Find-A-Friend

By Tengyu Liu (tliu12) and Xinyan Zhou (xzhou14)

Table of Contents

Abstract

Social networks have changed the way people interact with each other. While maintaining relationships online has been very popular and very well developed, it is not common to develop a friendship from online. On major barrier is that there is no good tool to utilize the digital nature to improve the experience of finding new friends. Through implementing an information retrieval system, Find-A-Friend (FAF) addresses the issue and finds potential best friend by analyzing the online profile programmatically.

Introduction

Beginning with Facebook in 2004, social network has been existing on the internet for 10 years. People have enjoyed the convenience of being able to communicate and share their life with their friends and other people online at almost zero cost and latency. The existence of the digital social network has provided computer scientists the opportunity to explore more in human nature as it is very easy to gain access to people's daily thoughts that are traditionally kept only in people's diaries.

With the big data available, we thought it would be viable to make a recommendation system that suggests potential friendship based on people's social network website history. The idea behind this is that people with similar interests will be more likely become good friends, and that the posts on social media reflects the interests of each person.

In Find-A-Friend, we take advantage of this convenience and designed a system that dig into people's status posts to find what topics people are interested in. We then suggest potential friendship based on the topic list that we mined. We have tried numeral methods for determining the topic for each person and finally decided to use topic clustering instead of querying by word.

Related Work

There is currently no pubic tool that explores potential friendship by analyzing people's postings, although we believe that there might exist similar tools being used internally when Twitter and Weibo recommends follows. There are public tools that analyze Facebook activities also, but none of them makes recommendation on friendship. For example Wolfram|Alpha Personal Analytics for Facebook analyzes user's Facebook activity and generates a report including the temporal activities, social network structure and the user's basic information.

Problem Definition

The FAF system expects the user's input to be a Facebook log in (which is a user id and an access token provided by Facebook), and the expected output will be a ranked list of Facebook users. The problem is split into three pieces: data accessing, data processing and result visualization.

Data Accessing

In data accessing, we need to gather all Facebook status posts of the querying user and of each of the friend of the querying user. All posts will be composed to a single file where each post of the same user

is concatenated with each other and each user occupies one line in the file. The posts of the querying user is in the first line.

Data Processing

In data processing, we need to take the composed file and cluster it into several topics. We will then find how likely each user is interested in each topic. We will then rank the list of friends of the querying user in the order of topic interest similarity and provide the similarity data as well.

Result Visualization

We will need to visualize the ranked list to the user as well as providing a word cloud for each result in the ranked list. The word cloud will represent the most interested topic between the querying user and the result user.

Feedback and Improvement

When the user gives feedback, we improve our ranking algorithm to accommodate to the information reflected in the feedback.

Methods

Data Accessing

Data accessing is done via Facebook API querying. Facebook API paginates automatically when the result is too big, so we need to automatically navigate through pages. We also need paginate manually when we use AJAX to write big file back to server if the file is too big.

Data Processing

Pre-Processing

Before actually clustering the file, we need to remove the words that are not useful in clustering (stop words) and stem similar words into one.

With the ultimate goal of digging just the topics from Facebook posts, the stop words that we need is different from most stop word lists available online. With several experiments, we decided to add a bunch of adjectives, adverbs and verbs into the stop word list that we had used in Assignment5.

Stemming was relatively easy in data pre-processing as we used the MeTA framework. Just using the tools provided in Assignment5 we were able to get a good result in stemming the words.

Topic Extraction

We used the unsupervised topic extraction from MeTA and experimented a range of methods. After comparing the results, we decided to use one-vs-all method for our classifier with stochastic gradient descent as the base.

Topic Analysis

After extracting the topics, we need to compare each document (posting history of each user) with the topic model we have. The topic we have from Topic Extraction is a vector form where each position in a vector represents a term in that topic. The value represents the weight (probability) of that term.

$$\text{i-th topic vector: } u_i = [\cdots \quad 0.000289465 \quad 0.000127486 \quad \ldots]$$

Although the topic model looks solid enough, it created some serious issue where basically every document is scored based one particular topic only. It was because that the sum over all elements in each topic vector is very different, or, $i \neq j \Rightarrow |u_i| \neq |u_j|$. In our experiment dataset, there is one topic whose Euclidian length is much larger than the others, which result in that each document is basically scored only based on its similarity to this particular topic.

To counter that, we normalized all topic vectors by multiplying each topic to a constant c so that

$$t_i = c \cdot u_i \text{ such that } |c \cdot u_i| = 1$$

We then take a dot product between each topic vector and the document term distribution vector where each entry represents $p(w|d)$. The resulting numbers are put into another vector v_{query}. This is equivalent to computing a cross product of the following form:

$$v_{query} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \end{bmatrix} \times vd_{query} \text{ where } vd_{query} = [p(w|d)|w \in d \wedge d = \text{posts from querying user}]$$

The resulted v_{query} represents the similarity between the querying user's posts and each topic model. We then need to compute if any of the user's friends has similar similarity behavior over the topic models.

To compare each document with the topics, we do the same thing to each of the documents.

$$\text{i-th similarity vector: } v_i = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \end{bmatrix} \times vd_i \text{ where } vd_i = [p(w|d_i)|w \in d_i \wedge d_i = \text{posts from i-th user}]$$

To compute the score for each document, we first adopted a naive approach where

$$score_i = v_{query} \cdot v_i$$

This is the most intuitive calculation, but it did not run well in our experiment dataset where there is one topic that is very popular in all documents. We then looked into the dataset and realized that what we need is not only that both topics are common, but that they are common in the same way. In another words, if a topic scores 8 in one document and 2 in another document, it should have a lower score than a document with score 4 in both documents. We then came up with the following equation

$$score_i = \left\| \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \end{bmatrix} \right\| \text{ where } s_j = \frac{v_{query_j} \times v_{i_j}}{(v_{query_j} - v_{i_j}) + 0.01}$$

The subtraction in the denominator compensates the case where the score is raised too high because one of the two documents are high. The additional 0.01 is added to normalize it so that when two documents have identical topic distribution the score between them does not go off to infinity.

We will rank all documents based on the score we obtained here and pass it, as well as v_i, along to Result Visualization.

Result Visualization

With the resulted ranked list, we are able to present to the users. In the resulted ranked list, we have the following information:

- The Facebook ID for each user in each location of the ranked list;
- The topic distribution vector v_{query} for the querying user;
- The topic distribution vector v_i for that user;
- The document d_i of that user, containing all his/her posting histories.

In order to make the results intuitive and to let users know instantly why we made that ranking, we decided to put a word cloud for each result next to the Facebook profile picture and link of that user. The word cloud is made by feeding the common terms between the querying user doc and the displaying user doc and their weights. These terms are selected from the topic j where $j = argmax\left(\frac{v_{query_j} \times v_{i_j}}{(v_{query_j} - v_{i_j}) + 0.01}, j\right)$ and the weights of each term $w_{term} = v_{query_{j_{term}}} \times v_{i_{j_{term}}}$.

Feedback and Improvement

Since this project is about the potential of being a good friend, it is not possible for us to have immediate feedback on our results because it takes time to build a relationship and to find whether a person can be a good friend. However, in order to practice query improvement based on feedback, we added a button next to each result, allowing users to label one as either “positive”, “negative” or “no opinion”.

The idea behind our query improvement is that the frequency for a person to bring up some topic on Facebook does not necessarily reflect the true willingness for him/her to discuss it with friends. For example, someone may brag about having to stay up late for studying all the time on Facebook but is not willing to talk about studying with friends.

In our improvement algorithm, if the result i is considered to be “positive”, then v_{query} will be adjusted correspondingly to increment the weight on the term that was most important in this particular result. Specifically,

$$P(i) \longrightarrow v_{query_j} *= 1.1 \text{ where } j = argmax(\frac{v_{query_j} \times v_{i_j}}{(v_{query_j} - v_{i_j}) + 0.01})$$

Similarly, if the result k is considered to be "negative", then v_{query} will be adjusted correspondingly to decrement the weight on the term that was most important in this particular result:

$$N(k) \longrightarrow v_{query_j} *= 1.1 \text{ where } j = argmax(\frac{v_{query_j} \times v_{kj}}{(v_{query_j} - v_{kj}) + 0.01})$$

The results will then be automatically refreshed based on the new query.

Evaluation and Sample Results

Evaluation

While the results of the querying looks very good to us, we cannot systematically determine the quality of the system because this project focuses on finding potential best friends. It is even not easy for human to determine the correctness of the result because during our testing, we often find that we are not familiar with the resulted person and cannot determine if we could become a good friend with them in the future. So we have to turn our evaluation into the case that given a specific criteria, will the query improvement help the search engine tilt towards the criteria over feedback.

Since our teammate Xinyan has a fanatic passion about sports, especially basketball, we decided to set the criteria to be that "if the word cloud is about basketball then the result is relevant". In the initial result, we see that only the 7th and 8th results out of 10 shown are relevant according to our criteria, with $precision = 0.2$. We then marked the three to be relevant and reran the query. The precision now becomes 0.5 where the relevant results locate at 3rd, 4th and 6th in the ranked list. After marking these three as positive again, we see that the first 4 and the 6th are all relevant.

We also tried the negative results. After clearing the browser cache and reloading the page, we see the initial result where the 7th and 8th results have word cloud relating to basketball. This time we marked them as irrelevant ("negative") and reran the query. The result now have only 1 result at 8th location. Marking as negative again results in complete disappearing of the topic basketball.

Deficiency

While this system satisfies us very well for a final project, it is far from a complete product yet. In addition the crappy user interface, there are still several key issues that needs to be addressed:

Search Domain

Currently we are searching within the querying user's friends. This does not make much sense when you are searching for a "new friend". We were forced to do so because we do not have the permission to view everyone's posts on Facebook. And even if we do, we cannot store them all in one place a course project scale.

For this problem, one approach is to ask for permission to access and store the user's posts and the user's friends' posts for future use. This way we can accumulate the search domain overtime and do not have a huge need for disk space and a super-efficient algorithm to begin with.

Also, due to the time restriction we only made it working with Facebook. It would be better to extend it to other social network websites. Twitter is a good place to continue, but Quora would work way better.

Topic Extraction

In this project we used unsupervised topic extraction method with a fixed number of topics. An ideal implementation would be to let the system automatically figure out how many topics there are. One potential way to do it is to make a big training dataset for the classifier so that it can learn from. We did not do it because it needs too much time and effort in it just to compose a dataset. However, we will definitely make one and keep it updated automatically if we were going to extend this project.

Result Visualization

When doing the result visualization, we simply took the intersection between the query user's posts and the result user's posts in one topic and put them into a word cloud. It was a rush work and can definitely be improved. One way of improving this could be to analyze the result user's posts and find a chunk of posts which is most relevant to the highest scored topic. We can display this chunk of post and show it with terms in the topic highlighted in red, just as what Google does.

Conclusion and Future Work

In this report, we have introduced a new search engine that searches for your potential best friend hidden somewhere in your social network where you can't see. Using what we learned in CS 410, we are able to find a list of potential friends and we are able to improve the result based on the feedback from users.

With the bloom of the data science, people are doing much more things than before. However, with the quick development there are always something left behind that is not benefited. Find-A-Friend is the first tool to use the data we have for us to improve our social network. This project extends what we can do with the data we have into a whole new category – friendship.

Although this project is complete for CS410, there are many deficiencies that can be improved. In addition to the problems discussed in Deficiency section in the previous chapter, we believe that there are a lot of better algorithms that we could adopt in this project. We also believe that research in sociology, communications and other humanities field would also contribute to the performance of this project.

Appendix

Individual Contributions

Tengyu Liu (tliu12)

- Data Access
- Result Visualization
- User Interface
- Query Improvement (shared task)

Xinyan Zhou (xzhou14)

- Data Processing
- Feedback and Query Improvement (task)

References

Wolfram|Alpha Personal Analytics for Facebook:
http://www.wolframalpha.com/input/?i=facebook%20report#

Data mining methods not covered in CS 410: CS 412 Lecture by Professor Jiawei Han