

Recommendation of TV shows and Movies based on Facebook data

Mathangi Venkatesan

mathangi@stanford.edu

Andy Mai

andymai@stanford.edu

Abstract

We seek to use Facebook data to solve the cold-start problem in recommending movies and TV shows. We use a person's "likes" on Facebook to predict what TV shows and movies he/she may like. To simulate the cold-start situation for our test users, we don't use any data about what movies or TV show pages that they've "like"-d on Facebook. This information is used only to find the accuracy of our recommendation engine.

We've implemented two major approaches in our recommendation engine. The two methods approach the same problem with different perspectives. Our first approach uses unsupervised learning to cluster together similar Facebook profiles. After getting clusters of profiles, we recommend movies and TV shows to each cluster. Our second approach is a two-step approach. In the first step, we use supervised learning to recommend top genres for each test user. In the second step, we recommend most popular movies and TV shows within those top genres.

1. Introduction

When a user first signs up for a Movie/TV recommendation service, the recommendation system has no information about what movies or TV shows the person has already watched and liked/disliked. This is called the cold-start situation. Despite not knowing anything about the user, the recommendation engine has to give good recommendations in order to retain the user's interest. The common solution to this problem is to recommend to new users TV shows and movies that are most "popular" among all users. Here, popularity could be measured by highest rating, most views, box-office revenue or any other reasonable metric based on available data. However, this solution is not personalized at all.

Our goal is to give personalized cold-start movie and TV show recommendations. The data we use for making recommendations is the pages a person "likes" on Facebook. The premise here is that there's a correlation between a person's taste in books, activities, music etc. and his/her taste in movies/TV shows.

2. Data

Through a Facebook data scraper, we acquired close to 16 thousand unique Facebook profiles. Most of the profiles had very less/no information. So, we discarded profiles of people who listed less than three "likes". In the end, we had 4761 profiles to work with. We randomly chose 20% of these to be our test-users.

We were given a list of 347 TV shows along with their genres and description. We found this dataset missing some of the most popular shows (like "F.R.I.E.N.D.S"). So, we manually added ~ 20 shows and their genre information from IMDb. In the end, we had shows from 109 unique genres.

3. Data Processing common to both methods

For each user, we aggregated all their “likes”, except for movies and TV shows, into a long string, which served as the document for that user. We stripped away tag information for the “likes” because most users had only very few “likes” in each category and sometimes the same “like” was in different categories for different users (this was most common for the activities and interests categories. For eg. some people listed swimming under interests while others listed it under activities).

We tokenized the documents and used a reverse-stemmer written in Matlab to get the root-form of each token. Then, we created our vocabulary consisting of stemmed form of words that appeared in at least one document.

We created a document-term matrix where each document corresponds to a user and each term corresponds to a word in our vocabulary. The (i,j) th element of the matrix is 1 if the j th term is present in the i th document, otherwise its 0. We applied idf (Inverse Document Frequency) weighting to this matrix so that more frequently occurring terms have lesser importance than less frequently occurring terms.

3. Features

We use the stemmed form of words that appear in any of the Facebook profiles as our features. For our dataset we had 43191 features.

4. Methodology and Results

4.1.1 Method 1: Clustering Facebook Profiles using K-means

We implemented K-means algorithm with cosine similarity to cluster similar Facebook profiles. By clustering similar profiles, we wanted to simulate real-life friend circles. Profiles in the same clusters are grouped together because they have the most similar profile data, and therefore can be treated as “friends,” since people tend to be friends with others who are share the same interests. After clustering, the 10 most popular movies and TV shows from each cluster are compiled and served as recommendations to test users who map to that cluster.

Because of the very large feature set, we couldn’t get a visual representation to see how well our clustering algorithm was doing. So, we varied the number of clusters from 3 to 40 to see for what cluster size we got the best accuracy.

In our first implementation of this algorithm, we naively used the document-term matrix without idf weighting and a non-stemmed vocabulary. After seeing very poor accuracy of recommendations, we investigated on a bunch of 100 profiles and observed that a lot of the same/similar words were being considered as different. So, we started reading about Natural Language Processing. We learnt about stemmers and used a Matlab implementation of reverse-stemmer on our vocabulary. During this phase, we also hit upon idf weighting concept on the Internet. The final version of our algorithm uses the idf weighted document-term matrix and stemmed vocabulary that we created(described in [3. Data processing] section).

Lastly, we implemented Latent Semantic Analysis(LSA) to reduce the dimension of the document-term matrix from 43191 to approximately 20,000 and then, tested our accuracy.

4.1.2 Results for Method 1

We define

$$\text{accuracy for each user} = (\text{hits} \div \text{number of shows / movies we recommended}) * 100$$

where

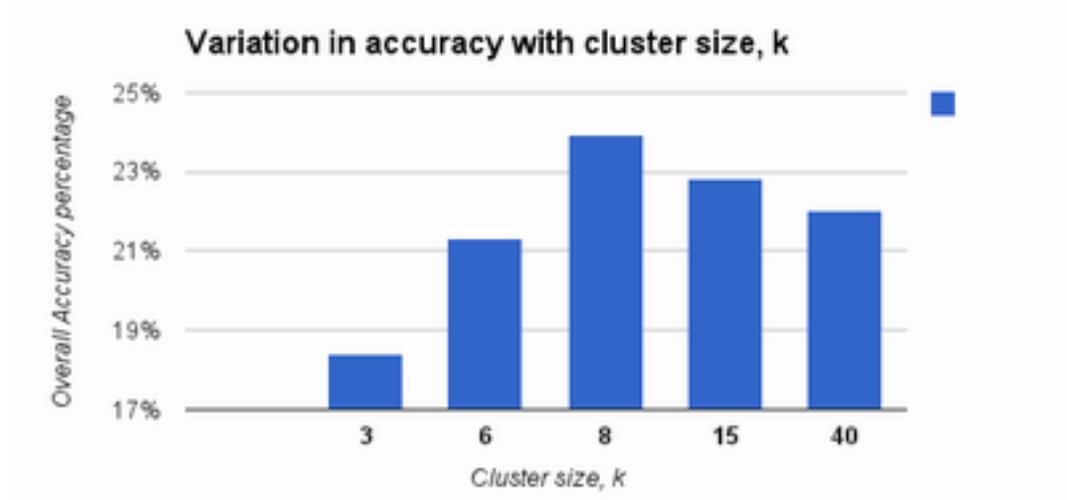
$$\text{hits} = \text{total number of recommended shows / movies that were listed in their profile} \\ \text{number of shows / movies we recommended} = 10$$

And, $\text{overall accuracy} = \text{accuracy for each test user} \div \text{number of test users}$

We chose recommending the 10 most frequently liked movies and TV shows in the training set as the baseline to compare our algorithm against. This baseline has an accuracy of 16.9%.

When we implemented K-means without stemming and idf weighting, we saw an overall accuracy of 8.3 % (for k = 8)

After stemming and idf weighting, our overall accuracy rose to 23.96%(for k = 8). As, we varied the number of clusters from 3 to 40, we saw an variation in accuracy between 18.42% and 23.96%. We chose k = 8 because that had the best overall accuracy. The following figure shows the accuracies for some values of k. Note: All accuracies in the figure were measured after implementing the algorithm with stemming and idf weighting.



After reducing dimensions using LSA, our overall accuracy rose from 23.96% to 24.4% (for k =8)

A summary of our results:

Metric	Baseline(recom mending top 10 movies/shows in training set)	K-means without stemming, idf weighting	K-means with stemming, idf weighting	K-means with stemming, idf weighting + LSA
Overall Accuracy	16.9%	8.3%	23.96%	24.4%

4.2.1 Method 2: Using Naive Bayes to get top genres for users

Our second approach to the problem centers around genres. Rather than recommending movies right away, we develop a model to determine users' favorite genres of movies and TV shows and recommend them popular movies and TV shows from those genres.

We label each user(or the document of each user) in our training set with the genre of shows they've liked. We also associate a weight for each genre they've liked. For eg. if they've liked 2 shows and the 1st show has genres comedy, satire, the 2nd show has genres comedy,drama. Then, for this user comedy has a weight of 2 and satire, drama have a weight of 1 each. We create a person(or document)-genre matrix with each entry having weights as described above.

We use the document-term matrix(as described in [3. Data processing] section) and the document-genre matrix to get the probability that a term in our vocabulary is associated with a specific genre.

For each test user, we use the words in their document and apply the Naive Bayes algorithm to compute the probability that they like each of the 109 genres. Essentially, it is like a binary classifier for each genre except we store the actual probability that they like a genre instead of just storing 0 or 1. Based on these probabilities, we recommend the top five genres for each test user. Then, for each of the top five genres recommended, we recommend the two most popular TV shows or movies for that genre in our training dataset.

Lastly, as in our previous method, we used Latent Semantic Analysis(LSA) to reduce the dimension of the document-term matrix from 43191 to approximately 20,000 and then, tested our accuracy.

4.2.2 Results for Method 2

After implementing method 1, we learnt our lesson and used the stemmed vocabulary and idf weighted document-term matrix from the beginning for method 2.

We have two accuracies to report for this algorithm. One is the accuracy with which we recommend genres for users and the other is the accuracy with which we recommend movies and TV shows for the users(from those genres). We define accuracy for each of these cases similar to how we defined it for Method 1.

We chose recommending the 5 most frequently "like"-d genres in our training dataset as the baseline to compare our algorithm against. This baseline has an accuracy of 48.36%.

In comparison, our algorithm for recommending the top 5 five genres for each test user has an overall accuracy of 70.98%. And, recommending the most frequently "like"-d two movies/TV shows from each of the five recommended genres has an overall accuracy of 32.84%.

After using LSA, our genre recommendation accuracy rose to 72.1% and our movie/TV recommendation accuracy rose to 32.88%.

5. Analysis

Both our methods suffered from the problem that our data was far too sparse. Another general problem is that people do not tend to list all the movies and TV shows they like on their Facebook profile.

The low accuracy of Method 1 could be because there are no well-defined clusters in our dataset or there were clusters that got broken when we weeded out profiles that listed very less information. Another reason could be that perhaps Facebook profiles cannot be clustered by simply using a term-document sparse matrix. The data that we have has only likes and interests, but it does not have any information on the social dynamic of the person on Facebook. Since our original goal was to cluster profiles to simulate friend circles, having information about who is who's "friend" would have helped us a lot.

The accuracy of Method 2 could have been higher if the list of popular shows and their genres that we started out with had been more exhaustive. The main advantage of our Method 2 over Method 1 is that in Method 2 we don't assume that there is any underlying pattern in the data(like clusters).

6. Challenges faced during implementation

The biggest hurdle during implementation was processing text in users' profiles. A surprisingly large number of people misspelled the simplest of words. Multiple Facebook pages(some with misspellings) for the same "like" only made things worse.

Very sparse data led to very large matrices and because of that the execution times of the algorithms were very long.

7. Future Work

The immediate next step in our work is to reduce the number of features(thereby reducing the sparsity of the matrix).

If we are able to collect "friends" information for each of our users, we can use that in our recommendations.

A more philosophical view of the project is that we are trying to summarize people by looking at what pages they've liked on Facebook. To this end, we need more data than just the title of the pages they've "like"-d. A future direction is to work on an algorithm that takes in the content of the pages they've liked, pages they visit often, their status updates/comments and information about closest friends to give a "summary" for a person which can then be used as the "document" for the person.

8. Acknowledgments

We would like to thank Andrew Maas for suggesting this project topic and giving valuable advice throughout the duration of the project. We would also like to thank Graham Darcey and Wayne Yurtin of Screaming Velocity for their help in collecting Facebook data and providing TV/Movie genres and descriptions.

9. References

Turney, Peter D. and Patrick Pantel. "From Frequency to Meaning: Vector Space Model of Semantics." *Journal of Artificial Intelligence Research* 37 (2010): 141-188