Next Link Prediction in Social Networks

Social and Economic Networks - Project

Guillem Bagaria Portet

June 24, 2018

Contents

Chapter 1

Motivation

The motivation for this project came from the classes during the course. As a future Data Scientist, finding underlying patterns from which to gain knowledge is essential to the field. The reasons why people connect in social networks might be complicated, however the network they generate is relatively simple.

Recommending systems in general have fascinated me and are one of the reasons I got interested in Data Science. Being able to use meager information, such as the connections in a network to predict who might be a friend, or finding underlying similarities between people based on their connections is not only interesting but a valuable resource.

This project will consist of finding a reasonable data set from which to extract a set of network information. Several similarity measures will be used in order to test their performance differences. On the way, I expect to encounter many of the challenges that a production version of this algorithm would encounter in a real world scenario.

Chapter 2

The data

The goal of this project is to implement and test the performance of a next-connection prediction algorithm in a social network with communities, that is a friend suggester algorithm. For this project I will be using a ready-made dataset provided by Stanford department that developed and maintains SNAP (Stanford Network Analysis Project) named *email-EuAll*.

2.1 Description

The network was generated using email data from a large European research institution. The information about all incoming and outgoing email between members of the research institution has been anonymized. There is an edge (u, v) in the network if person u sent person v at least one email. The e-mails only represent communication between institution members (the core), and the dataset does not contain incoming messages from or outgoing messages to the rest of the world.

The dataset also contains "ground-truth" community memberships of the nodes. Each individual belongs to exactly one of 42 departments at the research institute.

This network represents the "core" of the *email-EuAll* network, which also contains links between members of the institution and people outside of the institution (although the node IDs are not the same).

Table 2.1: Dataset Statistics

Nodes	1005
Edges	25571
Nodes in largest WCC	986 (0.981)
Edges in largest WCC	25552 (0.999)
Nodes in largest SCC	803 (0.799)
Edges in largest SCC	24729 (0.967)
Average clustering coefficient	0.3994
Number of triangles	105461
Fraction of closed triangles	0.1085
Diameter (longest shortest path)	7
90-percentile effective diameter	2.9

The dataset can be found in the SNAP email-Eu-core network Dataset information page at `https://snap.stanford.edu/data/email-Eu-core.html` [June 2018].

2.2 *A priori* data issues and caveats

- Large data set: The computation time required increases exponentially with the number of nodes.
- A random sample would be very weakly connected: Strongly connected graphs are those that keep their component number whenever an increasingly large amount of edges are removed. Weakly connected graphs, such as a randomly selected set of nodes or edges, will result in mostly line graphs, which contain too little topological similarity information to guess their next connections.

Chapter 3

Data Cleaning

The data can indeed be used *as is*, but several issues arise requiring some previous treatment. The node and edge list, as well as the community labeling will be cleaned of excess data that might hinder the performance or usefulness of the algorithm as well as to improve the performance during the testing stage.

- Fix the number of nodes, for computation and results manageability purposes.
- Fix the maximum number of edges per node to avoid weakly connected graphs heavily reliant on a hub. This turned out to be unnecessary.
- Fix the minimum number of edges per node to avoid nodes being isolated when randomly removing edges.

3.1 Breadth First Search

After many different attempts to sample the data it became evident that one of the main problems that such an algorithm faces is the sparsity of connections or weakly connected graph. In this scenario, removing links at random might cause communities to be lost making their recovery through similarity measures to be impossible.

By using a Breadth First Search network exploration algorithm we are following only connected components. This assures strong connectivity to a very high degree albeit there is still some randomness left in the sampling by selecting the starting node. By using hubs as starting points, the expected number of edges per node should be maximized.

Chapter 4

Similarity Measures

4.1 Cosine Similarity

Uses the pairwise projection of every vector in the adjacency matrix to quantify how "close" they are. Since we are using unweighted binary edges we can simplify the Cosine Similarity function:

$$Sim(A,B) = \frac{A \cap B}{|A| \cdot |B|}$$

4.2 Jaccard Similarity

Intersection over union. Shows how many connections two nodes share, compared to the ones they don't.

$$Sim(A,B) = \frac{A \cap B}{A \cup B}$$

4.3 Inverse log-weighted

The number of common neighbors weighted by the inverse logarithm of their degrees. This gives higher importance in low degree node edges. The *igraph* implementation was used, based on the paper *Friends and neighbors on the Web* by Lada A. Adamic and Eytan Adar, 2003.

$$Sim(A,B) = \frac{A \cap B}{log(\text{degree}(A \cap B)}$$

Chapter 5

Scoring

In order to evaluate the performance of the algorithm the guesses and the truth will be compared as follows:

1. A number of edges proportional to the network total, are removed at random and kept in a truth list.
2. The algorithm is run, outputting a list of edges ordered by likelihood (similarity) to be the next link, truncated at the same length as the truth list.
3. The truth and guessed lists are compared to see which (undirected) edges there are in common.
4. The result is the fraction of edges in truth found in the guessed list.

Chapter 6

Results

After running many tests, in Table 6 is an example of the results obtained in systematic tests.

6.1 Discussion

- The results are highly dependent on the network topology which in this case is a largely random process and dependent on the initially selected node.
- Jaccard Similarity shows consistently better performance in predicting connections than both Cosine Similarity and Inverse log-Weighted Similarity.
- The performance overall is lower than expected.

6.2 Next Steps

Clearly there is room for improvement. Here are some of the most immediate issues to address:

- Use larger numbers of edges per node, closer to a fully connected network
- Use the more complex methods in the literature review
- Use attribute weighing to achieve production level performance
- Devise other scoring methods that include the likelihood ranking
- Add Centrality as a model for likelihood for the next connection

	Similarity measure		
Number of Nodes	Cosine Similarity	Jaccard Similarity	Inverse log weight
25	0.091	0.182	0.091
35	0.040	0.200	0.040
45	0.222	0.194	0.194
55	0.035	0.122	0.070
65	0.050	0.088	0.050
100	0.028	0.119	0.073

Table 6.1: Score results for various network sizes

Chapter 7

Annex

R code

```
library(igraph)
library(dplyr)
library(reshape2)
getwd()
setwd('~')
edge_list <- read.csv2('Desktop/networks_data/email-Eu-core.txt', sep = ' ')
communities <-
    read.csv2('Desktop/networks_data/email-Eu-core-department-labels.txt', sep = '
    ')
colnames(communities) <- c('Node','Community')
communities <- arrange(communities,communities$Node) # Order by node number Node

# Remove 0's from the node list
# edge_list <- edge_list-1
# communities <- communities

#Rank nodes by number of connections to identify hubs
node_connections <- as.data.frame(table(unlist(edge_list)))
arrange(desc(node_connections$Freq))
colnames(node_connections) <- c('Node','EdgeQuant')
node_connect_comm <- merge.data.frame(x=node_connections, y=communities, by =
    'Node')

# Create a manageable dataset
Nodes <- 300
MinEdgesPerNode <- 10
MaxEdgesPerNode <- 600

# Delete nodes with less than 10 nodes and more than MaxEdgesPerNode
node_connections_useful <-
    node_connect_comm[node_connect_comm$EdgeQuant>=MinEdgesPerNode &
    node_connect_comm$EdgeQuant<=MaxEdgesPerNode,]

# Create a new edge list only containing nodes in previous list
edgelist_new <- edge_list[edge_list$X0 %in% node_connections_useful$Node &
    edge_list$X1 %in% node_connections_useful$Node,]

# Create a strongly connected graph using breadth first search
G <- graph_from_data_frame(d = edgelist_new, directed = FALSE)
```

```
f <- function(graph, data, extra) {
 data['rank'] == Nodes
}
bfsy  <- bfs(G, root=120, "all", order=TRUE, callback = f)
suby <- bfsy$order[1:Nodes]
newgraph = induced.subgraph(G, suby)

# Generate adjacency matrix from the edge list
Adj <- as_adjacency_matrix(graph = newgraph, type = 'both', sparse=FALSE, names =
    TRUE)
Adj <- Adj[1:Nodes,1:Nodes]
Adj <- Adj*upper.tri(Adj, diag = FALSE)

newgraph <- simplify(graph=newgraph, remove.multiple = TRUE, remove.loops = TRUE)
e <- get.edgelist(newgraph, names = FALSE) # Rename edges from 1 to #Nodes
colnames(e) = c('X0','X1')

# Plot graph
pg <- graph_from_data_frame(d = e, directed = FALSE)
plot(pg)

# Ravasz algorithm for community clustering, dendrogram plotting
X <- matrix(0, nrow = Nodes, ncol = Nodes)
for(i in 1:Nodes){
  print(paste0("Row: ",i))
  for(j in 1:Nodes){
    if(i!=j){
      print(j)
      di <- 0
      dj <- 0
      J <- 0
      for(k in 1:Nodes){
        print(k)
        J <- J+Adj[i,k]*Adj[j,k]
        di <- di+Adj[i,k]
        dj <- dj+Adj[j,k]
      }
      if(J>0){
        X[i,j] <- J/(min(di,dj))
        print('positive')
      } else {
        X[i,j] <- J/(min(di,dj)+1)
        print('negative')
      }
    }
  }
}

# Plot dendrogram
# prepare hierarchical cluster
hc = hclust(dist(X))
plot(hc)
```

```
# Remove edges at random
number_of_edges = length(e)/2
dropped <- round(runif(n=integer(0.1*number_of_edges), min = 1, max =
    number_of_edges))
new_edge_truth <- data.frame(e[dropped,])
e_missing <- data.frame(e[-dropped,])
Ge_m <- graph_from_data_frame(d = e_missing, directed = FALSE)
Ae_m <- as_adjacency_matrix(graph = Ge_m, type = 'both', sparse=FALSE, names =
    FALSE)
# colnames(Ae_m) <- c(1:Nodes)

# Predict the next edges that will be created
# Around 5%, very low performance
# 1. Cosine Similarity.
CosSim <- function(ix){
    Q = Ae_m[ix[1],]
    W = Ae_m[ix[2],]
    if(Q==0||W==0){
      return(0)
    }
    return(sum(Q*W)/sqrt(sum(Q^2)*sum(W^2)))
}
cmb <- expand.grid(i=1:Nodes, j=1:Nodes)
similarity_matrix <- matrix(data = apply(X = cmb, MARGIN = 1, FUN = CosSim), nrow =
    Nodes, ncol = Nodes)

# 3. Jaccard. Up to 30% performance!
similarity_matrix <- similarity(Ge_m, method = "jaccard")

# 4. Iverse log-weighted.
similarity_matrix <- similarity(Ge_m, method = "invlogweighted")


# Fix matrix
diag(similarity_matrix) <- 0
similarity_matrix[lower.tri(similarity_matrix)] <- 0


# Recover a list of edges with predicted high cosine similarity
candidate_edges <- melt(similarity_matrix, varnames = c('row', 'col'))
candidate_edges <- arrange(candidate_edges, desc(candidate_edges$value))

# Rating the performance
# Take the top link suggestions over number in truth
links <- length(new_edge_truth$X0)
top <- candidate_edges[1:links,]
r <- data.frame(X0=integer(), X1=integer())
for(i in 1:links){
  for(j in 1:links){
    if(top$row[i]==new_edge_truth$X0[j]){
      if(top$col[i]==new_edge_truth$X1[j]){
        r[nrow(r) + 1,] = list(top$row[i],top$col[i])
        break
```

```
      }
    }
    if(top$col[i]==new_edge_truth$X0[j]){
      if(top$row[i]==new_edge_truth$X1[j]){
        r[nrow(r) + 1,] = list(top$row[i],top$col[i])
        break
      }
    }
  }
}
score <- nrow(r)/links
```