Abstract
TODO...

Introduction

Ever since Facebook released their API, developers from all over the world have programmed applications using Facebook’s data. Most of these applications only use the data to post on a user’s wall, find friends who also have the application, or display statistics about user data. There are very few Facebook applications that do any in-depth analysis of Facebook user data.

Our goal is to create a Facebook application that applies data mining techniques to the data made available by Facebook’s API. Specifically we would like to apply a clustering algorithm to a user’s friends. This should reveal groups of friends that are closely connected by their activity on Facebook.

For example, let’s think about high school students. High school students cluster together creating cliques. We will define a clique as a group of people that interact with each other on a routine basis. In Facebook we will consider pictures, messages, post, etc as attributes that identify people who interact with each other. In high school, students typically join based cliques based on their interests or where they fit in. However on Facebook, users communicate with people from many different groups. Friends could be from high school, family, college, work, etc. Even if you look from within a group like college, friends can be from clubs, classes, dorms, Greek life, etc.

So what if Facebook users were forced to join one clique on Facebook. How would they form? What are the important factors of people joining cliques? We surveyed Facebook users to see what they considered the most important information on their profile and on their friend’s profiles. We then determined the distance between users based on the analyzed results. These distances between users are then used in our clustering algorithm.

Our clustering algorithm works by continuously grouping together the closest two friends on the user’s friend list. Once two friends are grouped together, they are considered a cluster and are treated as one friend. This process repeats until all friends are in one cluster, or until the closest friends or clusters in the list are considered too distant from each other that we won’t put them in the same clique.

Facebook puts restrictions on the data they give developers access to. The user’s profile information, wall posts, photos, friends, and friend’s data can be retrieved but only stored for a maximum of 24 hours. However any Facebook IDs can be stored indefinitely. For example, user IDs, picture IDs, event IDs, etc. /* more about Facebook restrictions and the influence I had on our program (Bill: well we could store the results of clustering as IDs only but we didn’t actually do that. Probably worth mentioning anyway) */
Once we evaluate the user’s friend data, we construct a distance matrix that stores the distances between the user’s friends. This matrix can be used to find the distance between the user and his or her friends, as well as the distances between any two of the user friends. This matrix allows us to analyze how friends interact with each other, not just how they interact with the user.

**Background/Related work**

TODO

**Problem/Solution**

TODO

**Implementation**

**Distance Algorithm:**

- Analysis of the amount user data

Retrieving and processing large amounts of data from Facebook’s servers in a short amount of time is a major concern of ours. There restrictions not only on the calls to Facebook’s API, but on how much server processing we can request as well. The connection from the server will time out if the response from the server takes to long. The first goal of our project was to determine how much data we could access.

We collect data from Facebook tables using the Facebook Query Language (FQL). There are 34 Facebook tables containing large amounts of data. We did not need to access most of the tables like permissions, link stats, and pages. However, there is a lot of information in the tables we do want to access. Let’s look at the user table which contains all the personal information. There are 54 columns in that table. For a user that has only 200 friends, there are over 10000 pieces of personal information stored in the table. However, it is only one query and quick to process. If you want to access information about two users, you can index the array using their user IDs. This information however can’t be immediately translated into a distance value.

Accessing photo tags is very useful for our distance value, but more complicated to retrieve. To get all photo ids from all friend’s albums and all friend’s tagged photos only requires 2 queries. The problem lies in processing the data. We explain the process in the distance algorithm’s implementation, however lets look at an example of how fast cpu cycles can add up.

Lets look at a user who has 100 friends, each friend is tagged in 200 pictures, each friend has 50 pictures, and each friend’s picture has 2 people tagged, This is not uncommon and, if anything, could be considered a low ball estimate. The resulting list from the two queries will have 2 million entries. We have to go through and turn it into
a of users with there corresponding picture id’s. A lot of entries will be thrown away since the friend’s picture tags won’t include the current list of 100 users. Now we have to compare each user with every other user and see if they share any pictures. If n is the number of friends and x is the number of photos each one has, the maximum run time would be $n^2x^2$, which in this case would result in 400 million cpu cycles. If each cycle takes a microsecond, it would take 6 minutes to process just the pictures.

Facebook won’t return us 2 million entries, but a combination of all multiple data sources could add up fast. Our server will also time out anywhere from 10-30 seconds, so we have to mine the most valuable data we can. Therefore we had to be careful and considerate with the data we mined from facebook. We wanted to try and cluster people using data they think is important.

- **Analysis of important user data**

We chose 5 different data sources that would provide a lot of information and would be easy and efficient to mine. These data sources consist of personal information, wall post, photos, groups, and events. We then surveyed 50 facebook users on what they considered the most important information in 4 different areas: personal information, friend’s information, information that determines how close they are with friends, and information they would use to form a clique. Photos contained the most votes across all areas and within 3 out of the 4 areas, where the 4th area was a tie between photos and personal information. Graph 1 shows the votes percentages across all areas.

![Graph 1: Votes across all areas](image)

Therefore, we decided to concentrate on mining photo information and construct distances based on that. The wall post count is kept within the user table with all the personal information and probably the easiest thing to mine. Since wall post had 22%, we decided to include the count in our distinct matrix as well.
- **Implementation of the Distance Algorithm**

The goal of the distance algorithm is to provide the distances between users. We do this by building a distance matrix using the amount of photos two users are tagged in together.

First we retrieve all of the photos from facebook, which returns a list of entries with a picture id with the person’s id that is tagged in that photo. If a photo has 5 people tagged, 5 entries are returned. We turn these results into a list of users with their corresponding picture ids. Now we go through each user and check if each user has any of the same photos in their list. Then we divide the number of the photos the user has by the number of photos the two users share. This gives us in inverse relationship where the more pictures they are in together, the less distance they have.

Once we have all the distances, we create a list of objects that implements the Groupable interface from the clustering algorithm. Now the algorithm can check the distance to another object, which just looks up the distance value in the matrix.

**Clustering Algorithm:**

In the initial stages of designing our Facebook application, we needed to decide what type of data mining algorithm we were going to apply to Facebook data. K-means clustering requires us to select a specific number of clusters and the cluster centers. There are a number of problems with this. First, discovering appropriate candidates for cluster centers requires additional analysis of friend data when we are already short on computation time. Second, two friends that may appear to be candidates for cluster centers may in fact be very close to each other, giving skewed cluster results. Third, it is very difficult to determine the appropriate number of clusters without lots of analysis. A user can have 50 friends who are all very active with each other and form one large clique, or a user can have 10 friends who don’t interact with each other at all giving 10 cliques.

We instead decided to use Hierarchical Clustering. Hierarchical Clustering allows us to form clusters of friends without requiring us to designate any friends as centers of a cluster.

Normally Hierarchical Clustering continues until all nodes are grouped into one cluster, and the hierarchy of clusters is simply cut (or stopped early) when the desired number of clusters has been reached. Since we do not naturally have a desired number of clusters, we decided instead to set a maximum distance threshold between groups. This threshold is a distance value where we consider two friends or two groups of friends to be disconnected from each other. The clustering algorithm stops when the smallest value between any two groups or friends is greater than the threshold value. We believe this gives us results that are most relevant to our goal, which is to discover
close-knit groups of friends. The size and number of clusters is less important to us.

In Hierarchical Clustering there are three common choices for deciding the distance between clusters of nodes. One technique is to consider only the smallest distance between any nodes from each cluster. This is called single-linkage clustering [cite]. Applied to our application, this would mean that to join a clique, someone would only need to be close to one person in that clique. This alone might not give us the results we want and might even give us very large, less meaningful, clusters. Another technique is to do the opposite, which is to consider the maximum distance between two nodes in each cluster. This is called complete-linkage clustering. In our application this would mean that to be in a clique, a friend must have a distance no less than our threshold to every member of a clique. In other words, someone must have interacted in a non-trivial way with the entire clique to join. While this might give us close-knit cliques, it may needlessly eliminate friends from cliques where they interact with all but maybe one person. The third common method is to compare average distances between all nodes in each cluster. This method makes the most sense for application since all members of a clique are taken into account which gives the most representative value for whether or not a person belongs within a clique.

Even though our algorithm is based primarily on average distances, the minimum and maximum value distances might also be useful for determining eligibility within a clique. Keeping this in mind, we developed our clustering method to be able to use any of the three clustering methods or a weighted combination of the three.

**Validation and Analysis**

TODO...

**Testing the algorithm against test data:**

To test and train our algorithm, we created a fake data set of ten facebook users, as shown in figure 1. Our primary user is number five, who will run the Cliques application. Every user is a friend to our primary user. We simulate four cliques by giving each user three photos. Each photo for a user has the members of their clique tagged in it. Thus, our results should show four cliques: two cliques of two users and two clique of three users, displayed in figure 1.
We use this as a training data set by increasing or decreasing the threshold in our clustering algorithm until it produces the set of clusters expected. Table 1 shows the threshold values and the corresponding clusters.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3-10</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>12&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Threshold to clusters for sample data set

As seen in table 1, threshold values 3-10 gave us the number of cliques expected. However, all threshold values in this range put user 5 in a clique with user 8, 9, and 10. Since our algorithm misplaced 1 user out of 10, we will say our algorithm has an error rate of 10%. However since those threshold values still gave 90% accuracy, we will chose a mid threshold of 6. One concern is that the data set is so small the threshold value might not apply to large data sets, and possibly give us skewed results. Therefore, we will change the threshold value in larger data set as well.

Testing the algorithm against real data:
For the large data set, we use Thomas Dvornik’s facebook profile. Within his friends there are six major cliques, which could possibly be broken up into smaller ones. Friends who don’t belong in these cliques should be considered outliers, thus creating their own individual clique or a clique together.

A cluster with over 50% nodes belonging to an expected clique will be considered correct. With a threshold value of 1, it created 59 cliques as seen in graph 1. This turned out to be so much that the majority of incorrect clusters where friends who were by themselves, when they should be in clique. In fact, as the threshold gets closer to zero, then number of cliques becomes the number of friends. We expected 8% of Thomas’s friends would be clustered in their own clique. So in this data set, as the threshold gets closer to zero, the error rate approaches 92%. In General, the error will approach the percentage of all members who should be in cliques.

As the threshold increases, the number of cliques approaches 1. In this case, the error rate would be 100% because each expected clique would contain less than 50% of that one cluster. In general, the higher your threshold gets, the more skewed your results will be, unless you only expect one clique. However, in that case, we would want out algorithm to try to find smaller cliques within that large one.

We increased our threshold from 1 to 10 and calculated the percentage of corrected clusters, which can be seen in graph 2. The highest percentage happened at a threshold of 3, and the next nigh point happened at a threshold of 6. However, we felt that 30 cliques is a too much. Also in our sample data set, we chose our threshold value to be 6. Therefore, we will use 6 as our threshold value.
Graph 2: Percent of correct cliques

**Analysis:**
Our clustering algorithm worked great on a small sample data, giving us the results we expected. However, on a larger data set we only achieved 26.7% accuracy. Therefore our clustering did not appropriately cluster the friends into cliques. This could be because people use Facebook in different ways and we only took into consideration one attribute because of physical limitations. If we could improve the efficiency of the algorithm or get a more powerful server, we could increase the number of attributes we use and therefore increase our accuracy.

It could have also gathered these results from the method of clustering. In some cases, it seemed to cluster users with no relation, which should give it a larger distance. However, maybe there is some real connection that we just can’t see.

**Future Work**

**Distance**

**Caching**

The amount of database queries and computations that are necessary to build distance matrix can get very large and may require a lot of time, therefore caching can be of great benefit. The distance matrix is completely constructed prior to the clustering algorithm and at this point it would need to be cached so the next time the page was loaded the distance matrix could be quickly re-built without any database queries and with minimal computation. Cached distance matrices should only be used for 24 hours because clusters could change with the addition of new friends, wall posts, and tagged photos.
Accuracy

The cliques that were produced from our clustering algorithm were not as accurate as they could be. The distance is computed by using wall posts and user tags in photographs. Although this distance does have a relationship between users it is weak and could be drastically improved by including more data into the distance algorithm.

To create a more accurate distance matrix all aspects of face should be used. This can greatly improve the cliques and find other connections between users. For example, some facebook users only leave comments on photos, posts, videos, and notes. These activities are not considered in our current distance algorithm.

An additional solution to gaining more accuracy would be to use weights for each data source in order to build the distance matrix. Using these weights would increase the accuracy of the distance matrix because there are some data sources that are more important than others.

Clustering
TODO

Conclusion
TODO