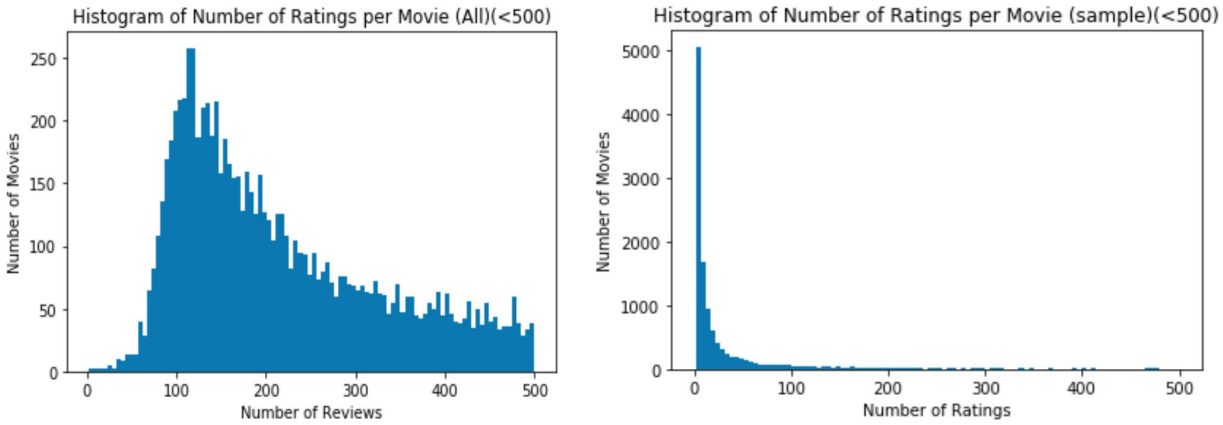
In order to proceed with our analysis, we needed to reduce the number of instances in the dataset. After some experimenting, we decided to drop any users with less than 30 ratings, and any movies with less than 214 ratings. This helped a bit, but our dataset was still massive (98,143,553 ratings), so after this we used simple random sampling without replacement to select 0.5% of the users, and dropped the rest, giving us a final smaller dataframe of the netflix data consisting of 510,852 ratings, 1,934 users, and 11,866 movies. After some experimentation, we felt this dataset had a reasonable dimensionality for us to proceed with the Item-Item Collaborative Filtering, and Linear Regression techniques mentioned previously. This initial data reduction process was built into a GUI to aid us in experimenting with these parameters, and saving off the downsampled dataframe contents.

Now, let's look at how the original dataset, and our reduced dataset compare:

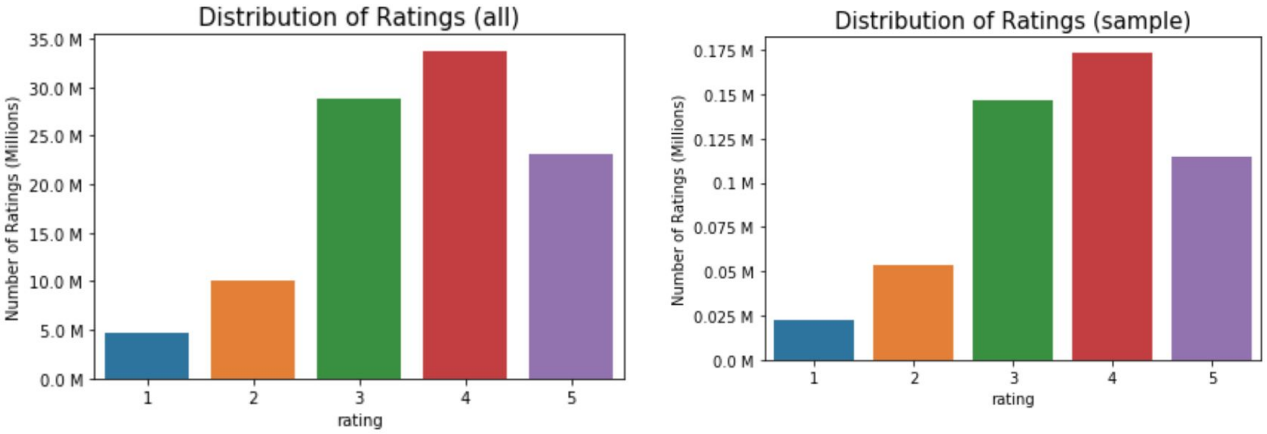


The graphs above show the distribution of the number of ratings each user made, both for the original dataset (left), and for our reduced sample (right). We restricted the graph to only include less than 100 review users, as otherwise the x-axis expands to about 17,100. Which implies that at least one users has rated every movie

netflix provided in the dataset. We can see that while some noise has been introduced, we've generally maintained the distribution behind the number of reviews per user after applying our data reduction techniques.



The graphs above shows the distribution of the number of ratings each movie has received by users, both for the original dataset (left), and for our reduced sample (right). We restricted the graph to only include less than 500 review movies, as otherwise the x-axis expands to about 200,000 reviews. Which implies that in the original data at least one movie had been reviewed by almost half the users netflix gave us data for. It's worth noting that despite us cutting out movies in the original dataset with less than 214 ratings, very few reviews in the reduced sample have more than 214 ratings. This is actually expected, as our Random Sampling took .5% of the users, and didn't affect the number of movies. As a result the number of ratings for every movie was reduced as well, while we didn't further decrease the number of movies.



The graphs above show the distribution of ratings both for the original dataset (left), and our reduced sample (right). You can see that in our downsampled dataset the distribution of focused in the upper range, between three and five (stars). These two graphs are consistent, showing that we very nicely maintaining the ratings distribution between the original dataset, and our reduced sample.

3.- Linear Regression

In setting up our Linear Regression approach we'll first identify a cluster of similar users. We compute the distance between permutations of user pairs, and then get a list of the most similar users for each user, based on the similarity of their ratings. We used the following distance metric:

$$d(A,B) = \frac{1}{n \cdot 5^2} \sum_i (r_{A_i} - r_{B_i})^2$$

Where r_{A_i} is the rating of user *A* to movie *i*, and *n* is the number of movies both users *A* and *B* have rated. After computing the distance between each user, which took about one and a half hours, we assigned to the same cluster all B_j such that $d(A, B_j)$ is less than 0.02, meaning that on average, the ratings of *A* and B_j for each common movie are within a 0.7 absolute distance (For example, $r_{A_i} = 5$ and $r_{B_i} = 4.3$). All the distances were saved in json format as we needed to experiment with this threshold without needing to re-compute the distance calculations over and over. The clusters using the final threshold value of 0.02 were also saved in json format. Note that the scikit-learn K-Means algorithm isn't well suited to this problem without some adaptation.

It is also worth noting that the clusters we are defining are not discrete, in that a user can (and will) belong to multiple clusters. 'Clusters' may not be the right nomenclature, as we'll have one of these clusters for each users in our reduced dataset. We'll use this concept of clustering to construct a predictor variable for our linear regression to use. For each user, we're going to construct the following dataframe:

movie	`cluster_avg_rating`	`user_A_rating`
movie_1	Known	Maybe Known
movie_2	Known	Maybe Known
...	....	...

We can then train a regression model for that specific user, and predict their ratings for movies they've not yet seen. We briefly considered attempting to use Lasso or Ridge regression techniques for this problem, but quickly realized that as those techniques are tailored toward different forms of feature reduction it wasn't really a good idea (as we only have one predictor to work with).

After experimenting with different distance thresholds, we found out 0.02 to be a good middle-ground in general. The overall R^2 = 0.65 while the training and testing set are big enough for most users (73 movies on average per user). The individuals R^2 vary greatly, getting perfect ones for some users and even negative ones for others. This suggests that this model is only suitable to some users, but that could also be used to identificate this abnormal users, which then can be dropped before training more complex models. The distance threshold could also be automatically adjusted for each user, in order to get the better predictive power for each of them. This would most likely result in a very small threshold, but less users for which we can predict.

4.- Item-Item Collaborative Filtering

After preprocessing the data and downsampling, we decided to try recommending movies based on Item-Item Collaborative Filtering. We'll compute a measure of similarity (correlation) between various movies using the user ratings information in our reduced sample. From there we'll be able to sort the movies for those that are the most similar to the movie a user has already watched, providing recommendations for what they should watch next. An important consideration in implementing this approach is which correlation to use for our comparisons, the next couple sections describe our thought process in considering some of the options.

4.1 Pearson R Correlation:

$$r = \frac{N \sum xy - \sum (x)(y)}{\sqrt{N \sum x^2 - \sum (x^2)][N \sum y^2 - \sum (y^2)]}}$$

r = Pearson r correlation coefficient
N = number of observations
∑xy = sum of the products of paired scores
∑x = sum of x scores
∑y = sum of y scores
∑x2= sum of squared x scores
∑y2= sum of squared y scores

The Pearson r correlation is one of the most widely used correlations in statistics which also makes it one of the most misused as well. It measures the degree to which two variables present as linearly related based on the provided sample. An example

would be if you wanted to measure how two stocks correlate with each other. You may use the Pearson r correlation to measure the degree of the relationship between the two.

Variables should also be normally distributed. Assumes data is interval or ratio level. The Pearson r correlation assumes the data has the properties of magnitude and equal intervals between adjacent units. This type of measurement is possibly appropriate for the Netflix dataset and shows the strength of the correlation. We believe there is a linear relationship between the variables and we acknowledge that the data set is not perfectly normally distributed which could skew the correlations some - though we don't know to what degree. Overall possibly one of the better suited correlation methods for this dataset.

4.2 Kendall Rank Correlation:

$$\tau = \frac{n_c - n_d}{\frac{1}{2}n(n-1)}$$

Nc= number of concordant
Nd= Number of discordant

The Kendall rank correlation is a non-parametric test that measures the strength of dependence between two variables. If you look at two samples a and b, where each sample is a size of n, then we know the total number of pairings with b is $n(n-1)/2$. Kendall's correlation is distribution free and does not depend upon the assumptions of

various underlying distributions. However this type of formula is applied where there are no tied ranks - which means that Kendall's rank correlation is not well suited for the Netflix dataset and won't be taken into consideration.

4.3 Spearman Rank Correlation:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2 - 1)}$$

ρ= Spearman rank correlation
di= the difference between the ranks of corresponding variables
n= number of observations

The Spearman rank correlation is also a non-parametric test that is used to measure the degree of association between two variables. This test does not carry any assumptions about the distribution of data. It is appropriate for correlation analysis when the variables are measured on a scale that is at least ordinal in nature. The Spearman correlation assumes that the data is ordinal in nature and the scores on one variable are related to the other variable. This appears to be the most appropriate correlation measure for the Netflix dataset. However, a problem with this type of ranking is that it does not show the strength of the relationship.

4.4 Pearson R and Spearman Rank Correlation Comparison

We decided on comparing both the Pearson r and Spearman rank correlations for the item-item collaborative filtering algorithm in order to see the similarity (if any)

compared against each other for the following movies: “Miss Congeniality”, and “The Godfather”. We expect that at least a few of the same titles may appear on the initial correlation lists prior to being paired down into the final movie recommendations. There is a small possibility that the same movies may end up on the same final recommendation lists.

5.- Analyzing Recommendations

Looking at the results between the “Miss Congeniality” movie for the Pearson r and Spearman rank correlations we see that the movie “Another Stakeout” made both lists but one was ranked second in the Spearman rank correlation list versus the Pearson r correlation list which ranked it third. When comparing the results between “The Godfather” we achieved essentially the same results as the “Miss Congeniality” movie correlations. The movie “Latter Days” was found on both of the lists, however with the Spearman correlation it was listed as second and on the Pearson r correlation it was listed as third. Both the Spearman rank and the Pearson r correlation differ in results - however the correlations that are in common between both lists seem to be quite close. We also have to take into account that Netflix perturbed much of the data which may also skew the recommendation results.

Pearson R Correlation Figure 1:

movie_title	PearsonR Correlation	number_ratings
Miss Congeniality	1.000000	1099
The Cowboy Way	0.715484	44
Another Stakeout	0.705242	32
Angel: Season 2	0.673434	39
Half Past Dead	0.659582	40

movie_title	PearsonR Correlation	number_ratings
The Godfather	1.000000	1084
Dawson's Creek: Season 1	0.804613	32
Latter Days	0.799671	35
South Park: Season 5	0.737044	53
Jonah: A VeggieTales Movie	0.729285	36

Spearman Rank Correlation Figure 2:

movie_title	Spearman	number_ratings
Miss Congeniality	1.000000	1099
Another Stakeout	0.676479	32
Eulogy	0.655227	35
Center Stage	0.651307	39
The Queen of the Damned	0.638307	66

movie_title	Spearman	number_ratings
The Godfather	1.000000	1084
Latter Days	0.924062	35
Night Shift	0.756997	45
Will & Grace: Season 2	0.697571	35
Sylvia	0.685595	34

Lastly, we compared both approaches. We selected some users and for each of them we looked at the correlation of movie A with another movie B. An agreement between the two approaches would be indicated by: correlation A-B high <-> rating for movie A high; correlation A-B low <-> rating for movie A low. A discrepancy would be indicated by the opposite fact.

We found that when we compared the clustering algorithm output along with the item-item collaborative filtering we found that they did not match. We performed a small experiment to compare the results. We looked at one user's preference for the "Ice Age" movie and tried to find another movie for that user which he has not yet watched.

```
c *= np.true_divide(1, fact)
```

movie_title	Correlation	number_ratings
Ice Age	1.000000	432
Look Who's Talking Now	0.596281	101
The Lion King 1 1/2	0.570257	105
Bewitched	0.539448	108
When a Man Loves a Woman	0.526263	105

```
In [29]: y_pred.iloc[8]

Out[29]: movie_id                 7381.000000
         user_predicted_score        3.067344
         Name: 12, dtype: float64

In [30]: title2 = movie_titles[movie_titles["id"] == str(7381)]["t
         corr_movie.loc[title2]

Out[30]: Correlation         0.338701
         number_ratings    146.000000
         Name: Flubber, dtype: float64
```

Using this code, the correlations themselves can also be evaluated

After doing some searching based on the clustering we found "Flubber" although correlated low might be something this user would be interested in watching. We then took the collaborative filtering solution to seek recommendations for both "Ice Age" as

well as “Flubber” to see if either of the movies were in each other lists. What we found was not that surprising since both approaches varied slightly.

movie_title	PearsonR Correlation	number_ratings
The Lover	1.0	31
Flubber	1.0	146
Everest: IMAX	1.0	37
Eulogy	1.0	35
Bread and Tulips	1.0	43

movie_title	Spearman	number_ratings
Everest: IMAX	1.0	37
Flubber	1.0	146
A Dirty Shame	1.0	35
The Lover	1.0	31
The Long Walk Home	1.0	31

movie_title	PearsonR Correlation	number_ratings
Ice Age	1.000000	431
Last of the Dogmen	0.815368	31
CSI: Miami: Season 1	0.782303	31
Rivers and Tides	0.774597	34
Crocodile Dundee in Los Angeles	0.709299	43

movie_title	Spearman	number_ratings
Ice Age	1.000000	431
CSI: Miami: Season 1	0.816727	31
Last of the Dogmen	0.782195	31
Eulogy	0.778216	35
Protocol	0.753464	32

“Ice Age” nor “Flubber” were on either of the correlation lists. The initial lists prior to being paired down to the top recommendations were expanded to incorporate 50 recommendations and still neither “Flubber” nor “Ice Age” were included in the lists.

Collectively as a group we learned about what goes into the construction of a very basic recommender system. We learned that recommender systems are more complex than we realized. We learned that pyQT5 is similar to Java Swing and it takes a fair amount of time integrate code into it. We experienced the curse of dimensionality on this project in regards to “Big Data”,(i.e. 100 million records, 480,000 users and 17,770 movies).

Future improvements regarding a Netflix Movie Recommendation System could include researching the implementation of Self Organizing Maps (SOM or SOFM) for this project. Utilize IMDB, MovieLens, Gross Revenue, metadata and other possible databases to augment and

improve the baseline data from Netflix. In order to ingest “Big Data” utilize parallel computing, Graphical Processing Units (GPU), and Neural Networks. It would be useful to experiment with sparse matrices to optimize collaborative filtering algorithms.

6.- Conclusion

We initially set out on this project with the goal of improving on the naive implementation of the netflix recommendation system (Cinematch) by utilizing the netflix prize dataset provided for the 2006 competition. Our experience working with this data has been rather challenging, from the realization that the data we'd selected is much larger than datasets we've worked with in the past, to figuring out how to do more traditional EDA on said data. Many of the algorithms we've thus far discussed in class aren't well suited to over 100 million entries, without a good deal of data reduction. Our efforts have yielded somewhat functional recommendation systems. One based on a relatively conventional Item-Item Collaborative Filtering Approach, and the other based on building up a linear regression problem by ultimately extracting a feature from users that are near-by.

Not surprisingly, we've confirmed that different approaches to getting recommendations will yield different results, though we did some similarities in the results they weren't in very strong agreement. Our original goal of analyzing our predictive capability with the netflix prize test set, was shattered by the realization as got deeper into the task, that we were unable to get our hands on the original test data. Due to a pending lawsuit, which was eventually settled, netflix never released the test data, instead only releasing the resulting RMSE values which were used in comparing the competitors. They also cancelled the second netflix competition, which would have included even more user information, as the lawsuit was specifically regarding the

anonymity of the data we've used for this project. Further research is needed, but we believe pulling in additional feature data for the Linear Regression based approach from external sources would drastically improve the accuracy of the somewhat custom recommendation approach we devised. We also know that ensemble collaborative filtering approaches have been extremely successful in this field (grand prize winners have used them in the past), and it would be very interesting to attempt adapting those approaches with additional feature data as well.

7.- Appendix A: Definitions

1. Algorithm - A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

2. Collaborative Filtering - Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is such that if person A has the same opinion of person B about something, then A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

3. Correlation - A correlation coefficient measures the extent to which two variables tend to change together. The coefficient describes both the strength and direction of the relationship.

9.- References

https://www.kaggle.com/netflix-inc/netflix-prize-data

https://www.netflixprize.com/faq.html

https://en.wikipedia.org/wiki/Netflix_Prize

https://www.statisticssolutions.com/correlation-pearson-kendall-spearman/

https://github.com/amir-jafari/Data-Mining

Stanford Lecture Series 41-45: Overview on Recommender Systems:

- https://www.youtube.com/watch?v=6BTLobS7AU8
- https://www.youtube.com/watch?v=2uxXPzm-7FY
- https://www.youtube.com/watch?v=1JRrCEgiyHM
- https://www.youtube.com/watch?v=h9gpufJFF-0
- https://www.youtube.com/watch?v=VZKMyTaLI00