Movie Recommendations Using Social Networks

Ankit Gupta, Rohan Jain, Shiwei Song
{gankit,rohanj,shiweis}@cs.stanford.edu
CS229, Stanford University
Stanford, CA 94305

Dec 12, 2008

Abstract

This paper explores utilization of information from social networks in making automatic movie recommendations. Implementations of three different algorithms (SVM, Clustering, and Ranking SVM) are implemented and evaluated. The general approach utilizes a large collection of Facebook profile information as training set in order to generate a list of movie recommendations for a particular user (client). A brief analysis of the importance of specific profile features in making movie recommendations is conducted using feature selection on each of the recommendation algorithms.

1. Introduction

Generating good movie recommendations has been a problem tackled by large companies such as Netflix and Amazon. Solutions by these companies often utilize large data sets of movie ratings by thousands of users and use a user's previous movie preferences (usage history on their site) to predict future movie preferences [1]. Although these solutions generally work, they have many shortcomings [2]. Because these sites are product based, their data do not capture the personality and tastes of a user. Furthermore, people tend to rent/buy movies for family and friends, making their data noisy. We believe many personal factors and characteristics (such as age, relationship status, political view) affect one's preference for movies. Since like minds gather, who one's (real world) friends are and what they like may also be a factor in recommending novel movies. With the advent of social networks and accessibility of personal profile information, it is now possible to build recommendation engines that incorporate personal profiles.

In this project, we utilize information available through the Facebook API to gather personal profile information about our client and his friends. Aside from just information about individuals, we also gather how they are connected with each other in the social graph. Using these pieces of information, we build a recommendation engine that can take advantage of the rich information available from social networks. Our motivation and goal is to explore the usage of these newly available data in building a better movie recommendations engine.

The outline of this paper is as follows. Section 2 looks at previous work in the area of movie recommendations and currently available recommendation services. Section 3 discusses how data is gathered and represented. Section 4 goes into details of our recommendation algorithms. Section 5 discusses our results and observations. Finally, Section 6 concludes this paper by discussing possible future extensions to our work.

2. Previous Work

The Netflix Prize [1] has sparked interest in movie recommendations research. A variety of approaches have been attempted, such as using k-means clustering [3] and logistic regression [4]. These approaches, however, still suffers from the limitations of the Netflix data set.

Many online movie recommendation engines such as Yahoo! Movies and MovieLens uses collaborative filtering to generate movie recommendations. This approach works well given a large data set of the user's movie viewing history and ratings, which often involves filling out lengthy surveys.

Golbeck [5] presents work in using social networks for movie recommendations. The approach presented uses a trust based social network in conjunction with collaborate filtering. However, this approach does not use personal profile information of users and still requires lengthy surveys.

We hope to develop an effective movie recommendations engine that utilizes existing personal profile information available on social networks without the need for further user input.

3. Data Gathering and Feature Representation

3.1 Data Gathering

The Facebook API allows access to personal profile information of a client and his friends. After receiving permission from our friends (clients), we use a Ruby script that gathers the following raw information about each client and his friends: favorite movies, favorite books, favorite music, birthday, location, name, relationship status, sex, activities, and political view. We hypothesized that items in this list of information that we gather will have influence over one's movie tastes. Between each two friends of the client, we check if they are friends as well. This information is used to compose the social graph of the client. The social graph information is stored as an adjacency matrix G with G_{ij} indicating friend i is also friends with friend j.

The raw information gathered above cannot be used directly as features in our algorithms. One problem is that the location information itself does not work well as a feature. Rather, the relative distances between cities is needed (in order to find possible movie preferences for particular geographical regions). To accomplish this, we use Yahoo! Geocoding API for each city that appears in our data set to find its longitude and latitude. We then create a table of distances between any two cities in our data set.

Another major problem is that movie, book, and music names in the raw data are not standardized across user inputs. Due to spelling and punctuation inconsistencies, the same movie title may appear to be separate titles when used in training set directly. Therefore, these features need to be standardized. Standardization of features is done by using the results of Google Search (using the Google Ajax Search API) with the user inputs as part of the query strings. To standardize movies, we construct queries with the format `<movie name> imdb`. A movie title that appears in the top 3 results is then used as the standardized name. Similarly, to standardize books, `<book title> wiki` is used as query string. A book title from wikipedia in the top 3 results is used. Finally, to standardize music written as band or song name, the query is `<band/song name> last.fm`. A music title from the website last.fm in the top 3 results is used as the standardized name.

We gathered the entire social graph for 10 users and their friends. Combined, we got the profile information for over 2000 users. Section 5.2 describes how the data is used in our tests.

3.2 Data Representation

The standardized data for each user is converted into feature vectors. A user's feature vector is a concatenation of representation of each pieces of information collected from his profile. The representations are as follows.

1. **Sex**: A bit vector with [1 0] for male and [0 1] for female.

2. **Relationship Status, Political View**: Bit vectors with 1 for the specific relationship status/political view of the user.

3. **Location**: Vector with an entry for each city. Let d_i be the distance between city i and the user's location. The i^{th} entry in the vector is equal $\frac{\max_j d_j - d_i}{\sum_k (\max_j d_j - d_k)}$. This is the inverted normalized distance from the user location.

4. **Age**: Vector with entry corresponding to each age bucket. Age is divided into buckets for users under 16, 16 to 30 with step of 2, and over 30. Let v_i be the i^{th} entry in the vector. If the user's age falls into bucket i, then $v_i = 0.50, v_{i-1} = v_{i+1} = 0.25$ and all other entries equals 0. E.g. a person of age 21 would have the age vector as [0 0.25 0.50 0.25 0 0 0].

5. **Books, Music**: Bit vectors with 1 for the books/music the user listed in his profile.

In case some feature was not present, we gave equal weights to all of the possible entries. The sum of entries for each vector is 1.

For the cases when we want to use the social graph in training, we added another feature for every user with the value as number of friends he has in common with the person we are trying to predict movies for. This feature tries to capture the closeness of two people based on the number of friends they have in common. For the client himself, the value of this feature is the number of friends he has. Although there are many other ways to incorporate the social graph information, we could not explore this space much due to lack of time. See section 6 for possible alternatives/improvements.

4. Recommendation Algorithms

4.1 SVM

The first approach we tried was to train an SVM for each movie and find a separating hyperplane between the users who liked the movie and users who didn't. We recommend movies in the order of the largest margin for the client. Intuitively, this is recommending movies we are most confident that the client will like.

Let the set of all unique movies in the training set be M. For client x and users in the training set $x^{(k)}$'s (represented by feature vectors), we do the following: For each movie $m^{(i)} \in M$, we train an SVM to get $\alpha^{(i)}, b^{(i)}$. We define $y^{(k)} = 1$ if $x^{(k)}$ liked movie $m^{(i)}$. To make recommendations for x, we compute

$$score(i) = \sum_k \alpha_k^{(i)} y^{(k)} K(x^{(k)}, x)$$

which is effectively the margin for x on the SVM for $m^{(i)}$. We then rank the movies in decreasing order of $score(i)$ and recommend high scoring movies.

Both linear and gaussian kernels were tried and results are presented in section 5.

4.2 Clustering

The second approach we tried is doing k-means clustering for each movie on the users who liked that movie.

The training step consists of the following: For each movie $m^{(i)}$ liked by some subset of users $X^{(i)}$ from the training set, find up to $k^{(i)}$ clusters in $X^{(i)}$. Model each cluster $c_j^{(i)}$ as a normal distribution $N(\mu_j^{(i)}, (\sigma_j^{(i)})^2)$. In our tests, $|X^{(i)}|$ is generally fairly small (less than 10). In order to not over fit the data, we used $\forall i, k^{(i)} = 1$.

To make a movie recommendation for client x, first calculate the score for each movie with

$$score(i) = \min_j \left(\frac{\|x - \mu_j^{(i)}\|_2}{(\sigma_j^{(i)})^2} \right)$$

Then sort the movies by their score in descending order and recommend movies with high scores.

Intuitively, this algorithm works by first finding stereotypes of people who like a particular movie. Scores indicate how closely the client fits into one of these stereotypes. Recommend movies with high scores to the client.

This algorithm has the benefit of being very easy to implement and very intuitive to understand.

4.3 Adapted Ranking SVM

The third approach was using an adapted Ranking SVM. The Ranking SVM Algorithm [6] represents a ranking problem in terms of an SVM. Given a set of favorite movies for each person, the goal is to come up with a ranking of the movies for the client.

Let $w^{(i)}$ be movie specific weights for each movie $m^{(i)}$ such that user $x^{(k)}$ likes $m^{(i)}$ more than $m^{(j)}$ (represented as $m^{(i)} >_{x^{(k)}} m^{(j)}$) if $(w^{(i)})^T x^{(k)} > (w^{(j)})^T x^{(k)}$. The SVM for this problem can be modeled as

$$\begin{aligned} \min_{W=\{w^{(1)}, w^{(2)}, \ldots, w^{(|M|)}\}} \quad & \frac{1}{2} \sum_{i=1}^{|M|} (w^{(i)})^T w^{(i)} + C \sum_{i,j,k} \epsilon_{i,j,k} \\ \forall (m^{(i)} >_{x^{(k)}} m^{(j)}) \quad & (w^{(i)})^T x^{(k)} > (w^{(j)})^T x^{(k)} + 1 \\ & - \epsilon_{i,j,k} \\ \forall i, j, k \quad & \epsilon_{i,j,k} \geq 0 \end{aligned}$$

Each condition $m^{(i)} >_{x^{(k)}} m^{(j)}$ can be derived if $x^{(k)}$ has listed $m^{(i)}$ but not $m^{(j)}$ in his favorite movies list.

Let $W = [w^{(1)}; w^{(2)}; \ldots; w^{(|M|)}]$. Let $\theta_{i,j,k}$ be a vector of size $|M||x^{(k)}|$ such that if $m^{(i)} >_{x^{(k)}} m^{(j)}$, $\theta_{i,j,k} = [0; 0; \ldots; x^{(k)}; 0; \ldots; -x^{(k)}; \ldots; 0]$ (with $x^{(k)}$ and $-x^{(k)}$ at i^{th} and j^{th} position respectively). We can then remodel the previous SVM as a standard SVM

$$\begin{aligned} \min_W \quad & \frac{1}{2} W^T W + C \sum_{i,j,k} \epsilon_{i,j,k} \\ \forall (m^{(i)} >_{x^{(k)}} m^{(j)}) \quad & W^T \theta_{i,j,k} > 1 - \epsilon_{i,j,k} \\ \forall i, j, k \quad & \epsilon_{i,j,k} \geq 0 \end{aligned}$$

This SVM finds $|M|$ weight vectors w_i for each movie representing weights given to individual user features.

To make a prediction for client x, we find the values $(w^{(i)})^T x$ for all movies $m^{(i)}$ and rank the movies based on this value.

Due to the enormous size of the feature vectors and the large number of constraints, this algorithm suffers from poor running time. If the average user lists f favorite movies and the training set contains $|P|$ users, then the size of the feature vector is $|M||x|$ and the number of constraints is $2f|M||P|$. For example, with a total of 200 movies and 100 people each having 80 features, 5 favorite movies, the size of the feature vector is 16,000 and the number of constraints is 200,000.

5. Results and Evaluations

5.1 Evaluation Criteria

The subjective nature of movie preferences makes evaluating the algorithms difficult. There are several issues that standard evaluation criteria fail to capture.

1. Users don't really know the ranking of their favorite movies. They can only tell the few movies they like a lot.
2. Although user's movie preferences may remain constant over time, their list of favorite movies changes. Movies seen recently are more likely to be remembered and become favorites.

3. Users don't spend a lot of time listing favorite movies on Facebook, thus reducing the reliability of their favorite movies list data.

Due to the above reasons, the "ground truth" data we gather from Facebook is not reliable. So our evaluation criteria should not take the entire favorite movies list into account or there would be too much variance in outcome. Instead, we used a metric which reports the score of the highest ranked movie present in a user's favorites list.

Let $F_k = \{m_{f_k1}, m_{f_k2}, \ldots, m_{f_kn}\}$ be a list of favorite movies of person p_k. Let $Rank_k(m)$ represent the rank of movie m in the ranked output of the recommended movies for p_k after running some recommendation algorithm. The Maximum Reciprocal Rank ($MaxRR$) for p_k is

$$MaxRR(p_k) = \frac{1}{\max_{m \in F_k} Rank(m)}$$

Let M be the set of all possible movies we can recommend and $T = \{p_{t1}, p_{t2}, \ldots, p_{tn}\}$ be the test set, the evaluation metric we use is

$$Average\ MaxRR = \frac{1}{|T|} \sum_{p \in T} MaxRR(p)$$

$$score = \frac{100}{|M| Average\ MaxRR}$$

The score represents the average percentile in which one of the movies in a person's the favorites list appear in our recommendations list. For example, a score of 2 for $|M| = 100$ means on the average, one of our top 2 recommendations will be in the client's favorite list. The score is normalized for different $|M|$ ($|M|$ depends on the training set).

5.2 Test Results

The data gathered is used to construct two separate tests. Test 1 evaluates the results of the algorithms when social graph is taken into consideration. Test 2 evaluates the algorithm when only using profile information.

In Test 1, for each of the 10 users we have social graph for, we used the set of his friends as training data to make recommendations for him. The social graph is factored into the feature vectors as described in section 3.2. The scores reported for each algorithm is the average score for the 10 users. The average $|M| = 350$.

In Test 2, the data is randomly split into a training set and test set. The Training set has about 1500 users and test set has about 500 users. Scores are calculated as described in section 5.1. This large training set gives $|M| = 1800$.

Due to the running time problems Ranking SVM encounters for large data sets, a randomly sampled smaller training set is used for it in Test 2. Additionally, only the top 100 most frequently occurring movies are considered by Ranking SVM. Intuitively, this is taking the movies with the highest prior probability of being liked.

Algorithm	Percentage
Clustering	3.1254%
Linear SVM	4.05577%
Gaussian SVM	4.3177%
RankingSVM	9.1025%

Table 1: Test 1 scores (lower = better)

Algorithm	Percentage
Clustering	0.8109%
Linear SVM	0.3096%
Gaussian SVM	0.1531%
RankingSVM	1.3561%

Table 2: Test 2 scores (lower = better)

5.3 Feature Selection

To find the importance of each of the features we used in making movie recommendations, feature selection is conducted on linear SVM, gaussian SVM, and the clustering algorithm in Test 2.

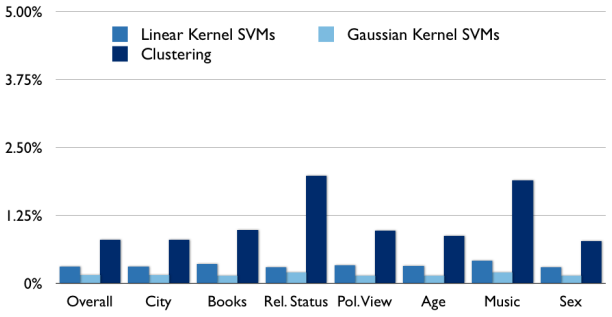

Figure 1: Scores when ignoring each of the features

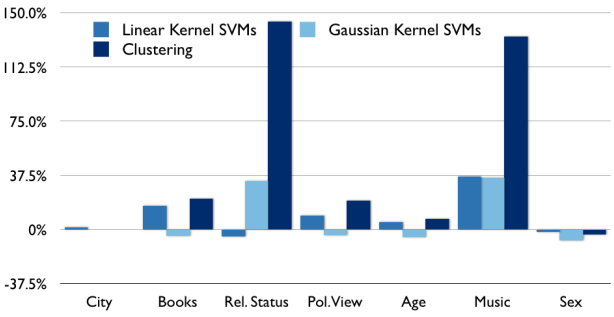

Figure 2: Percent change in scores when ignoring each of the features

The scores for each algorithm are calculated when each of the features are ignored. The results are plotted in Figure 1 and Figure 2. An increase in score indicates that removing the feature made the recommendations worse.

5.4 Observations

From our results, we can make several observations:

1. The clustering algorithm works quite well, indicating that movies do appeal to certain stereotypes. We expect that the performance of this algorithm to increase with a larger data set as clustering becomes more accuate.

2. The Ranking SVM does not do well probably because we had to sample the data and reduce the number of movies we were considering for it.

3. Gaussian kernel is expected to do better than a linear kernel, which turns out to be the case while conducting Test 2. In Test 1, both have comparable performances for the smaller data sets.

4. There is high correlation between tastes in books and music and taste in movies as expected.

5. Surprisingly, features like sex and the location reduce performance. Sex is a bad indicator because our data set was biased towards males. Location does not help because people tend to list their hometowns rather than where they currently reside.

6. Conclusion

Our project shows that it is possible to use relatively fast algorithms such as SVM and clustering to make fairly accurate movie recommendations without the need for any additional input from the user. The techniques used here can be utilized by a production social network based movie recommendation engine to make seed recommendations (when it has no additional user data yet). As more data is collected, collaborative filtering or other online learning algorithms can be used to improve results.

There are a few things to explore for continued work on this subject:

1. One way to improve the algorithms is to incorporate movie specific features from IMDB such as movie rating, genre, and release date. We can also use information retrieval methods to generate textual features from the plot, title, user comments etc.

2. An intuitive way to incorporate information from the social graph in the Ranking SVM is by using a kernel. This can be done by using a diffusion kernel [7]. This approach defines a matrix H as

$$H_{ij} = \begin{cases} 1 & \text{if edge from } i \text{ to } j \\ -d_i & \text{if } i = j \\ 0 & \text{Otherwise} \end{cases}$$

   Where d_i is the degree of node i. Using the kernel $\exp(\beta H)$ for the SVM incorporates this information. Here, the parameter β controls how much importance is given to the links in the graph. Thus, increasing β will increase the probability of making decisions based on the choices of your friends and vice versa.

3. Currently, we are not using interests and activities from a user's Facebook profile as it is very hard to standardize. In the future, we can semantically cluster them to use as features in our algorithms. Since books and music turn out to be good indicators of one's movie preferences, we believe that including activities and interests will help as well.

References

[1] Bennett, James, and Stan Lanning. "The Netflix Prize." *Proceedings of KDD Cup and Workshop 2007* (2007).

[2] Chen, Harry. "Why Netflix Movie Recommendation Doesnt Work." Weblog post. *Harry Chen Thinks Aloud*. 3 Oct. 2006. 12 Dec. 2008 <http://harry.hchen1.com/2006/10/03/391>.

[3] Sa, Brian, and Patrick Shih. "K-Means for Netflix User Clustering." *CS229* (2006).

[4] Sadovsky, Adam, and Xing Chen. "Evaluating the Effectiveness of Regularized Logistic Regression for the Netflix Movie Rating Prediction Task." *CS229* (2006).

[5] Jennifer Golbeck. "Generating Predictive Movie Recommendations from Trust in Social Networks." *Proceedings of the Fourth International Conference on Trust Management* (2006).

[6] T. Joachims. "Optimizing Search Engines Using Clickthrough Data." *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), ACM* (2002)

[7] Risi Imre Kondor, John Lafferty. "Diffusion Kernels on Graphs and Other Discrete Input Spaces." *Proceedings of the Nineteenth International Conference on Machine Learning* (2002)