FRAUD IDENTIFICATION AND ANALYSIS

A PROJECT REPORT

Submitted by

VAIBHAV SHARMA [Reg No: RA1411003010740]
J. SUJIT SHANKAR [Reg No: RA1411003010749]

Under the guidance of

Mr. S. SELVAKUMAR, M.Sc
(Assistant Professor, Department of Computer Science & Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

MAY 2017

Dedicated to our Parents

SRM UNIVERSITY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**FRAUD IDENTIFICATION AND ANALYSIS**" is the bonafide work of " **VAIBHAV SHARMA [Reg No: RA1411003010740], J. SUJIT SHANKAR [Reg No: RA1411003010749 , ,** ", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mr. S. SELVAKUMAR, M.Sc
GUIDE
Assistant Professor
Dept. of Computer Science & Engineering

SIGNATURE

Dr. B. AMUTHA, Ph.D
HEAD OF THE DEPARTMENT
Dept. of Computer Science & Engineering

Signature of the Internal Examiner

Signature of the External Examiner

ABSTRACT

Enron Corporation was one of the world's major electricity, natural gas, communications and pulp and paper companies, with claimed revenues of nearly $101 billion during 2000. At the end of 2001, it was revealed that its reported financial condition was sustained by institutionalized and creatively planned accounting fraud, better known as Enron scandal. In the resulting Federal investigation, there was a significant amount of typically confidential information entered into public record, including tens of thousands of emails and detailed financial data for top executives. Enron has since become a well-known example of wilful corporate fraud and corruption. This report aims at answering whether top level Enron employees had incriminating evidence in their office emails or uncover any unusual patterns in the months leading up to the scandal through an exploratory data analysis. We also try to mine the email inboxes and financial data of Enron to identify persons of interest in one of the greatest corporate fraud cases in human history.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my guide, Mr. S.Selvakumar for his valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing research. All through the work, in spite of his busy schedule, he has extended cheerful and cordial support to us for completing this research work.

Vaibhav Sharma & J Sujit Shankar

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 The Enron Dataset

The dataset used is obtained from multiple sources based on usability and reliability. The main dataset contains emails , about 5 lakh, exchanged between 150 odd users. This data was collected and prepared by the CALO Project. Other datasets include emails associated with certain energy bids obtained by MIT and another email dataset made available by Kaggle. With respect to text analysis, the dataset is focused on one consisting of 855 emails pertaining to energy bids. This data was obtained in 2010 at the Text Mining Conference.

Differing datasets allow for focused and restricted application of analysis and prediction methods. For our purposes, it is imperative to look at Sent mails since the inbox could be filled with a bunch of spam mails that are interspersed with the Received mails. In regards to analysis our purpose is to first apply IDA and then the more complex and insightful EDA. We further our cause by attempting to predict our POI by using various ML algorithms.

1.2 Data Processing

Any analysis process requires a clean and pre-processed dataset to avoid error and inclusion of outliers. Let us look at an example of a raw email text:

The amount of unwanted data in these emails is quite large. It would be cumbersome to deal with while conducting our text and exploratory analysis. In the best case scenario, we would require the body of the email. Since the email text is so structured, Regex expression were used in Python to extract the body of the emails. These mails could be written down as new files using a Python script.

Figure 1.1: Sample Email

```
Message-ID: <13514081.1075840283517.JavaMail.evans@thyme>
Date: Sat, 12 Feb 2000 01:33:00 -0800 (PST)
From: rosalee.fleming@enron.com
To: russell.servat@enron.com
Subject: KLL's e-mail address
Cc: dorothy.barnes@enron.com
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Bcc: dorothy.barnes@enron.com
X-From: Rosalee Fleming
X-To: Russell Servat
X-cc: Dorothy Barnes
X-bcc:
X-Folder: \Kenneth_Lay_Dec2000\Notes Folders\'sent
X-Origin: LAY-K
X-FileName: klay.nsf

Russell, I think it will be Kenneth.Lay@Enron.com.  Do we capitalize
all the words, but com? I noticed that mine is Rosalee.Fleming@Enron.com.
I had sent something to my home e-mail address. This may not be a good
test as I am sending it from my home computer, from his mail-box, but not
from his computer.  Oh, well, we'll see!!
Rosie
```

Since our data pertains to only those that are within the sent and sent items categories, using R, mails satisfying this criteria are retained and the rest removed. This gives a centralized, spam filtered dataset. With respect to text analysis, the dataset has to be cleaned and processed with respect to the words used. A corpus is created which has responsive values based on whether it pertains to energy bids. The corpus undergoes stemming among other text mining operations.

With respect to Machine Learning, all features in the dataset are either financial data or features extracted from emails. Financial data includes features like salary and bonus while the email features include number of messages written/received and to whom/from.

We are interested in labeling every person in the dataset into either a POI or a non-POI (POI stands for Person Of Interest). More than that, if we can assign a probability to each person to see what is the chance she is POI, it would be a much more reasonable model given that there is always some uncertainty.

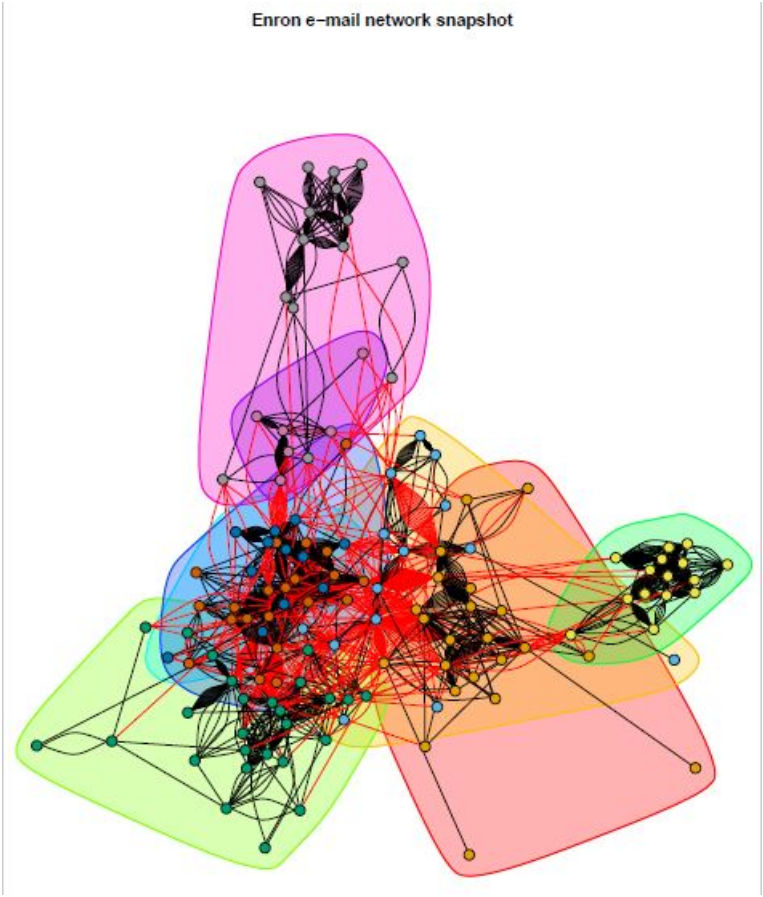
CHAPTER 2

VISUALIZING THE DATA

2.1 Network Overview

In order to understand the grand scheme of things and to easily understand the detailed corruption in Enron, Data Visualization is employed. The Enron email network is visualized using igraph package in R (on the CALO dataset). We restrict the emails to only those sent and received within the @enron.com domain name.

Figure 2.1: Network snapshot



The graph gives an idea about clusters being formed within the network. There are certain nodes in each of the clusters that are distance from activity. It can also be noticed that certain clusters are overlapping, which is fairly common in many large organizations.

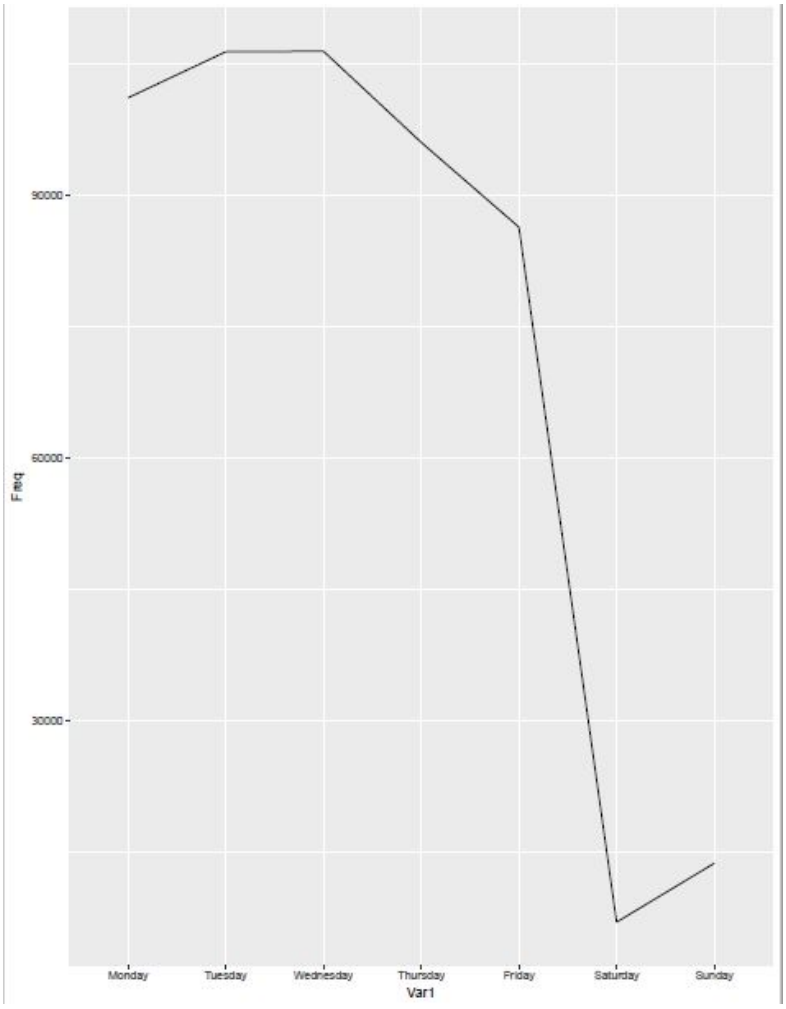
2.2 Frequency of Access

The sheer number of mails that were communicated is large. An inchoate look into the time and number is given below.

2.2.1 Emails Sent Everyday

The following plot gives us the total number of mails sent everyday over the entire period until the collapse of Enron. The bulk of the mails were sent midweek. There is a justified sharp drop from Friday to Saturday. This does not indicate any outliers.

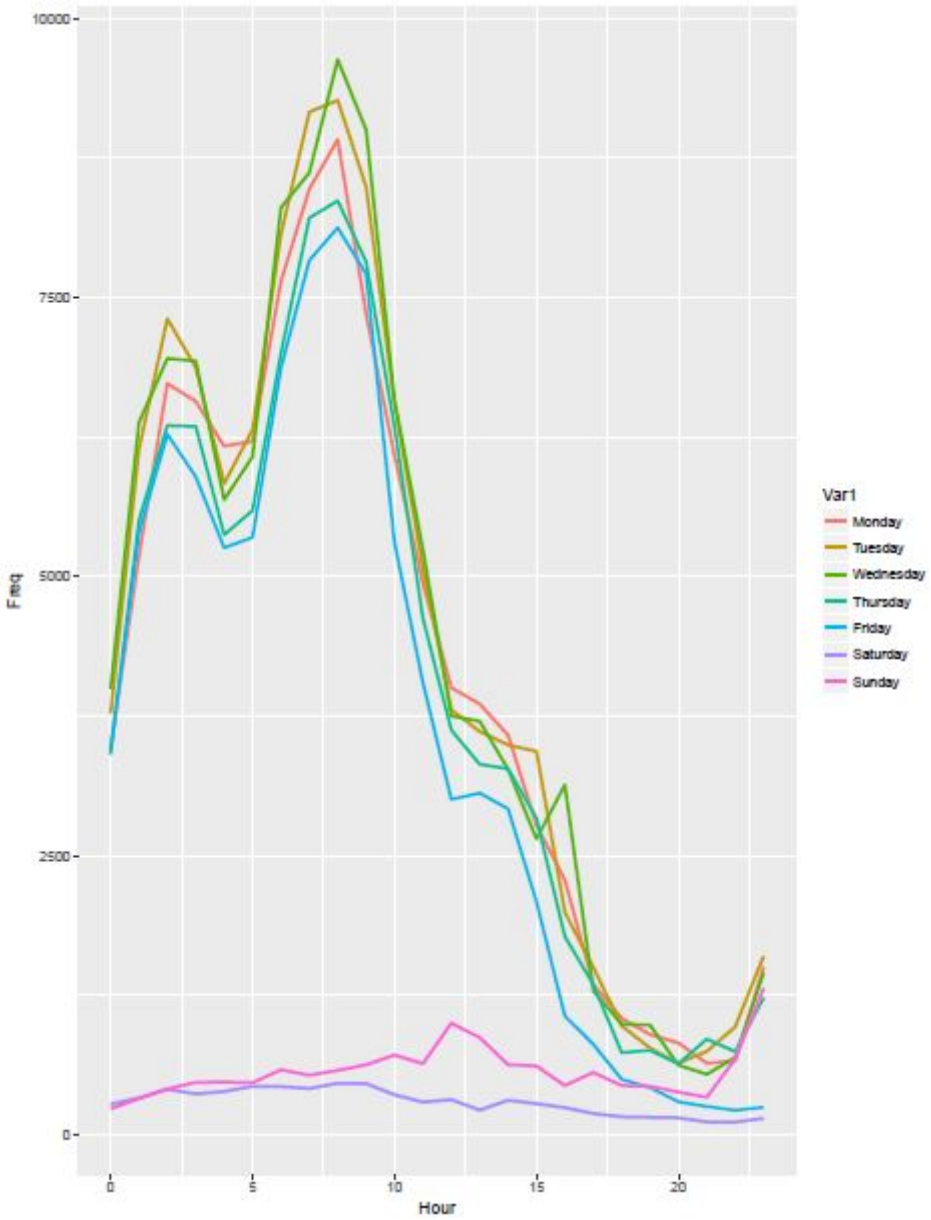
Figure 2.2: Emails Sent Everyday



2.2.2 Frequency of Emails per Hour

This graph gives a plot for the frequency of emails per hour for each of the seven days of the week. The weekdays have a huge peak in the morning. Some doubt falls on late night Sundays as the emails sent dramatically increases. This could be due to different working hours, but is not seen so during other days.

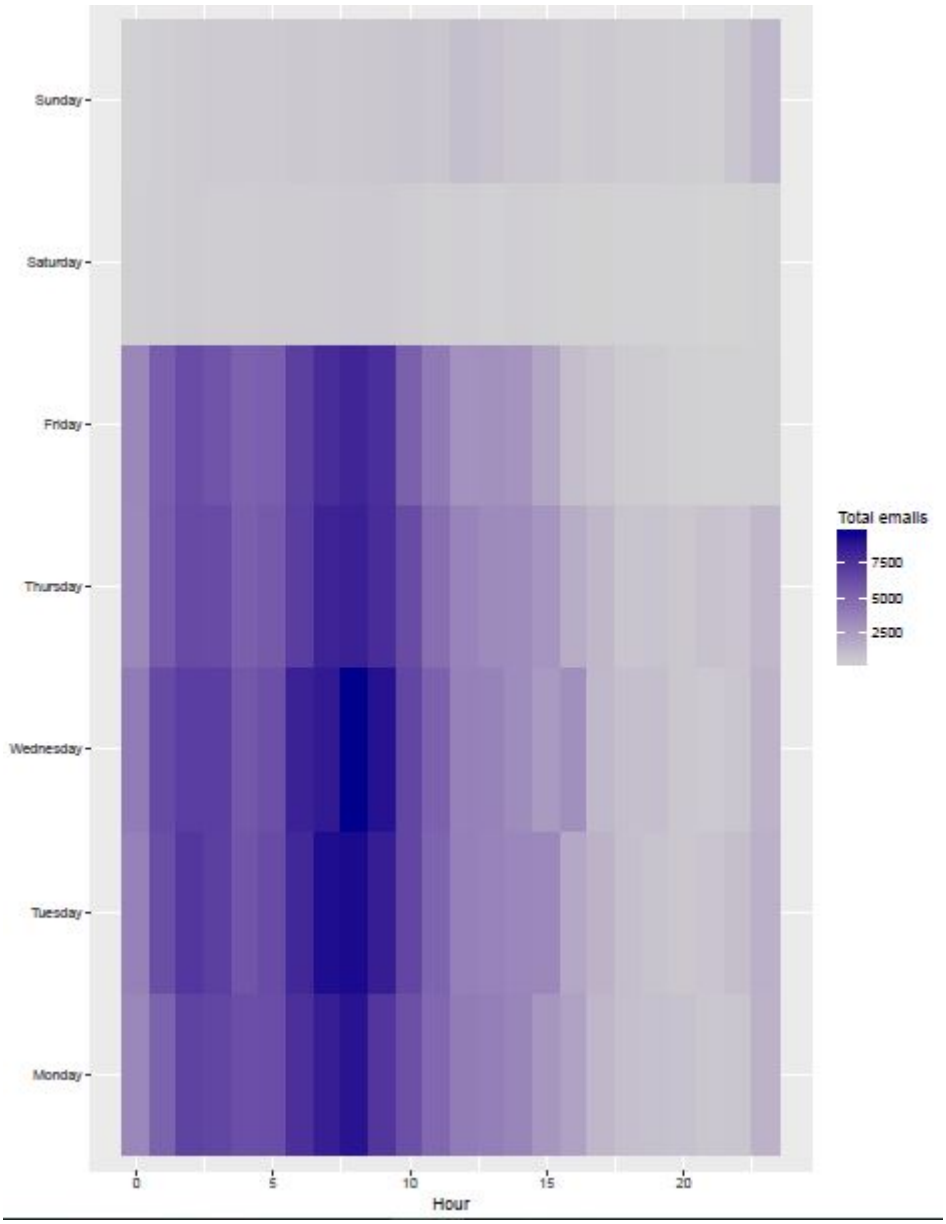
Figure 2.3: Emails Sent Per Hour Per Day



2.2.3 Heat-Map

A heat-map of the above metrics yields a more eye-friendly visualization.

Figure 2.4: Heat Map



The working of the company in such a large scale seems alright, but it is this same working intensity that has masked the inner misgivings of Enron.

2.3 Top Users

We select the top 20 people who come under the following sections:

2.3.1 Sent Most Mails

Figure 2.5: Top 20 sent most mails.

```
> head(sort(table(enron$From), decreasing = TRUE), n=20)

           kay.mann@enron.com      vince.kaminski@enron.com      jeff.dasovich@enron.com
                        16735                         14368                         11411
         pete.davis@enron.com       chris.germany@enron.com     sara.shackleton@enron.com
                         9149                          8801                          8777
enron.announcements@enron.com          tana.jones@enron.com       steven.kean@enron.com
                         8587                          8490                          6759
         kate.symes@enron.com     matthew.lenhart@enron.com         eric.bass@enron.com
                         5438                          5265                          5158
         no.address@enron.com   debra.perlingiere@enron.com        sally.beck@enron.com
                         5112                          4387                          4343
        mark.taylor@enron.com        susan.scott@enron.com       gerald.nemec@enron.com
                         4111                          4000                          3888
        drew.fossum@enron.com        john.arnold@enron.com
                         3706                          3578
>
```

2.3.2 Received Most Mails

Figure 2.6: Top 20 received most mails.

```
> head(sort(table(enron$To), decreasing = TRUE), n=20)

                                pete.davis@enron.com      tana.jones@enron.com
                        9724                    9155                      5677
sara.shackleton@enron.com        vkaminski@aol.com   jeff.dasovich@enron.com
                        4974                    4870                      4350
     kate.symes@enron.com   all.worldwide@enron.com     mark.taylor@enron.com
                        3517                    3324                      3295
       kay.mann@enron.com    gerald.nemec@enron.com     steven.kean@enron.com
                        3085                    3074                      3030
  louise.kitchen@enron.com      sally.beck@enron.com  vince.kaminski@enron.com
                        2626                    2550                      2404
benjamin.rogers@enron.com    daren.farmer@enron.com richard.shapiro@enron.com
                        2249                    2064                      2027
    all.houston@enron.com   suzanne.adams@enron.com
                        1953                    1800
```

2.3.3 Sent and Received Most Mails

Figure 2.7: Top 20 most mails.

```
> head(sort(table(enron$user), decreasing = TRUE), 20)

  kaminski-v   dasovich-j       kean-s       mann-k      jones-t shackleton-s     taylor-m
       28465        28234        25351        23381        19950        18687        13875
    farmer-d    germany-c       beck-s      symes-k      nemec-g      scott-s     rogers-b
       13032        12436        11830        10827        10655         8022         8009
      bass-e    sanders-r   campbell-l    shapiro-r     guzman-m        lay-k
        7823         7329         6490         6071         6054         5937
```

Kenneth Lay was the founder of Enron, but did not step into the waters directly. Other guilty people are Jeff Dasovich at number 2, Nemec at 14 among others.

CHAPTER 3

EXPLORATORY ANALYSIS

In statistics, exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods.

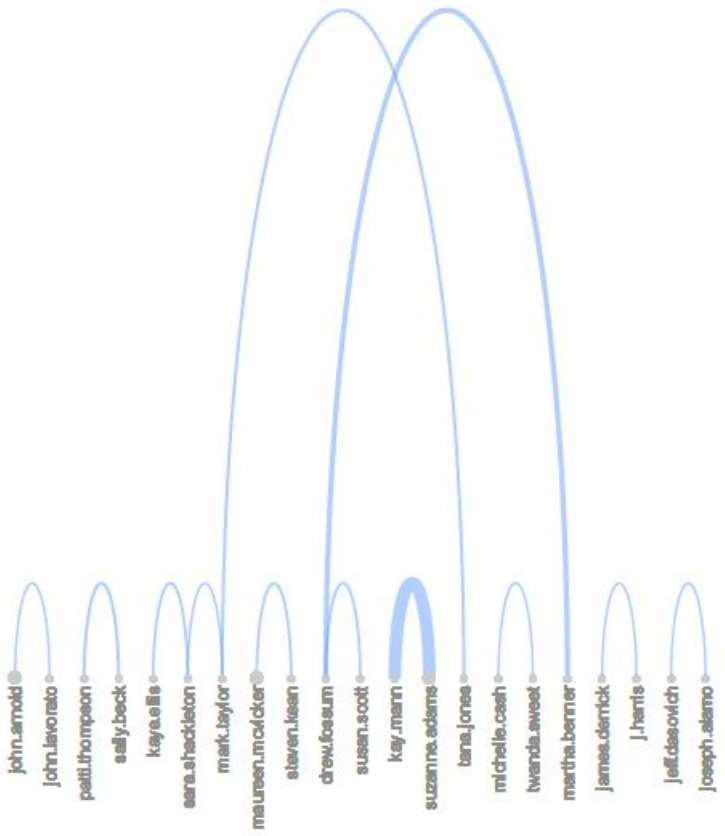
A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task The objectives of EDA are to: Suggest hypotheses about the causes of observed phenomena Assess assumptions on which statistical inference will be based Support the selection of appropriate statistical tools and techniques Provide a basis for further data collection through surveys or experiments As explained, the sent emails are only considered. Also emails from one person to exactly one person is analysed.

Although the data set contains many emails sent to several recipients, parsing the To: field of the emails was skipped for time constraints. Only emails sent directly to one recipient are kept. A complicated parser is required to exactly choose all emails we need. This will be explored later.

3.1 Most Mails Sent

The first analysis looks only at emails that are sent between people whose mailbox is in the data set. To do that, only emails are kept that are sent to someone that has also sent a mail him/herself.

Figure 3.1: Most Mails Sent



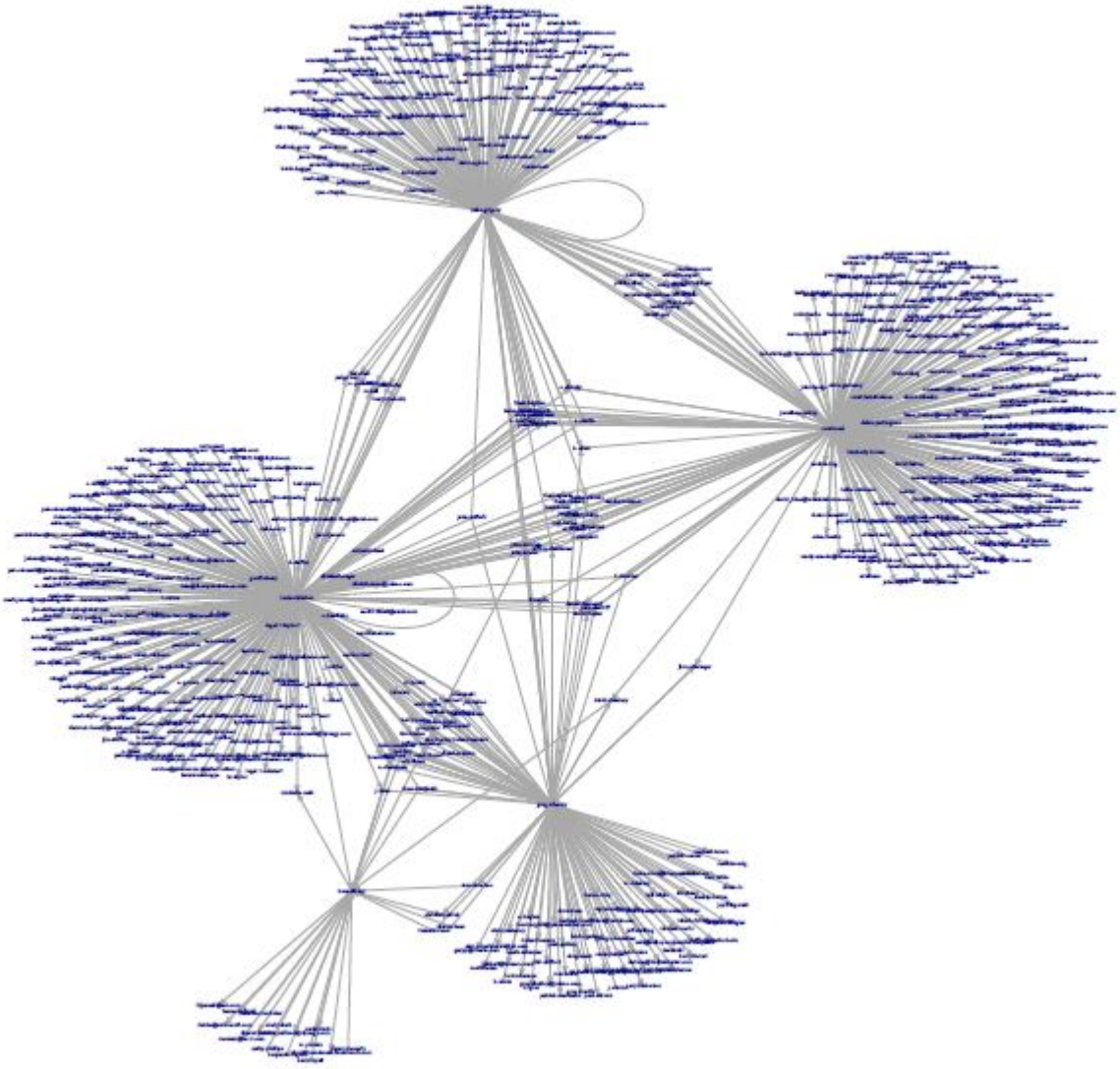
The graph shows connections between people where more than 150 emails have been sent. This number reduces the data set to only 21 people, a number that can still be plotted nicely.

The thickness of each arch in the graph shows the volume of email exchanged. The size of each vertex is proportional to the number of different people that person has sent emails to overall.

3.2 Analysis: Important People

A paper by Shetty and Adibi suggests these names: Louise Kitchen, Mike Grigsby, Greg Whalley, Scott Neal, Kenneth Lay.

Figure 3.2: Analysis: Important People



We first simply plot the whole network of people that have sent mails to or received mails from one of the members in the list of important people.

While there are a lot of people that only exchange emails with one of the important people there is a group of people (in the center of the graph) that is connected to nearly all of the important people. (E.g. John Lavorato, Jeffrey Shankman, etc)

In fact Jeff Shankman is in custody right now for hiding partnerships illegally.

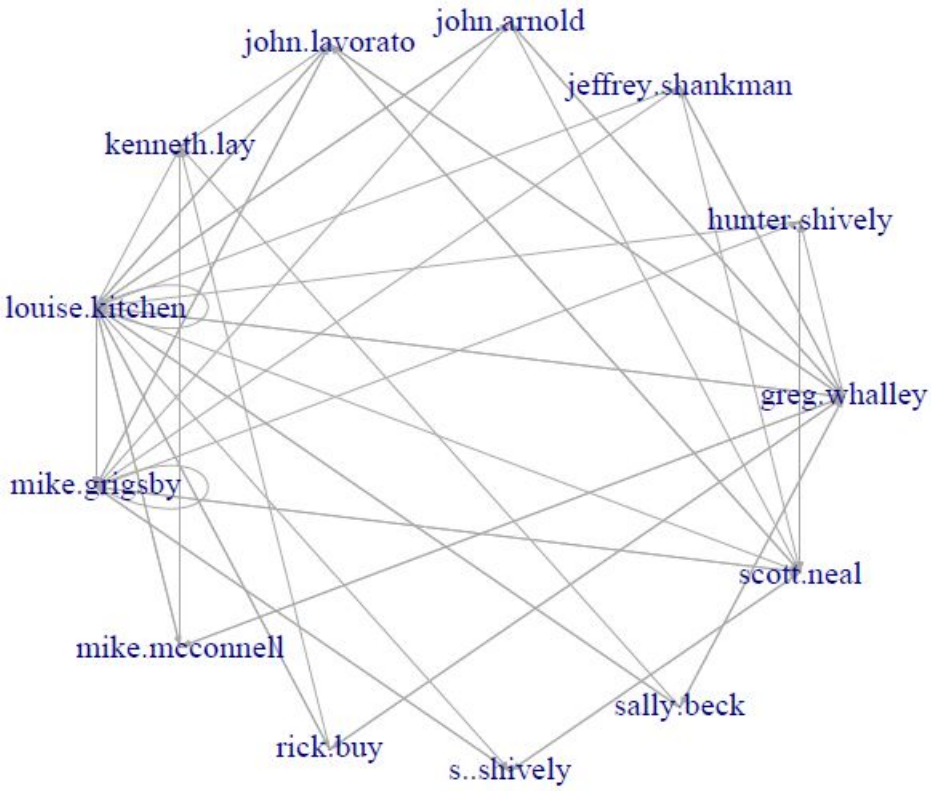
The government indicted Shankman in September 2013 on 24 counts including bankruptcy fraud and concealment of assets, alleging he schemed to hide assets and defraud creditors and the trustee who was appointed to collect and dispose of all his assets in his bankruptcy case. Shankman had been an executive in Enron's global markets division.

This is for the whole network and as result not a lot of sense can be made from it.

We can see from the graph that, while there are many people that exchange mails with only one of the "important" people selected above, there are some individuals who exchange mails with nearly all of them.

3.3 Analysis: Network of Well Connected

Figure 3.3: Analysis: Network of Well Connected



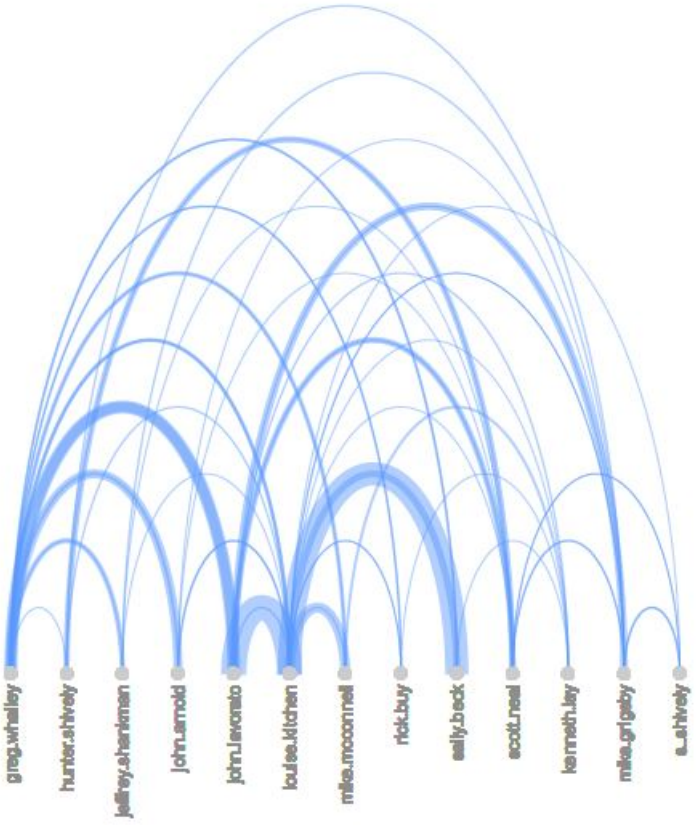
The graph shows only those people in the network that have more than 4 connections.

Within the selected data set, this will be people that are connected to most of the 'important people' in the network. One can see that nearly everybody is connected to Louise Kitchen.

Louise Kitchen was a young British trader spearheading Enron's entry into Europe's energy markets. She wasn't a top executive and hadn't even turned 30. But Kitchen cooked up a plan for the company's online trading operation.

3.4 Network of Well Connected

Figure 3.4: Network of Well Connected



The last graph shows the network of 'well connected' people with thickness of lines indicating the number of emails sent from one person to the other.

The thickness of the vertex shows the number of mails that the person sends in general.

The connection between Louise Kitchen and John Lavarato and Sally Beck is exceptionally strong. Furthermore, John Lavarato has quite strong connections to all of the 'important people' mentioned above.

CHAPTER 4

PREDICTIVE CODING

Predictive coding software uses a mathematical model and artificial intelligence programming to scan electronic documents and locate data that is relevant to a legal case. The software, which is capable of learning from its mistakes, first reviews a sample cluster of documents that have been tagged and categorized manually by a human legal team.

The predictive coding program is then given a new set of documents and asked to identify which documents are relevant and should be reviewed by humans. The legal team then reviews the software's decisions to determine whether an acceptable level of confidence has been achieved.

Predictive coding is a new technique in which attorneys manually label some documents and then use text analytics models trained on the manually labeled documents to predict which of the remaining documents are responsive.

The data set contains just two fields: email: the text of the email in question, responsive: a binary (0/1) variable telling whether the email relates to energy schedules or bids.

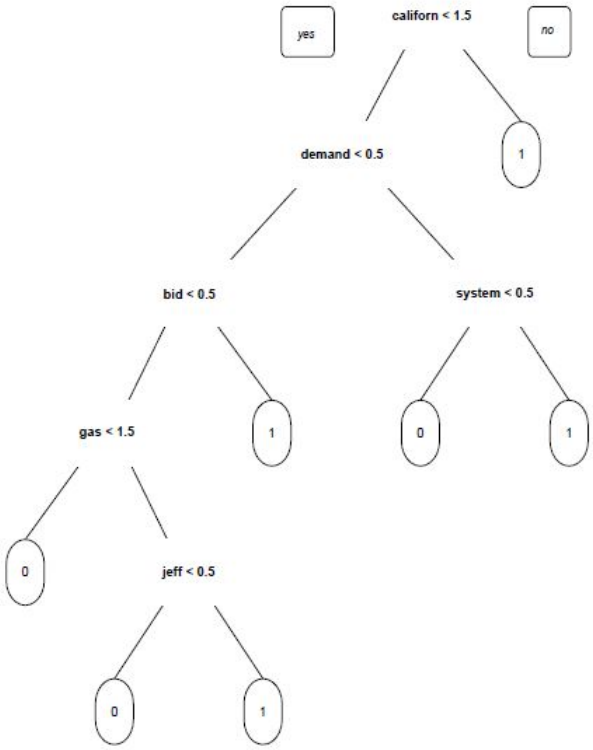
The labels for these emails were made by attorneys as part of the 2010 text retrieval conference legal track, a predictive coding competition.

4.1 Steps Done to Obtain CART Model

1. LOADING THE DATA 2. CREATING A CORPUS - Converting text to lower case, Removing punctuation, Removing stop words, Stemming, 3. BAG OF WORDS - Create a Document Term Matrix, Remove sparse terms, Creating a Data Frame from the DTM, Split data in training/testing sets with splitRatio = 0.7

EMAIL CART MODEL

Figure 4.1: CART MODEL



If californ appears at least twice in an email, we are going to take the right path and predict that a document is responsive. It is somewhat unsurprising that California shows up, because we know that Enron had a heavy involvement in the California energy markets. Further down the tree, we see a number of other terms that we could plausibly expect to be related to energy bids and energy scheduling, like system, demand, bid, and gas. Down at the bottom is jeff, which is perhaps a reference to Enron's CEO, Jeff Skillings, who ended up actually being jailed for his involvement in the fraud at the company.

4.2 Out-of-Sample Performance of the Model

Now that we have trained a model, we need to evaluate it on the test set. We build an object predictCART that has the predicted probabilities for each class from our CART model, by using the predict() function on the model emailCART and the test data with newdata = test. Overall accuracy of this CART model is 0.856 Sensitivity = TP rate = 25 / 42 = 0.595 Specificity = (1 - FP rate) = 195 / 215 = 0.907

Figure 4.2: Accuracy Calculation

```
cmat_CART <- table(test$responsive, predictCART.prob >= 0.5)
cmat_CART
##
##     FALSE TRUE
##   0   195   20
##   1    17   25

accu_CART <- (cmat_CART[1,1] + cmat_CART[2,2])/sum(cmat_CART)
```

4.3 Comparison with baseline model

The accuracy of the baseline model is then 0.8366. We see just a small improvement in accuracy using the CART model, which is a common case in unbalanced data sets. However, as in most document retrieval applications, there are uneven costs for different types of errors here. Typically, a human will still have to manually review all of the predicted responsive documents to make sure they are actually responsive.

Figure 4.3: Baseline Model Performance

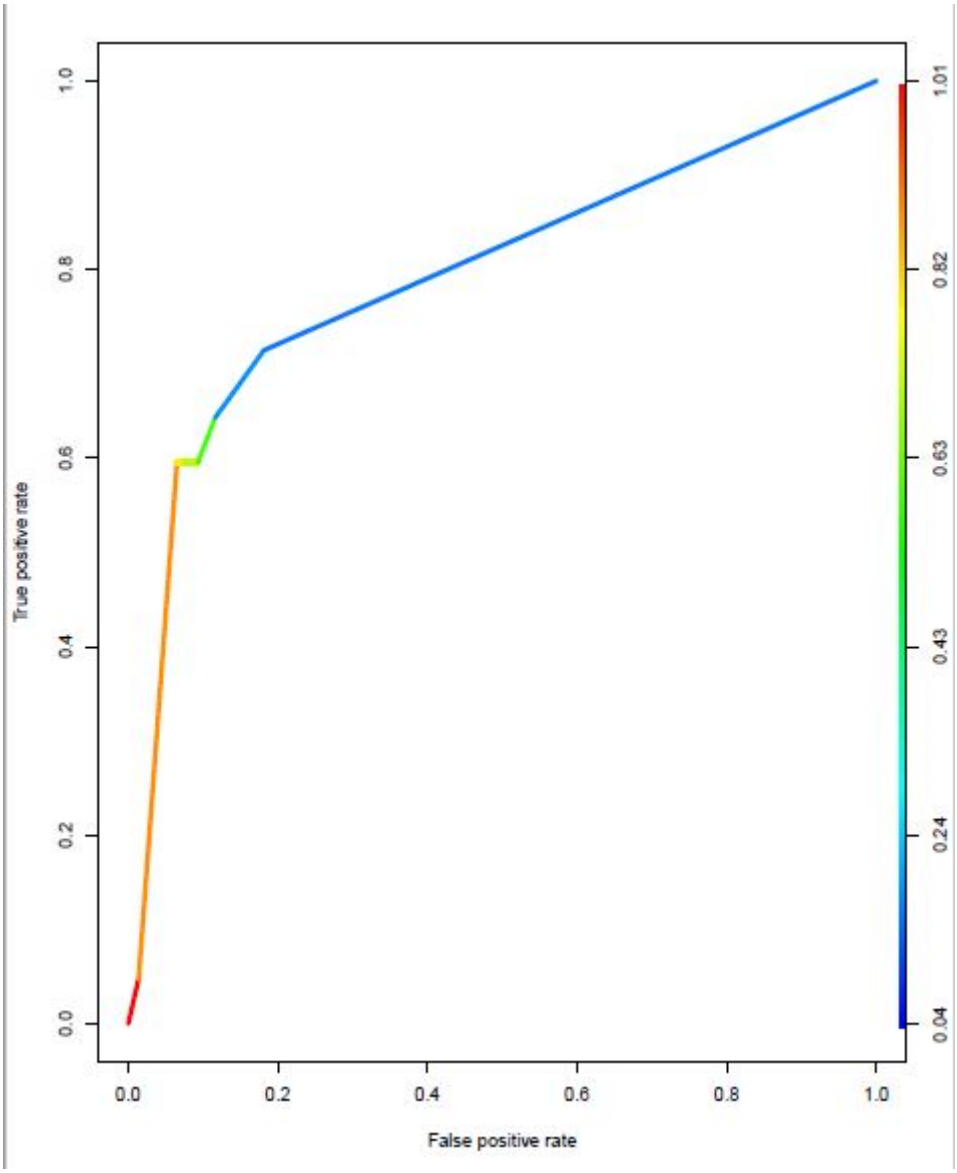
```
cmat_baseline <- table(test$responsive)
cmat_baseline
##
##   0   1
## 215  42

accu_baseline <- max(cmat_baseline)/sum(cmat_baseline)
```

If we have a false positive, i.e. a non-responsive document labeled as responsive, the mistake translates to a bit of additional work in the manual review process but no further harm, since the manual review process will remove this erroneous result. On

the other hand, if we have a false negative, i.e. a responsive document labeled as non-responsive by our model, we will miss the document entirely in our predictive coding process. Therefore, we are going to assign a higher cost to false negatives than to false positives, which makes this a good time to look at other cut-offs on our ROC curve.

4.4 ROC Curve

Figure 4.4: ROC Curve



A choice that might look promising could be in the part of the curve where it becomes flatter (going towards the right), where we have a true positive rate of around 70 percent (meaning that we're getting about 70 percent of all the responsive documents),

and a false positive rate of about 20 percent (meaning that we are making mistakes and accidentally identifying as responsive 20 percent of the non-responsive documents.)

4.5 Area Under Curve (AUC)

The AUC of the CART models is 0.7936, which means that our model can differentiate between a randomly selected responsive and non-responsive document about 79.4 percent of the time.

CHAPTER 5

MACHINE LEARNING: PREDICITION

Machine learning algorithms are used to predict the existenceof a POI in the emails analysed. The training time and accuracy measures are observed and the result is compared.

5.1 Naive Bayes

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator.

5.2 Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are:

Effective in high dimensional spaces. Still effective in cases where number of dimensions is greater than the number of samples. Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include: If the number of features is much greater than the number of samples, the method is likely to give poor perfor-

mances. SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

5.3 Decision Trees

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Some advantages of decision trees are: Simple to understand and to interpret. Trees can be visualised. Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values. The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree. Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable.

The disadvantages of decision trees include: Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning , setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble. There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Figure 5.1: Training Time

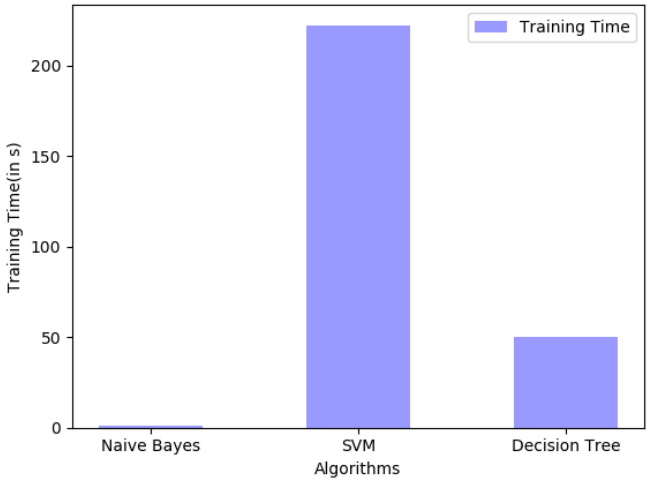
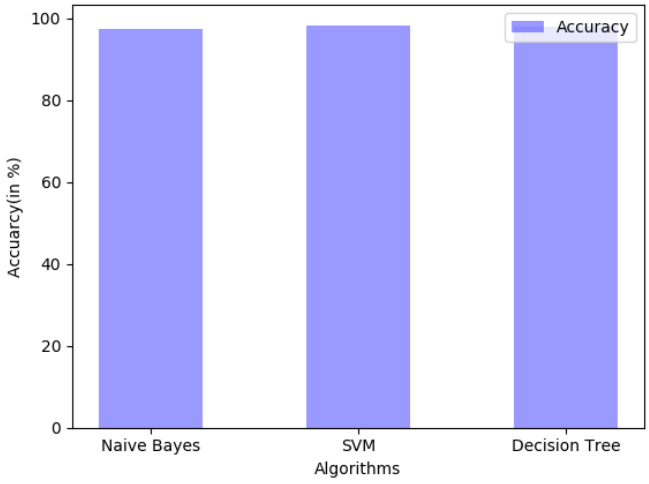


Figure 5.2: Accuracy



5.4 Random Forest

Ensembles of decision trees (such as Random Forests, which is a trademarked term for one particular implementation) are very fast to train, but quite slow to create predictions once trained. More accurate ensembles require more trees, which means using the model becomes slower. In most practical situations this approach is fast enough, but there can certainly be situations where run-time performance is important and therefore other approaches would be preferred.

Of course, it is important to recognize that it is a predictive modelling tool, not a descriptive tool - if you are looking for a description of the relationships in your data, you should look at other options.

Overall, for fast, simple, flexible predictive modelling, ensembles of decision trees are probably the most useful single pragmatic tool available today - but like any method they do have their limitations.

5.5 Adaboost

Adaptive Boosting uses a committee of weak base classifiers to vote on the class assignment of a sample point. The base classifiers can be decision stumps, decision trees, SVMs, etc.. It takes an iterative approach. On each iteration - if the committee is in agreement and correct about the class assignment for a particular sample, then it becomes down weighted (less important to get right on the next iteration), and if the committee is not in agreement, then it becomes up weighted (more important to classify right on the next iteration). Adaboost is known for having good generalization (not overfitting).

5.6 K Nearest Neighbour

We use KNN since, they are robust to noisy training data(especially if we use inverse square of weighted distance as the distance) and effective if the training data is large, which is so in this case.

The problems arise since, K needs to be determined and computation cost to find the distance is quite high.

Figure 5.3: Training Time

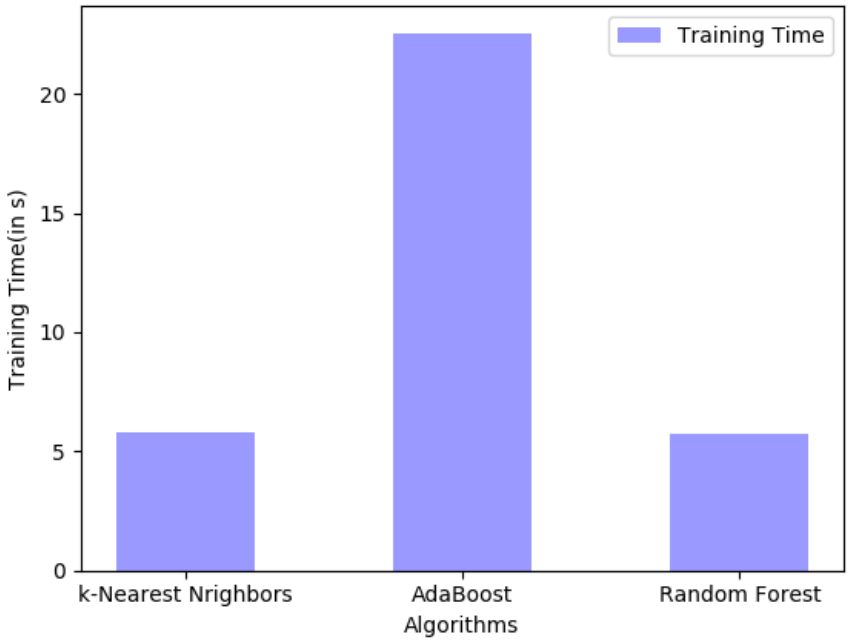
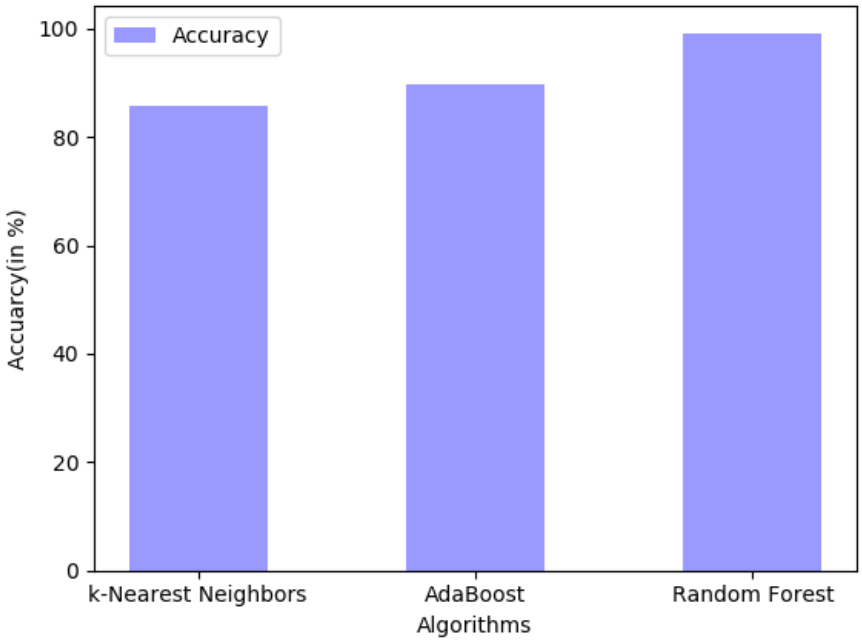


Figure 5.4: Accuracy



CHAPTER 6

CODING

6.1 Code in R

The following deals with the usage of R in cleaning, processing and visualizing data.

6.1.1 Enron.R

```
setwd("E:/SSJ/Sujit/IntrotoML/enron_mail_20150507/")
#Always run this script in the folder where maildir exists
#Enron Email Dataset: https://www.cs.cmu.edu/~./enron/

#Load specialised libraries
library(stringr) #String manipulation
library(igraph, warn.conflicts=F) #Network analysis

#E-mail corpus consists of nested folders per user
#with e-mails as text files

#Create list of all available e-mails
emails <- list.files("maildir/", full.names=T, recursive=T)
length(emails)

#Filter by inbox only
emails <- emails[grep("/inbox", emails)]
length(emails)

#Create list of sender and receiver (inbox owner)
```

```
inboxes <- data.frame(
from=apply(as.data.frame(emails), 1, function(x)
{readLines(x, warn=F)[3]}),
to=emails,
stringsAsFactors=F)

#Keep only enron.com and strip all but username
inboxes <- inboxes[grepl("@enron.com", inboxes$from),]
inboxes$from <- str_sub(inboxes$from, 7,
nchar(inboxes$from)-10)
to <- str_split(inboxes$to, "/")
inboxes$to <- sapply(to, "[", 3)

#Create username list
users <- data.frame(user=paste0("maildir/", unique(inboxes$to)))

#Remove those without sent mails
sent <- apply(users, 1, function(x){sum(grepl("sent",
dir(x)))})
users <- subset(users, !sent==0)

#Replace username with e-mail name
users$mailname <- NA
for (i in 1:nrow(users)){
sentmail <- dir(paste0(users$user[i], "/sent_items/"))
name <- readLines(paste0(users$user[i], "/sent_items/",
sentmail[1]), warn=F)[3]
name <- str_sub(name, 7, nchar(name)-10)
users$mailname[i] <- name
}
users$user <- str_sub(users$user, 9)
inboxes <- merge(inboxes, by.x="to", users, by.y="user")
```

```
inboxes <- data.frame(from=inboxes$from,
to=inboxes$mailname)

inboxes$from <- as.character(inboxes$from)
inboxes$to <- as.character(inboxes$to)

#Only e-mails between inbox users
inboxes <- inboxes[inboxes$from %in% inboxes$to,]

#Remove no.address
inboxes <- subset(inboxes,
from!="no.address" & to!="no.address")

#Remove mail to self
inboxes<- subset(inboxes, inboxes$from!=inboxes$to)

#Define network
g <- graph_from_edgelist(as.matrix(inboxes),
 directed=F)
coms <- spinglass.community(g)

#Plot network
par(mar=c(0,0,2,0))
plot(coms, g,
vertex.label=NA,
layout=layout.fruchterman.reingold,
vertex.size=3,
main="Enron e-mail network snapshot"
)

#Analyse network
degree(g)[order(degree(g), decreasing = T)]
```

6.1.2 SampleAnalysis.R

```
setwd("E:/SSJ/Sujit/IntrotoML/")


# This R script
#  * requires the library "plyr"
#  * requires 'arcdiagram' which needs to be downloaded
# from github.
#  Use these commands:
#    > library(devtools)
#    > install_github("arcdiagram",
# username = "gastonstat")
# You need to have the library "devtools"
# installed for this.
library(plyr)
library(arcdiagram)
require(igraph)

# Function to read a single email message's
# headers from a file.
# Adapted from Conway, White: Machine Learning
# for Hackers
get.message.headers  <- function(path){
tryCatch({
con <- file(path, open="rt", encoding="ascii")
text <- readLines(con)
# Headers of an email end with the first empty
```

```
# line in the file.
headers <- text[seq(1,which(text=="")[1]-1, 1)]
close(con)
return(headers)
},
error = function(e){

return(NA)
})
}

# This function extracts the "To:" and "From:"
#fields.
parse.message <- function(headers){
if(is.na(headers)){
return(c(NA, NA))
}

from <- grep("^from:",headers,ignore.case=TRUE,
value=TRUE)[1]
# Just keep the email address without the From: field
from <- sub("^From:\\s*", "", from, ignore.case=TRUE)

to <- grep("^to:",headers,ignore.case=TRUE,value=TRUE)[1]
# Just keep the email address without the To: field
to <- sub("^to:\\s*", "", to, ignore.case=TRUE)

# If the email was sent to several receipients,
# they are separeted by
# comma and appear on several consecutive lines.
# This would require a proper parser.
# Given the time constraints
```

```
# while developing this script,
# we throw away emails that have more than one
# receipient at this point.
if(!is.na(to)){
if(regexpr(",",to,fixed=TRUE)[1] != -1){
to <- NA
}
}
return(c(from, to))
}

# Get all 'Sent' folders from the mailboxes
# contained in the dataset
mail_folders <- file("sent_folders", open="rt",
encoding="ascii")
mail_files_paths <- readLines(mail_folders, warn = TRUE)
close(mail_folders)

# Construct paths for all files
mail_files <- unlist(lapply(mail_files_paths,
function(path){
paste(path, dir(path), sep="/")
}))
# Read and parse all mails (this step takes time)
mails <- sapply(mail_files, function(filename){
parse.message(get.message.headers(filename))
})

# Turn mail data into an R data frame, removing cases that
# could not be parsed above.
mails <- na.omit(data.frame(mails[1,], mails[2,]))
colnames(mails) <- c("From", "To")
```

```
# Get all mail addresses from which mails originated.
mails.sender <- unique(mails$From)
# Keep only those mails that were sent to one of these people.
mails.connected <- subset(mails,
To %in% mails.sender)

mails.counted <- ddply(mails.connected, .(From, To),
summarise, weight = length(To))
# Remove @enron.com domain for better readability
mails.counted$From <- sub("@enron.com", "",
mails.counted$From)
mails.counted$To <- sub("@enron.com", "",
mails.counted$To)

# Produce a graph from these connections.
g0 <- graph.data.frame(mails.counted,directed=TRUE)

g <- as.undirected(g0, mode="collapse")
# Remove all connections where fewer than 150
# emails have been sent.
gg <- delete.edges(g, which(E(g)$weight < 150))
# Remove vertices that are now disjunct.
# This leaves us with 21 nodes,
# which is still nicely plottable.
gg <- delete.vertices(gg, which(degree(gg) < 1))

# Plot this as a graph to a pdf file.
# We take only those people who
# have sent more than 150mails
pdf("00.most.mails.pdf")
```

```
arcplot(get.edgelist(gg),height=5,width=5,unit="in",
lwd.arcs = E(gg)$weight/100, cex.labels = 0.75,
pch.nodes = 21,
lwd.nodes = 2, line = -0.7, cex.nodes = degree(gg)*0.5)
dev.off()



# Important people according to the paper cited above
important.people <- c("louise.kitchen@enron.com",
"mike.grigsby@enron.com",
"greg.whalley@enron.com",
"scott.neal@enron.com", "kenneth.lay@enron.com")
# Keep only mails that are either
# From or To the "important" people.
mails.important <- subset(mails,
From %in% important.people | To %in% important.people)
# Count how many mails were exchanged between each member.
mails.important.counted <- ddply(mails.important, .(From, To),
summarise,
weight = length(To))
# Remove enron domain for better readability
mails.important.counted$From <- sub("@enron.com", "",
mails.important.counted$From)
mails.important.counted$To <- sub("@enron.com", "",
mails.important.counted$To)

# Produce a graph from these connections
g <- graph.data.frame(mails.important.counted,directed=TRUE)
# This function produces a nice layout for plotting
l <- layout.fruchterman.reingold
(g,niter=550,area=vcount(g)^2.3,
```

```
   repulserad=vcount(g)^2.8)

# Produce a plot showing the network.
pdf("01.important.people.pdf", height=34, width=34)
#arcplot(as.matrix(mails.important.counted.10)[,1:2],
lwd.arcs = mails.important.counted.10$weight/10,
cex.labels = 0.5,
pch.nodes = 20, lwd.nodes = 1, line = -0.2)
plot.igraph(g, layout=l, vertex.size=0)
dev.off()



# Remove alls nodes which are connected to less than 5 other nodes.
gg2 <- delete.vertices(g, which(degree(g) <5))
# Print to a PDF
pdf("02.important.people.well.connected.pdf", height=20,
 width=20)
#arcplot(as.matrix(mails.important.counted.10)[,1:2],
lwd.arcs = mails.important.counted.10$weight/10,
cex.labels = 0.5,pch.nodes = 20, lwd.nodes = 1, line = -0.2)
plot.igraph(gg2, layout=layout.circle, vertex.size=0,
vertex.label.cex=4,margin=0.1)
dev.off()


pdf("03.important.people.well.connected.weights.pdf")
arcplot(get.edgelist(gg2), lwd.arcs = E(gg2)$weight/3,
cex.labels = 0.75,pch.nodes = 21, lwd.nodes = 2, line = -0.6)
dev.off()
```

6.1.3 TextAnalysis.R

```
# Load the necessary packages. Read from CRAN what
# each package does.
library("tm")
library("SnowballC")
library("caTools")
library("rpart")
library("rpart.plot")
library("ROCR")



# Loading the data and having a look at it.
emails <- read.csv("E:/SSJ/Sujit/Enron/energy_bids.csv.gz",
stringsAsFactors = FALSE)
str(emails)

#Lets look at a few examples to see what these
#particular emails consist of
strwrap(emails$email[1])

# What is its 'responsive' value?
emails$responsive[1]

# Email 2
strwrap(emails$email[2])

#Its responsive
```

```
emails$responsive[2]

# No of emails responsive to our query
table(emails$responsive)


corpus <- Corpus(VectorSource(emails$email))

corpus

# Converting the text to lower case

corpus <- tm_map(corpus, tolower)
corpus
# corpus <- tm_map(corpus, PlainTextDocument)
corpus


# Remove punctuation
corpus <- tm_map(corpus, removePunctuation)
corpus
# No stop words are removed. The default
# english stop words list
# provided in the 'tm' package.
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus
# Our data set is now stemmed.
corpus <- tm_map(corpus, stemDocument)
corpus
strwrap(corpus[[1]])
# Now extract the word frequencies to
# be used for prediction
```

```
# tm package has the function, DocumentTermMatrix()
# corpus <- Corpus(VectorSource(corpus))
DTM <- DocumentTermMatrix(corpus)

DTM

# Let's remove the terms that are infrequent
sparse_DTM <- removeSparseTerms(DTM, 0.97)
sparse_DTM
# Creating a dataframe from the DTM
labeledTerms <- as.data.frame(as.matrix(sparse_DTM))

# Use make.names for easy R use.
colnames(labeledTerms) <-
make.names(colnames(labeledTerms))

# Adding the responsive part
labeledTerms$responsive <- emails$responsive

str(labeledTerms)

# Splitting data to create training/testing sets
# Generate pseudo-random numbers with 144 as the starting point
set.seed(144)

# 70% of responsive variables are set to TRUE
# when dividing into training set
split <- sample.split(labeledTerms$responsive, SplitRatio = 0.7)

train <- subset(labeledTerms, split == TRUE)
test <- subset(labeledTerms, split == FALSE)
```

```
# Now lets build a CART Model. CART -
# Classification and regression Trees is a
# tool for predictive model.

emailCART <- rpart(responsive ~ . , data = train, method = "class")

# Plots the model
prp(emailCART)



#  Now that we have trained it. It is time to
# evaluate it on the test set.

predictCART <- predict(emailCART, newdata = test)
predictCART[1:10,]

# We need the predicted probability of the document
# being responsive and
# it would be convenient to handle it separately
predictCART.prob <- predictCART[ , 2]

# Here we create the confusion matrix from the test set.
cmat_CART<- table(test$responsive, predictCART.prob >= 0.5)
cmat_CART

# Then we calculate the accuracy (out of sample)
accu_CART <- (cmat_CART[1,1] + cmat_CART[2,2])/sum(cmat_CART)

# Comparing with the baseline model.
# The baseline model always predicts
# non-responsive(most common value of the dependant variable).
# In this case the average of the dependent variable.
```

```
cmat_baseline <- table(test$responsive)
cmat_baseline

accu_baseline <- max(cmat_baseline)/sum(cmat_baseline)
accu_baseline



# Make false-negative costs higher.

# ROC Curve - to understand the performance of the model
# at different cutoffs.

predROCR <- prediction(predictCART.prob, test$responsive)
predROCR <- performance(predROCR, "tpr", "fpr")

# Plot it

plot(predROCR, colorize = TRUE, lwd = 4)
predROCR <- prediction(predictCART.prob, test$responsive)
# AUC
auc_CART <-
as.numeric(performance(predROCR, "auc")@y.values)
auc_CART
```

6.1.4 ExploreEnron.R

```
### Libraries

setwd("E:/SSJ/Sujit/IntrotoML/")
#library(formattable) # output is easier to read
# an well formatted
library(stringr) # String manipulation, Regex
library(plyr)
library(ggplot2)
library(tm)
library(SnowballC)
library(RColorBrewer)
library(wordcloud)

### read emails.csv

enron <- read.csv("E:/SSJ/Sujit/IntrotoML/emails.csv",
stringsAsFactors = FALSE)

###  locate the blank line "\n\n"

breaks <- str_locate(enron$message, "\n\n")


### Extract headers and bodies


headers <- str_sub(enron$message, end = breaks[,1] - 1)
```

```
bodies <- str_sub(enron$message, start = breaks[,2] + 1)


### Splitting the email header

parseHeader <- function(header){
MessageID <-
str_sub(str_extract(header, "^Message-ID:.*"), start = 12)
Date <- str_sub(str_extract(header,"Date:.*"), start = 7)
From <- str_sub(str_extract(header,"From:.*"), start = 7)
To <- str_sub(str_extract(header,"To:.*"), start = 5)
Subject <- str_sub(str_extract(header,"Subject:.*"),
start = 10)
#X-cc <- str_sub(str_extract(header,"X\\-cc:.*"),
#start = 7)
#X-bcc <- str_sub(str_extract(header,"X\\-bcc:.*"),
#start = 8)

headerParsed <- data.frame(MessageID, Date, From, To, Subject,
stringsAsFactors = FALSE)
return(headerParsed)
}

headerParsed <- parseHeader(headers)

### Conversion of dates


## UTC time
datesTest <- strptime(headerParsed$Date,
format = "%a, %d %b %Y %H:%M:%S %z")
## localtime
```

```
datesLocal <- strptime(headerParsed$Date,
format = "%a, %d %b %Y %H:%M:%S")



### Copy dates


headerParsed$Date <- datesTest
headerParsed$DateLocal <- datesLocal
# remove dates Test
rm(datesTest)
rm(datesLocal)

### File column


## split
fileSplit <- str_split(enron$file, "/")
fileSplit <-rbind.fill(lapply(fileSplit,
function(X) data.frame(t(X))))

### Creating one dataset


enron <- data.frame(fileSplit, headerParsed,
bodies, stringsAsFactors = FALSE)
colnames(enron)[1] <- "User"

### Cleaning up

rm(headerParsed)
```

```
rm(bodies)
rm(headers)
rm(breaks)
rm(fileSplit)

# garbage collection
gc()


## Some Top 20s
### Mail writers
head(sort(table(enron$From), decreasing = TRUE), n=20)

### Mail recipients
head(sort(table(enron$To), decreasing = TRUE), n=20)

### User
head(sort(table(enron$User), decreasing = TRUE), 20)


## Weekdays and Hour of day
# extract weekday
enron$Weekday <- weekdays(enron$DateLocal)
# extract Hour of day
enron$Hour <- enron$DateLocal$hour



## Weekdays Analysis
```

```
WeekdayCounts <- as.data.frame(table(enron$Weekday))
str(WeekdayCounts)
WeekdayCounts$Var1 <- factor(WeekdayCounts$Var1, ordered=TRUE,
levels=c( "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday","Saturday", "Sunday"))
DayHourCounts <- as.data.frame(table(enron$Weekday,
enron$Hour))
str(DayHourCounts)
DayHourCounts$Hour <-
as.numeric(as.character(DayHourCounts$Var2))
DayHourCounts$Var1 <- factor(WeekdayCounts$Var1,
ordered=TRUE,
levels=c( "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday","Saturday", "Sunday"))


### Plot number of emails per Weekday


ggplot(WeekdayCounts, aes(x=Var1, y=Freq)) +
geom_line(aes(group=1))



### Plot number of emails per Hour per Day

ggplot(DayHourCounts, aes(x=Hour, y=Freq)) +
geom_line(aes(group=Var1, color=Var1), size=1)


### Heatmap: emails per Hour per Day
```

```
ggplot(DayHourCounts, aes(x = Hour, y = Var1)) +
geom_tile(aes(fill = Freq)) +
scale_fill_gradient(name="Total emails",
low = "lightgrey", high = "darkblue") +
theme(axis.title.y = element_blank())
```

6.2 Code in Python

We apply machine learning algorithms through the following python code.

6.2.1 dtauthorid

```
#!/usr/bin/python

"""
This is the code to accompany the Lesson 3
(decision tree) mini-project.
Use a Decision Tree to identify emails from
the Enron corpus by author:
Sara has label 0
Chris has label 1
"""

import sys
from time import time
sys.path.append("../tools/")
from email_preprocess import preprocess



### features_train and features_test are the features
### for the training
### and testing datasets, respectively
### labels_train and labels_test are the corresponding
### item labels
features_train, features_test, labels_train,
labels_test = preprocess()
```

```
print len(features_train[0])

from sklearn import tree
from sklearn.metrics import accuracy_score

clf = tree.DecisionTreeClassifier(min_samples_split=40)

clf.fit(features_train, labels_train)
pred = clf.predict(features_test)

acc = accuracy_score(labels_test, pred)
print "Accuracy: ", acc
```

6.2.2 knnauthorid

```
#!/usr/bin/python

"""
Use a k- Nearest Neighbors Classifier to identify
emails by their authors
authors and labels:
Sara has label 0
Chris has label 1
"""

import sys
from time import time
sys.path.append("../tools/")
from email_preprocess import preprocess

features_train, features_test, labels_train,
```

```
labels_test = preprocess()



## K- Nearest Neighbors Algorithm
print "\nK- Nearest Neighbors Algorithm\n"

# import KNeighborsClassifier from sklearn.neighbors
from sklearn.neighbors import KNeighborsClassifier
# also import time
from time import time

# create a classifier
clf_kNN = KNeighborsClassifier()
# note the start of training time
t0 = time()
# train the classifier
clf_kNN.fit(features_train, labels_train)
# print the training time
print "training time:", round(time() - t0, 3), "s"

# note the start of testing time
t0 = time()
# predict the labels of features_test
predicted = clf_kNN.predict(features_test)
# print the testing time
print "testing time:", round(time() - t0, 3), "s"

# import accuracy_score from sklearn.metrics to
calculate accuracy
from sklearn.metrics import accuracy_score
# calculate accuracy
```

```
acc = accuracy_score(labels_test, predicted)
# print the accuracy
print "accuracy:", acc
print "\n"
```

6.2.3 rfauthorid

```
#!/usr/bin/python

"""
Use a Random Forest Classifier to identify emails
by their authors
authors and labels:
Sara has label 0
Chris has label 1
"""

import sys
from time import time
sys.path.append("../tools/")
from email_preprocess import preprocess


### features_train and features_test are the features for
### and testing datasets, respectively
### labels_train and labels_test are the corresponding item labels
features_train, features_test, labels_train,
labels_test = preprocess()
```

```
## Random Forest Algorithm
print "\nRandom Forest Algorithm\n"

# import RandomForestClassifier from sklearn.ensemble
from sklearn.ensemble import RandomForestClassifier
# also import time
from time import time

# create a classifier
clf_RF = RandomForestClassifier(n_estimators=15,
min_samples_split=50)
# note the start of training time
t0 = time()
# train the classifier
clf_RF.fit(features_train, labels_train)
# print the training time
print "training time:", round(time() - t0, 3), "s"

# note the start of testing time
t0 = time()
# predict the labels of features_test
predicted = clf_RF.predict(features_test)
# print the testing time
print "testing time:", round(time() - t0, 3), "s"

# import accuracy_score from sklearn.metrics to
calculate accuracy
from sklearn.metrics import accuracy_score
# calculate accuracy
acc = accuracy_score(labels_test, predicted)
# print the accuracy
```

```
print "accuracy:", acc
print "\n"
```

6.2.4 svmauthorid

```
#!/usr/bin/python

"""
Sara has label 0
Chris has label 1
"""
activate python27
import sys
from time import time
sys.path.append("../tools/")
from email_preprocess import preprocess


### features_train and features_test are the features for
### the training
### and testing datasets, respectively
### labels_train and labels_test are the corresponding item labels
features_train, features_test, labels_train,
labels_test = preprocess()



from sklearn.svm import SVC
clf = SVC(kernel="rbf", C = 10000)

#features_train = features_train[:len(features_train)/100]
#labels_train = labels_train[:len(labels_train)/100]
```

```
t0 = time()
clf.fit(features_train, labels_train)

print "Training time:", round(time()-t0, 3), "s"

t0 = time()
pred = clf.predict(features_test)

print "Prediction time:", round(time()-t0, 3), "s"

from sklearn.metrics import accuracy_score
print "Accuracy:", round(accuracy_score(pred, labels_test),
3)

#print "predictions: 10=", pred[10], " 26=", pred[26],
" 50=", pred[50]

print sum(pred)
```

6.2.5 adaboostauthorid

```
#!/usr/bin/python

import sys
from time import time
sys.path.append("../tools/")
from email_preprocess import preprocess


### features_train and features_test are the features for
```

```
### the training
### and testing datasets, respectively
### labels_train and labels_test are the corresponding
### item labels
features_train, features_test, labels_train,
labels_test = preprocess()



## AdaBoost Algorithm
print "\nAdaBoost Algorithm\n"

# import AdaBoostClassifier from sklearn.ensemble
from sklearn.ensemble import AdaBoostClassifier
# also import time
from time import time

# create a classifier
clf_AdaBoost = AdaBoostClassifier(n_estimators=20)
# note the start of training time
t0 = time()
# train the classifier
clf_AdaBoost.fit(features_train, labels_train)
# print the training time
print "training time:", round(time() - t0, 3), "s"

# note the start of testing time
t0 = time()
# predict the labels of features_test
predicted = clf_AdaBoost.predict(features_test)
# print the testing time
print "testing time:", round(time() - t0, 3), "s"
```

```
# import accuracy_score from sklearn.metrics to calculate
accuracy from sklearn.metrics import accuracy_score
# calculate accuracy
acc = accuracy_score(labels_test, predicted)
# print the accuracy
print "accuracy:", acc
print "\n"
```

6.2.6 nbauthorid

```
#!/usr/bin/python

"""
authors and labels:
Sara has label 0
Chris has label 1
"""

import sys
from time import time
sys.path.append("../tools/")
from email_preprocess import preprocess

### features_train and features_test are the
### features for the training
### and testing datasets, respectively
### labels_train and labels_test are labels
features_train, features_test, labels_train,
labels_test = preprocess()
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

clf = GaussianNB()
t0 = time()
clf.fit(features_train, labels_train)
print "training time: ", round(time()-t0, 3), "s"

t1 = time()
pred = clf.predict(features_test)
print "predicting time: ", round(time()-t1, 3), "s"

accuracy = accuracy_score(labels_test, pred)

print accuracy
```

CHAPTER 7

CONCLUSION

The application of initial and exploratory data analysis allows us to understand and visualize the data to get a brief idea about where the problem lies. Predictive coding is used to get a measure of the types of words used and encorporates text mining concepts. Through our CART model we were able to identify Jeff Skilling's involvement in some shady energy bids.

We applied 6 machine learning algorithms to predict the author of the e-mails, namely, naive bayes, support vector machine, decision trees, adaptive boosting, random forest, and k-nearest neighbors. Among these learning algorithms Naive Bayes is the fastest in training the dataset with a good amount of accuracy. If we have to calculate, the author in real time, this would be most applicable.

Adaboost and K-Nearest Neighbours had an accuracy of about 85 percent but they take more time to predict our designated author. If we are not in a hurry to find our authors, it would be best to apply SVM or Random Forest as they both have best accuracy.

CHAPTER 8

FUTURE ENHANCEMENT

Cross-validation can be done to get a fair idea of accuracy, performance, training time and testing time among other metrics. Some more paramaters can be tweaked so as to increase the performance of the classifiers.

Instead of just sent mails, all mails must be parsed and included to ensure certain important data is not missed.

Some people who were involved in scandal but not part of the company require application of more testing methods since they may not be part of the email dataset. Therefore we need other techniques to prove their involvement.

APPENDIX A

PREDICTION ALGORITHMS

A.1 Naive Bayes Rule

Naive Bayes methods are a set of supervised learning algorithms based on applying BayesâĂŹ theorem with the âĂIJnaiveâĂİ assumption of independence between every pair of features.

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

A.2 Support Vector Machine

Support Vector Machine (SVM) is primarily a classier method that performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables. For categorical variables a dummy variable is created with case values as either 0 or 1.

A.3 Adaboost

AdaBoost, short for "Adaptive Boosting", is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire who won the GÃűdel Prize in 2003 for their work. It can be used in conjunction with many other types of learning algorithms to improve their performance.

A.4 K Nearest Neighbours

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

Matrix multiplication is a binary operation that takes a pair of matrices, and produces another matrix. Numbers such as the real or complex numbers can be multiplied according to elementary arithmetic. On the other hand, matrices are arrays of numbers, so there is no unique way to define multiplication of matrices. As such, in general the term “matrix multiplication” refers to a number of different ways to multiply matrices.

REFERENCES

1. Healy, P. M. and Palepu, K. G. (2003). "The fall of enron." *Journal of Economic Perspectives*, Volume 17, Number 2.

2. Li, Y. (2010). "The case analysis of the scandal of enron." *International Journal of Business and Management*, Vol. 5, No. 10.

3. Matloff, N. (2011). *The Art of R Programming*. William Pollock.

4. Muller, A. and Guido, S. (2016). *Introduction to Machine Learning with Python*. O'Reilly.

5. Shetty, J. and Adibi, J. (2004). "The enron email dataset database schema and brief statistical report." *Information Sciences Institute Technical Report, University of Southern California*.