Facebook Friend Suggestion

Eytan Daniyalzade and Tim Lipus

1. Introduction

Facebook is a social networking website with an open platform that enables developers to extract and utilize user information and relationships. Our goal was to create a "friendship model" to be used for the following purposes:

1) Recommend a new friend to a user given the user's existing friends.

2) Given a user's friends, determine which of those friends would be users' "best friends."

To make these predictions, we gathered user profile information such friends, groups, interests, music, and activities. We trained our models based on these features using techniques from both supervised and unsupervised learning. For the first goal, we implemented logistic regression and Naïve Bayes learning algorithms to predict which users are most likely to be friends. For the second task, we used PCA and K-means clustering to compare a given user's friends to each other.

It is highly likely that the friendship structure of a certain person would be very different than that of another person. Furthermore, there is a large variation among the profile information of different people (e.g. we will not necessarily have the same features for each person, and the relative importance of features may be different for different users). Therefore, we decided to train our hypothesis separately for every person whose friends we would try to understand. We use the term "central user" to denote the user for whom we are training the algorithm. Instead of defining features for individual users, we defined the features for each sample as a function of a central user and the sample we are comparing against. For example, a feature would not be the number of friends of a user, but rather the number of friends a user has in common with the central user.

2. Data Acquisition

We used the Facebook API to collect user data. Data collection steps were as follows: choose central users, find their friends (positive samples), choose a set of non-friends (negative samples), and gather profile information for every positive and negative sample with respect to the central user.

Limitations of the API and privacy settings created some obstacles in data collection. First, some profiles allow only limited access, which in some cases makes it impossible to see any data other than the name of the user. For this reason, we limited ourselves to profiles within the Stanford network, since most non-Stanford profiles allow us very little access. Second, it is not possible to directly query for all of a given user's friends. One can only query whether two particular users are friends. Therefore, we needed to start with a large set of user ids, and it was not possible to find all the friends of a user unless our starting set contained all of them. The third problem was the lack of an easy way to get a list of users according to arbitrary criteria (such as Stanford students). We had to collect data in a less direct way; we started with a Stanford student-group, found all the Stanford students in that group, then found other groups to which those users belong, and repeated until we had around 4000 samples. It is possible that this method could have introduced some bias into our data set that could have been avoided had it been possible to select 4000 random Stanford users.

Because a given user's friends make up a small percentage of our network, we chose to include in our training and test sets all of the central user's friends (that we found in our set of users) and up to 200 non-friends. We made this choice to include as much information about friends as possible, but we also had to keep in mind the implications of having a training set with a different distribution than the overall data, which we discuss in more detail below.

3. Model Evaluation

One of the major challenges of this project was determining how to evaluate our models. One reason this is difficult is that although the models we used for friend suggestion are typically used for classification, our application is not quite classification. Rather than outputting y=1 (friend) or y=0 (non-

friend) for each candidate user, we instead want to select the most likely candidates out of a given pool. Therefore, measuring test error is not as simple as finding the percentage of misclassifications on a test set. One property that an error metric should have is a higher emphasis on precision than recall; we do not need to return all likely candidates, but we want the ones we return to be good.

A second problem with the friend suggestion task is that what we are testing is not quite the same as the goal we are trying to accomplish. Our goal is to create an application that suggests new friends that are not already the central user's friends, but we are testing the ability of the model to predict whether a given user is already friends with the central user. Accurately testing the first goal would require a large set of hand-labeled data; we decided it would be better to use the large amount of already existing data as a proxy. For the second task of determining a central user's closest friends, we do not have any labeled training data at all, so it is difficult to make a precise evaluation of our model's performance.

4. Friend Recommendation using Logistic Regression

We trained a linear logistic classifier to classify non-labeled samples as friends or non-friends. Initially, we used a training set of 50 non-friends and approximately 30 friends for each user. For finding the optimal classifier, we preferred Newton's Method due to its rapid convergence and the non-singularity of our feature set. Features used were the ones mentioned above, i.e. number of common friends, groups, activities etc. We used K-Fold Cross Validation (with K=10) to find our test error. Accuracy rates on our first run were highly satisfactory given the acceptable level of accuracy for our application. Classification error was on average between 15% and 20%. The test error rate increased as we increased the number of features included, despite the fact that each feature was individually a decent friendship indicator. **Figure 1** shows how testing error varied with increasing number of training samples, and different graphs shows error rates for different number of features included.

Figure 1 Data reduced to 2-Dimensions for Visualization

As seen in the plots, training error was significantly less than test error with higher number of features. Further, test error declined with increasing number of training samples. We interpreted these results as indicators of high variance and acquired more data to increase our training set to 200 non-friends and around 60 friends. Expanding our training set significantly improved our results, lowering error rates to 12%, 11% and 10% for 2, 4 and 6 features respectively.

4.1 Weighted Logistic Regression

Basic logistic regression's main drawback is that it punishes false negatives the same as false positives. However, since our application would only be reporting the samples that it would classify as positive, we should lower error on positive guesses as much as possible. This could be achieved by weighing negative samples more than positive samples while training. This approach would put more emphasis on maximizing the likelihood of negative samples and shift the separating line closer to positive samples, making our hypothesis less likely to guess positive samples. As seen in **Figure 2**, as the weight assigned to negative samples increased, the percentage error on positively guessed samples (i.e. mislabeling of negative data) and the number of positive guesses decreased.

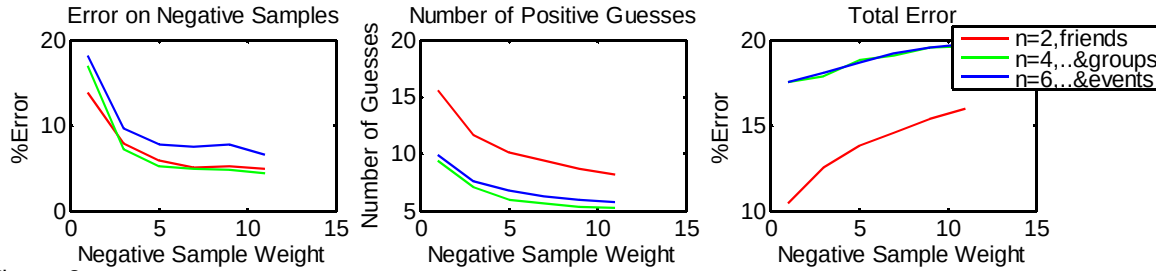

Figure 2

4.2 Conclusions on Logistic Regression

A major challenge is figuring out the optimal weight on the negative samples. We deferred tackling this issue until we gathered sufficient user feedback to decide whether the number of recommendations or their accuracy is a bigger priority.

5. Naïve Bayes

We also trained a NB model based on features that each person might have in common with the central user. We trained both a Bernoulli and a multinomial model. In the Bernoulli model, for a given feature of a user (e.g., groups), we let $P(x_i = 1|y)$ be the probability that the user's i^{th} group was also shared by the central user. However, we considered the possibility that some groups might be more indicative than others. Therefore, we also created a lexicon of the groups to which the central user belongs and trained a multinomial model in which we let $P(x_i = j|y)$ be the probability that the user's i^{th} group was the same as the central user's j^{th} group. If $j = 0$, then it is the probability that the user's i^{th} group is not shared by the central user. (Note that the multinomial model distinguishes between each group to which the central user belongs, but groups to which he does not belong are treated symmetrically.)

5.1 Modeling the Prior

One difficulty in applying NB is modeling the prior $P(y=1)$. We would normally set this parameter to be the percentage of the training examples which were positive, but as discussed above, our training set is not representative of the actual distribution. Furthermore, even if we base our prior on all of the data (not just the data in the training set), it may not match the prior for the question we are actually trying to answer; the probability that given user is friends with the central user may be different than the probability that a given non-friend is someone with whom the central user might like to be friends.

However, the nature of our goal allows us to solve this problem. As mentioned above, instead of performing classification, we instead want to select the most likely candidates. Therefore, we rank the users by the likelihood ratio $P(x|y=1)/P(x|y=0)$, which is equivalent to ranking by $P(y=1|x)$. This way, we can avoid modeling the prior.

5.2 Measuring Accuracy

Unfortunately, as mentioned above, this application of Naïve Bayes makes measuring the test error difficult. A simple method would be to let F be the number of friends in the test set, rank the test set by the likelihood ratio, and let the error be equal to the percentage of users in the top F scores that were actually non-friends. This metric (which we call metric A) has the advantage of focusing on false positives. However, in practice, we would probably only report the top few candidates, so we care the most about the ones at the top. Therefore, we also used metric B, which awards a higher weight to the test samples that the model ranks higher; specifically, $w_i = F+1–i$ (with weights renormalized to sum to 1). We also let metric C be the same as A, but with F/2 in place of F.

The following plots show the test results for our model, where each column shows the test samples for a single central user. Friends are shown in blue, non-friends in red. We found the best model (**Figure 3-b**) to be the one that treats common friends by a multinomial model and other features as a Bernoulli model. For comparison, the model with all features treated as Bernoulli is shown in **Figure 3-a**. The error according to each of the three metrics is given as well.

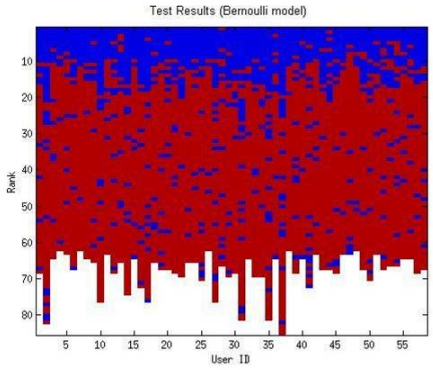

Figure 3-a A = 24.5% B = 12.2% B = 7.33%

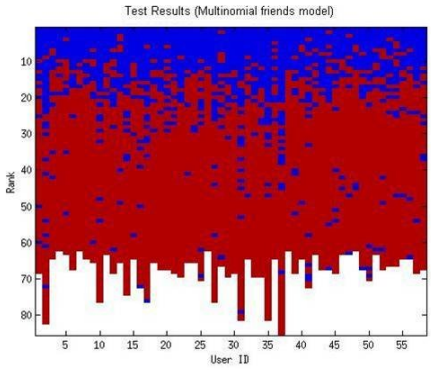

Figure 3-b A = 20.7% B = 11.2% C = 7.65%

6. Unsupervised and Semi-Supervised Learning for Best Friend Suggestion

The goal here is to cluster a user's friends based on their similarity, and report the members of the group closest to the user as the "best friends."

6.1. Reduction to a Single Dimension using PCA

An obvious challenge in unsupervised learning is defining a metric that determines the cluster to report. We tackled this issue by assuming that features were positively correlated with friendship level. To test this assumption, we created a feature set consisting of our friends and reduced the data to a single dimension by applying PCA. This single dimension, the principal eigenvector, indicates the direction of highest variation; hence, we postulated that the projection of the better friends would lie further away from the origin, indicating a higher number of common features. Using ourselves as the central user, this metric correctly identified the people we would consider as good friends. Although we could not prove results rigorously, this experiment indicated that projecting on the principal component could indicate level of friendship.

6.2. K-Means Clustering

We clustered our data into four clusters (the number of clusters was arbitrarily decided) using the 8-dimensional feature set and reported a cluster based on our metric of closeness to the central user. **Figure 4-a** shows the results. It is noteworthy that we applied clustering in higher-dimensional space, rather than reducing data to a lower dimensional space and then reporting points based on their distance from the origin. The two methods would cluster points in similar but not identical ways. Whereas the former method clusters points based on their similarity with each other, the latter clusters them based on their similarity with the central user. **Figure 4-b** shows the results of the latter method. As further discussed below, it is hard to know which method yields better results without actually getting feedback from users.

6.3. Constrained K-Means Clustering

The clustering methods covered so far did not accommodate user feedback, so we implemented a constrained K-Means clustering algorithm that would incorporate user feedback as labeling on data to be adhered to while clustering. User feedback would be in the form of "good friend" or "not good friend." Our algorithm would ensure that sample points with same label would be assigned to the same cluster, and no cluster would contain points with different labels. In terms of implementation, constrained K-Means differs from regular K-Means in the assignment of the labeled samples to a centroid. While regular K-Means assigns each labeled sample to a centroid with the objective of minimizing that specific sample's Euclidian distance from the centroid, constrained K-Means finds the centroid yielding the smallest value for the sum of distances of all the samples with a specific label and assigns them to that centroid. To test this algorithm, we labeled the top 10 outputs of unconstrained K-Means and ran constrained K-Means on the new semi-labeled data set. The results are shown in **Figure 4-c**.

6.4 Conclusions on Unsupervised and Semi-Supervised Learning

Testing was a major challenge on our "best friend suggestion" algorithms. Given the nature of the problem, we could only test the algorithm on our own friends, and we did not have a coherent metric for gauging the accuracy of results. However, results were encouraging; 75% of the 20 people reported by the

unconstrained K-Means algorithm were people that we would classify as good friends. Furthermore, a significant drawback of the semi-supervised learning algorithm we used was that it only adjusted clusters, not the definition of "good friends", based on the user feedback. An algorithm that would adjust the weights of different features, such as "Distance Metric Learning [1]", could be more appropriate for the question at hand.

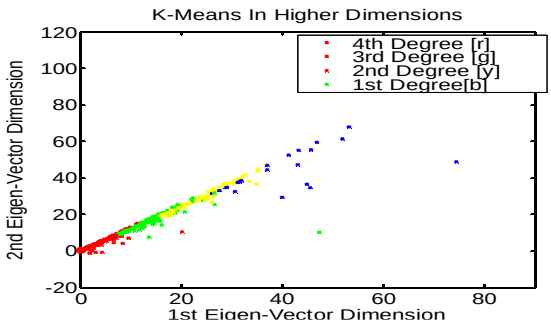

Figure 4-a Reduced to 2-Dimensions for Visualization

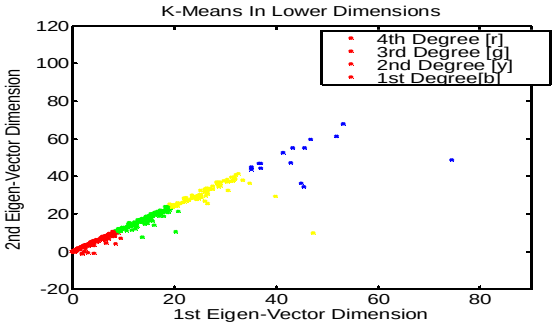

Figure 4-b

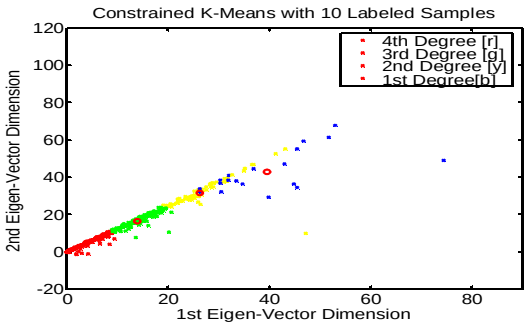

Figure 4-c Reduced to 2-Dimensions for Visualization

7. Conclusion

In all of our algorithms, we found the best feature to be the percentage of friends that users had in common, followed by the percentage of common groups and events. One reason for this result is the sparseness of the data for many of the features. For example, many people have few events listed, so it is common to see users who have no events in common with the central user. For other features, such as activities and music, the problem is even worse because the entries for these categories are user generated (i.e. prone to spelling errors or writing the same thing in different ways), which makes it even less likely to see commonality among users. The sparseness of these features makes them harder to use than features with denser data, such as number of common friends.

However, we believe the errors we found for our friend suggestion models show promise. Depending on the metric, test errors are roughly in the 10-20% range. We believe that friend suggestion could be useful even if only a much smaller fraction (say, one out of five) of our results were relevant. However, we also realize that our error measures are by no means perfect. As discussed above, since we were testing the ability to predict current friends rather than suggesting new ones, we would ultimately need to test with actual users, and this is even more the case with our "best friend" predictor. Therefore, the next step in our work would be to incorporate this model into a Facebook application and gather user feedback.

8. References

[1] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan and Stuart Russel. *Distance Metric Learning, with application to clustering with side-information.* University of California, Berkeley.