Identification of spam, ham and phish

Project on Machine learning

A project report submitted to the
Intro to DBMS class in partial fulfilment of
the class overall requirements

By
Namrata Sharma
Nithya Ramachandran

Acknowledgement

We must find time to stop and thank the people who make a difference in our lives..
- **John F. Kennedy**

There are many people that worked really hard to make this project something of real value. Even though we worked arduously, however, it is certain that we are in the debt of the following individuals:

The three biggest influencers on this project were surely Prof J.J. Hsieh, Jose Nazario, and Brandon Azad. Prof J.J. Hsieh is easily one the most influential person who helped us come up with the outline of this project. Jose maintained the data corpus of the phishing emails which are really hard to find online. He has helped us by giving access to the data corpus. He was always willing to talk. Brandon helped me with selection of the tools to convert the mbox files.

Table of Contents

Abstract

Spam is a real problem in today's world. Every day we receive a surge of spam emails and many times we reply to the spam emails with our personal credentials. Spam email is a form of commercial advertising which is economically viable because email is a very cost-effective medium for the sender. If just a fraction of the recipients of a spam message purchase the advertised product, the spammers are making money and the spam problem is perpetuated.

Each month, more attacks are launched with the aim of making web users believe that they are communicating with a trusted entity for the purpose of stealing account information, logon credentials, and identity information in general. This attack method, commonly known as "phishing," is most commonly initiated by sending out emails with links to spoofed websites that harvest information. We present a method for detecting these attacks, which in its most general form is an application of machine learning on a feature set designed to highlight user-targeted deception in electronic communication. This method is applicable, with slight modification, to detection of phishing websites, or the emails used to direct victims to these sites.

We evaluate this method on a set of approximately 7000 such phishing emails, and 4450 non-phishing emails, and correctly identify over 95% of the phishing emails while only mis-classifying on the order of 1.25% of the legitimate emails. We conclude with thoughts on the future for such techniques to specifically identify deception, specifically with respect to the evolutionary nature of the attacks and information available.

Introduction

Email spam—also known as junk email—is a type of electronic spam where unsolicited messages are sent by email. Many email spam messages are commercial in nature but may also contain disguised links that appear to be for familiar websites but in fact lead to phishing web sites or sites that are hosting malware. Spam email may also include malware as scripts or other executable file attachments. Email spam has steadily grown since the early 1990s. Phishing attacks impersonate legitimate transactional electronic communications from trustworthy entities in order to steal a user's personal information, often targeting usernames and passwords, credit card numbers, or social security numbers. Phishing emails are designed to resemble transactional email from the impersonated institution closely so that the user believes she is interacting with the legitimate institution. These attacks have become a serious problem with the rise of digital commerce, since both money and information access are put at risk. Flagging suspicious messages before the user opens them is one way to mitigate the threat posed by phishing email attacks. While the task of identifying phishing emails is similar to that of identifying spam emails, less research has been conducted into the former. The difficulty of classifying phishing emails is compounded since the emails are designed to look like legitimate transactional email.

What is a Spam Email?

Unwanted commercial email – also known as "spam" – can be annoying. Worse, it can include bogus offers that could cost you time and money. Spam email is a form of commercial advertising which is economically viable because email is a very cost-effective medium for the sender.

If just a fraction of the recipients of a spam message purchase the advertised product, the spammers are making money and the spam problem is perpetuated. Spammers harvest recipient addresses from publicly accessible sources, use programs to collect addresses on the web, and simply use dictionaries to make automated guesses at common usernames at a given domain. Spamming is politically debated in several countries, and has been legislated in some places with varying results.

Spammers often conceal or forge the origin of their messages to circumvent laws, service provider regulations, and anti-spammer lists used by anti-spam software. At the present more than 95% of email messages sent worldwide is believed to be spam, making spam fighting tools increasingly important to all users of email.

What is a Ham Email?

Ham email is a term sometimes used as opposed to spam messages. Ham is then all "good" legitimate email messages, that is to say, all messages solicited by the recipient through an opt-in process.

"**Ham**" is e-**mail** that is not Spam. In other words, "non-spam", or good **mail**". It should be considered a shorter, snappier synonym for "non-spam". Its usage is particularly common among anti-spam software developers, and not widely known elsewhere; in general it is probably better to use the term "non-spam", instead.

What is a Phishing email?

Phishing scams are typically fraudulent email messages appearing to come from legitimate enterprises (e.g., your university, your Internet service provider, your bank). These messages usually direct you to a spoofed website or otherwise get you to divulge private information (e.g., passphrase, credit card, or other account updates). The perpetrators then use this private information to commit identity theft.

One type of phishing attempt is an email message stating that you are receiving it due to fraudulent activity on your account, and asking you to "click here" to verify your information. See an example below.

Phishing scams are crude social engineering tools designed to induce panic in the reader. These scams attempt to trick recipients into responding or clicking immediately, by claiming they will lose something (e.g., email, bank account). Such a claim is always indicative of a phishing scam, as responsible companies and organizations will never take these types of actions via email.

What is Machine Learning?

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data.

The process of machine learning is similar to that of data mining. Both systems search through data to look for patterns. However, instead of extracting data

for human comprehension -- as is the case in data mining applications -- machine learning uses that data to detect patterns in data and adjust program actions accordingly.

Supervised and Unsupervised algorithms: Machine learning algorithms are often categorized as being supervised or unsupervised. Supervised algorithms can apply what has been learned in the past to new data. Unsupervised algorithms can draw inferences from datasets. In supervised learning, the output datasets are provided which are used to train the machine and get the desired outputs whereas in unsupervised learning no datasets are provided, instead the data is clustered into different classes.

Supervised learning draws conclusions with specific One variable, there is a dependent variable and a few independent variables. We are trying to understand the effect of all variables on the dependent variable. Where as in unsupervised learning we divide the data into the groups how similar the data is to each other no specific importance to specific terms.

With unsupervised learning it is possible to learn larger and more complex models than with supervised learning.

The Learning Problem

Users may have little prior knowledge of ham, spam and phishing emails and the ideas they bring to the topic may be misconceptions of what the current developments in the field provide. Furthermore, there has been little effort by practitioners in the field of identifying phishing emails. Lastly, these agents are designed for use with web content, yet there are no existing principles for how to identify the phishing emails.

Describing the learning problem.

This project aims to examine techniques that can be used to identify spam, ham and phishing emails with high accuracy. The emails are first examined under the bag of words model, a multinomial event model used for natural language processing. The emails are analyzed using a naive Bayesian classifier to establish a baseline classification accuracy. The bag of words model of the email is then extended to include features extracted from the emails in the dataset and reclassified with naive Bayes. Many of these features have already been examined by the literature on phishing mail classification. Next, maxent and support vector machines (SVMs) are examined as replacements for naive Bayes. Finally, I attempt to use a random forest classifier based on to capture the meaning inherent in the emails better than the bag of words model.

Datasets

We have used the spam and ham dataset available in apache.org website to test and train our algorithms. For phishing emails, collected over 7000 emails from Nazario public corpus. We used mbox viewer to view the mbox files of phishing emails. We converted the phishing files to csv files using mbox converter. Even though the ham and spam emails are tar.bz2 files (unix files), "R" supports them, we were able to use the dataset directly.

- Ham Dataset – 4450 emails collected from SpamAssassin (tar.bz2)
  http://spamassassin.apache.org/publiccorpus/
- Spam Dataset – 1900 emails collected from SpamAssassin (tar.bz2)
  http://spamassassin.apache.org/publiccorpus/
- Phishing Dataset – 7133 emails collected from Nazario Public Corpus (mbox)
  http://monkey.org/jose/wiki/doku.php

Limitations

This dataset is likely not representative of phishing and non-phishing emails received at a typical inbox since it was compiled from multiple sources at widely different times; however, there are very few publicly available phishing corpora, and (at the time of this writing) none that I could find with both legitimate transactional email and phishing email. It would have been better to have a single corpus containing phishing, legitimate transactional, and ham email, but since no such corpus could be located, this compilation is the best I could achieve.

Text Analysis

Text mining, also referred to as text data mining, roughly equivalent to text analytics, is the process of deriving high-quality information from text. High-quality information is typically derived through the devising of patterns and trends through means such as statistical pattern learning. Text mining usually involves the process of structuring the input text (usually parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database), deriving patterns within the structured data, and finally evaluation and interpretation of the output. 'High quality' in text mining usually refers to some combination of relevance, novelty, and interestingness. Typical text mining tasks include text categorization, text clustering, concept/entity extraction, production of granular taxonomies, sentiment analysis, document summarization, and entity relation modelling (i.e., learning relations between named entities).

Text analysis involves information retrieval, lexical analysis to study word frequency distributions, pattern recognition, tagging/annotation, information extraction, data mining techniques including link and association analysis, visualization, and predictive analytics. The overarching goal is, essentially, to turn text into data for analysis, via application of natural language processing (NLP) and analytical methods.

A typical application is to scan a set of documents written in a natural language and either model the document set for predictive classification purposes or populate a database or search index with the information extracted.

Text Analysis Code

```
# we will use spam and ham dataset.
# Set below variables with the path
> spam.path <- "C:/Users/dell/Documents/R/data/spam/" #500 Emails
> ham.path <-  "C:/Users/dell/Documents/R/data/easy_ham/" #2500 Emails

# function to read the file
> get.msg <- function(path) {
 +   con <- file(path,open="rt",encoding="latin1")
 +  text <- readLines(con)
 +  msg <- text[seq(which(text=="")[1]+1,length(text))]
 +  close(con)
 +  return(paste(msg,collapse="\n"))

# files extracted are compatible with unix, hence have a file cmd which contains unix commands
# select all files except cmd
> spam.docs <- dir(spam.path)
> spam.docs <- spam.docs[which(spam.docs!="cmds")]
> all.spam <- sapply(spam.docs,function(p)get.msg(paste(spam.path,p,sep="")))

# we do the same thing for all the files containing "ham" email
> ham.docs <- dir(ham.path)
> ham.docs <- ham.docs[which(ham.docs!="cmds")]
> all.ham <- sapply(ham.docs,function(p)get.msg(paste(ham.path,p,sep="")))


spam.docs[1]
00001.7848dde101aa985090474a91ec93fcf0

# we can create our Corpus, document-term matrix (dtm) and term-document matrix (tdm) objects
> library(tm)
>control<-
list(stopwords=TRUE,removePunctuation=TRUE,removeNumbers=TRUE,minDocFreq=2)
> spam.corpus <- Corpus(VectorSource(all.spam))
> spam.tdm <- TermDocumentMatrix(spam.corpus,control)
> spam.dtm <- DocumentTermMatrix(spam.corpus,control)
> ham.corpus <- Corpus(VectorSource(all.ham))
```

```
> ham.tdm <- TermDocumentMatrix(ham.corpus,control)
> ham.dtm <- DocumentTermMatrix(ham.corpus,control)


# stopwords=TRUE     - Removes all common words from the document, such as a, and, or, but, etc.
# removePunctuation=TRUE- Eliminate punctuation marks from the corpus
# removeNumbers=TRUE       - Eliminate numbers from the corpus
# minDocFreq=5               - Ignore all words that appear in less than 5 documents in your corpus
# minWordLength=2    - Ignore all "words" less than 2 characters

# summary of spam corpus
> summary(spam.corpus)
A corpus with 500 text documents
> summary(ham.corpus)
A corpus with 2500 text documents

> head(spam.tdm)
A term-document matrix (6 terms, 500 documents)

Non-/sparse entries: 18/2982
Sparsity           : 99%
Maximal term length: 25
Weighting          : term frequency (tf)

> inspect(spam.tdm[1:3,1:3])
<<TermDocumentMatrix (terms: 3, documents: 3)>>
Non-/sparse entries: 0/9
Sparsity           : 100%
Maximal term length: 22
Weighting          : term frequency (tf)

                                        Docs
Terms                                   1 2 3
  \033b\033b                            0 0 0
  \033bal\033bvipmail                   0 0 0
  \033bckkonsh\033bpt\033bwlshy\033b    0 0 0

> inspect(spam.dtm[1:3,1:3])
<<DocumentTermMatrix (3 documents, 3 terms)
Non-/sparse entries: 0/9
Sparsity           : 100%
```

Maximal term length: 22
Weighting : term frequency (tf)

	Terms		
Docs	\033b\033b \033bal\033bvipmail	\033bckkonsh\033bpt\033bwlshy\033b	
00001.7848dde101aa985090474a91ec93fcf0	0	0	0
00002.d94f1b97e48ed3b553b3508d116e6a09	0	0	0
00003.2ee33bc6eacdb11f38d052c44819ba6c	0	0	0

#Remove atleast 80 percent sparse words
> new.spam.tdm.2 <- removeSparseTerms(spam.tdm,0.8)
> head(new.spam.tdm.2)
A term-document matrix (6 terms, 500 documents)

Non-/sparse entries: 909/2091
Sparsity : 70%
Maximal term length: 11
Weighting : term frequency (tf)

Find how many words have occured minimum 300 times
> length(findFreqTerms(spam.dtm,300))
[1] 29

We can inspect elements of a corpus if we call them by the number they are indexed.
> inspect(spam.corpus[40])
A corpus with 1 text document

The metadata consists of 2 tag-value pairs and a data frame
Available tags are:
create_date creator
Available variables in the data frame are:
MetaID

$`00040.949a3d300eadb91d8745f1c1dab51133`
Dear Sir or Madam:

Please reply to
Receiver: China Enterprise Management Co., Ltd. (CMC)

E-mail: unido@chinatop.net

As one technical organization supported by China Investment and Technical Promotion Office of United Nation Industry Development Organization (UNIDO), we cooperate closely with the relevant Chinese Quality Supervision and Standardization Information Organization. We provide the most valuable consulting services to help you to open Chinese market within the shortest time:

1. Consulting Service on Mandatory National Standards of The People's Republic of China.

2. Consulting Service on Inspection and Quarantine Standards of The People's Republic of China.

3. Consulting Service for Permission to Enter Chinese Market

We are very sorry to disturb you!

More information, please check our World Wide Web: http://www.chinatop.net

Sincerely yours

--
Irish Linux Users' Group: ilug@linux.ie
http://www.linux.ie/mailman/listinfo/ilug for (un)subscription information.
List maintainer: listmaster@linux.ie

```
> findFreqTerms(spam.dtm,300)
 [1] "arial"          "body"           "border"         "borderd"        "business"
 [6] "center"         "div"            "email"          "facedarial"     "faceverdanafont"
[11] "font"           "free"           "height"         "heightd"        "helvetica"
[16] "html"           "input"          "list"           "money"          "option"
[21] "people"         "please"         "receive"        "sansserif"      "size"
[26] "sized"          "table"          "width"          "widthd"

> intersect(findFreqTerms(spam.dtm,100),findFreqTerms(ham.dtm,100))
 [1] "access"      "address"     "business"    "call"        "company"     "computer"
 [7] "contenttype" "day"         "email"       "free"        "government"  "help"
[13] "here"        "home"        "html"        "information" "internet"    "life"
[19] "link"        "list"        "mail"        "mailing"     "message"     "million"
[25] "money"       "name"        "people"      "please"      "report"      "send"
[31] "sent"        "service"     "size"        "software"    "time"        "web"
[37] "you"
```

```
# We can also examine word clouds
# we need to merge all the documents in all.spam and all.ham into a data frame with
two columns, one for each email type
library(wordcloud)
> allspam <- paste(all.spam,sep="",collapse=" ")
> allham <- paste(all.ham,sep="",collapse=" ")
> tmpText = data.frame(c(allham,allspam),row.names=c("HAM","SPAM"))
> ds <- DataframeSource(tmpText)

# Create a new corpus
> corp = Corpus(ds)
> corp = tm_map(corp,removePunctuation)
> corp = tm_map(corp,tolower)
> corp = tm_map(corp,removeNumbers)
> corp = tm_map(corp,function(x){removeWords(x,stopwords())})

# Once we have the tdm, we can create a word cloud:
> tdm <- TermDocumentMatrix(corp)
term.matrix <- as.matrix(tdm)
v <- sort(rowSums(term.matrix),decreasing=TRUE)
d <- data.frame(word=names(v),freq=v)
wordcloud(d$word,d$freq,max.words=150)
par(mfrow=c(1,2))
comparison.cloud(term.matrix,max.words=200,random.order=FALSE,
        colors=c("#999999","#000000"),main="Differences Between
        HAM and SPAM")
commonality.cloud(term.matrix,max.words=100,random.order=FALSE,
        color="#000000",asp=2,main="Similarities Between HAM
        and SPAM")
```

Text Analysis Results

- Word Cloud formed by R package WordCloud

- Comparision Cloud formed by R package wordcloud

Naïve Base Classifier

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Naive Bayes has been studied extensively since the 1950s. It was introduced under a different name into the text retrieval community in the early 1960s, and remains a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. It also finds application in automatic medical diagnosis.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

Naïve Base Code

```
# This takes two required paramters: a file path to an email to classify, and
# a data frame of the trained data.  The function also takes two
# optional parameters.  First, a prior over the probability that an email
# is SPAM, which we set to 0.5 (naive), and constant value for the
```

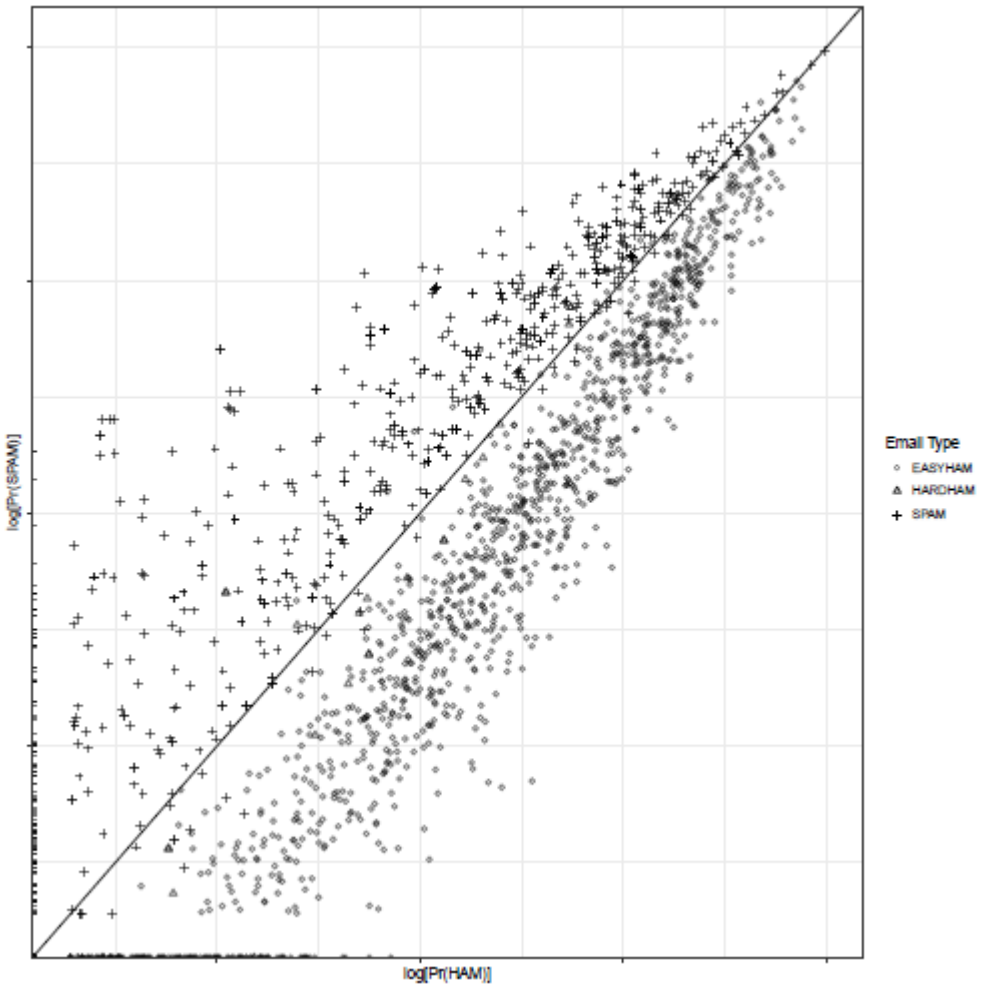
```
# probability on words in the email that are not in our training data.
# The function returns the naive Bayes probability that the given email
# is SPAM.

classify.email <- function(path, training.df, prior = 0.5, c = 1e-6)
{
  # Here, we use many of the support functions to get the
  # email text data in a workable format

  msg <- get.msg(path)
  msg.tdm <- get.tdm(msg)
  msg.freq <- rowSums(as.matrix(msg.tdm))
  # Find intersections of words
  msg.match <- intersect(names(msg.freq), training.df$term)
  # Now, we just perform the naive Bayes calculation
  if(length(msg.match) < 1)
  {
    return(prior * c ^ (length(msg.freq)))
  }
  else
  {
    match.probs <- training.df$occurrence[match(msg.match, training.df$term)]
    return(prior * prod(match.probs) * c ^ (length(msg.freq) - length(msg.match)))
  }
}
```

Naïve Base Results

- Plot of ham and spam email classification using package ggplot in R



- **Result Table**

Email Type	%classifies as ham	%classified as spam
Easy Ham	0.78	0.22
Hard ham	0.73	0.27
Spam	0.15	0.85

SVM and MAXENT

Support Vector Machines (SVM) was introduced in COLT-92 by Boser, Guyon & Vapnik. SVM became popular since, because it's a theoretically well motivated algorithm - developed from Statistical Learning Theory. SVM have successful applications in many fields - bioinformatics, text, image recognition, and may. A large and diverse community work on them - from machine learning, optimization, statistics, neural networks, functional analysis, and so on.

A Support Vector Machine is a discriminative classifier formally defined by a separating hyperplane. In machine learning, support vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Support Vectors: Support vectors are the data points that lie closest to the decision surface and they are the most difficult to classify. They have direct bearing on the optimum location of the decision surface. Support vectors are the elements of the training set that would change the position of the dividing hyper plane if removed. Support vectors are the critical elements of the training set.

The concept of ***Maximum Entropy*** can be traced back along multiple threads to Biblical times. It was introduced to NLP area by Berger in 1996. Steven Phillips et al (2006) proposed a MAXENT approach to species distribution modelling using presence-only data in a spectacularly successful paper (cited 600 times in 2012), and Bill Shipley et al (2006) proposed a MAXENT method for using traits to understand community structure in the journal Science. Ever since there has been a flurry of work to understand MAXENT and its properties.

One reason for the appeal of MAXENT is that it tends to perform well in comparisons of predictive performance, some also see the MAXENT "minimum assumption" philosophy as attractive

The principle of maximum entropy states that, subject to precisely stated prior data (such as a proposition that expresses testable information), the probability distribution which best represents the current state of knowledge is the one with largest entropy. For classification, take precisely stated prior data or testable information about a probability distribution function. Consider the set of all trial probability distributions that would encode the prior data. According to this principle, the distribution with maximal information entropy is the proper one.

The Max Entropy classifier is a probabilistic classifier which belongs to the class of exponential models. Unlike the Naive Bayes classifier that we discussed in the previous article, the Max Entropy does not assume that the features are conditionally independent of each other. The MaxEnt is based on the Principle of Maximum Entropy and from all the models that fit our training data, selects the one

which has the largest entropy. The Max Entropy classifier can be used to solve a large variety of text classification problems such as language detection, topic classification, sentiment analysis and more.

Due to the minimum assumptions that the Maximum Entropy classifier makes, MaxEnt is when we don't know anything about the prior distributions and when it is unsafe to make any such assumptions. Maximum Entropy classifier is used when we can't assume the conditional independence of the features. This is particularly true in Text Classification problems where our features are usually words which obviously are not independent.

SVM and MAXENT Code

```
> spam.path <- "C:/Users/dell/Documents/R/data/spam/"
> spam2.path <- "C:/Users/dell/Documents/R/data/spam_2/"
> easyham.path <- "C:/Users/dell/Documents/R/data/easy_ham/"
> easyham2.path <-  "C:/Users/dell/Documents/R/data/easy_ham_2/"
> hardham.path <- "C:/Users/dell/Documents/R/data/hard_ham/"
> hardham2.path <- "C:/Users/dell/Documents/R/data/hard_ham_2/"
> library(NLP)
> library(tm)

# same function as we wrote for text analysis
> get.msg <- function(path) {
+     con <- file(path,open="rt",encoding="latin1")
+     text <- readLines(con)
+     msg <- text[seq(which(text=="")[1]+1,length(text))]
+     close(con)
```

```
+     return(paste(msg,collapse="\n"))
+ }

> spam.docs <- dir(spam.path)
> spam.docs <- spam.docs[which(spam.docs!="cmds")]
> all.spam <- sapply(spam.docs,function(p)get.msg(paste(spam.path,p,sep="")))
> spam2.docs <- dir(spam2.path)
> spam2.docs <- spam2.docs[which(spam2.docs!="cmds")]
> all.spam2 <- sapply(spam2.docs,
+                     function(p)get.msg(paste(spam2.path,p,sep="")))
> easyham.docs <- dir(easyham.path)
> easyham.docs <- easyham.docs[which(easyham.docs!="cmds")]
> all.easyham <- sapply(easyham.docs,
+                       function(p)get.msg(paste(easyham.path,p,sep="")))
> easyham2.docs <- dir(easyham2.path)
> easyham2.docs <- easyham2.docs[which(easyham2.docs!="cmds")]
> all.easyham2 <- sapply(easyham2.docs,
+                        function(p)get.msg(paste(easyham2.path,p,sep="")))
> hardham.docs <- dir(hardham.path)
> hardham.docs <- hardham.docs[which(hardham.docs!="cmds")]
> all.hardham <- sapply(hardham.docs,
+                       function(p)get.msg(paste(hardham.path,p,sep="")))
> hardham2.docs <- dir(hardham2.path)
> hardham2.docs <- hardham2.docs[which(hardham2.docs!="cmds")]
> all.hardham2 <- sapply(hardham2.docs,
+                        function(p)get.msg(paste(hardham2.path,p,sep="")))
> easy_ham.dfr    <- as.data.frame(all.easyham)
> easy_ham_2.dfr  <- as.data.frame(all.easyham2)
> hard_ham.dfr    <- as.data.frame(all.hardham)
> hard_ham_2.dfr  <- as.data.frame(all.hardham2)
> spam.dfr        <- as.data.frame(all.spam)
```

```
> spam_2.dfr      <- as.data.frame(all.spam2)
> rownames(easy_ham.dfr)    <- NULL
> rownames(easy_ham_2.dfr)  <- NULL
> rownames(hard_ham.dfr)    <- NULL
> rownames(hard_ham_2.dfr)  <- NULL
> rownames(spam.dfr)        <- NULL
> rownames(spam_2.dfr)      <- NULL



# Assign 2 for ham and 4 for spam
> easy_ham.dfr$outcome    <- 2
> easy_ham_2.dfr$outcome  <- 2
> hard_ham.dfr$outcome    <- 2
> hard_ham_2.dfr$outcome  <- 2
> spam.dfr$outcome        <- 4
> spam_2.dfr$outcome      <- 4
> names(easy_ham.dfr)   <- c("text", "outcome")
> names(easy_ham_2.dfr) <- c("text", "outcome")
> names(hard_ham.dfr)   <- c("text", "outcome")
> names(hard_ham_2.dfr) <- c("text", "outcome")
> names(spam.dfr)       <- c("text", "outcome")
> names(spam_2.dfr)     <- c("text", "outcome")
> train.data  <- rbind(easy_ham.dfr, hard_ham.dfr, spam.dfr)
> train.num   <- nrow(train.data)
> train.data  <- rbind(train.data, easy_ham_2.dfr, hard_ham_2.dfr, spam_2.dfr)
> set.seed(2012)
> train_out.data <- train.data$outcome
> train_txt.data <- train.data$text

# Train your data using RTextTools
```

```
> library(RTextTools)

> matrix <- create_matrix(train_txt.data, language="english", minWordLength=3,
removeNumbers=TRUE, stemWords=FALSE, removePunctuation=TRUE,
weighting=weightTfIdf)

> container <- create_container(matrix,t(train_out.data), trainSize=1:train.num,
testSize=(train.num+1):nrow(train.data), virgin=FALSE)

> maxent.model    <- train_model(container, "MAXENT")

> svm.model       <- train_model(container, "SVM")

> svm.result    <- classify_model(container, svm.model)

> svm.analytic  <- create_analytics(container, svm.result)

> svm.doc       <- svm.analytic@document_summary

> svm_spam.doc  <- svm.doc[svm.doc$MANUAL_CODE==4, ]

> svm_ham.doc   <- svm.doc[svm.doc$MANUAL_CODE==2, ]

> svm.true.pos  <- nrow(svm_spam.doc[svm_spam.doc$CONSENSUS_CODE==4,]) /
nrow(svm_spam.doc)

> svm.false.neg <- nrow(svm_spam.doc[svm_spam.doc$CONSENSUS_CODE==2,]) /
nrow(svm_spam.doc)

> svm.true.neg  <- nrow(svm_ham.doc[svm_ham.doc$CONSENSUS_CODE==2,]) /
nrow(svm_ham.doc)

> svm.false.pos <- nrow(svm_ham.doc[svm_ham.doc$CONSENSUS_CODE==4,]) /
nrow(svm_ham.doc)

> maxent.result   <- classify_model(container, maxent.model)

> maxent.analytic <- create_analytics(container, maxent.result)

> maxent.doc      <- maxent.analytic@document_summary

> maxent_spam.doc <- maxent.doc[maxent.doc$MANUAL_CODE==4, ]

> maxent_ham.doc  <- maxent.doc[maxent.doc$MANUAL_CODE==2, ]

> maxent.true.pos <-
nrow(maxent_spam.doc[maxent_spam.doc$CONSENSUS_CODE==4,]) /
nrow(maxent_spam.doc)

> maxent.false.neg<-
nrow(maxent_spam.doc[maxent_spam.doc$CONSENSUS_CODE==2,]) /
nrow(maxent_spam.doc)

> maxent.true.neg <-
nrow(maxent_ham.doc[maxent_ham.doc$CONSENSUS_CODE==2,]) /
nrow(maxent_ham.doc)
```

```
> maxent.false.pos<-
nrow(maxent_ham.doc[maxent_ham.doc$CONSENSUS_CODE==4,]) /
nrow(maxent_ham.doc)

> View(spam.matrix)
```

SVM and MAXENT Result

- Table for SVM result

Email Type	TRUE	FALSE
Spam	86.8%	13.2%
Ham	96.8%	3.2%

- Table for Maxent result

Email Type	TRUE	FALSE
Spam	85.3%	14.7%
Ham	99.6%	0.4%

Phishing Emails Features

Phishing Websites Features

As per our research, we have defined particular features of the phishing emails that have proved to be sound and effective in predicting phishing websites. In addition, we proposed some new features, experimentally assign new rules to some well-known features and update some other features. Many of these features might be used in our analysis.

Using the IP Address

If an IP address is used as an alternative of the domain name in the URL, such as "http://125.98.3.123/fake.html", users can be sure that someone is trying to steal their personal information. Sometimes, the IP address is even transformed into hexadecimal code as shown in the following link "http://0x58.0xCC.0xCA.0x62/2/paypal.ca/index.html".

*Rule: IF{*If The Domain Part has an IP Address → PhishingOtherwise→ Legitimate

Long URL to Hide the Suspicious Part

Phishers can use long URL to hide the doubtful part in the address bar. For example: http://federmacedoadv.com.br/3f/aze/ab51e2e319e51502f416dbe46b773a5e/?cmd=_home&dispatch=11004d58f5b74f8dc1e7c2e8dd4105e811004d58f5b74f8dc1e7c2e8dd4105e8@phishing.website.html... To ensure accuracy of our study, we calculated the length of URLs in the dataset and produced an average URL length. The results showed that if the length of the URL is greater than or equal 54 characters then the URL classified as phishing. By reviewing our dataset we were able to find 1220 URLs lengths equals to 54 or more.

Rule: IF $\{URL\ length$<54 $\rightarrow\ feature$=Legitimate $else\ if\ URL\ length$≥54 and ≤75 $\rightarrow$ $feature$=$Suspicious\ otherwise\rightarrow\ feature$=Phishing

We have been able to update this feature rule by using a method based on frequency and thus improving upon its accuracy.

Using URL Shortening Services "TinyURL"

URL shortening is a method on the "World Wide Web" in which a URL may be made considerably smaller in length and still lead to the required webpage. This is accomplished by means of an "HTTP Redirect" on a domain name that is short, which links to the webpage that has a long URL. For example, the URL "http://portal.hud.ac.uk/" can be shortened to "bit.ly/19DXSk4".

*Rule: IF{*TinyURL → PhishingOtherwise→ Legitimate

URL's having "@" Symbol

Using "@" symbol in the URL leads the browser to ignore everything preceding the "@" symbol and the real address often follows the "@" symbol.

Rule: IF {Url Having @ Symbol→ PhishingOtherwise→ Legitimate

Redirecting using "//"

The existence of "//" within the URL path means that the user will be redirected to another website. An example of such URL's is: "http://www.legitimate.com//http://www.phishing.com". We examin the location where the "//" appears. We find that if the URL starts with "HTTP", that means the "//" should appear in the sixth position. However, if the URL employs "HTTPS" then the "//" should appear in seventh position.

Rule: IF {ThePosition of the Last Occurrence of "//" in the URL > 7→ PhishingOtherwise→ Legitimate

Adding Prefix or Suffix Separated by (-) to the Domain

The dash symbol is rarely used in legitimate URLs. Phishers tend to add prefixes or suffixes separated by (-) to the domain name so that users feel that they are dealing with a legitimate webpage. For example http://www.Confirme-paypal.com/.

Rule: IF {Domain Name Part Includes (−) Symbol → PhishingOtherwise → Legitimate

Sub Domain and Multi Sub Domains

Let us assume we have the following link: http://www.hud.ac.uk/students/. A domain name might include the country-code top-level domains (ccTLD), which in our example is “uk”. The “ac” part is shorthand for “academic”, the combined “ac.uk” is called a second-level domain (SLD) and “hud” is the actual name of the domain. To produce a rule for extracting this feature, we firstly have to omit the (www.) from the URL which is in fact a sub domain in itself. Then, we have to remove the (ccTLD) if it exists. Finally, we count the remaining dots. If the number of dots is greater than one, then the URL is classified as “Suspicious” since it has one sub domain. However, if the dots are greater than two, it is classified as “Phishing” since it will have multiple sub domains. Otherwise, if the URL has no sub domains, we will assign “Legitimate” to the feature.

Rule: IF {Dots In Domain Part=1 → LegitimateDots In Domain Part=2 → SuspiciousOtherwise→ Phishing

Using Non-Standard Port

This feature is useful in validating if a particular service (e.g. HTTP) is up or down on a specific server. In the aim of controlling intrusions, it is much better to merely open ports that you need. Several firewalls, Proxy and Network Address Translation (NAT) servers will, by default, block all or most of the ports and only open the ones selected. If all ports are open, phishers can run almost any service they want and as a result, user information is threatened.

Rule: IF{ Port # is of the Preffered Status→ PhishingOtherwise→ Legitimate

Phishing Results

Email Type	TRUE	FALSE
Phishing	94.14%	5.86%
Non-Phishing	1.24%	98.75%

Conclusion

All the classifiers examined achieved over 92 percent accuracy, although Maxent and the SVM with the word bag performed best. Naive Bayes misclassified over 15 percent of spam emails, however, while the SVM misclassified 13 percent of phishing emails. Performance of the SVM was on par with naive Bayes and maxent operating on the reduced feature set, making the SVM a practical real world method of flagging potential spam and phishing emails before they reach the user

References

[1] I. Androutsopoulos, J. Koutsias, K. Chandrinos, G. Paliouras, and C. Spyropoulos. An evaluation of naive bayesian anti-spam _ltering. In Proc. of the workshop on Machine Learning in the New Information Age, 2000.

[2] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, and C. D. Spyropoulos. An experimental comparison of naive bayesian and keyword-based anti-spam _ltering with personal e-mail messages. In SIGIR '00:Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, pages 160{167, New York, NY, USA, 2000. ACM Press.

[3] Anti-Phishing Working Group. http://www.antiphishing.org/.

[4] M. W. Berry, editor. Survey of Text Mining: Clustering, Classi_cation, and Retrieval. Springer, 2004.

[5] L. Breiman. Random forests. Machine Learning, 45(1):5{32, October 2001.

[6] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. Classi_cation and Regression Trees. Chapman & Hall/CRC, 1984.

[7] M. Chandrasekaran, K. Narayanan, and S. Upadhyaya. Phishing email detection based on structural properties. In NYS Cyber Security Conference, 2006.

[8] H. A. Chipman, E. I. George, and R. E. McCulloch. Bayesian CART model search. Journal of the American Statistical Association, 93(443):935{947, 1998.

[9] H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian Additive Regression Trees. Journal of the Royal Statistical Society, 2006. Ser. B, Revised.

[10] L. F. Cranor, S. Egelman, J. Hong, and Y. Zhang. Phinding phish: An evaluation of anti-phishing toolbars. Technical Report CMU-CyLab-06-018, CMU, November 2006.