Final Project Report

The *read* module contains four public functions: *read_edge, read_feat, read_egofeat, and read_featname*. The *read_edge* function takes the filename of all edges in the network as input and returns a vector containing the pairs of nodes in each edge. The *read_feat* function takes in the path to the file that records the features for each of the nodes in the network, and returns a Hashmap whose keys are the names of the nodes and the values are their corresponding features. The *read_egofeat* function takes in the name of the ego-node in the network, and returns a Hashmap with the name of this ego-node as its key and the features of this ego-node as its value. The *read_featname* function takes in the path to the file that records the featurenames of its corresponding feature file, and returns a Hashmap with the position in the feature file as its key and the feature name as its value. The *graph* module contains a *Graph* struct with fields *edges* and *node_features*. The *edges* field records all pairs of nodes that form an edge in the graph, and the *node_features* field records all nodes and their corresponding features. The *Graph* struct has three private methods: *find_all_indices*, *convert_feat_to_featname*, and *pair_up*. The *find_all_indices* method finds all indices of '1's in the user's feature vector to see what features that user has, the *convert_feat_to_featname* method converts each feature index to its corresponding feature name, and the *pair_up* method pairs up each user node and the feature names it has. The *create_graph* is a public method of the *Graph* struct that takes in the paths to all files needed in the network and returns a new *Graph* object. The *compare* module contains the private functions *greater_featuresize* and *similarity* and a public function *all_similarity*. The *greater_featuresize* function takes in two vectors and returns the larger vector size in f64 type. The *similarity* function takes in the pair of friend-users that form an edge in the network and the *Graph* object containing the information of the network, and calculates the similarity between this single pair of friend users. The *all_similarity* function takes in a *Graph* object and iterates through all pairs of friend users and calculates the similarity of each pair, and finally returns a vector containing the similarities of all pairs of friend users. The *histogram* module contains a *Histogram* struct with fields *range* and *frequency*. The *range* field records all labels for the similarity ranges, and the *frequency* field records the corresponding frequency of each label. The *Histogram* struct has a private method *count_frequency* and a public method *create_histo*. The *count_frequency* method takes in a vector of all similarities and returns a HashMap with the label codes as its keys and the corresponding frequency as its values. The *create_histo* method takes in a vector of all similarities and returns a new *Histogram* object. The *main* module contains the *main* function, that first constructs a *graph* object for the network, and then calculates the similarities of all friend users in the *graph* object and builds a corresponding histogram, and finally uses a for loop to visualize the outcome (the ranges of similarities and the how many pairs of friends in the network are having each range of the similarities).

The outcome of this project indicates that the 20%-30% similarity of two individuals are the most likely for them to become friends, the 30%-40% similarity of two individuals are the second-most likely for them to become friends, and the 10%-20% similarity of two individuals are the third-most likely for them to become friends. The outcome also indicates that the 80%-90% similarity of two individuals are the least likely for them to become friends.