Modeling Social Data: Homework 3

Anshul Doshi

Anshul Doshi (ad3222) Modeling Social Data Assignment: 3

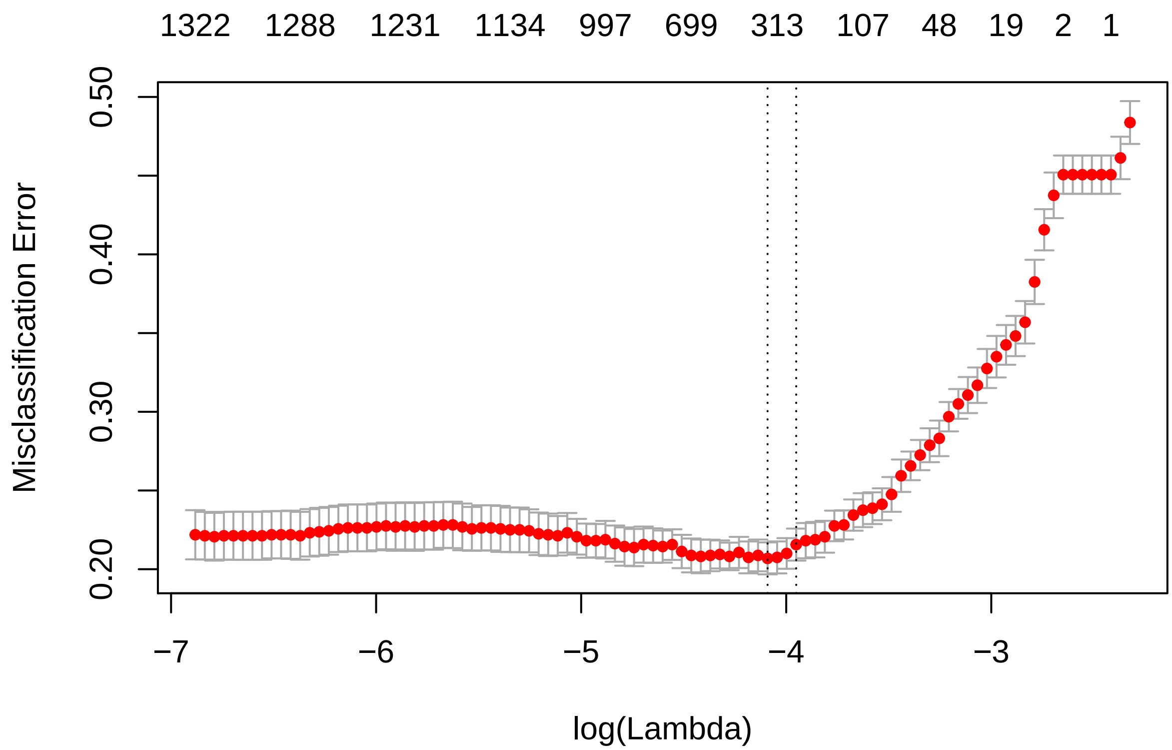
Problem 1: Logisitic Regression for Article Classification

```r
#set working directory
setwd("~/Desktop/Spring2017/Modeling Social Data/HW3_ad3222")

# read business and world articles into one data frame
business_data <- read.delim(file = "business.tsv", sep = "\t")
world_data = read.delim(file = "world.tsv", sep = "\t")

article_data <- rbind(business_data, world_data)

# create a Corpus from the article snippets
snippet_corpus <- VCorpus(VectorSource(article_data$snippet))
snippet_corpus
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 2000
```

```r
# create a DocumentTermMatrix from the snippet Corpus
# remove punctuation and numbers
dtm <- DocumentTermMatrix(snippet_corpus, control =
              list(removePunctuation =TRUE, removeNumbers=TRUE, stopwords = TRUE))

# convert the DocumentTermMatrix to a sparseMatrix, required by cv.glmnet
# helper function
dtm_to_sparse <- function(dtm) {
 sparseMatrix(i=dtm$i, j=dtm$j, x=dtm$v, dims=c(dtm$nrow, dtm$ncol), dimnames=dtm$dimnames)
}

sparse_dtm <- dtm_to_sparse(dtm)

# create a train / test split
set.seed(42)

num_articles <- nrow(sparse_dtm)
frac_train <- 0.8
num_train <- floor(num_articles * frac_train)

# randomly sample rows for the training set
ndx <- sample(1:num_articles, num_train, replace=F)

# used to fit the model
articles_train <- sparse_dtm[ndx, ]
section_train <- vector()
for (i in 1:num_train) {
```

```r
  section_train[i] = article_data$section[as.numeric(rownames(articles_train)[i])]
}

# used to evaluate the fit
articles_test <- sparse_dtm[-ndx, ]
section_test <- vector()
for (i in 1:(num_articles-num_train)) {
  section_test[i] = article_data$section[as.numeric(rownames(articles_test)[i])]
}

# cross-validate logistic regression with cv.glmnet, measuring auc
#cvfit <- cv.glmnet(articles_train, section_train, family="binomial", type.measure = "auc")
#plot(cvfit)
cvfit <- cv.glmnet(articles_train, section_train, family="binomial", type.measure = "class")
plot(cvfit)
```




```r
# evaluate performance for the best-fit model
df <- data.frame(actual_num=section_test, log_pred = predict(cvfit,
                      articles_test, s="lambda.min", type="class")) %>%
  mutate(actual = ifelse(actual_num == 1, 'business', 'world')) %>%
  mutate(pred = ifelse(X1 == 1, 'business', 'world'))
head(df)
```

```
##   actual_num X1   actual     pred
## 1          1  1 business business
## 2          1  1 business business
## 3          1  1 business business
## 4          1  2 business    world
## 5          1  1 business business
## 6          1  1 business business
```

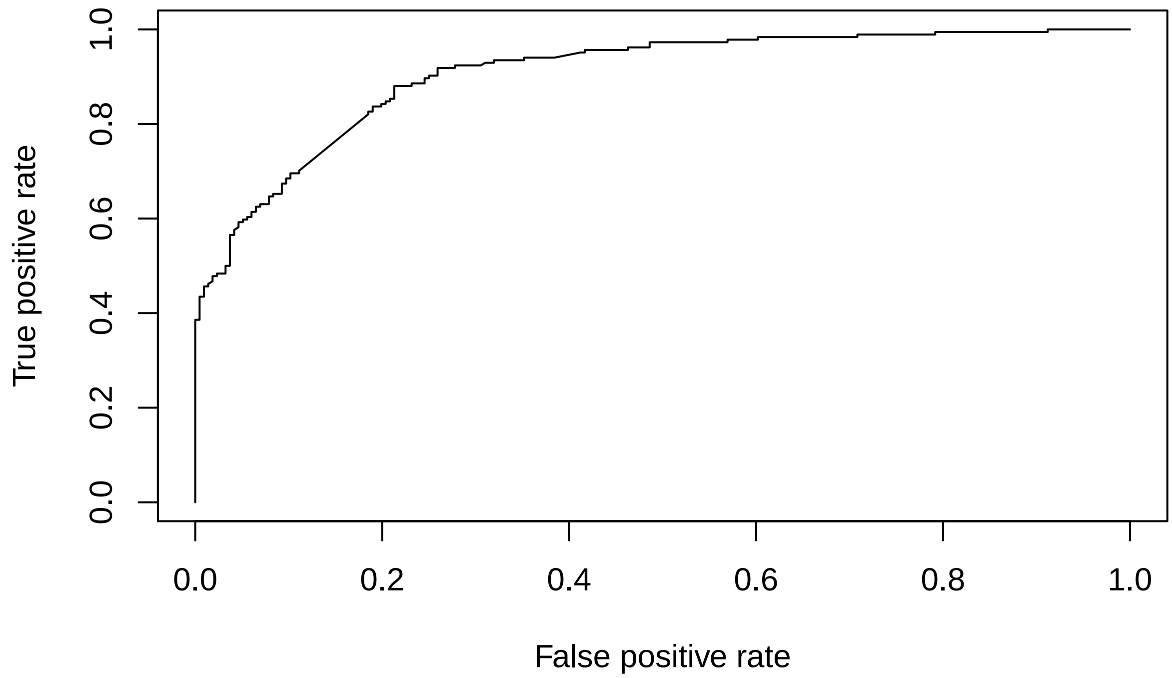
```r
#Confusion matrix
table(actual = df$actual, predicted = df$pred)
```

```
##           predicted
## actual     business world
##   business      163    53
##   world          21   163
```

```r
# accuracy: fraction of correct classifications
df %>%
  summarize(acc = mean(pred == actual))
```

```
##     acc
## 1 0.815
```

```r
# plot ROC curve and output accuracy and AUC
probs <- predict(cvfit, articles_test, type="response",  s="lambda.min")
pred <- prediction(probs, section_test)
perf_lr <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf_lr)
```

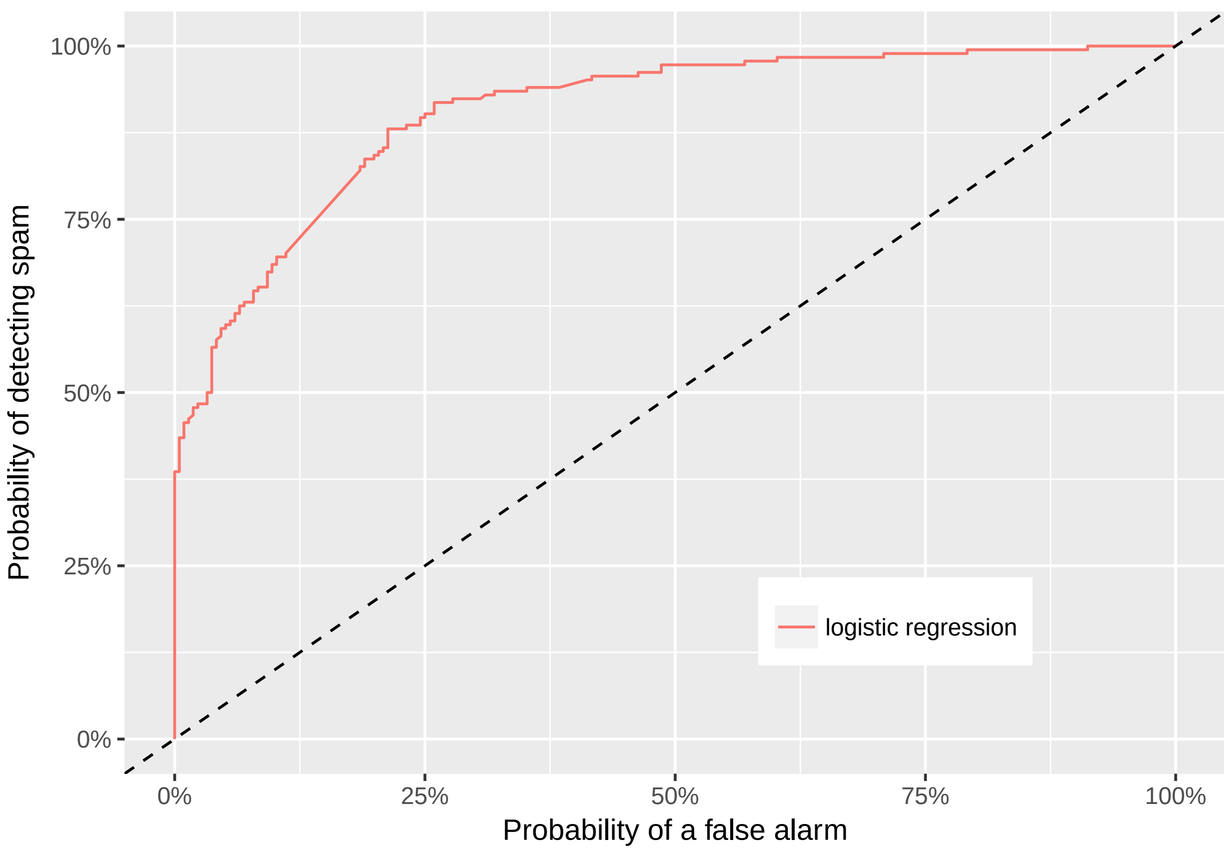



```r
performance(pred, 'auc')
```

```
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
```

```
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.906011
##
##
## Slot "alpha.values":
## list()
```

```
#ROC curve
roc_lr <- data.frame(fpr=unlist(perf_lr@x.values), tpr=unlist(perf_lr@y.values))
roc_lr$method <- "logistic regression"
roc_lr %>%
  ggplot(data=., aes(x=fpr, y=tpr, linetype=method, color=method)) +
  geom_line() +
  geom_abline(linetype=2) +
  scale_x_continuous(labels=percent, lim=c(0,1)) +
  scale_y_continuous(labels=percent, lim=c(0,1)) +
  xlab('Probability of a false alarm') +
  ylab('Probability of detecting spam') +
  theme(legend.position=c(0.7,0.2), legend.title=element_blank())
```




```
# extract coefficients for words with non-zero weight
# helper function
get_informative_words <- function(crossval) {
  coefs <- coef(crossval, s="lambda.min")
  coefs <- as.data.frame(as.matrix(coefs))
```

```r
  names(coefs) <- "weight"
  coefs$word <- row.names(coefs)
  row.names(coefs) <- NULL
  subset(coefs, weight != 0)
}

# show weights on words with top 10 weights for business
get_informative_words(cvfit) %>%
  arrange(weight) %>%
  head(10)
```

```
##       weight      word
## 1  -1.853283   company
## 2  -1.561869   pricing
## 3  -1.367786   billion
## 4  -1.345750       tax
## 5  -1.323066  companys
## 6  -1.292382 financial
## 7  -1.240644  business
## 8  -1.174210      firm
## 9  -1.172665 companies
## 10 -1.090536    spicer
```

```r
# show weights on words with top 10 weights for world
get_informative_words(cvfit) %>%
  arrange(desc(weight)) %>%
  head(10)
```

```
##       weight     word
## 1  1.2964063   russia
## 2  1.0635509  killing
## 3  1.0554619   leader
## 4  0.9966137   region
## 5  0.9745761    north
## 6  0.9433456     vice
## 7  0.9154837   police
## 8  0.8270328 military
## 9  0.8186487  license
## 10 0.8160355   failed
```

Problem 2: Close vs. Distant Friends

The Problem:

Consider A to be a "close-friend" network where each person has a directed edge to 10 of their closest friends.

Consider B to be a "distance-friend" network where each person is connected to their 21st thorugh 30th ranked friends.

Let C be the average number of people a person can reach in six steps from the close-friend network described in A and let D be the avaerage number of people a person can reach in six steps from the distant network reffered to in B. Which number will be consistently larger and why?

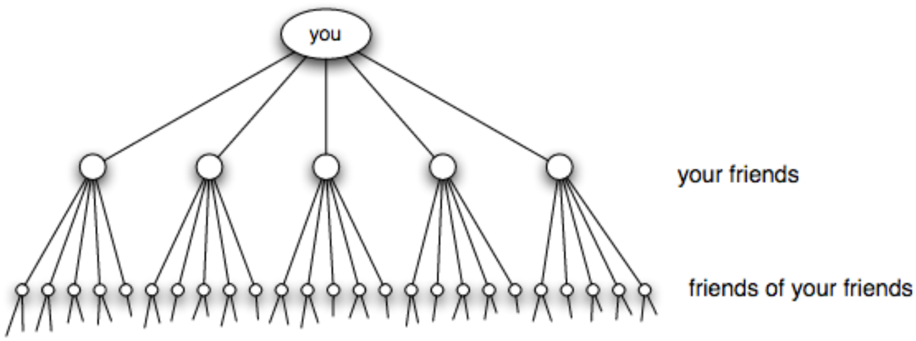
Solution

First lets attack this intuitively and then base our intuition with class notes and rules. Intuitively, D should be larger. In a close-friend network, it should be common for a person's close-friends to be friends themselves. To get the largest quantity of average number reached in six steps, we'd want each person to be connected to as many unique people as possible.

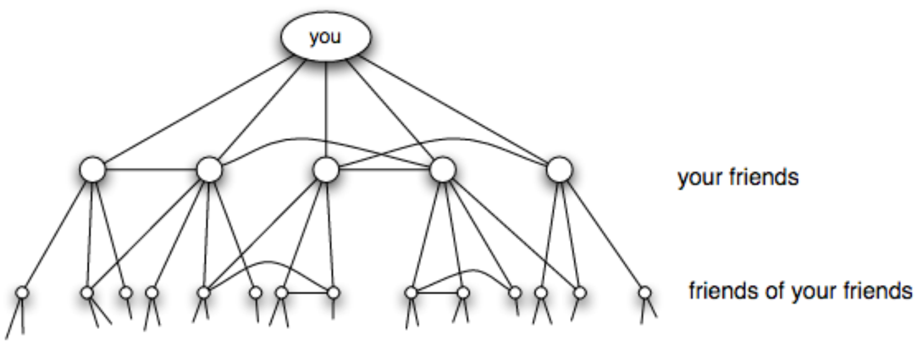
Thus, in this case, the largest quantity we'd get is a 10-ary tree where we fix a person at the root. The largest size is then,

$$1(root) + 10(depth1) + 100 + 1000 + 10000 + 100000 + 1000000 = 1111111$$

This quantity would be the max average six degrees of separation obtained if each friend has another unique 10 friends at each depth such that no edges connect backwards or to anyone already seen in the tree. Visually this would mean:



(a) *Pure exponential growth produces a small world*



(b) *Triadic closure reduces the growth rate*

The exponential version would result in a larger six degrees of separation quantity whereas the backward ties in the second tree would make this quantity smaller. Our intuiotion says that close-friends would be more like the second tree in that theres a higher likelihood that my two best friends are friends themselves and their friends are friends and so on.

Thus, these groups created from a close-friends network limit our average people reached. The distant friend network on the other hand with be more spread out as there's a higher chance distant friends don't know each other so we will be able to reach a larger amount of unknown or uniqe people.

Lets formulate some rules that support this theory. The effect we are examining here is related to the triadic closure effect. The triadic closure effect says that "if two people in a social network have a friend in common, then there is an increased likelihood that they will become friends themselves at some point in the future" (Granovetter, 347)

Quantities C and D will be maximized when the clustering coefficient is minimized such that if we select a person at random, the probability that two randomly selected friends of that person are not friends. Thus, we want to minimize triadic closure since as the graph becomes more interconnected, the greater the clustering coefficient becomes, We want clustering coefficient to be small so we can reach more people.

Graph A will be our close-friends graph which will be filled with strong ties and graph B will be distant friends containing weak ties. According to Granovetter, If a node A has edges to nodes B and C, then the B-C edge is especially likelyto form if A's edges to B and C are both strong ties.

Thus, by this formulation, we can see that Graph A consisted of strong ties so they are especially likely to satisfy the triadic cosure property just as our intuition led us to believe. By satisfying this property, clustering coefficient will increase for graph A and quantity C will go down.

Therefore, D > C because graph B contains weak ties and graph A contains strong ties meaning people in graph A have a higher probability to satisfy the triadic closure property.

Problem 3: Email Network Statistics

a.) Read in edge list 2010 data and plot distribution of edge weights for the entire network

```
theme_set(theme_bw())

col_header <- c("User1", "User2", "Weight")
edge_list_2010 <- read.table("undweighted10.dat", header=FALSE, col.names = col_header)

graph_2010 <- graph.data.frame(edge_list_2010[, 1:2], directed = FALSE)
E(graph_2010)$weight <- as.numeric(edge_list_2010[,3])

gorder(graph_2010)
```

```
## [1] 2066
```

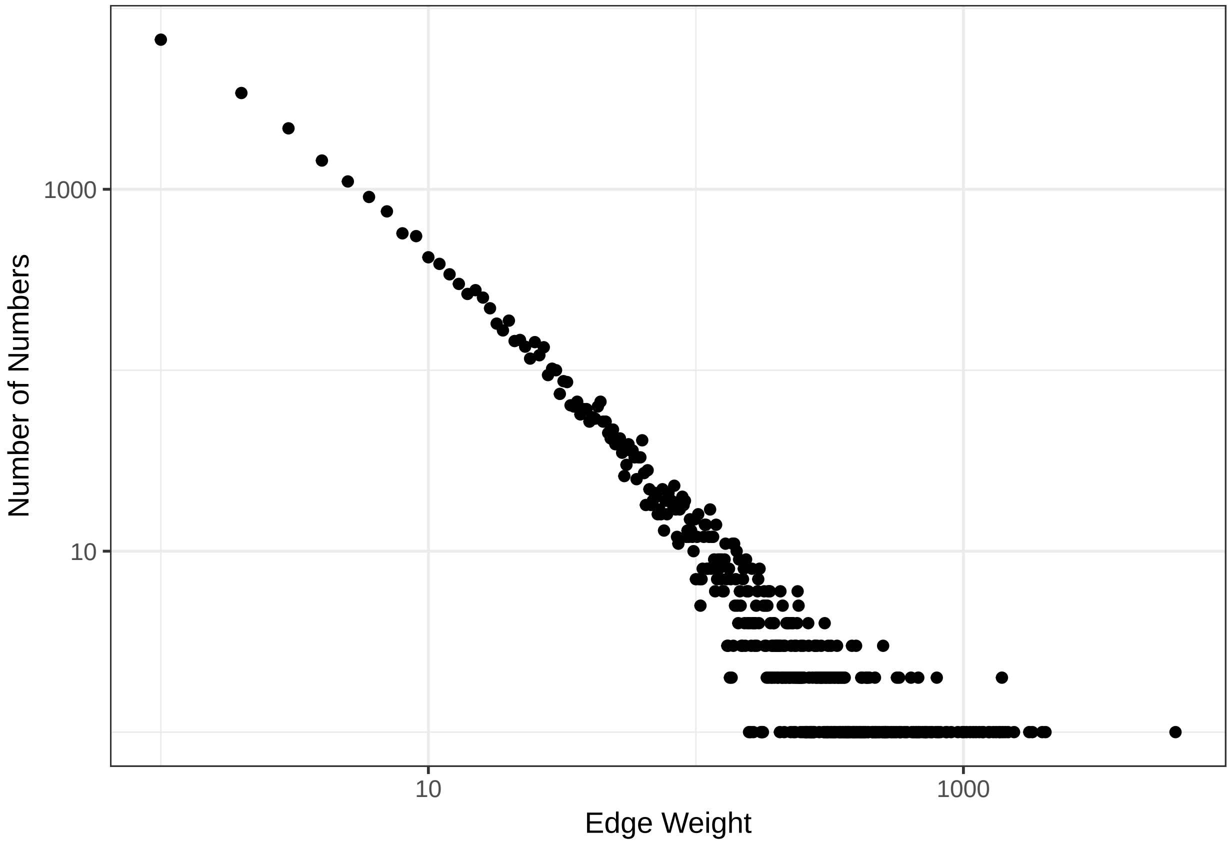
```
max(components(graph_2010)$csize)
```

```
## [1] 2064
```

```
edge10_dist <- edge_list_2010 %>%
  group_by(Weight) %>%
  summarize(num_nodes = n())
edge10_dist
```

```
## # A tibble: 459 × 2
##    Weight num_nodes
##     <int>     <int>
## 1       1      6714
## 2       2      3405
## 3       3      2172
## 4       4      1442
## 5       5      1105
## 6       6       907
## 7       7       755
## 8       8       571
## 9       9       551
## 10     10       421
## # ... with 449 more rows
```

```r
ggplot(edge10_dist, aes(x=Weight, y=num_nodes)) +
  geom_point() +
  xlab("Edge Weight") +
  ylab("Number of Numbers") +
  scale_y_log10() +
  scale_x_log10()
```

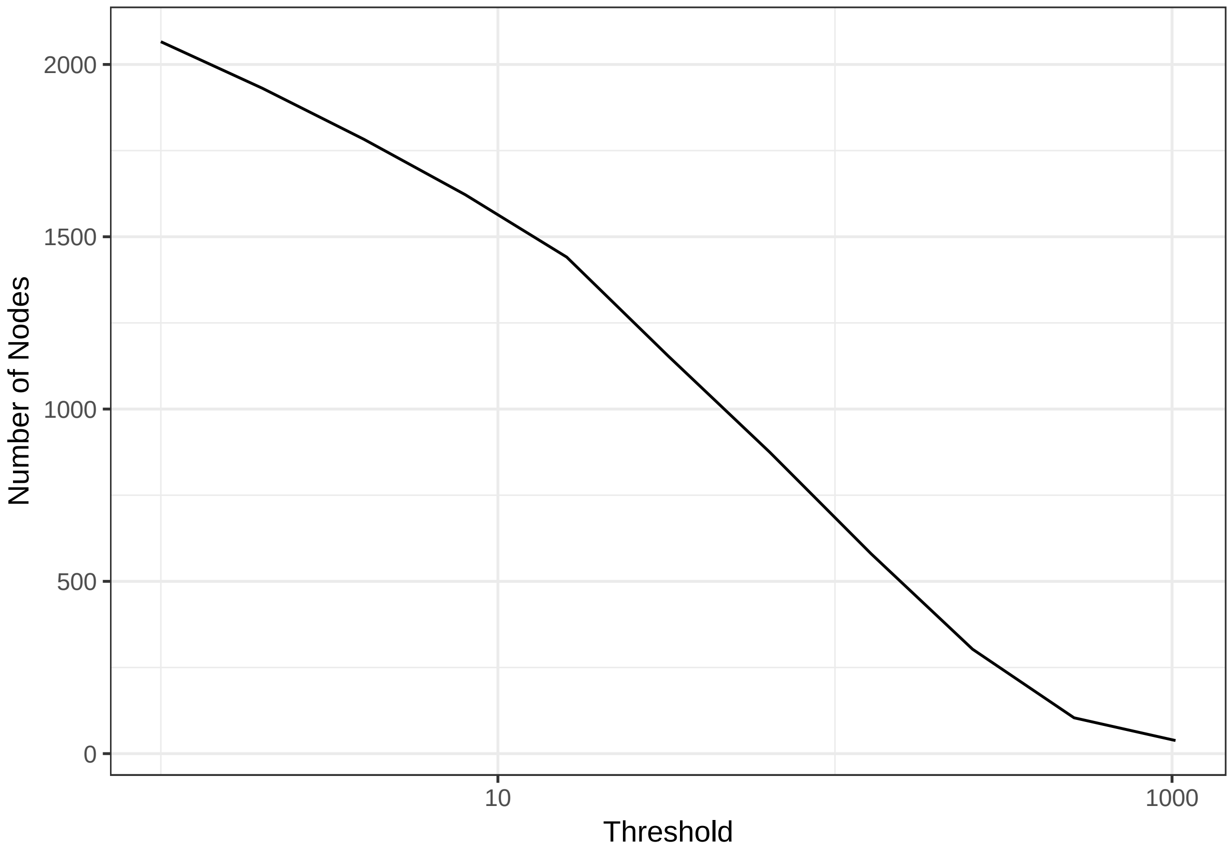


The distribution of edge wights show that most nodes are weakly tied and as edge weight rises, the number of nodes fall.

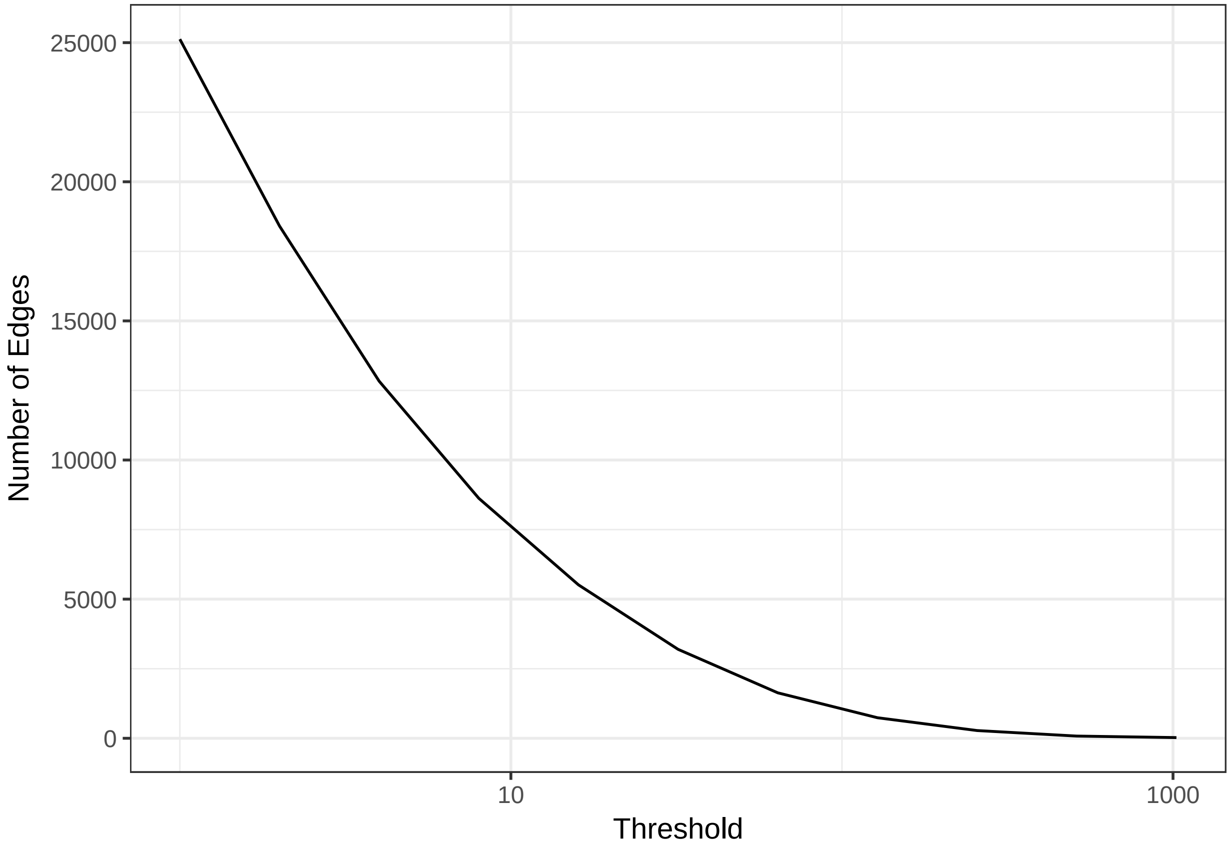
b.) Define a sequence of 12 thresholds (0. . . 1024). Remove all edges whose weight is below a given threshold and compute various stats.

```r
thresholds <- vector()
thresholds[1] = 0
for(i in 1:11) {
  thresholds[i+1] <- 2^(i-1)
}

num_nodes <- vector()
num_edges <- vector()
num_connected_comp <- vector()
frac_nodes <- vector()
avg_dists <- vector()

for(t in 1:12) {
  #Remove values below threshold
  edge_data <- edge_list_2010 %>%
```

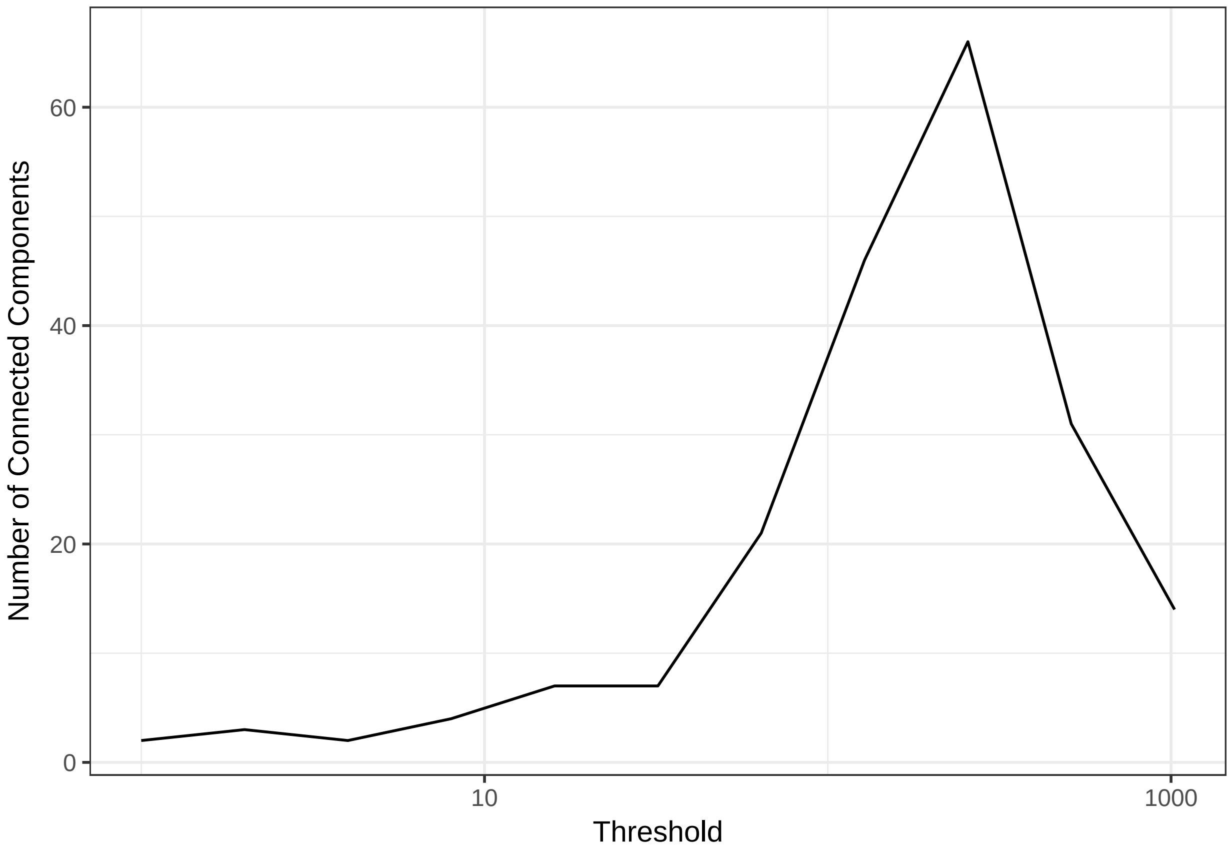
```
    filter(Weight >= thresholds[t])

  #Create graph
  my_graph <- graph.data.frame(edge_data[, 1:2], directed = FALSE)
  E(my_graph)$weight <- as.numeric(edge_data[,3])

  #number of nodes
  num_nodes[t] <- gorder(my_graph)

  #number of edges
  num_edges[t] <- ecount(my_graph)

  #num_connected_comp
  num_connected_comp[t] <- components(my_graph)$no

  #fraction of nodes contained in the largest connected component of the network
  frac_nodes[t] <- max(components(my_graph)$csize) / gorder(my_graph)

  # The average distance between all pairs of nodes in the network
  avg_dists[t] <- mean_distance(my_graph, directed = FALSE, unconnected = TRUE)
}

#Plot num_nodes
plot_data <- data.frame(cbind(thresholds, num_nodes))
ggplot(plot_data[-1, ], aes(x=thresholds, y=num_nodes)) +
  geom_line() +
  xlab("Threshold") +
  ylab("Number of Nodes") +
  scale_x_log10()
```



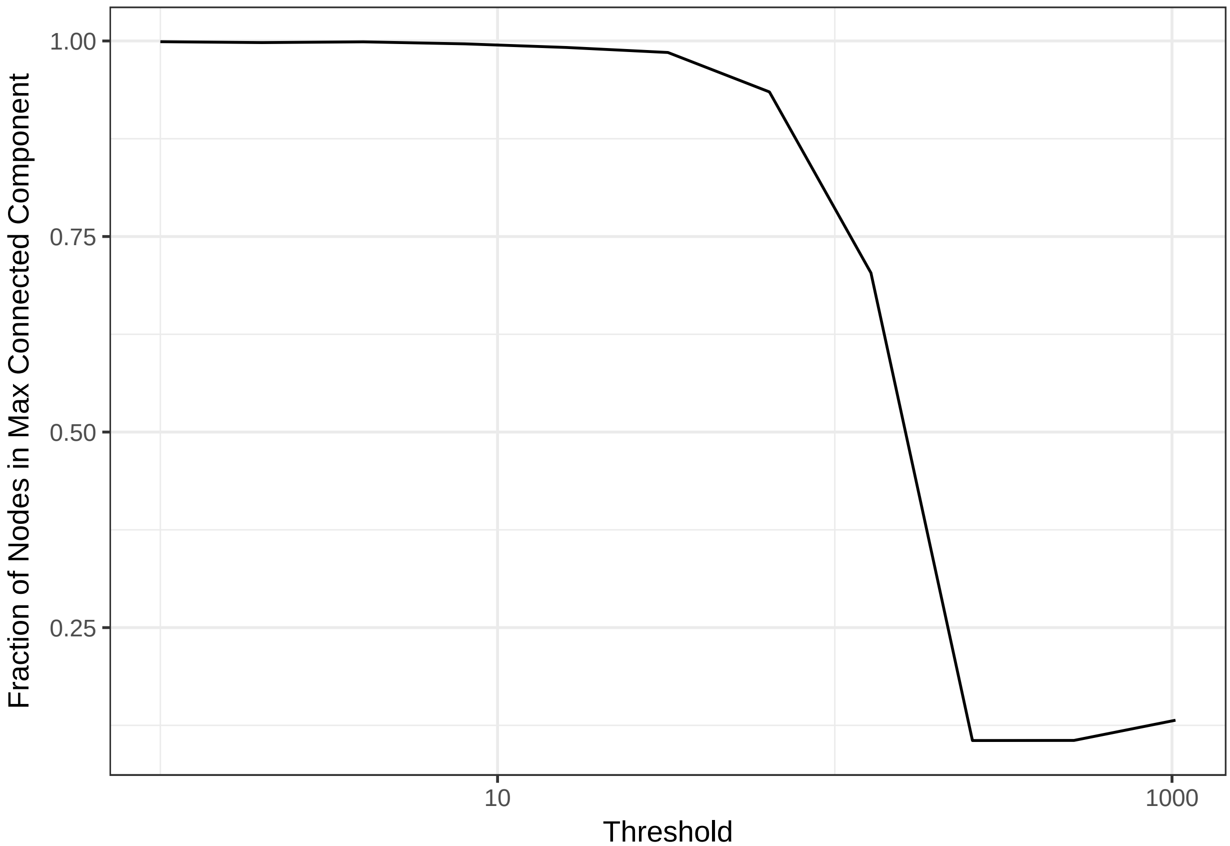

```
#Plot num_edges
plot_data <- plot_data %>%
  cbind(num_edges)
ggplot(plot_data[-1, ], aes(x=thresholds, y=num_edges)) +
  geom_line() +
  xlab("Threshold") +
  ylab("Number of Edges") +
  scale_x_log10()
```



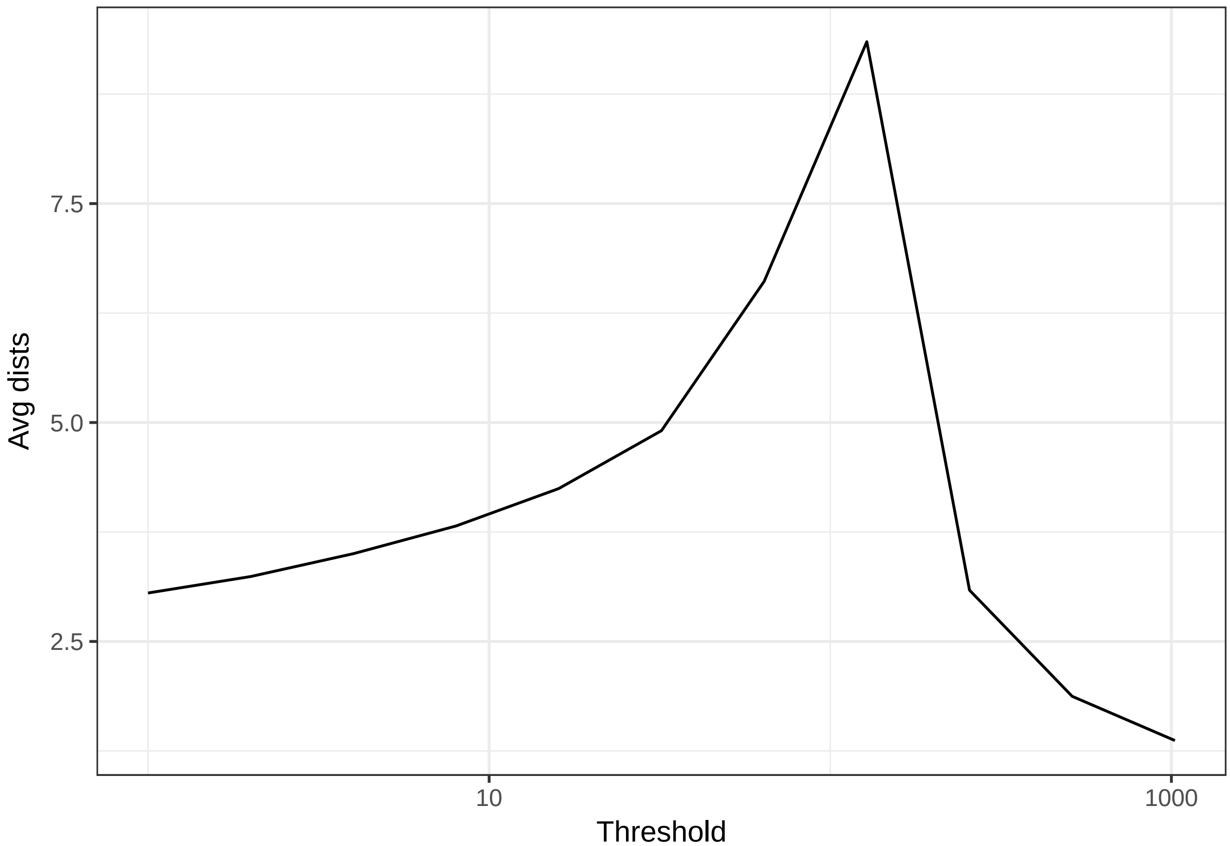

```
#Plot connected components
plot_data <- plot_data %>%
  cbind(num_connected_comp)
ggplot(plot_data[-1, ], aes(x=thresholds, y=num_connected_comp)) +
  geom_line() +
  xlab("Threshold") +
  ylab("Number of Connected Components") +
  scale_x_log10()
```




```
#Plot fraction nodes
plot_data <- plot_data %>%
  cbind(frac_nodes)
ggplot(plot_data[-1, ], aes(x=thresholds, y=frac_nodes)) +
  geom_line() +
  xlab("Threshold") +
  ylab("Fraction of Nodes in Max Connected Component") +
  scale_x_log10()
```




```r
#Plot avg distance
plot_data <- plot_data %>%
  cbind(avg_dists)
ggplot(plot_data[-1, ], aes(x=thresholds, y=avg_dists)) +
  geom_line() +
  xlab("Threshold") +
  ylab("Avg dists") +
  scale_x_log10()
```



Thus, we get 5 different graphs repesting five quantities. Lets examine the shape of each of them:

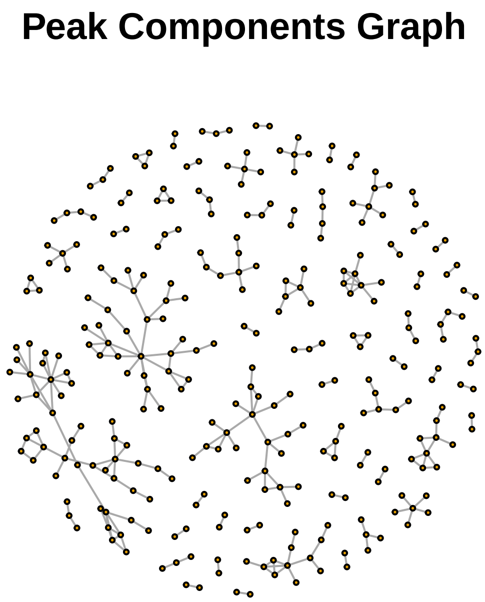
- Number of Nodes
- As threshold increases, we see the number of nodes decrease which makes sense since the greater the threshold is, the more nodes we have to remove that live under said threshold.
- Number of Edges
- As the threshold increases, the more edges we lose. In the beginning we lose them quickly since most edges are weak or have small weights. Later we lose them at a smaller rate as not as many strong edges remain.
- Number of Connected Components
- The number of connected components increases as we start remove more and more edges but then exxperiences a peak and declines. One possibility is that in the beginning since we are removing weak ties, we remove these bridges increasing number of connected components. Later then, when we remove the stronger ties, we start to lose those whole components, decreasing the number of components.
- Fraction of Nodes in largest Connected Component
- At the beginning, our fraction is close to 1 as we haven't delted that many edges. Later, as the threshold increases, our number of components increase and there is a huge decline until the number of compnents hits a peak. At this point, as whole components are removed, the number of components decreases and the fraction in the largest moves slightly up due to this.
- Average Distance
- As threshold increases, avg path increases to a peak before decreasing again. This makes sense as in the beginning we are removing some weaker paths and thus the distance needed to get to certain nodes increases. However, once whole components start to dissappear and the network is a bunch of small connected portions, we get the avg path start to decline again.

c.) Notice the peak in the connected components graph. Lets investigate why this is happening and plot the graph at this point and the next.

```r
max_connected_comp <- edge_list_2010 %>%
  filter(Weight >= thresholds[which.max(num_connected_comp)])

thresholds[which.max(num_connected_comp)]
```

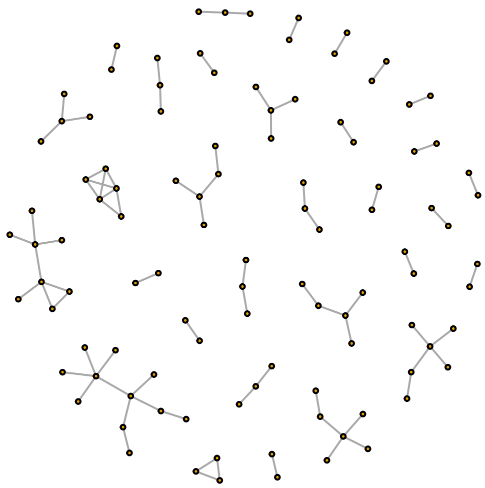
```
## [1] 256
```

```r
max_connected_graph <- graph.data.frame(max_connected_comp[, 1:2], directed = FALSE)
E(max_connected_graph)$weight <- as.numeric(max_connected_comp[,3])

plot(max_connected_graph, vertex.size=2, vertex.label=NA, main="Peak Components Graph")
```




```r
nxt_connected_comp <- edge_list_2010 %>%
  filter(Weight >= thresholds[which.max(num_connected_comp)+1])

nxt_connected_graph <- graph.data.frame(nxt_connected_comp[, 1:2], directed = FALSE)
E(nxt_connected_graph)$weight <- as.numeric(nxt_connected_comp[,3])

plot(nxt_connected_graph, vertex.size=2, vertex.label=NA, main="Next Components Graph")
```

Next Components Graph



It seems that our intuition is right. At the peak we have many components that contain just two nodes connected by an edge making our compnent number high. Thus, in the next iteration, when our threshold moves up, those edges are deleted, wiping out the whole component decreasing component number. We reach a peak at a certain threshold when are able to hold as many compnents with a small number of edges as possible but then these edges are later destroyed as our threshold gets too high.