A Movie Recommendation System that uses Content Based Filtering, Collaborative Filtering and Clustering

María E. Ramos Morales
CS-685: Data Mining

May 1, 2014

1 Project Description

For this project I created a movie recommendation system, a program that essentially predicts the ratings that users would give to movies they have not seen based on ratings they and other users gave. I tested the system with the MovieLens dataset. I used content based filtering (the cosine similarity algorithm) and collaborative filtering (the Slope One algorithm) for the recommendation system, as well as clustering. Content based filtering serves to find similar movies/items so to predict ratings based on such similarities, collaborative filtering serves to predict ratings based on the ratings given by other users/viewers and clustering serves to improve the scalability and quality of the results acquired from collaborative filtering.

2 Related Work

The papers I used as reference for the project are "Big and Personal: data and models behind Netflix recommendations" by Xavier Amatriain, "TV Predictor: Personalized Program Recommendations to be displayed on SmartTVs" by Christopher Krauss, Lars George and Stefan Arbanowski, "Clustering Items for Collaborative Filtering" by Mark OConnor and Jon Herlocker, and "Design of Movie Recommendation System by Means of Collaborative Filtering" by Debadrita Roy and Arnab Kundu.

The paper "Big and Personal: data and models behind Netflix recommendations" talks about the way in which the Netflix website approaches the problem of personalization and of engaging users into interacting with the recommendation system results, what it looks to accomplish with its personalization system, how to design the system for ranking movies so to know what to recommend to users, and what data is used and how it is used to base the recommendations on. This paper supports my work because it discusses the necessary components of a movie recommendation system, what is needed to build it and to improve its performance. It is a general paper and it will be a helpful basis for my project.

The paper "TV Predictor: Personalized Program Recommendations to be displayed on SmartTVs" talks about a recent project, which took place in Germany. This project had to do with creating a recommendation system for smartTVs (which, in their most simple form, are just commonl TVs but with connection to the internet) that will recommend to a user

TV channels that they may like, and even automatically change the channel when a suited TV show appears in a different channel. They used both content based and collaborative filtering, as well as other high level data mining techniques: clustering, association rules and Support Vector Machines (SVM). This paper supports my project because it discusses how to create a movie recommendation system using content based and collaborative filtering, as well as clustering. It explains the mathematical basis of the two filtering methods. It is a moderately technical paper and it will be a substantial reference piece for my project.

The paper "Clustering Items for Collaborative Filtering" discusses the application of data partitioning/clustering algorithms to ratings data in collaborative filtering. They discuss how they partition sets of items based on user rating data so to then compute the rating predictions independently within each partition. They propose clustering as a way to improve the quality of collaborative filtering predictions and increase the scalability of collaborative filtering systems. I used this paper as reference to my project because I needed to clarify my understanding of clustering and how it could be used to improve the accuracy and speed of rating predictions.

The paper "Design of Movie Recommendation System by Means of Collaborative Filtering" talks more in detail about the implementation of clustering and collaborative filtering on a set of data consisting of users, ratings and movie details. I used this paper as help for me to understand how to implement clustering on user, rating and movie data to improve the run time of collaborative filtering.

3 Method

3.1 Mathematical Abstraction

We have a utility matrix of the form:

$$\begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{array} \begin{array}{c} \begin{array}{cccc} m_1 & m_2 & \dots & m_n \end{array} \\ \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ r_{31} & r_{32} & \dots & r_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \dots & r_{nn} \end{pmatrix} \end{array}$$

Where u_i stands for user i, m_j stands for movie j and r_{ij} stands for the rating user i gave to movie j. Usually, a movie's rating will be between 1 and 5 ($1 \leq r_{ij} \leq 5$), as it is in this project, though in some cases the rating may range from 1 to 10 [2]. An element in the matrix will be 0, $(i, j) = 0$, when there is no existing rating given by a user i for a movie j. The goal of a recommendation system is to change the entries in the matrix that are 0 to non-zero entries by predicting what rating a user will give movies they have not watched.

We also have a movie matrix of the form:

$$\begin{array}{c} \\ m_1 \\ m_2 \\ m_3 \\ \vdots \\ u_n \end{array} \begin{array}{c} \begin{array}{cccc} I_1 & I_2 & \dots & I_n \end{array} \\ \begin{pmatrix} f_{11} & f_{12} & \dots & f_{1n} \\ f_{21} & f_{22} & \dots & f_{2n} \\ f_{31} & f_{32} & \dots & f_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \dots & r_{nn} \end{pmatrix} \end{array}$$

Where m_i stands for movie i, I_j stands for the type of feature information stored and $f_{i,j}$ is the value of the feature for movie i. Possible types of feature information are: genre, actors, average rating, directors, etc. If some of the features stored in the matrix are genres, then one column, I_j, would correspond to a genre. For example, I_1 may correspond to the romance genre and I_2 may correspond to the comedy genre. The values of f_{ij} may be quite different, depending on what they represent. Usually, if it represents a genre than its value will be binary. A 1 for yes and a 0 for no, meaning that the movie it corresponds to is or isn't part of a certain genre. In the MovieLens dataset, the only feature information I use is the genre information. An example of a possible movie matrix is the following:

$$\begin{array}{c} \\ m_1 \\ m_2 \\ m_3 \end{array} \begin{array}{c} \begin{array}{cccc} action & romance & comedy & fiction \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

In this project I implemented content based filtering and collaborative filtering. Content based filtering focuses on the properties of items/movies. The similarity of such items is determined by measuring the similarities in their properties. Collaborative filtering focuses on the relationship between users and items. Similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items.

To perform content based filtering, I had to compare the similarity between movies. To do this, we have to have two profile vectors, one for users and one for items/movies. The user vector corresponds to one user and has the ratings for all movies in the data set (dimension of matrix is $1 \times m$, m is the number of movies). The user vector is derived from the utility matrix mentioned earlier. The item vector corresponds to one movie and will have all of the information available for that movie. The item vector is derived from the movie matrix mentioned earlier. The columns in the movie matrix correspond to all of the genres mentioned in the data set. Finally, the cosine similarity must be calculated between the movies. The cosine similarity is calculated with the following equation:

$$cSim(e_1, e_2) = cos(\vec{E_1}, \vec{E_2}) = \frac{\vec{E_1} \cdot \vec{E_2}}{|\vec{E_1}| \times |\vec{E_2}|}$$

Where e_1 and e_2 are the elements of the movies to be compared, such as genre information. $\vec{E_1}$ and $\vec{E_2}$ are vectors representing all features of the movies being compared. In my implementation of the cosine similarity algorithm, $\vec{E_1} = a_1, a_2, a_3, ..., a_n$ where a_i is a binary value, n is 19, the number of genres, and each element corresponds to a genre, determined by the position of the element in the vector. If $a_i = 0$, then the movie is not that certain genre, and if it is $a_i = 1$, then it is that certain genre. In the end, we chose the rating of the movie

most similar to the movie we are trying to rate as the predicted rating.

To perform collaborative filtering, I implemented the Slope One algorithm to calculate the similarity between movies and users. The Slope One algorithm is divided into two parts. One of them is the prediction value for user u and item j:

$$pre(u,j) = \frac{\sum_{i \in I_j}(r_{u,i} - dev(i,j))}{|I_j|}$$

Where I_j is the set of all relevant items (the set of items previously rated by user u) to be compared with item j, and $|I_j|$ is its cardinality. r_u, i is the rating of user u for item i. The resulting value is the predicted rating of this user u for the current item j.

The second part of the Slope One algorithm is a method to get the average deviation of two items:

$$dev(i,j) = \frac{\sum_{u \in U_{i,j}}(r_{u,i} - r_{u,j})}{|U_{i,j}|}$$

Where i and j are the items, $r_{u,i}$ is the rating user u gave to item i. U_{ij} is the set of users that rated both items i and j, and $|U_{i,j}|$ is the set's cardinality. The higher $|U_{ij}|$ is, the better the prediction. The result is the deviation of the rating of an item.

In this project I also implemented clustering, so to cluster users with the purpose of improving the scalability, quality and running time of collaborative filtering. I used a pre-existing function from a python library that implemented the k-means clustering algorithm. k-means clustering is a type of partitioning method, it involves constructing a partition for a database D of n objects into a set of k clusters. In k-means, each cluster is represented by the center of the cluster, which is denoted as the centroid. The following is a pseudocode of k-means Clustering:

```
1 Select K points as the initial centroids.
2 repeat
3   Form K clusters by assigning all points to the closest centroid.
4   Recompute the centroid of each cluster.
5 until The centroids don't change
```

3.2 Computational Algorithm

Initially I wrote the program so that the user, movie and rating data was stored in a list where each element was another list, with the purpose of representing the utility matrix. The movie data was also stored in the same type of structure. The problem with this approach was that going through the matrix and processing it was very time consuming. In the end, my implementation of the cosine similarity algorithm took 1 hour to predict 50,000 ratings (each iteration took a little less than half a second), while I wasn't even able to finish running my implementation of the Slope One algorithm, for which each iteration took 1 minute and 30 seconds, so it would have taken approximately 52 days to predict 50,000 ratings! Terribly inefficient.

After finding out that the Slope One algorithm would take more time to run than the amount of time available for the project, I decided to try a different approach. I stored the information into hash tables. It took up more memory, but the running time improved by a lot. The cosine similarity algorithm only took 14 minutes with this new and improved implementation. A great improvement! For the Slope One implementation, each iteration for rating entries took varying seconds, a maximum of 9 seconds. So in total this new implementation would take up to 5 days to run. A great improvement from the first implementation, but, unfortunately, this faster implementation was completed in less than 5 days before the deadline for this project, so this version of the program could not be ran completely before having to turn in this project. Thus, the results of the program for my implementation of the Slope One algorithm are inconclusive. The final implementation for this project uses hash tables to store and process the data.

After implementing Content Based filtering and Collaborative filtering in a more efficient manner, I started to work on clustering the user data. The idea was to group users into clusters based on their rated movies with highest ratings (ranging from 3 to 5), in order to minimize the running time of the Slope One algorithm by making it so that the deviation may be calculated on a smaller set of users, one with more similarities to the user in question, rather than all of the users that have rated the two items being compared in the calculation. At first I had some difficulty understanding how to store the data in a way that I could have a matrix in which each row corresponded to one user and thus apply a pre-existing clustering algorithm on this data.

Later I decided that a good course of action would be to first, for each user, choose the rated entries from the hash table that had a rating greater than or equal to 3 and then adding the genre features of the movies of all the highly rated entries of the same user. The idea was that if the addition for a genre, (say, romance), was greater than that of another genre (say, horror) then that would mean that the user prefers romance over horror. This new data would be stored in a matrix I denoted as the cluster matrix. Then I would perform clustering on the cluster matrix, so to create 10 clusters. The position where the movie addition was stored in the cluster matrix would correspond to the position of the userID in the list of keys of the hash table. I would use this position convention to later store the userIDs into their calculated and designated clusters. Finally, these sets would be used to calculate the deviation for the Slope One calculating. Instead of having to compare the user in question with all of the users that rated two particular items at the same time, we would only have to compare the user with a couple of other users that are more similar to them, acquired from the cluster set.

In the end, the addition of clustering the data improved the running time of the Slope One algorithm. By using clusters to minimize the set of information used to calculate the deviation in the Slope One calculations, running time per iteration became less than a second, and an ocassional maximum of 1 second, in comparison to the previous implementation in which each iteration took a maximum of 9 seconds. I calculated how long the program would now take to rate 50,000 entries if each iteration were to take 1 second to run and it turns out that it would take 13 hours to run. From 5 days to 13 hours, it is a good improvement. Unfortunately, once more, I finished the implementation within less than 13 hours of having to turn it in.

It may sound bad, but ... I had finished improving the Slope One algorithm about 3 days before the due date (the 52 day version had been completed long ago) and then I finally figured out how to do the clustering the day it was due, with less than 13 hours left. Not having much luck with timing. Thus, I decided to simply run the program so that it would rate 10,000 entries, which should take a maximum of 3 hours.

Pseudocode for Content Based Filtering:

```
Loop through all unrated entries (first half of information):
{
  if user ID of unrated entry is within user IDs of rated items (second half),
     then:
  {
    Loop through movies rated by the user in question (of the the unrated entry
       ):
    {
      Calculate the cosine similarity using vector of unrated movie and vector
         of other movie already rated by the user.
    }
    Pick the largest calculated cosine similarity as the predicted rating for
       the unrated movie.
  }
}
```

Pseudocode for creating the Cluster data matrix:

```
Initialize cluster matrix

Loop through all rated entries (acquired from Hash table containing rated data)
   :
{
  addition = empty list, will store cumulative addition of movie genre
     information for all movies per user with high ratings

  Loop through movies rated by the user in question (rated entry):
  {
    if rating is greater than or equal to 3, then:
    {
      Add each genre element of movie vector for user entry to each genre
         element of addition list
    }
  }
  Append final addition list to cluster matrix
}
```

Pseudocode for grouping users into 10 clusters based on Cluster data matrix:

```
Create 10 clusters using python library.

Store cluster matrix row and cluster id pairs into a list called user_cluster.

Store user IDs into a list called cluster that will have 10 elements, each a
   list containing user IDs. Do this using the user_cluster list created
   previously.
```

Pseudocode for Slope One, prediction:

```
Loop through all unrated entries (first half of information):
{
  sum = 0

  if user ID of unrated entry is within user IDs of rated items (second half), then:
  {
    cluster = cluster in which user belongs in

    Loop through movies rated by the user in question (of the the unrated entry):
    {
      rating u,i =  rating of movie we want to rate given by user of rated entries
      sum = sum + (rating u,i - dev(unrated movie vector, rated movie vector, cluster))
    }
    relevant = number of movies rated by user in question, for whom we are predicting the movie rating
    prediction = sum / relevant

    if prediction is less than 1, then:
    {
      predicted rating result is estimated as 1
    }
    else:
    {
      predicted rating result is rounded up
    }
  }

  Predicted rating for this entry is: prediction
}
```

Pseudocode for Slope One, deviation:

```
def dev(item1, item2,cluster):
{
  sum = 0
  cardinality = 0 #this corresponds to the number of users that rated both item1 and item2

  Loop through users within cluster:
  {
    if user we are currently on rated both items item1 and item2, then:
    {
      rating u,item1 = rating of item1 given by user of rated entries
      rating u,item2 = rating of item2 given by user of rated entries

      sum = sum + (rating u,item1 - rating u,item2)
      cardinality = cardinality + 1
    }
  }
  return the deviation, sum / cardinality
}
```

Pseudocode for calculating Root Mean Square Error (RMSE):

```
sum = 0
N = number of newly rated entries, the entries we want to verify

Loop i from 0 to N:

  predicted_rating = absolute value of calculated rated from entry i
  real_rating = absolute value of real (original) rating of entry i (from
      movieLens dataset)

  sum = sum + (predicted_rating - real_rating)

RMSE = sum / N
```

4 Results

4.1 Datasets

The data set used to test this project was the MovieLens data set (grouplens.org).The data set used contained 100,000 ratings from 943 users and 1,682 movies, with each user rating at least 20 items. The ratings in the MovieLens data were explicitly entered by users, and are integers ranging from 1 to 5. The data set also contains simple demographic information for the users (age, gender, occupation, zip). The data was collected from September 19, 1997 to April 22, 1998, so we won't expect any movies that came out after April 22 1998, nor any release dates that are after that.

There are different files available. One file is denoted as u.data. It is the full user(u) data set, with 100,000 ratings by 943 users on 1682 movies. Users and items are numbered consecutively from 1. The data is randomly ordered. There is a tab separated list of: user id, item id, rating and time stamp. Another file, denoted u.info, has the number of users, items and ratings that are in the u data set. Another file, denoted as u.item, has the information about the items (movies). It is a tab separated list of: movie id, movie title, release date, video release date, IMDb URL, unknown, action, adventure, animation, children's, comedy, crime, documentary, drama, fantasy, film-noir, horror, musical, mystery, romance, sci-fi, thriller, war, western. The last 19 fields correspond to the genres; a value of 1 indicates that the movie is of that genre, and a 0 indicates it is not of that genre. A movie can be in several genres at once. The movie ids are the ones used in the u.data data set. Another file, called u.genre, is a lot of the genres. Another, called u.user, is a list of the demographic information about the users, a tab separated list of: user id, age, gender, occupation, zip code. The user ids are the ones used in the u.data data set. And finally, the file u.occupation is a list of the occupations. There are other files in the MovieLens data set but they are irrelevant to this project (I did not use them).

4.2 Evaluation Criteria

I wrote my program in Python 3.3. and I ran it in a laptop with an AMD A8-3500M APU processor with Radeon(tm) HD Graphics 1.50GHz and an installed memory (RAM) of 6.00GB, of which 5.48GB are usable.

I evaluated the accuracy of my algorithms by calculating the error of the final rating prediction. The easiest way to do this is by calculating the Root Mean Square Error (RMSE), for which we would be substracting the real value from the predicted value. The RMSE is calculated with the following equation:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} |p_i - q_i|^2}{N}}$$

Where N is the number of items rated by the algorithms, p_i is the predicted rating for item i and q_i is the real rating for item i. The lower the error, the better the algorithm. For example, in the Netflix Prize contest, the winners achieved an RMSE of 0.8567 [1]. Another example is the RMSE acquired by the recommendation system created for the TV predictor explained in [2] which was 3.77. I concluded that a preferred RMSE should be below 3.77, and a reasonable one would be above 0.90, since my intention was not to create a better recommendation system, but to simply create one, and to get such a small error in my first try would most likely imply that I did something wrong.

4.3 Summary of Results

The resulting root mean square error (RMSE) for the predictions acquired with the cosine similarity algorithm (content based filtering) after rating 50,000 movies based on 50,000 already rated movies is 1.4457. The lower the root mean square error, the better, so this result means that the algorithm did a good job in predicting ratings for movies, but not very impressive. The running time was 14 minutes.

The RMSE that resulted from the Slope One algorithm (collaborative filtering) after rating 10,000 movies based on 50,000 already rated movies is 3.7099. Not as good as the error from the cosine similarity algorithm. It is very slightly less than the RMSE of the TV predictor created in Germany [2], so it is hanging on the edge. I would conclude that my implementation did not work as well as hoped. Possible reasons for this are: 1.) the fact that I only ran it for 10,000 entries due to the lack of time, which makes only a few predicted ratings to compare with the original and apparently make it seem like the algorithm has too large of an error, and 2.) it is possible that my decisions on storing the data for clustering may have affected negatively the clustering results and so the deviation does not draw from good users for calculations. The running time for this algorithm ran only on 10,000 ratings was about 1 hour.

My implementation of the cosine similarity algorithm was significantly faster than that of the Slope One algorithm. While cosine similarity took 14 minutes to predict 50,000 ratings, the Slope One algorithm would take from 13 hours (with clustering) to 5 days (without clustering) to do the same.

5 Future Work

As future work I may attempt to improve the accuracy of the content based filtering algorithm by using additional movie information acquired from IMDb. This information may be acquired by performing text mining on IMDb metadata available online. The idea is as follows:

I would get the descriptions of the movies and I would compare the descriptions between movies with text mining. If, say, 50% (for example) of the words are the same between two movie descriptions, then they are similar, and there is a good chance that a user that likes one will like the other. This data would form part of the movie matrix, as one feature column.

After downloading the IMDb metadata file that contains the plots to all the movies in IMDb (a very large file), I would proceed to do the following:

1. First, clean up the data; delete the movies that are not mentioned in the MovieLens dataset, so to speed up the analysis and parsing of the file.

2. Second, clean up the data some more by removing commonly used words in english. This would be done by using a data set with the 100 most used words in English so to find common words in the descriptions and then remove them.

3. Finally, within the program, I can look for the description of two particular movies in the new smaller file (do parsing for this) and compare the two movies with text mining methods (text frequency).

The down side to this proposed approach is that we would then be limited to the English language. All the descriptions must be in English, or it won't work.

Another goal I may attempt for future work is to improve the running time of the program (by this I mean decrease the running time). I would do this by reasearching more efficient data structures and applying them, as well as using multi-threading for program segments that don't depend on each other. I'd specially make two threads, one to run content based filtering and the other to run collaborative filtering, so to run both programs at the same time, since they don't depend on each other in my implementation.

Finally, I may also do additional preprocessing to the data. According to a classmate named James, the MovieLens dataset had a couple of repetitions both in the u.data file (which contains the rating data) and the u.items file (which contains the movie features information). So I would work on removing such inconsistencies to get more accurate results from the algorithms I implemented.

6 Project Experience

I found it to be a very fulfilling experience. My understanding of recommendation systems and data mining techniques increase as well as my confidence in programming. This was the first program I have written which dealt with great amounts of data. I encountered a couple of difficulties that taught me more about the challenges of dealing with big data. For example, in this project I managed to write code that would take 52 days to run. A personal record. More suprisingly, I was able to optimize the program to only need 5 days to run, and then I optimized it further to only need 13 hours to run. It took a lot of time to do this project but I enjoyed the time I spent on it, and I would not mind continuing to improve this code further or to keep running the collaborative filtering code past turning this project in so to later compare results and find out if my implementation was good or not. But I would also

enjoy to dive into a new data mining project. It was a fun experience, and I'm excited to continue such line of work for my independent study and masters project.

References

[1] Xavier Amatriain, *Big & personal: data and models behind netflix recommendations.* 1-6. KDD 2013: Chicago, IL, USA - BigMine Workshop

[2] Christopher Krauss, Lars George, Stefan Arbanowski, *TV predictor: personalized program recommendations to be displayed on SmartTVs.* 63-70. KDD 2013: Chicago, IL, USA - BigMine Workshop

[3] Mark OConnor & Jon Herlocker, *Clustering Items for Collaborative Filtering.*

[4] Debadrita Roy & Arnab Kundu *Design of Movie Recommendation System by Means of Collaborative Filtering.* 3-4.

[5] Anand Rajaraman, Jure Leskovec, Jeffrey D. Ullman, *Mining of Massive Datasets.* Book.