DATA 607 Project 4 - SPAM/HAM

KoohPy <- Koohyar Pooladvand

2024-04-28

Intro

The goal of this project is to work with a database to identify **spam emails**. Being able to classify new "test" documents using already classified "training" documents is crucial. A common scenario involves using a corpus of labeled spam and ham (non-spam) emails to predict whether a new document is spam or not. For this project, we can begin with a spam/ham dataset and then predict the class of new documents—either withheld from the training dataset or another source, such as your spam folder. An example corpus can be found at SpamAssassin Public Corpus.

Similar to other projects, I'll start by initializing R Studio and then proceed step-by-step toward the end goal.

Initialization

```
## [1] "All required packages are installed"
```

Strategy and method

Upon researching this topic, I have come across some repositories on GitHub and articles that focus on email spam detection. The main concepts presented in these resources can be summarized in the following steps:

1. **Finding and Importing Data**:
    - Gather relevant email data (both spam and ham).
    - Import the data into your analysis environment.
2. **Exploratory Data Analysis (EDA)**:
    - Explore the dataset to understand its structure, features, and potential issues.
    - Identify any missing data or outliers.
3. **Creating a Corpus**:
    - Clean up the data by removing unnecessary characters, formatting, and noise.
    - Handle missing data appropriately.
    - Prepare the data for training by creating a corpus.
4. **Applying a Machine Learning Algorithm**:
    - Choose an appropriate algorithm (e.g., Naive Bayes, SVM, or Random Forest).

- Train the model using the cleaned data.
- Fine-tune hyperparameters as needed.

5. **Model Evaluation**:
   - Assess the model's performance using metrics such as accuracy, precision, recall, and F1-score.
   - Consider cross-validation to validate the model's generalization ability.

These are the general steps, each with several sub-steps to be performed. As you work on your project, feel free to utilize available resources and code from other references, citing them appropriately when used.

All in all our goal is to use the source of legitimate (ham) and illegitimate emails (spam) and use them to train a network to be able to identify the future email an one of these categories. It is a fun project, and one thing that I have never done before.

In this project, the data collection has been done and we have access to the corpus of ham and spam data, which makes my life easier. First step i to load those data into R.

1- Importing data: Reading files into R

```
# Paths of spam and ham folders
#spam_path <- "C:/Users/kohya/OneDrive/CUNY/DATA 607/DATA 607/Data/spam"
#ham_path <- "C:/Users/kohya/OneDrive/CUNY/DATA 607/DATA 607/Data/ham"
spam_path <- "Data/spam"
ham_path <- "Data/ham"


# Function to read emails from a folder
read_emails <- function(folder_path) {
  # List all files in the folder
  email_files <- list.files(folder_path, full.names = TRUE)

  # Initialize list to store email content
  email_contents <- vector("list", length(email_files))

  # Loop through each file
  for (i in seq_along(email_files)) {
    # Read the content of the email
    email_content <- readLines(email_files[i],encoding = "UTF-8") #use "UTF-8 for multibyte encodidng t

    # Store the email content
    email_contents[[i]] <- paste(email_content, collapse = "\n")
  }

  # Return the list of email contents
  return(email_contents)
}

# Read emails from spam and ham folders
spam_email_contents <- read_emails(spam_path)
```

```
## Warning in readLines(email_files[i], encoding = "UTF-8"): incomplete final line
## found on 'Data/spam/00136.faa39d8e816c70f23b4bb8758d8a74f0'
```

```r
ham_email_contents <- read_emails(ham_path)

# Create dataframes for spam and ham emails
spam_emails_df <- data.frame(Content = unlist(spam_email_contents), Label = "spam", stringsAsFactors = F
ham_emails_df <- data.frame(Content = unlist(ham_email_contents), Label = "ham", stringsAsFactors = FALS

#Combined the two spam and ham database together for
emails_df <- rbind(spam_emails_df,ham_emails_df)
```

```r
# Print the first few rows of each dataframe
#knitr::kable(head(spam_emails_df, 3),format = "markdown")
# Convert the data frame to a Markdown table
message("Three examples of Spam emails")
```

```
## Three examples of Spam emails
```

```r
#markdown_table <- knitr::kable(data.frame(head(spam_emails_df, 3)), format = "markdown", sanitize = TR
#markdown_table <- pander::pander(head(spam_emails_df, 3), style = "pipe")
# Print the Markdown table
#cat(markdown_table)
cat("\n")
```

```r
message("Three examples of Ham emails")
```

```
## Three examples of Ham emails
```

```r
#knitr::kable(head(ham_emails_df, 3),format = "markdown")
# Convert the data frame to a Markdown table
#markdown_table <- knitr::kable(data.frame(head(ham_emails_df, 3)), format = "markdown", sanitize = TRU
#markdown_table <- pander::pander(head(ham_emails_df, 3), style = "pipe")
# Print the Markdown table
#cat(markdown_table)
cat("\n")
```

2- Data Processing: Exploratory Data Analyses

Once we have our dataset, we'll need to preprocess the data. This typically involves tasks like tokenization, removing stop words, stemming or lemmatization, and possibly handling issues like spelling mistakes or special characters.

```r
# Regular expression to find multibites characters
error_pattern <- "[[:cntrl:]]"

# Function to identify potential error indices
find_multibyte_errors <- function(text) {
  error_indices <- grep(error_pattern, text, perl = TRUE)
  return(error_indices)
}

# Apply the function to spam Content
```

```r
potential_errors <- lapply(spam_emails_df$Content, find_multibyte_errors)


# Define replacement character for multi-bites
replace_char <- "?"

# Replace multi-byte characters with the placeholder
spam_emails_df$Content <- str_replace_all(spam_emails_df$Content, "[[:cntrl:]]", replace_char)
ham_emails_df$Content <- str_replace_all(ham_emails_df$Content, "[[:cntrl:]]", replace_char)
emails_df$Content <- str_replace_all(emails_df$Content, "[[:cntrl:]]", replace_char)

# Now you can use nchar or strwidth on the modified columns
df_EDA <- data.frame(spam = nrow(spam_emails_df), ham = nrow(ham_emails_df))

df_EDA <- rbind(df_EDA, min_email = c(min(nchar(spam_emails_df$Content)), min(nchar(ham_emails_df$Conte

df_EDA <- rbind(df_EDA, max_email = c(max(nchar(spam_emails_df$Content)), max(nchar(ham_emails_df$Conte

# Set row names
rownames(df_EDA) <- c("Emails #", "Min char length", "Max char Length")

# Print the dataframe
print(df_EDA)
```

```
##                    spam   ham
## Emails #            501  2551
## Min char length     867   367
## Max char Length  232374 90428
```

2-1- Cleaning and tidying data

Exploring the data it shows it contains many unuseful and additional irrrelevant data. The SPAM email should be mostly recognized by its sender email address, its domain, the contains of the email, to how many recipient is was sent, subject. Additionally, there can be some other parameters like when it was sent, it is a reply/forwarded or a new email, and what is its sentiment. These additional information are those that can amply improve the quality of the process.

I must create a set of functions that goes thru the email and extract above information for each emails. Specifically by starting reading the HTML and text and seperate them to diffrentiate the body text from other informtion.

In this particular project, I only attmp to use the body text of the message for the analyses not the entire message.

```r
#the goal of this section is to extract the body of a message text and store them in the database not t
```

3- Feature Extraction: Creating a Corpus

Th goal of this section is to use the data to be able to extract some numerical data to be used later for machine learning algorithm. As I was learning about the topic, there are different techniques to be used with Bag-of-Words (BoW) and Term Frequency-Inverse Docuement Frequency (TF-IDF) are the most striagt forwrd and mostly used.

- **Bag-of-Words (BoW)**: Representing each document as a vector where each element corresponds to the frequency of a word in the document.
- **TF-IDF (Term Frequency-Inverse Document Frequency)**: Weighing the importance of words in a document relative to their frequency across all documents in the corpus.

3-1- BOW

To work on the BoW, one may need to follwo the follwoign steps:

1. **Creating Corpus and Preprocessing**:
   - create a Corpus from the email contents using `Corpus(VectorSource())`.
   - Then apply various preprocessing steps to the corpus, such as converting text to lowercase, removing punctuation, numbers, whitespace, and stopwords, and stemming the words.
2. **Creating Document-Term Matrix (DTM)**:
   - convert the preprocessed corpus into a Document-Term Matrix (DTM) using `DocumentTermMatrix()`.
   - The DTM represents each document as a row and each unique word as a column, with the cell values indicating the frequency of each word in each document.
   - Use hunspell_ckeck to remove all the nonenglish and unrelated word from the column.
3. **Removing Sparse Terms**:
   - Remove sparse terms (words that appear in only a small fraction of documents) from the DTM using `removeSparseTerms()`.
4. **Converting DTM to Dataframe**:
   - Convert the DTM to a dataframe `email_matrix_spam` and `email_matrix_ham`.
   - It also adds the label column (spam/ham) to each dataframe.

SPAM email analyses

```
# Create a Corpus from the email contents
corpus_spam <- Corpus(VectorSource(spam_emails_df$Content))

# Convert to lowercase
corpus_spam <- tm_map(corpus_spam, content_transformer(tolower))

#Convert to plain text
#corpus_spam <-  tm_map(corpus_spam, PlainTextDocument)

# Remove punctuation
corpus_spam <- tm_map(corpus_spam, removePunctuation)

# Remove numbers
corpus_spam <- tm_map(corpus_spam, removeNumbers)

# Remove whitespace
corpus_spam <- tm_map(corpus_spam, stripWhitespace)
```

```r
# Remove stopwords
corpus_spam <- tm_map(corpus_spam, removeWords, stopwords("english"))

# Stemming
corpus_spam <- tm_map(corpus_spam, stemDocument)

# Convert the corpus to a DocumentTermMatrix
#The values in the matrix are the number of times that word appears in each document.
dtm_spam <- DocumentTermMatrix(corpus_spam)


#Remove sparse terms
dtm_spam <- removeSparseTerms(dtm_spam, 0.95)

dtm_spam
```

```
## <<DocumentTermMatrix (documents: 501, terms: 559)>>
## Non-/sparse entries: 38208/241851
## Sparsity           : 86%
## Maximal term length: 44
## Weighting          : term frequency (tf)
```

```r
# Convert the DocumentTermMatrix to a dataframe
email_matrix_spam <- as.data.frame(as.matrix(dtm_spam))

#use the make.names function to make the variable names.
colnames(email_matrix_spam) = make.names(colnames(email_matrix_spam))

#Use hunspell to check is the columns is actually an english word
#hunspell_check(colnames(email_matrix_spam))
#Use hunspell_check with which to only keep the column that have menaingful neglish words.
email_matrix_spam <- email_matrix_spam[, which(hunspell_check(colnames(email_matrix_spam)))]

# Add labels to the dataframe
email_matrix_spam$Label <- spam_emails_df$Label

# Print the first few rows of the processed dataframe
kable(head(email_matrix_spam, 10),
      align = c("l", "c", "c"),  # Align columns left, center, center
      width = "80%",  # Width of the table
      booktabs = TRUE,  # Use booktabs style
      longtable = TRUE,  # Allow table to span multiple pages if needed
      escape = FALSE)  # Allow LaTeX commands
```

access	best	buy	can	click	dollar	email	end	error	even	ever	fast	font	free	get	help	import	invest	less	let	life	list	lowest	make	money	[illegible]	pay	please	[illegible]	public	request	[illegible]	save	shop	size	start	state	take	thousand	type	week	will	wish	[illegible]	[illegible]	
1	3	1	1	1	1	2	2	1	1	1	1	2	3	1	1	1	1	1	1	5	1	1	1	1	1	1	1	1	1	1	1	5	1	5	1	1	1	1	2	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	2
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0
3	3	0	1	1	1	1	0	1	0	0	0	0	8	4	0	0	0	0	0	0	0	0	0	0	0	0	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

access	best	buy	can	click	dollar	email	end	error	even	ever	fast	font	free	get	help	import	invest	less	let	life	list	lowest	make	money	new	pay	pleas	post	public	request	sat	save	shop	size	start	state	take	thousand	type	week	will	wish	claim	day	
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	2
0	0	0	1	0	0	4	0	1	0	1	0	0	1	2	1	0	0	0	0	0	1	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	3	1
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
0	0	0	2	0	0	1	0	0	0	0	0	8	1	0	0	2	0	0	0	3	0	0	1	0	0	0	0	1	0	0	0	1	0	25	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	5	0	1	0	1	0	0	2	2	1	2	0	0	0	0	1	0	2	2	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	4	1	0	3	1
0	0	0	0	0	0	2	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	18	0	0	0	0	0	0	1	1	0	0	0

HAM email Analyses

```r
#Do the same for the ham
corpus_ham <- Corpus(VectorSource(ham_emails_df$Content))
corpus_ham <- tm_map(corpus_ham, content_transformer(tolower))
#corpus_ham <- tm_map(corpus_ham, PlainTextDocument)
corpus_ham <- tm_map(corpus_ham, removePunctuation)
corpus_ham <- tm_map(corpus_ham, removeNumbers)
corpus_ham <- tm_map(corpus_ham, stripWhitespace)
corpus_ham <- tm_map(corpus_ham, removeWords, stopwords("english"))
corpus_ham <- tm_map(corpus_ham, stemDocument)
dtm_ham <- DocumentTermMatrix(corpus_ham)
dtm_ham <- removeSparseTerms(dtm_ham, 0.95)
dtm_ham
```

```
## <<DocumentTermMatrix (documents: 2551, terms: 399)>>
## Non-/sparse entries: 154473/863376
## Sparsity           : 85%
## Maximal term length: 92
## Weighting          : term frequency (tf)
```

```r
email_matrix_ham <- as.data.frame(as.matrix(dtm_ham))
colnames(email_matrix_ham) = make.names(colnames(email_matrix_ham))
email_matrix_ham <- email_matrix_ham[, which(hunspell_check(colnames(email_matrix_ham)))]
email_matrix_ham$Label <- ham_emails_df$Label

# Print the first few rows of the processed dataframe
kable(head(email_matrix_ham, 10),
      align = c("l", "c", "c"),  # Align columns left, center, center
      width = "80%",  # Width of the table
      booktabs = TRUE,  # Use booktabs style
      longtable = TRUE,  # Allow table to span multiple pages if needed
      escape = FALSE)  # Allow LaTeX commands
```

actual	can	code	come	data	day	develop	discuss	error	express	get	happen	hit	like	line	list	local	log	mail	make	new	part	run	still	subject	sun	that	time	today	use	version	wed	without	work	contact	email	fair	free	group	high	internet	network	seek	sponsor	unknown	
1	1	1	2	1	1	1	1	1	1	1	1	3	1	1	4	1	1	1	1	1	1	1	1	3	1	1	1	1	1	2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	4	0	0	0	0	0	2	0	0	0	0	1	0	0	0	0	1	1	1	1	3	1	2	1	1	1	2	1
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	2	0	0	2	0	0	0	0	0	2	0	0	0	0	1	0	0	0	0	1	1	0	1	3	1	0	1	1	1	2	0

actual	and	code	come	data	day	develop	discuss	error	express	get	happen	hit	like	line	list	local	log	mail	make	new	part	run	still	subject	sum	that	time	today	use	version	wed	without	work	contact	email	fair	free	group	high	internet	network	send	sponsor	unkno
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	3	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	2	0	0	3	0	0	1	0	0	1	0	0	1	0	3	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	2	0	0	0	0	0	2	0	0	1	0	5	0	0	0	1	1	1	0	1	3	0	0	1	1	1	2
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	4	0	0	0	0	0	2	0	1	0	0	3	0	0	0	0	1	1	0	1	3	0	0	1	1	1	2
0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	2	0	0	4	0	0	0	0	0	2	0	0	0	0	1	0	0	0	0	1	1	0	1	3	0	0	1	1	1	2
1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	3	0	0	2	0	0	0	0	0	2	0	1	0	0	2	0	0	1	0	1	1	0	1	3	0	1	1	1	1	2
0	0	0	0	0	4	1	0	0	2	1	0	2	2	0	2	0	0	4	0	0	0	0	0	2	0	0	5	0	3	0	0	1	0	1	1	1	1	3	0	0	1	1	1	2

Combined Email analyses

```r
#let's work on the combined emails for the purpose of this work
corpus <- Corpus(VectorSource(emails_df$Content))
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, stemDocument)
dtm <- DocumentTermMatrix(corpus)
dtm <- removeSparseTerms(dtm, 0.95)
dtm
```

```
## <<DocumentTermMatrix (documents: 3052, terms: 406)>>
## Non-/sparse entries: 180093/1059019
## Sparsity           : 85%
## Maximal term length: 62
## Weighting          : term frequency (tf)
```

```r
email_matrix <- as.data.frame(as.matrix(dtm))
colnames(email_matrix) = make.names(colnames(email_matrix))
email_matrix <- email_matrix[, which(hunspell_check(colnames(email_matrix)))]
email_matrix$Label <- emails_df$Label

# Print the first few rows of the processed dataframe

kable(head(email_matrix, 10),
      align = c("l", "c", "c"),  # Align columns left, center, center
      width = "80%",  # Width of the table
      booktabs = TRUE,  # Use booktabs style
      longtable = TRUE,  # Allow table to span multiple pages if needed
      escape = FALSE)  # Allow LaTeX commands
```

access	best	can	click	dear	email	error	even	ever	free	get	help	import	less	let	life	list	make	money	new	pay	please	public	sat	size	start	state	take	wed	will	wish	daily	day	group	link	maintain	member	support	use	user	web	window	date	te	mal
1	3	1	1	2	2	1	1	1	3	1	1	1	1	1	5	1	1	1	1	1	1	1	1	1	5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	1	2	1	1	1	1	1	2	1	1	0	0

access	best	can	click	email	[illegible]	error	even	ever	free	get	help	important	less	let	life	list	make	money	[illegible]	pay	please	[illegible]	public	sat	size	start	state	take	wed	will	wish	daily	day	group	link	maintain	member	support	use	user	web	window	date	[illegible]	[illegible]
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	1	0	1	0	0	1	1	0	1	1	1	1	0
3	3	1	1	1	0	1	0	0	8	4	0	0	0	0	0	0	0	0	0	3	1	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	3	1	0	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	2	1	1	1	1	1	2	0	1	0	0	0
0	0	1	0	4	0	1	0	1	1	2	1	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0	2	1	0	3	1	0	0	1	0	0	0	0	0	1	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
0	0	2	0	1	0	0	0	0	1	0	0	2	0	0	3	0	1	0	0	0	0	1	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0
0	0	1	0	5	0	1	0	1	2	2	1	2	0	0	0	1	2	2	1	0	1	1	0	0	0	0	0	0	0	4	1	0	3	1	0	0	1	0	0	0	0	0	1	1	0
0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	18	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3-2- TF-IDF

The dataframe is created above will be used below to created the TF-IDF. The data preparation steps are the same, thus they are not repeated but the additional required step now listed below are taken:

- Calculate TF-IDF weights using `weightTfIdf()`.
- Convert the TF-IDF weighted matrix to a dataframe.
- Add the labels to the dataframe.

```r
# Calculate TF-IDF weights
tfidf_transformer_spam <- weightTfIdf(dtm_spam)

# Convert TF-IDF weighted matrix to a dataframe
tfidf_df_spam <- as.data.frame(as.matrix(tfidf_transformer_spam))

# Add labels to the dataframe
tfidf_df_spam$Label <- spam_emails_df$Label

# Print the first few rows of the processed dataframe
#kable(head(tfidf_df_spam))


#do the same for the ham
# Calculate TF-IDF weights
tfidf_transformer_ham <- weightTfIdf(dtm_ham)

# Convert TF-IDF weighted matrix to a dataframe
tfidf_df_ham <- as.data.frame(as.matrix(tfidf_transformer_ham))

# Add labels to the dataframe
tfidf_df_ham$Label <- ham_emails_df$Label

# Print the first few rows of the processed dataframe
#kable(head(tfidf_df_ham))


#do the same for the combined emals
# Calculate TF-IDF weights
tfidf_transformer <- weightTfIdf(dtm)
```

```r
# Convert TF-IDF weighted matrix to a dataframe
tfidf_df <- as.data.frame(as.matrix(tfidf_transformer))

# Add labels to the dataframe
tfidf_df$Label <- emails_df$Label

# Print the first few rows of the processed dataframe
kable(head(tfidf_df))
```

access	aug	best	can	charset-iso-content-transfer-encoding	click	[illegible]	[illegible]	fetchmail-forfree	[illegible]	[illegible]	localhost	mail-web-store	[illegible]	[illegible]	phobos-labs-spamassassin-taint-org	[illegible]	postfix	[illegible]	return-path	[illegible]
0.0[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]05	[illegible]
0.00[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]05	[illegible]
0.00[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]05	[illegible]
0.0[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]05	[illegible]
0.00[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]05	[illegible]
0.00[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]05	[illegible]

4- Applying a Machine Learning Algorithm

In this section we will be creating a few different models and will be evaluating their pefroamnce unsing the definition of accuracy, precision, recall, and F1 score as define below. We analyze the data to determine the true positive, false positive, true negative, and false negative values for the dataset, using the prediction model created using train datset on the test datset. We will display results in three confusion matrices, with the counts of TP, FP, TN, and FN.

Accuracy = (TP+TN)/(all (TP+TN+FP+FN)) Precision = TP/(TP+FP) Recall = TP / (TP+FN) F1 Score = 2 x (Precision x Recall) / (Precision + Recall)

For findign the confusion matrix, I will use the confusionMatrix() function from the caret package to generate a confusion matrix.

4-1- Explanatory Data Analyses

After cleaning and creating the DTM, I use a basic EDA to show some basics like wordscloud and histograms of words repeated in this Spam and ham emails. to just give us some intuitive ad additional information on what is going on.

It is surprising see some unrelated words like `localhost`, months, i.e., `oct`, `sep`, weeks of the day, i.e., `tue`, `wed`, and so one be repeated itself. It shows I need to do some additional data cleaning in the emails to remove some title and just work on the email bodies.

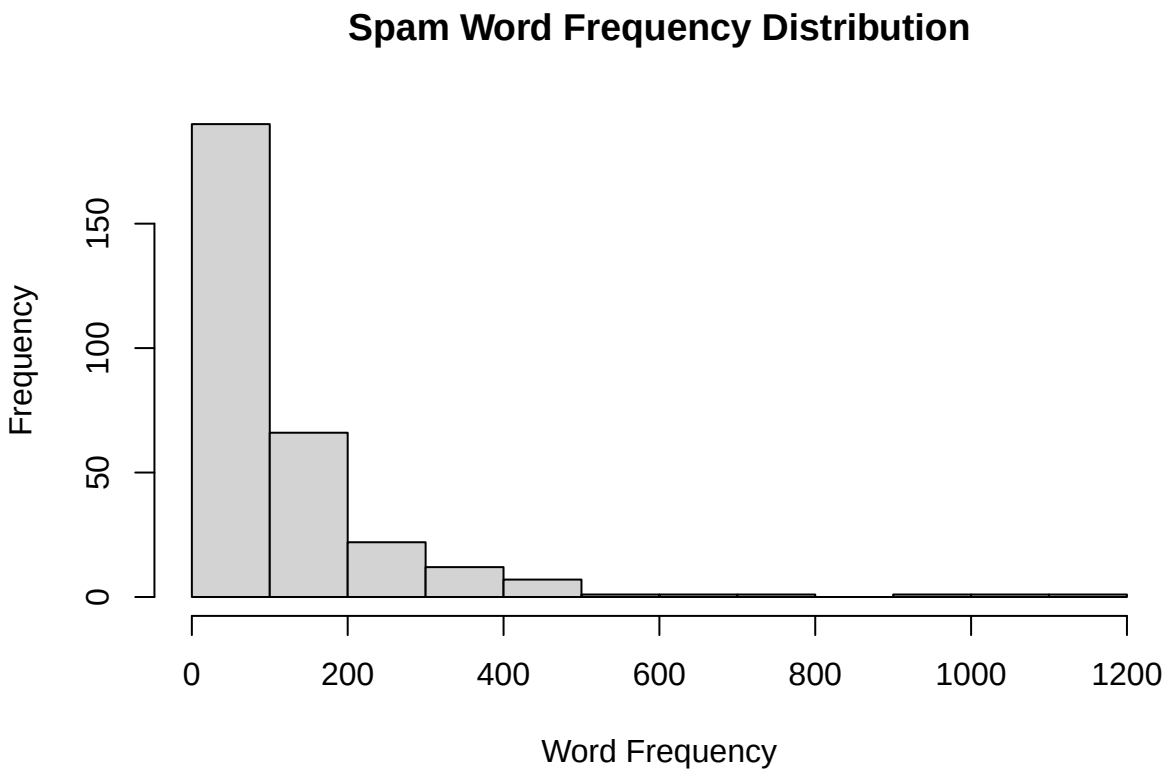
I have not figure it out yet, need sometimes to work. At the moment, I have decided to move on, knowing this is a lot of room for improvement in the DTM has been created for the ham and spam data.

SPAM EDA

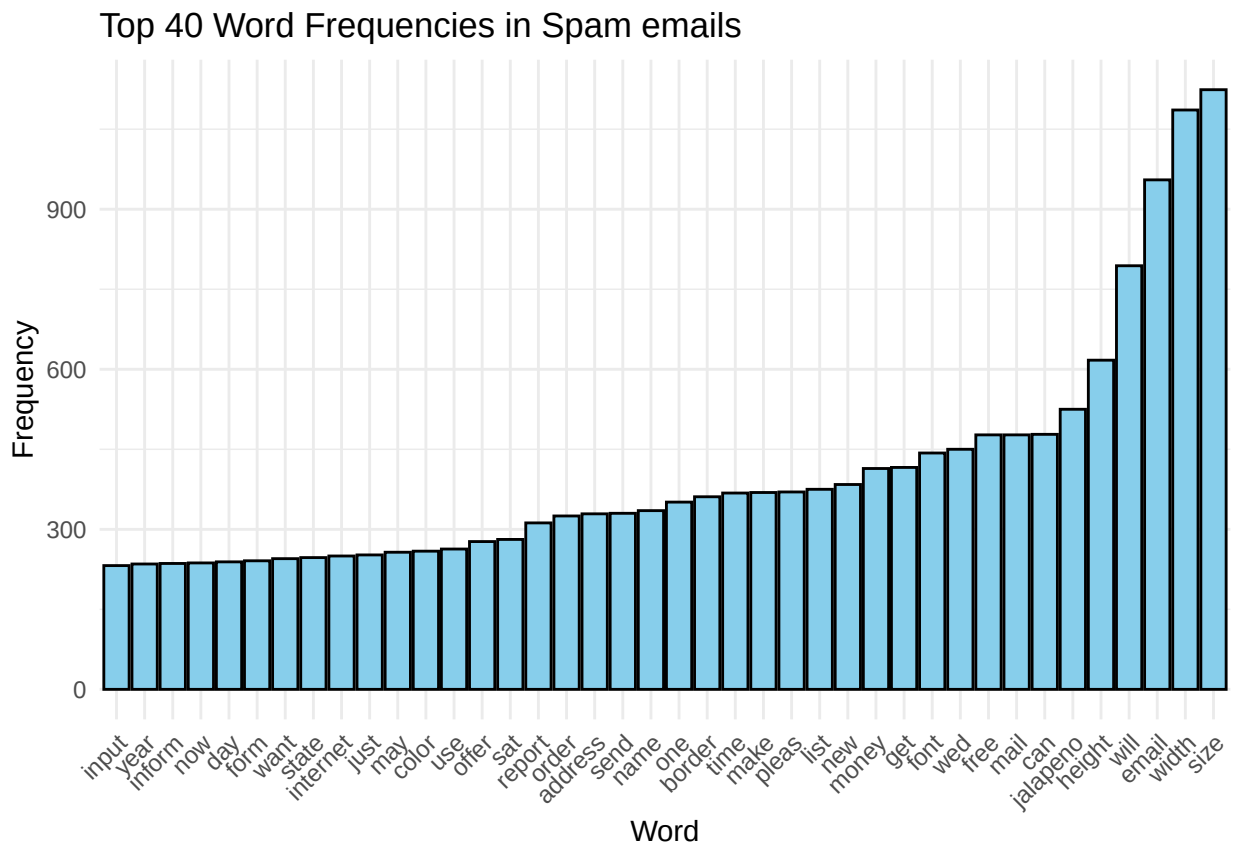
```
# Calculate the sum of each column (word frequency) excluding the "Label" column
word_frequencies_spam <- colSums(email_matrix_spam[, !colnames(email_matrix_spam) %in% "Label"])


# Sort the word frequencies in descending order
sorted_word_frequencies_spam <- sort(word_frequencies_spam, decreasing = TRUE)

# Print the sorted word frequencies
kable(head(sorted_word_frequencies_spam, 20),
      align = c("l", "c", "c"),  # Align columns left, center, center
      width = "20%",  # Width of the table
      booktabs = TRUE,  # Use booktabs style
      longtable = TRUE,  # Allow table to span multiple pages if needed
      escape = FALSE)  # Allow LaTeX commands
```

	x
size	1124
width	1086
email	955
will	794
height	617
jalapeno	525
can	478
free	477
mail	477
wed	450
font	443
get	416
money	414
new	384
list	375
pleas	370
make	369
time	368
border	361
one	351

```r
# Visualize word frequencies using a histogram
hist(word_frequencies_spam, main = "Spam Word Frequency Distribution", xlab = "Word Frequency", ylab = "
```




```r
# Convert word frequencies to a dataframe
word_frequencies_df_spam <- data.frame(word = names(word_frequencies_spam), frequency = word_frequencies

word_frequencies_df_spam <- word_frequencies_df_spam [order(word_frequencies_df_spam$frequency,decreasi

# Select the top 40 words
top_40_words_spam <- head(word_frequencies_df_spam, 40)


# Create a histogram using ggplot2 for the top 40 words
top_40_words_spam |> ggplot(aes(x = reorder(word, frequency), y = frequency)) +
  geom_bar(stat = "identity", fill = "skyblue", color = "black") +
  labs(title = "Top 40 Word Frequencies in Spam emails",
       x = "Word",
       y = "Frequency") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
# Visualize word frequencies using a word cloud
wordcloud(names(word_frequencies_spam), freq = word_frequencies_spam, max.words = 70, random.order = FA
```

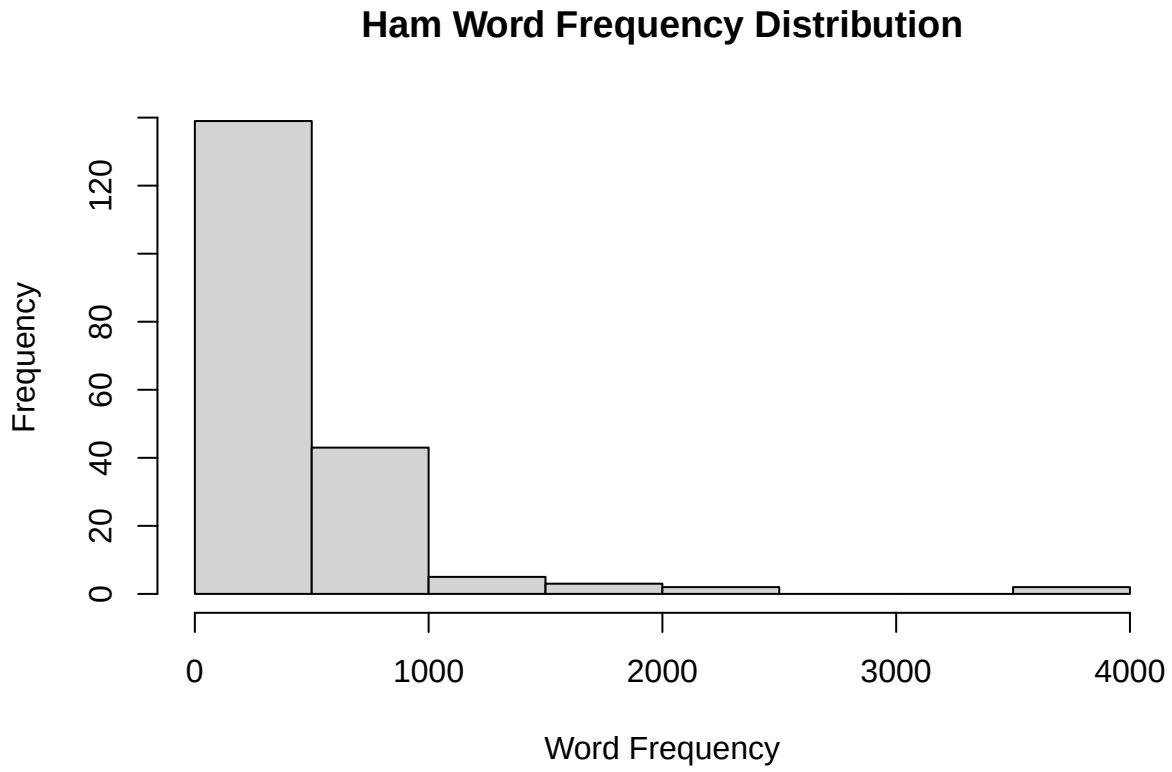
HAM EAD

```
# Calculate the sum of each column (word frequency) excluding the "Label" column
word_frequencies_ham <- colSums(email_matrix_ham[, !colnames(email_matrix_ham) %in% "Label"])

# Sort the word frequencies in descending order
sorted_word_frequencies_ham <- sort(word_frequencies_ham, decreasing = TRUE)

# Print the sorted word frequencies
kable(head(sorted_word_frequencies_ham, 20),
      align = c("l", "c", "c"),  # Align columns left, center, center
      width = "20%",  # Width of the table
      booktabs = TRUE,  # Use booktabs style
      longtable = TRUE,  # Allow table to span multiple pages if needed
      escape = FALSE)  # Allow LaTeX commands
```

	x
wed	3849
jalapeno	3652
use	2389
hit	2253
mail	1707
list	1565
can	1504
will	1421

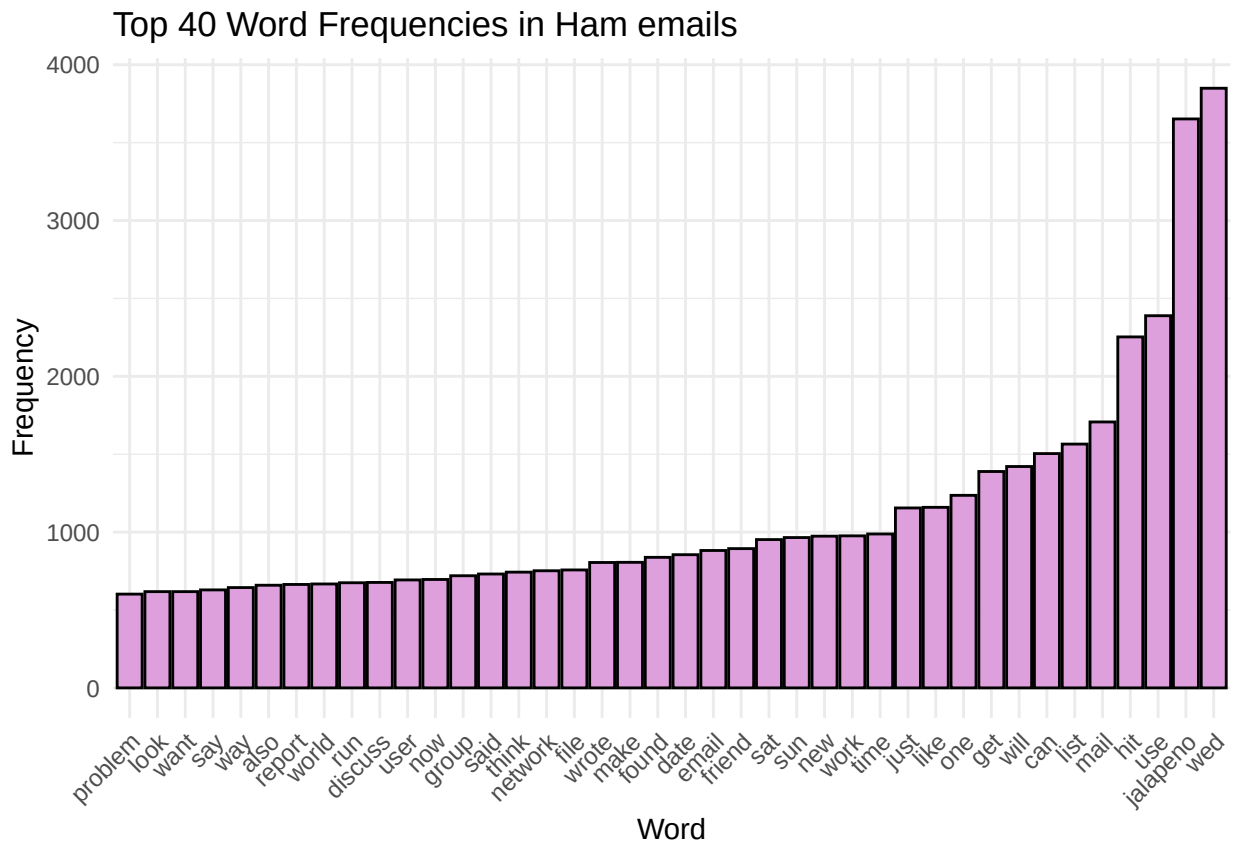
	x
get	1389
one	1236
like	1159
just	1155
time	988
work	976
new	974
sun	965
sat	952
friend	894
email	882
date	855

```r
# Visualize word frequencies using a histogram
hist(sorted_word_frequencies_ham, main = "Ham Word Frequency Distribution", xlab = "Word Frequency", yla
```

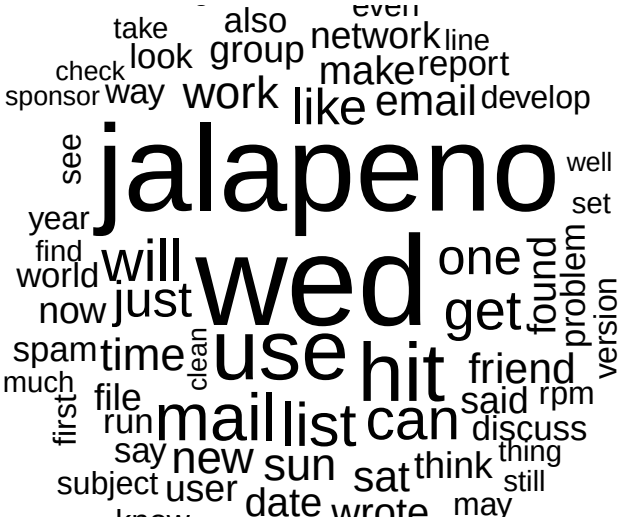



```r
# Convert word frequencies to a dataframe
word_frequencies_df_ham <- data.frame(word = names(word_frequencies_ham), frequency = word_frequencies_

word_frequencies_df_ham <- word_frequencies_df_ham [order(word_frequencies_df_ham$frequency,decreasing =

# Select the top 40 words
top_40_words_ham <- head(word_frequencies_df_ham, 40)
```

```r
# Create a histogram using ggplot2 for the top 50 words
top_40_words_ham |> ggplot(aes(x = reorder(word, frequency), y = frequency)) +
  geom_bar(stat = "identity", fill = "plum", color = "black") +
  labs(title = "Top 40 Word Frequencies in Ham emails",
       x = "Word",
       y = "Frequency") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```r
# Visualize word frequencies using a word cloud
wordcloud(names(word_frequencies_ham), freq = word_frequencies_ham, max.words = 70, random.order = FALSE
```



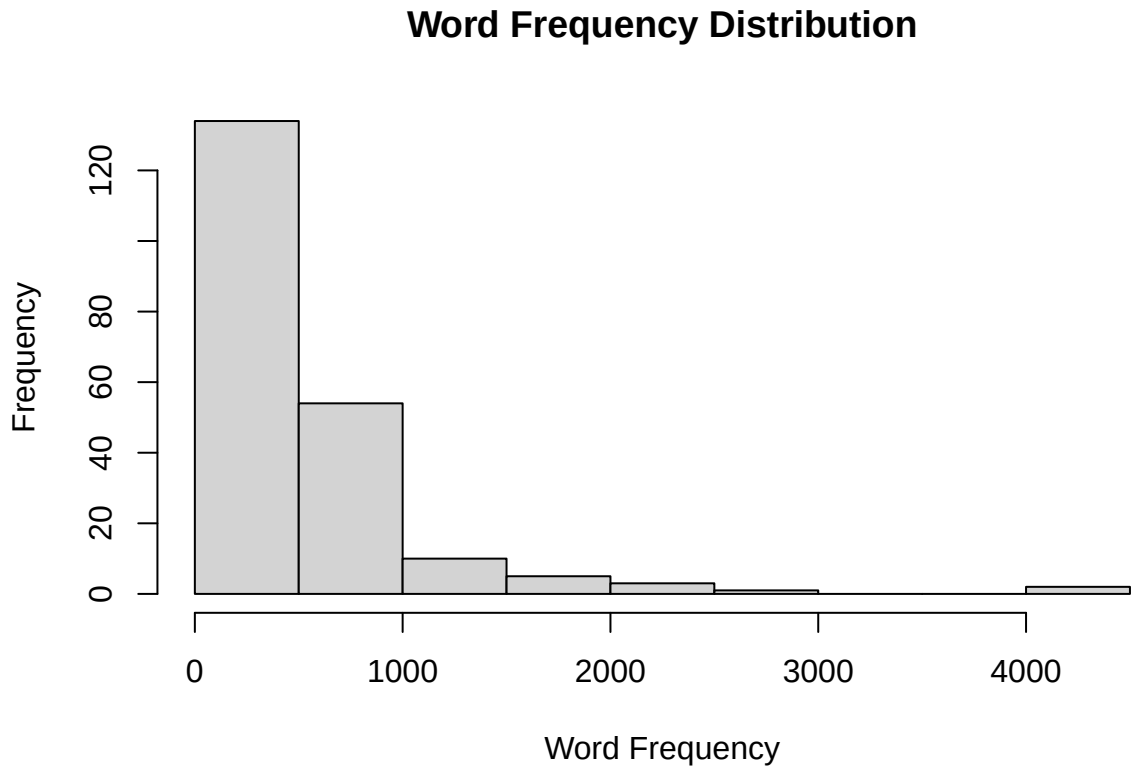
Combined EAD

```
# Calculate the sum of each column (word frequency) excluding the "Label" column
word_frequencies <- colSums(email_matrix[, !colnames(email_matrix) %in% "Label"])

# Sort the word frequencies in descending order
sorted_word_frequencies <- sort(word_frequencies, decreasing = TRUE)

# Print the sorted word frequencies
kable(head(sorted_word_frequencies, 20),
      align = c("l", "c", "c"),  # Align columns left, center, center
      width = "20%",  # Width of the table
      booktabs = TRUE,  # Use booktabs style
      longtable = TRUE,  # Allow table to span multiple pages if needed
      escape = FALSE)  # Allow LaTeX commands
```

	x
wed	4299
jalapeno	4177
use	2652
hit	2269
will	2215
mail	2184
can	1982
list	1940

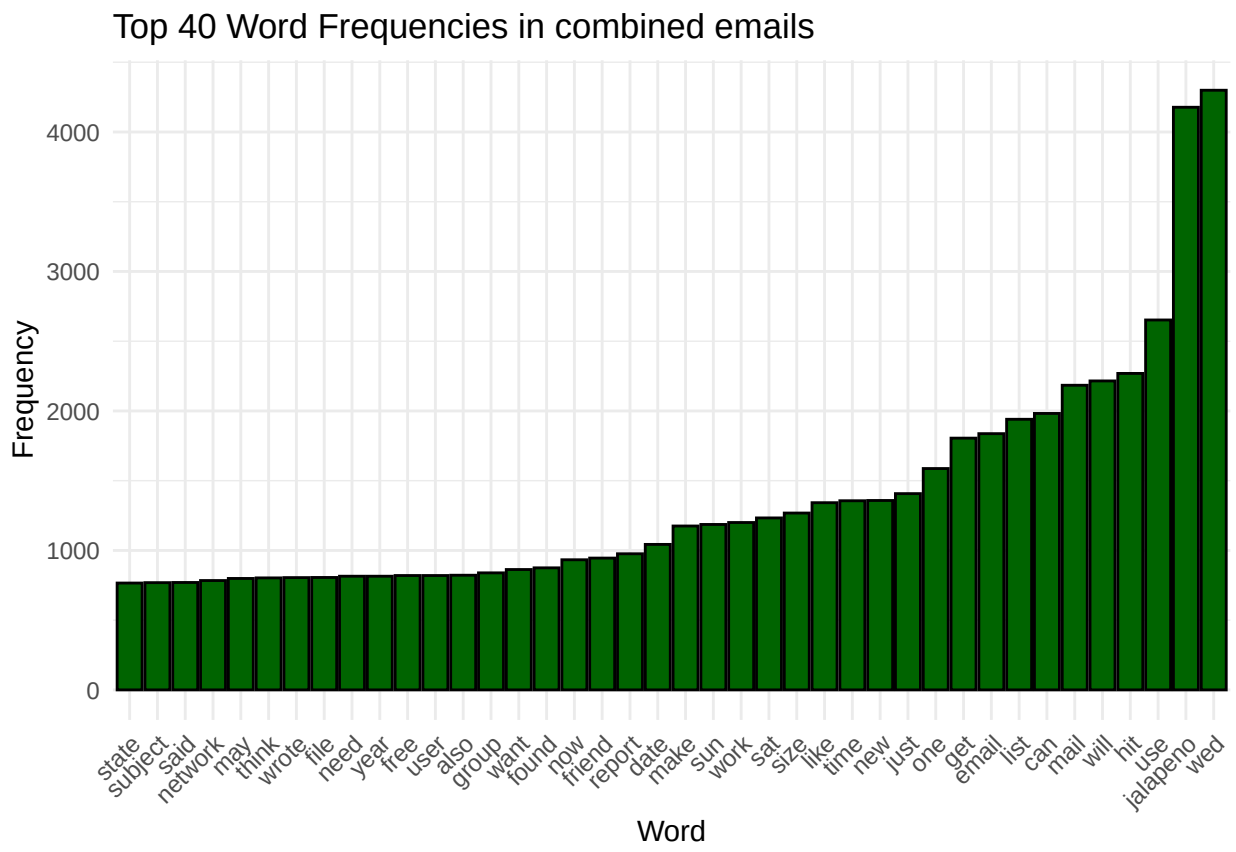
	x
email	1837
get	1805
one	1587
just	1407
new	1358
time	1356
like	1342
size	1268
sat	1233
work	1200
sun	1186
make	1175

```r
# Visualize word frequencies using a histogram
hist(sorted_word_frequencies, main = "Word Frequency Distribution", xlab = "Word Frequency", ylab = "Fr
```




```r
# Convert word frequencies to a dataframe
word_frequencies_df <- data.frame(word = names(word_frequencies), frequency = word_frequencies)

word_frequencies_df <- word_frequencies_df [order(word_frequencies_df$frequency,decreasing = TRUE),]

# Select the top 50 words
top_40_words <- head(word_frequencies_df, 40)
```

```r
# Create a histogram using ggplot2 for the top 50 words
top_40_words |> ggplot(aes(x = reorder(word, frequency), y = frequency)) +
  geom_bar(stat = "identity", fill = "darkgreen", color = "black") +
  labs(title = "Top 40 Word Frequencies in combined emails",
       x = "Word",
       y = "Frequency") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```r
# Visualize word frequencies using a word cloud
wordcloud(names(word_frequencies), freq = word_frequencies, max.words = 70, random.order = FALSE)
```

4-2- Further Data Cleaning:

Looking into the bar-chart into the wordcloud one can also identify someother words that seems to be unimportant in identifying the spam vs ham such as `localhost`, `esmtp`, `jmlocalhost`, `returnpath`, `fetchmailfor`, `imap`, `contenttyp`, `smtp`, and `messageid`. So, I will remove them from the database manually.

```
# Define the list of words to remove manually
manual_word <- c('localhost', 'esmtp', 'jmlocalhost', 'returnpath', 'fetchmailfor', 'imap', 'contenttyp

# Subset the DTM to remove columns containing the manual words
email_matrix_clean <- email_matrix[, !(colnames(email_matrix) %in% manual_word)]
```

4-3- Logestic Regression set up

In this section, we will use the data that has been created for training and testing logistic regression and later for machine learning (ML) to identify spam emails.

Unfortunately, logistic regression did not work well for this data, and the solution did not converge. As a result, no further action was taken.

Additionally, Recursive Partitioning and Regression Trees also seem to work, but upon closer examination, none of the three methods reveal a good solution for SPAM recognition. The failure seems to be rooted in the fact that we have not yet cleaned up the emails to contain only the body text, excluding additional irrelevant data.

```r
set.seed(2014)

email_matrix_clean$Label = as.factor(email_matrix_clean$Label)

spl = sample.split(email_matrix_clean$Label, 0.7)

train = subset(email_matrix_clean, spl == TRUE)
test = subset(email_matrix_clean, spl == FALSE)

#spamLog = glm(Label~., data=train, family="binomial")

#summary(spamLog)

#spamCART = rpart(Label~., data=train, method="class")
#prp(spamCART)
```

4-4: Naive Bayes model (NB)

We can also use **Naive Bayes classifiers**, which are available through the `e1071` package. Although I don't know all the details about the package, I've discovered it and will give it a try.

Naive Bayes classifiers are simple probabilistic classifiers based on Bayes' theorem, with the "naive" assumption of independence between features.

In R, you can use the `naiveBayes()` function from the `e1071` package to train a Naive Bayes classifier.

```r
nb_model <- naiveBayes(Label ~ ., data = train)

summary(nb_model)
```

```
##           Length Class  Mode
## apriori     2    table  numeric
## tables    209    -none- list
## levels      2    -none- character
## isnumeric 209    -none- logical
## call        4    -none- call
```

```r
# Make predictions on the test dataset
predictions_nb <- predict(nb_model, newdata = test)

# Create a confusion matrix
conf_matrix_nb <- confusionMatrix(predictions_nb, test$Label)

# Print the confusion matrix
print(conf_matrix_nb)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction ham spam
##       ham  726   22
##       spam  39  128
```

```
##
##                Accuracy : 0.9333
##                  95% CI : (0.9152, 0.9486)
##     No Information Rate : 0.8361
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7674
##
##  Mcnemar's Test P-Value : 0.0405
##
##             Sensitivity : 0.9490
##             Specificity : 0.8533
##          Pos Pred Value : 0.9706
##          Neg Pred Value : 0.7665
##              Prevalence : 0.8361
##          Detection Rate : 0.7934
##    Detection Prevalence : 0.8175
##       Balanced Accuracy : 0.9012
##
##        'Positive' Class : ham
##
```

```
# Extract accuracy from the confusion matrix
accuracy_nb <- conf_matrix_nb$overall['Accuracy']
print(paste("Accuracy:", accuracy_nb))
```

```
## [1] "Accuracy: 0.933333333333333"
```

4-5- Support Vector Machines (SVM):

Now we're using another method to detect spam emails—**Support Vector Machines (SVM)**. SVM is a popular algorithm for classification tasks and is part of the `e1071` package. Here's how we used the train data to create an SVM model, made predictions on the test data, and evaluated its performance using the confusion matrix. The results show a significant improvement in accuracy compared to Naive Bayes (NB) with accuracy incresed to 96%.

```
svm_model <- svm(Label ~ ., data = train)

summary(svm_model)
```

```
##
## Call:
## svm(formula = Label ~ ., data = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  590
##
```

```
##  ( 231 359 )
##
##
## Number of Classes:  2
##
## Levels:
##  ham spam
```

```r
# Make predictions on the test dataset
predictions_svm <- predict(svm_model, newdata = test)

# Create a confusion matrix
conf_matrix_svm <- confusionMatrix(predictions_svm, test$Label)

# Print the confusion matrix
print(conf_matrix_svm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction ham spam
##       ham  762   28
##       spam   3  122
##
##                Accuracy : 0.9661
##                  95% CI : (0.9523, 0.9769)
##     No Information Rate : 0.8361
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8675
##
##  Mcnemar's Test P-Value : 1.629e-05
##
##             Sensitivity : 0.9961
##             Specificity : 0.8133
##          Pos Pred Value : 0.9646
##          Neg Pred Value : 0.9760
##              Prevalence : 0.8361
##          Detection Rate : 0.8328
##    Detection Prevalence : 0.8634
##       Balanced Accuracy : 0.9047
##
##        'Positive' Class : ham
##
```

```r
# Extract accuracy from the confusion matrix
accuracy_svm <- conf_matrix_svm$overall['Accuracy']
print(paste("SVM Accuracy:", accuracy_svm))
```

```
## [1] "SVM Accuracy: 0.966120218579235"
```

4-6- Deep learning

Another option is **deep learning**, and the **Keras** and **TensorFlow** packages in R can help set this up. While I've learned how to use and configure them, I don't fully understand the inner workings yet. I copied the model structure and layers from the internet, and it utilizes the **ReLU** function, which is particularly suited for binary classification.

Below is the code to set up deep learning in R. When working with this data, it is recommended to normalize it. As a first step, I will normalize the train and test data. There are two standard methods for normalization: **min/max scaling** and **Z-score normalization**. I've chosen to apply both methods by defining a function and using `lapply`.

```
# Z-score normalization function
z_score_normalize <- function(x) {
  (x - mean(x)) / sd(x)
}

# Min-max normalization function
min_max_normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

# Apply min-max normalization to your data
nor_train_MM <- as.data.frame(lapply(train[, -which(names(train) == "Label")], min_max_normalize))
nor_train_MM$Label <- train$Label


# Apply z-score normalization to your data (excluding the label column)
nor_train_ZS <- as.data.frame(lapply(train[, -which(names(train) == "Label")], z_score_normalize))
nor_train_ZS$Label <- train$Label


# Apply min-max normalization to your data
nor_test_MM <- as.data.frame(lapply(test[, -which(names(test) == "Label")], min_max_normalize))
nor_test_MM$Label <- test$Label


# Apply z-score normalization to your data
nor_test_ZS <- as.data.frame(lapply(test[, -which(names(test) == "Label")], z_score_normalize))
nor_test_ZS$Label <-test$Label


  # Define the neural network architecture
model_DP <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = ncol(train) - 1) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

# Compile the model
model_DP %>% compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_adam(),
  metrics = c("accuracy")
)
```

```r
# Train the model
history_MM <- model_DP %>% fit(
  x = as.matrix(nor_train_MM[, -which(names(nor_train_MM) == "Label")]),   # Features
  y = as.numeric(nor_train_MM$Label),  # Labels
  epochs = 10,
  batch_size = 32,
  validation_split = 0.2
)
```

```
## Epoch 1/10
## 54/54 - 2s - loss: 0.2001 - accuracy: 0.7431 - val_loss: 0.1707 - val_accuracy: 1.0000 - 2s/epoch - 3
## Epoch 2/10
## 54/54 - 0s - loss: -1.6915e+00 - accuracy: 0.7946 - val_loss: 1.6410e-04 - val_accuracy: 1.0000 - 243
## Epoch 3/10
## 54/54 - 0s - loss: -7.2464e+00 - accuracy: 0.7946 - val_loss: 2.0750e-13 - val_accuracy: 1.0000 - 327
## Epoch 4/10
## 54/54 - 0s - loss: -2.1100e+01 - accuracy: 0.7946 - val_loss: 4.6988e-33 - val_accuracy: 1.0000 - 269
## Epoch 5/10
## 54/54 - 0s - loss: -4.8763e+01 - accuracy: 0.7946 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - 287
## Epoch 6/10
## 54/54 - 0s - loss: -9.5055e+01 - accuracy: 0.7946 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - 214
## Epoch 7/10
## 54/54 - 0s - loss: -1.6586e+02 - accuracy: 0.7946 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - 207
## Epoch 8/10
## 54/54 - 0s - loss: -2.6298e+02 - accuracy: 0.7946 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - 267
## Epoch 9/10
## 54/54 - 0s - loss: -3.9403e+02 - accuracy: 0.7946 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - 215
## Epoch 10/10
## 54/54 - 0s - loss: -5.6346e+02 - accuracy: 0.7946 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - 206
```

```r
# Evaluate the model
evaluation_DP_MM <- model_DP %>% evaluate(
  x = as.matrix(nor_test_MM[, -which(names(nor_test_MM) == "Label")]),   # Features
  y = as.numeric(nor_test_MM$Label)   # Labels
)
```

```
## 29/29 - 0s - loss: -6.1008e+02 - accuracy: 0.8361 - 97ms/epoch - 3ms/step
```

```r
summary (evaluation_DP_MM)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -610.0783 -457.3497 -304.6211 -304.6211 -151.8925    0.8361
```

```r
# Make predictions on the test dataset
predictions_DP_MM <- model_DP %>% predict(
  x = as.matrix(nor_test_MM[, -which(names(nor_test_MM) == "Label")])  # Features
)
```

```
## 29/29 - 0s - 204ms/epoch - 7ms/step
```

```r
# Convert predicted probabilities or classes to labels
predicted_labels_DP_MM <- ifelse(predictions_DP_MM > 0.5, "spam", "ham")

predicted_df_DP_MM <- data.frame(Label = factor(predicted_labels_DP_MM))

# Create a confusion matrix
conf_matrix <- table(predicted_df_DP_MM$Label, nor_test_MM$Label)

# Print the confusion matrix
print(conf_matrix)
```

```
##
##        ham spam
##   spam 765  150
```

```r
#conf_matrix_DP_MM <- confusionMatrix(predicted_labels_DP_MM, nor_test_MM$Label)

# Print the confusion matrix
#print(conf_matrix_DP_MM)

# Extract accuracy from the confusion matrix
#accuracy_DP_MM <- conf_matrix_DP_MM$overall['Accuracy']

#print(paste("DP for MM normalized data Accuracy:", accuracy_sum))
```

4-6- Random Forest

Another method I used for spam evalaution is Random Forest, that is among the very famous machine learning model to be used for classification specifically for binary.

The method presented in reference [1] is used here for this modeling. here is code is used.

```r
# Train the random forest model
rf_model <- randomForest(Label ~ ., data = train)

# Print summary of the random forest model
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = Label ~ ., data = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 14
##
##         OOB estimate of  error rate: 0.98%
## Confusion matrix:
##       ham spam class.error
## ham  1782    4 0.002239642
## spam   17  334 0.048433048
```

```r
# Make predictions on the test dataset
predictions_rf <- predict(rf_model, newdata = test)

# Create a confusion matrix
conf_matrix_rf <- confusionMatrix(predictions_rf, test$Label)

# Print the confusion matrix
print(conf_matrix_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction ham spam
##       ham  761    8
##       spam   4  142
##
##                Accuracy : 0.9869
##                  95% CI : (0.9772, 0.9932)
##     No Information Rate : 0.8361
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9516
##
##  Mcnemar's Test P-Value : 0.3865
##
##             Sensitivity : 0.9948
##             Specificity : 0.9467
##          Pos Pred Value : 0.9896
##          Neg Pred Value : 0.9726
##              Prevalence : 0.8361
##          Detection Rate : 0.8317
##    Detection Prevalence : 0.8404
##       Balanced Accuracy : 0.9707
##
##        'Positive' Class : ham
##
```

```r
# Extract accuracy from the confusion matrix
accuracy_rf <- conf_matrix_rf$overall['Accuracy']
print(paste("Random Forest Accuracy:", accuracy_rf))
```

```
## [1] "Random Forest Accuracy: 0.986885245901639"
```

5- Model Evaluation

I evaluated the performance of different models, including logistic regression, Naive Bayes (NB), Support Vector Machine (SVM), Random Forest, and Deep Learning. Using metrics such as accuracy, precision, recall, and F1-score, it became evident that Random Forest performed the best, achieving the highest accuracy among the evaluated models.

For a spam/ham classification problem, four methods were recommended, and they successfully predicted spam with accuracies ranging from 90% to 98.6%. Despite not achieving significant success in cleaning the

SPAM and HAM emails during this process, we still achieved remarkable effectiveness with the Random Forest method, reaching 98.6% accuracy.

I also attempted Deep Learning, but unfortunately, I couldn't complete it due to time constraints and not knowing enough about the topic.

All in all, this gives me hope that with better data cleaning, we may further improve the model's performance, possibly reaching 99.9%.

All in all, it was an interesting and challenging problem to tackle, requiring extensive research. I utilized resources from GitHub and R-Pubs and contributed my portion of the code where needed.

References:

[1] https://github.com/Prakash-Khatri/Text_spam_ham_detection

[2] RPubs - Spam and Ham Detection of Email