Data Mining on Spam Email Detection

INFOSYS 722

Iteration 4

Di Tian

6825548

Dtia298

https://github.com/DiTian1993/infosys722-iteration-4

(I acknowledge that the submitted work is my own original work in accordance with the University of Auckland guidelines and policies on academic integrity and copyright.I also acknowledge that I have appropriate permission to use the data that I have utilised in this project.)

Business Understanding

1. Situation understanding

Email spam, which is often referred to as junk-mail, is a creation of the fast-booming modern information technology. Email span has grown sharply from 1990's and its' form and structure are constantly evolving. It is mainly used for commercial advertising purpose and it is extremely cost-effective in sender's perspective. For instance, if only a fraction of the recipients of the spam email purchased the product or service the spam advertises, the sender is making money because the cost of sending spam email can be estimated to nothing. Spam email is a cheap and efficient advertising method because spammers are getting recipients' physical addresses or IP address from open-sourced websites such as search engines like Google, online shopping sites like Amazon or public review website like Zomato. What spammers basically do is using crawling software to automatically get users' personal information from the websites: users' comment on restaurants or reviews on products they purchased. In general, at the first glance, spam Email does not seem to be harmful and is just an advertising strategy, but we need to take a step deeper to discover the potential danger it brings.

To begin with, spam email might be deceptive and fraud. Senders could use false identities to set up various online accounts as their fraud base, and they use stolen credit card numbers to pay for these accounts. The Simple Mail Transfer Protocol (SMTP) has no authentication by default, so senders can cover their true IP address and pretend the email was originally coming from trustworthy addresses. In addition, span emails are now increasingly sent from computers infected by computer viruses. Spammers are now using Trojan Horse virus as annex in the spam emails. When victims click on the link the spam email provides, their local device instantly automatically download the virus, which help spammers to gain

access to victims' device. It is estimated that in the recent decade, nearly 90% of emails are actually spam emails.

1.1 Identify the objectives of the situation

Spam email is a hot topic now. The public views it from quite different perspectives. Some believe that the creation of spam email is just a side effect of the information booming era, and there is no need to exaggerate. On the other hand, others believe spam email could have the potential to endanger the internet environment and make it become a grey area where unlawful people can do all sorts of criminal activities without being punished. Spamming is also politically debated in many countries, international organisations and academia. Only some countries published specific law against spamming, while the law of other countries remain blank in this area. As spamming technology evolving in a fast path, it was estimated by IPwarmup.com (2019) that over 90% of the total emails worldwide are considered as spam emails after studying over 100 million mailboxes. The cost of spam is also enormous. A 2004 survey estimated that lost productivity costs Internet users in the United States $21.58 billion annually, while another reported the cost at $17 billion, up from $11 billion in 2003. In 2004, the worldwide productivity cost of spam has been estimated to be $50 billion in 2005. Even though there are various kind of anti-virus and IP-protectors out there on the market, it is still not enough effort in terms of protecting the public from the harm of spam emails. If there is a way to train a machine learning model to predict the occurrence of certain words or words combinations could potentially identify whether or not an email is a spam, it could be used as the core algorithm in the later anti-spam software. In this way, we can reduce the risk of being fraud by spam emails and create a cleaner cyber environment. To do so, we need to find datasets that contain key words that often used in spam emails, and also a class attribute indicating whether a instance is actually spam or not. In addition to be predictable, other related questions can also be solved: what key factor contribute to a word that made it become "spammable"; what combination of words are often not "spammable" but when they get together, they become suspicious, and which kind of wording structures we need to avoid using to avoid being auto detected as spam. Therefore, by mining the above data, we can get the following goals:

- Analyse the featured words and chars that are most reverent in spam email detection, we can give priority to these features when constructing prediction model.
- Build a model from the dataset of commonly used spam words and let the model make the model to successfully identify spam emails if an email contains certain words or words combinations.

The target can be evaluated as achieved if the below conditions are met:

- Find three of the most commonly used word sequence by spammers.
- The prediction model has an average 75 percent accuracy, the confusion matrix contains more accurately identified spam emails than wrongly classified normal emails.

1.2 Assess the situation

1. Personnel

Online spamming is studied by countless scholars and researchers. Many of them came up with brilliant ideas and theories about the topic itself and ways of defeat spammers. For our specific method, we can consult a natural language expert and database expert to work alongside to analyse and process the data. After all, language is another deep topic that involves studies on human behaviour. When we construct words sequence, we can consult the expert on why it might be the case that this particular sequence would contribute to spam detection. We can relate our study with psychology to do some further research. This research project can be used as preliminary background knowledge for other personnel to further expand the scope of this topic.

2. Data Sources:

Spam email detection has always been a popular topic in computer science area. There are various types of spam related data available publicly: there are datasets that contains spam email screenshot pictures for deep learning research purpose; there are spamming content related datasets for data mining and crawling. I can access to public databases such as Kaggle and UCI Machine Learning to get well-prepared dataset for my research purpose.

3. Requirements

Following points were considered and evaluated for requirements:

- The dataset will be chosen on open source databases (Kaggle and UCI), all the datasets are publicly available, so no legal restrictions presented. As for security, my project is for research purpose so there is no security restrictions.
- Personnel aligned on the project will be me along, so if I stick on my schedule, no personnel related issue will occur.
- This research is for personal study purpose for now, so there is no deployment schedule now. If the project operates on the expected level, then it might be used for future research.

4. Assumptions

- There is no economic factor that might affect the project, because all the resources needed are open sourced and publicly accessible.
- There are some data quality assumptions. First, I assume that the dataset I found in the open source databases would have a fairly good quality (less NA or missing values, data well-organised). Also, I assume the model we build can identify underlying relationships between features.
- Since this project is for personal study purpose at the moment, I am expected to visualise the result as well as understanding the statistics the model creates.

5. Constraints

- There is no password required to access the data.
- The dataset I found is fully licensed, as long as I do not use it for commercial purpose, I verified all legal constraints.
- There is no financial constraints, the reason is mentioned above.

6. Risks and Contingencies

Risks include:

- I might not get the perfect dataset for my project. After all, I will access open source databases such as Kaggle, where the reliability of the data is not guaranteed. The dataset itself may not be rich enough or is poorly structured with large number of noisey data.

- The prediction model built might perform extremely poor in terms of accuracy. If that happens, the prediction model itself would become meaningless.

contingencies:

- To overcome the data quality risk, I will need to search in databases and look for the best dataset among same topics. I might ask for authors of datasets' permission to combine multiple datasets into one if necessary.
- To overcome the poorly perform model risk, I will study the model itself from different angles and use different statistical approaches to modify it.

1.3 Determine data mining goals

The data mining goals of this study are:
Use the given dataset to find out patterns related to spam email. I will conduct data mining process to build a model to identify potential spam emails through analysing frequent words behaviors as well as the total length of an email. Once our model found some correlations between frequent words and the distribution of spam emails, I will based on our findings to interpret the result: what kind of frequent words contribute the most in identifying potential spam attack and how often do they occur in a spam email. Through the analysis, we can analysis underlying patterns in the wording structure of an spam email. In the future, we can conduct more research on this topic with more samples to further refine our study.

1.3.1 Data mining success criteria

In this study, basic machine learning and data mining methods will be used to predict potential spam emails. In terms of performance, I will record running time for different algorithms I select, and measure their compatibility while operating (test algorithm on if they can successfully run on given dataset). In terms of assessing the accuracy, I will document all the findings and compare them along with the measurement of accuracy.

1.3.2 Success benchmark

Success is evaluated based on whether the chosen algorithm compile on given dataset, and get highaccuracy, the efficiency of the model should also be taken into account. In this case, we can get a reasonably well performed data mining model on predicting potential spam emails.

1.3.3 Subjective measurements and arbiter of success

In this study, the measurement of project success is to build a data mining model which has high efficiency in terms of running, and to get a overall more than 80% accuracy while predicting.

1.3.4 Success deployment of model results a part of data mining success

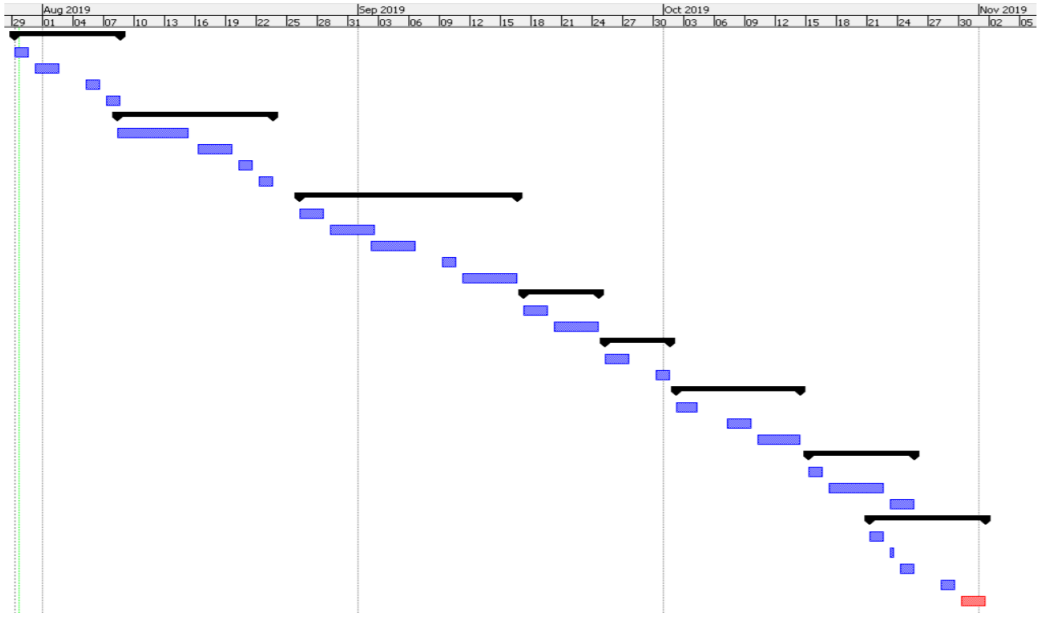
The modelling results would be deployed on a software which take our model as an indicator, while the software operates, it take the content of an email as input, and then use our model as a predictor to evaluate the content, and then send a feedback to end user indicating if the given email is spam or not. Ultimately, the motives behind this is to help creating a better and safer online environment by helping internet surfers filtering spam emails and keep them safe from spam attack.

1.4 Produce a project plan

Phase	Time	Resources	Risk
Situation understanding	0.5 week	All analysis	Situation change
Data understanding	0.5 week	All analysis	Data quality issue
Data preparation	0.5 week	Database analyst & Natural language experts	Data quality issue
Modelling	0.5 week	Database analyst & Natural language experts	Model poorly constructed
Evaluation	0.5 week	All analysis	Poorly performed model result in inability in implementing results
Evaluate	0.5 week	Database analyst & Natural language experts	Poorly performed model result in inability in implementing results

This is just a preliminary idea of the whole structure of this project. Now I construct a specific timeline for this study:

Task name	Start date	uratio	Finish date
1 Business Understanding	**7/29/2020**	**9**	**8/8/2020**
1.1 Identify business objectives	7/29/2020	2	7/30/2020
1.2 Assess the situation	7/31/2020	3	8/2/2020
1.3 Determine data mining goal	8/3/2020	2	8/6/2020
1.4 Peoduce a project plan	8/7/2020	2	8/8/2020
2 Data Understanding	**8/8/2020**	**11.75**	**8/23/2020**
2.1 Collect initial data	8/8/2020	5	8/15/2020
2.2 Describe data	8/16/2020	2	8/19/2020
2.3 Explore data	8/20/2020	2	8/21/2020
2.4 Verify data	8/22/2020	2	8/23/2020
3 Data Preparation	**8/26/2020**	**16**	**9/16/2020**
3.1 Select data	8/26/2020	3	8/28/2020
3.2 Clean data	8/29/2020	3	9/2/2020
3.3 Construct data	9/2/2020	5	9/6/2020
3.4 Integrate various data source	9/7/2020	2	9/10/2020
3.5 Dormat data as required	9/11/2020	4	9/16/2020
4 Data Transformation	**9/17/2020**	**6**	**9/24/2020**
4.1 Reduce data	9/17/2020	3	9/19/2020
4.2 Porject data	9/20/2020	3	9/24/2020
5 Data-mining method selection	**9/25/2020**	**5**	**10/1/2020**
5.1 Match and discuss objectives of data-mining	9/25/2020	3	9/27/2020
5.2 Select the appropriate method	9/28/2020	2	10/1/2020
6 Data-mining method selection	**10/2/2020**	**8.5**	**10/14/2020**
6.1 Conduct exploratory analysis and discuss	10/2/2020	2.25	10/4/2020
6.2 Select data-mining algorithm	10/5/2020	3	10/9/2020
6.3 Build model(s) and choose parameter setting	10/10/2020	2.5	10/14/2020
7 Data-mining	**10/15/2020**	**9**	**10/25/2020**
7.1 Create and justify test designs	10/15/2020	2	10/16/2020
7.2 Conduct data-mining	10/17/2020	4	10/22/2020
7.3 Search for patterns	10/23/2020	3	10/25/2020
8 Interpretation	**10/21/2020**	**10**	**11/1/2020**
8.1 Study and discuss the mined patterns	10/21/2020	2	10/23/2020
8.2 Visualize the data, result, models and patterns	10/23/2020	1	10/23/2020
8.3 Interprete the result, models and patterns	10/24/2020	2	10/25/2020
8.4 Assess and evaluate result, model, and patterns	10/26/2020	2	10/29/2020
8.5 Iterate prior steps as required	10/30/2020	3	11/1/2020



2. Data Understanding

2.1 Collect initial data

The spam email frequently used word matrix dataset used in this project is from:
Kaggle: it can be downloaded from the following url:
https://www.kaggle.com/somesh24/spambase
Kaggle account is needed for download. Author information is included in the original webpage.

2.2 Describe the data

There are many related dataset out there online, however, the chosen dataset was collected because it is a predefined word-matrix type of dataset, which means we do not need to perform text-mining before hand, in terms of efficiency. We will see how each feature is correlated with the class attribute.

- This dataset contains non-spam email contents that were retrieved from real work and personal emails, the spam email contents were retrieved from spam emails. The dataset contains 4602 instances and 57 attributes and 1 class attribute that indicate whether this instance is spam or not.
- Value types:
  - 48 continuous real [0,100] attributes of type. for instance, wordfreqWORD = percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A “word” in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
  - 6 continuous real [0,100] attributes of type char. For instance, freqCHAR = percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurrences) / total characters in e-mail.
  - 1 continuous real [1,…] attribute of type capitalrunlength_average = average length of uninterrupted sequences of capital letters.
  - 1 continuous integer [1,…] attribute of type capitalrunlength_longest = length of longest uninterrupted sequence of capital letters.
  - 1 continuous integer [1,…] attribute of type capitalrunlength_total = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail.

- 1 nominal {0,1} class attribute of type spam = denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.

- Coding schemes:

For the 48 continuous real [0,100] attributes wrodfreq and 6 continuous real [0,100] of type char, each value represents the proportion that this word occurred in this email. Value in capitalrunlength indicates the length of capital letters in the email. For the class attribute, 1 = spam email and 0 = real email.

```
import findspark
findspark.init('/home/ubuntu/spark-2.1.1-bin-hadoop2.7')
import pyspark
import numpy as np
import pandas as pd
from pyspark.sql import SparkSession
# from pyspark.sql import approxQuantile
spark = SparkSession.builder.appName('operations').getOrCreate()
```

```
dfs = spark.read.csv('dataset/spam.csv', inferSchema=True, header=True)
```

General setup for pyspark

```
dfs.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- word_freq_make: double (nullable = true)
 |-- word_freq_address: double (nullable = true)
 |-- word_freq_all: double (nullable = true)
 |-- word_freq_3d: double (nullable = true)
 |-- word_freq_our: double (nullable = true)
 |-- word_freq_over: double (nullable = true)
 |-- word_freq_remove: double (nullable = true)
 |-- word_freq_internet: double (nullable = true)
 |-- word_freq_order: double (nullable = true)
 |-- word_freq_mail: double (nullable = true)
 |-- word_freq_receive: double (nullable = true)
 |-- word_freq_will: double (nullable = true)
 |-- word_freq_people: double (nullable = true)
 |-- word_freq_report: double (nullable = true)
 |-- word_freq_addresses: double (nullable = true)
 |-- word_freq_free: double (nullable = true)
 |-- word_freq_business: double (nullable = true)
 |-- word_freq_email: double (nullable = true)
 |-- word_freq_you: double (nullable = true)
 |-- word_freq_credit: double (nullable = true)
 |-- word_freq_your: double (nullable = true)
 |-- word_freq_font: double (nullable = true)
 |-- word_freq_000: double (nullable = true)
 |-- word_freq_money: double (nullable = true)
 |-- word_freq_hp: double (nullable = true)
 |-- word_freq_hpl: double (nullable = true)
 |-- word_freq_george: double (nullable = true)
 |-- word_freq_650: double (nullable = true)
 |-- word_freq_lab: double (nullable = true)
 |-- word_freq_labs: double (nullable = true)
 |-- word_freq_telnet: double (nullable = true)
```

```
|-- word_freq_technology: double (nullable = true)
|-- word_freq_1999: double (nullable = true)
|-- word_freq_parts: double (nullable = true)
|-- word_freq_pm: double (nullable = true)
|-- word_freq_direct: double (nullable = true)
|-- word_freq_cs: double (nullable = true)
|-- word_freq_meeting: double (nullable = true)
|-- word_freq_original: double (nullable = true)
|-- word_freq_project: double (nullable = true)
|-- word_freq_re: double (nullable = true)
|-- word_freq_edu: double (nullable = true)
|-- word_freq_table: double (nullable = true)
|-- word_freq_conference: double (nullable = true)
|-- char_freq_%3B: double (nullable = true)
|-- char_freq_%28: double (nullable = true)
|-- char_freq_%5B: double (nullable = true)
|-- char_freq_%21: double (nullable = true)
|-- char_freq_%24: double (nullable = true)
|-- char_freq_%23: double (nullable = true)
|-- capital_run_length_average: double (nullable = true
|-- capital_run_length_longest: integer (nullable = tru
|-- capital_run_length_total: integer (nullable = true)
|-- word_trial: string (nullable = true)
|-- word_test: string (nullable = true)
|-- class: integer (nullable = true)
```

Figure 1 Check all data type

It is visible that all the variable types belong to either Float or integer along with two string columns. This result is achieved by using Pyspark printscheme.

We used a build-in function in pandas to get a rough idea of the dataset. See the figure below, we can visualize some potential challenges such as data outliers. We will conduct more investigation in section 2.3.

summary	capital_run_length_total	capital_run_length_longest	capital_run_length_average
count	4601	4601	4601
mean	283.28928493805694	52.17278852423386	5.191515105411881
stddev	606.3478507248461	194.89130952646354	31.72944874021063
min	1	1	1.0
max	15841	9989	1102.5

Figure indicate huge gap

As shown in above Figure, for a particular feature “capital_run_length_total’, the smallest value for this feature is 1, while the largest value is 15841, which indicate potential data quality problem. We need to perform analysis on this area later.

```
dfs.describe('capital_run_length_total', 'capital_run_length_longest', 'capital_run_length_average').show()
```

Figure print largest and smallest 2 values

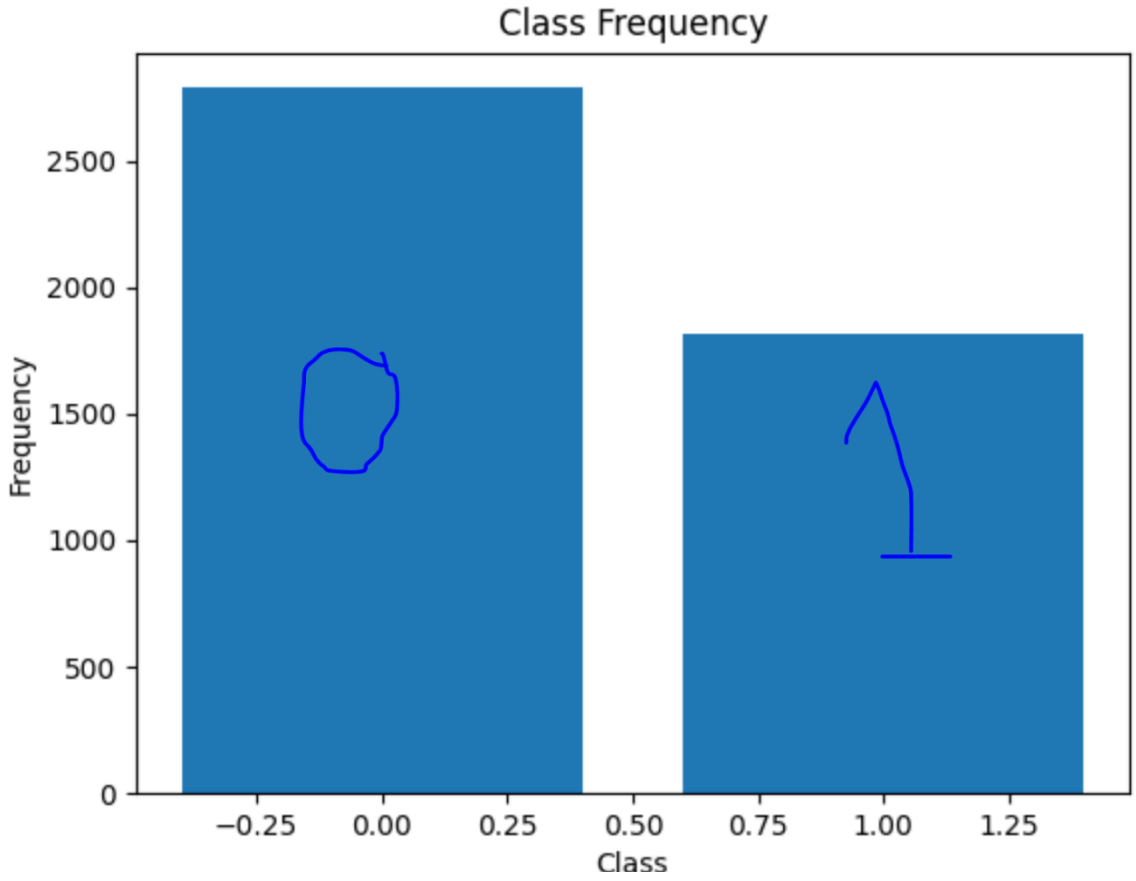
2.3 Explore the data

In my dataset, in total 59 features (exclude the id column) are presented, and they are mainly proportion based continuous variables, so I would imagine the histogram for most of the features would be highly skew to left. I will make some statistical assumptions and use Python to test them.
Assumptions:

- I assume that features that have the most impact on prediction would be all the frequent words and the three capital letter lengths related variables. Some of the illegal chars in the dataset would contribute less in terms of model building and predicting. I will perform feature selection to discover underlying relationships between features.
- Statistical Analysis:
  Outlier and extreme analysis
  I performed outlier and extreme analysis first. Because it is important to address this issue before we start any investigation on the dataset. Outliers and extremes are common in data mining and analysis. Outliers and extreme values are unnecessarily bad when it comes about model building, however, there is a huge trap between outliers and extreme values than noisy data. Outliers and extreme values that we think are sometimes noisy data. Noisy data is data that is corrupted, or distorted, or has a low signal-to-noise ratio. It is often time referred to as “man-made” mistakes when recording the data: wrong typing, mistyping or deliberately input misleading data. Noisy data harms data analysis in various ways: first, it affect the model building process, when we construct our machine learning model, the model itself might view noisy data as normal input and use them as key classifiers while making prediction. Second, noisy data cannot be easily understood by neither the user nor the system, therefore compromise the overall dataset quality. We need to perform this analysis first to weed out some potential noisy data.

  Feature selection
  Another assumption is that I assume some features will have greater prediction power over other features.

First, I need to visualise the class distribution for this dataset. I created a pandas dataframe and used the matlpotlib package to draw the class attribute distribution.



The distribution of class attribute is shown on above graph. Around 60.6% of total instances are labeled as normal emails, where 39.4% are labeled as spam emails. This dataset is imbalanced, but is in an acceptable range. However, we still need to take this factor into account when constructing the research model.

```
unique, counts = np.unique(y, return_counts=True)
plt.bar(unique, counts)
plt.title('Class Frequency')
plt.xlabel('Class')
plt.ylabel('Frequency')
```

Figure plot the class attribute

By visualize the features, I chose four of them to conduct further investigation. They are: ‘Word_freq_your’, ‘Word_freq_000’, ‘Word_freq_remove’ and ‘char_freq_%24’.

As for the first four key important variables, it is not difficult to visualize that for about all variables, most frequencies stay in the top side of the distribution barplot.

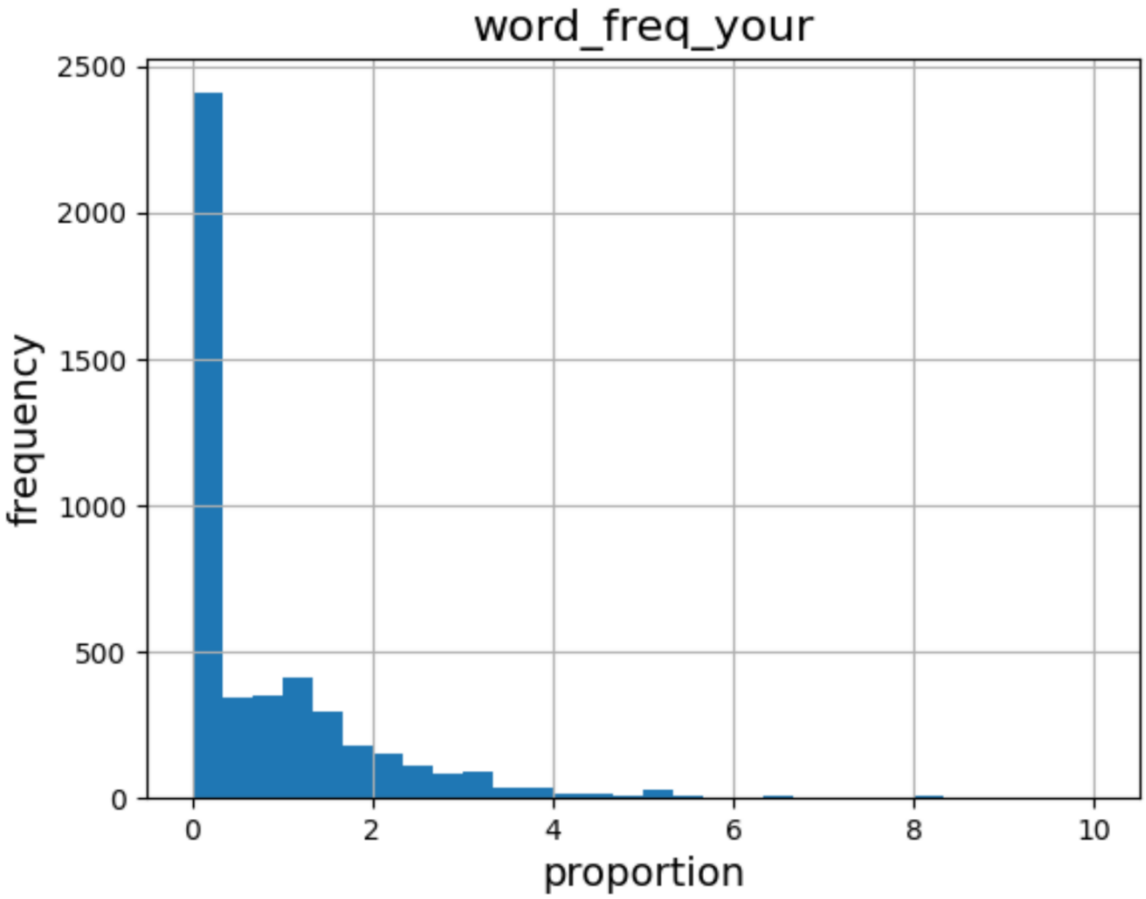


Figure frequency distribution for word_your

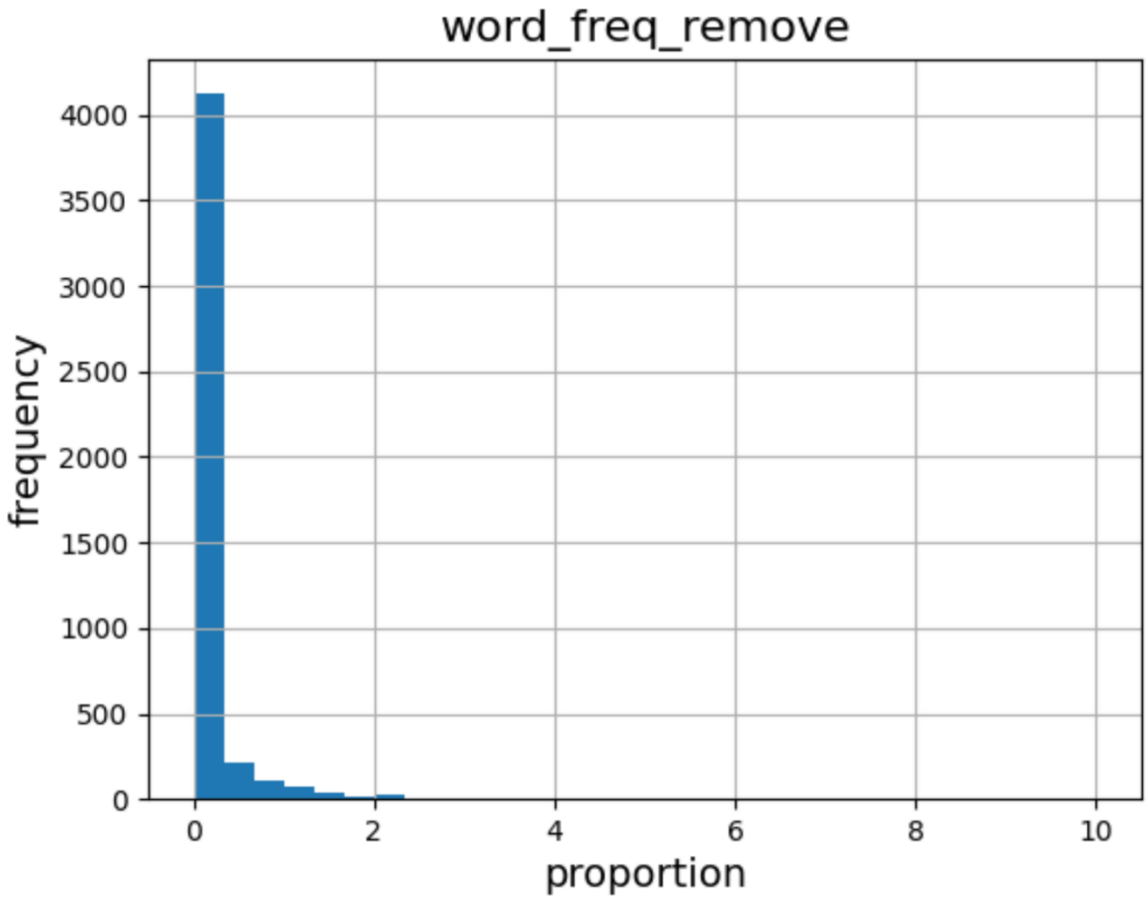


Figure frequency distribution for word_remove

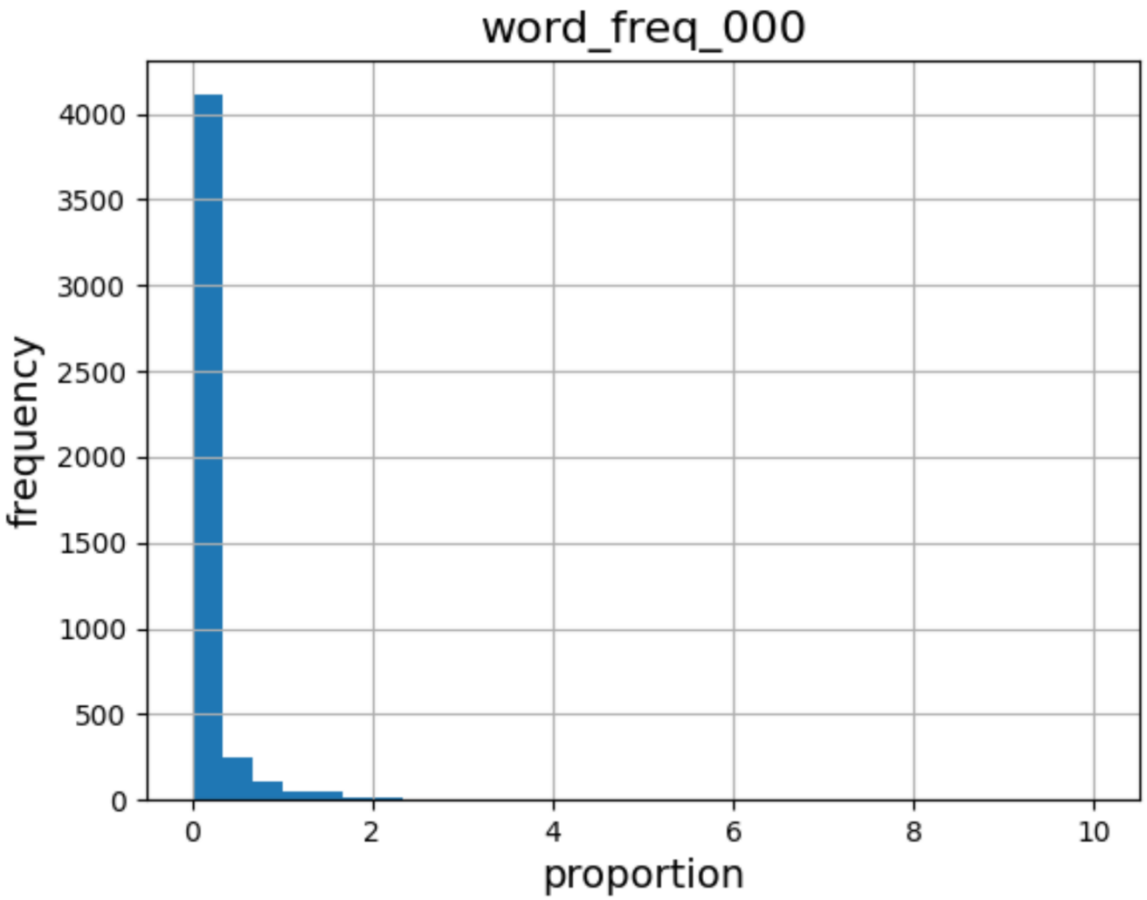


Figure frequency distribution for word_000

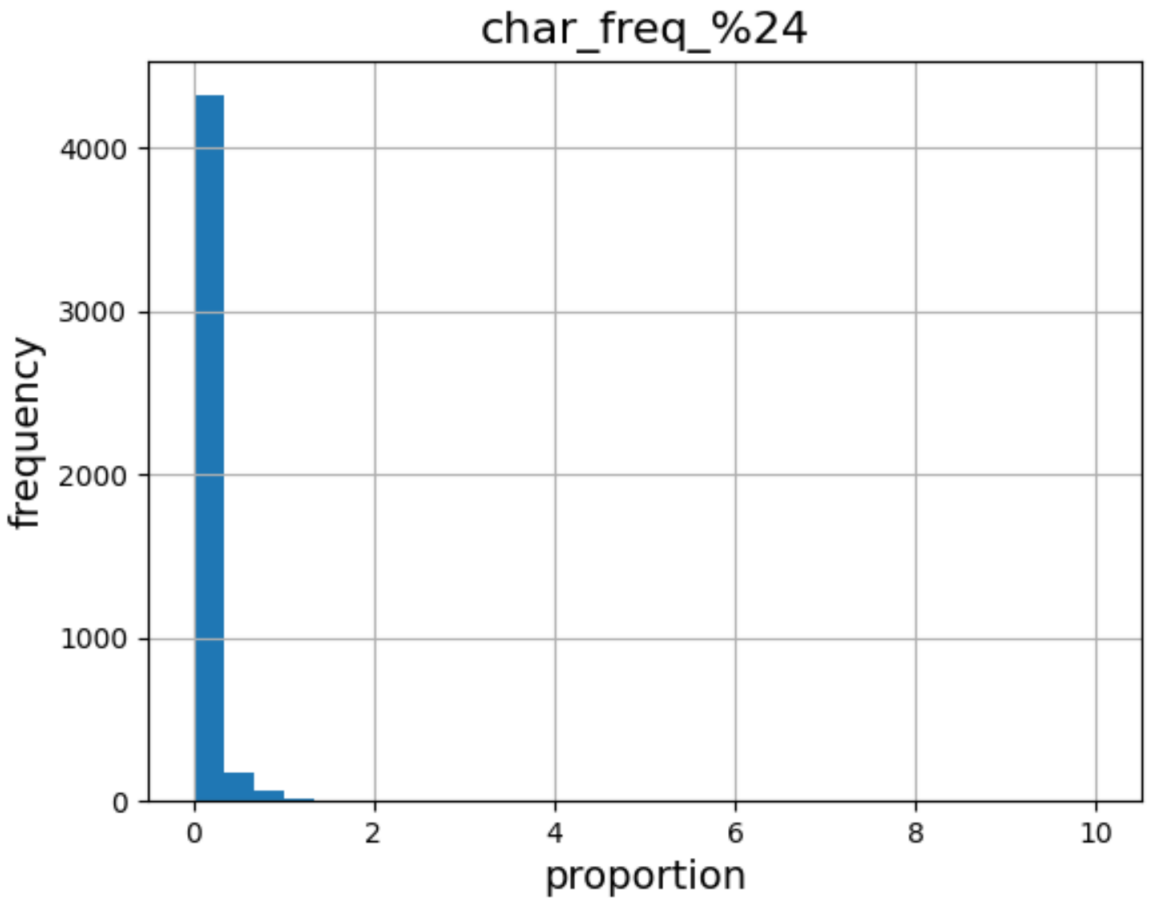


Figure frequency distribution for char_%24

All of the plots above are implemented by using matplotlib package:

```
vec = ["word_freq_your", "word_freq_remove", "word_freq_000", "char_freq_%24"]

for i in vec:
    df[i].hist(bins = 30, range = [0, 10])
    plt.title(i, fontsize = 16)
    plt.xlabel('proportion', fontsize = 14)
    plt.ylabel('frequency', fontsize = 14)
    plt.show()
```

As we can see from the frequency distributions of the four features they tend to share the same skewness. It is common that the frequency distribution for these specific words are quite similar. Also, for these frequency distributions, I noticed that there are some potential extreme value or outliers among distributions, at this point we cannot decide if these values will potentially harm our model's prediction. I will log this situation for now and come back if necessary. Apart from direct frequency distribution, I decided to make some barplots with the top four most important features against the class attribute:

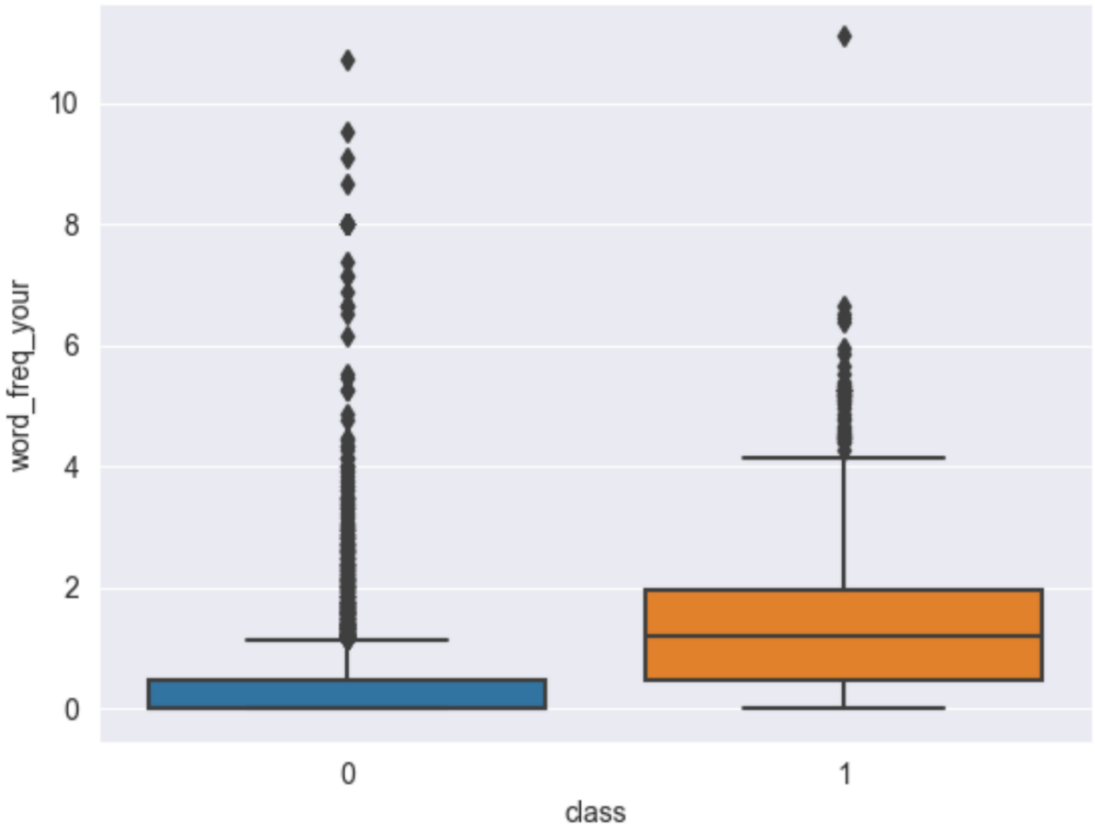


Figure frequency vs class distribution for word_your

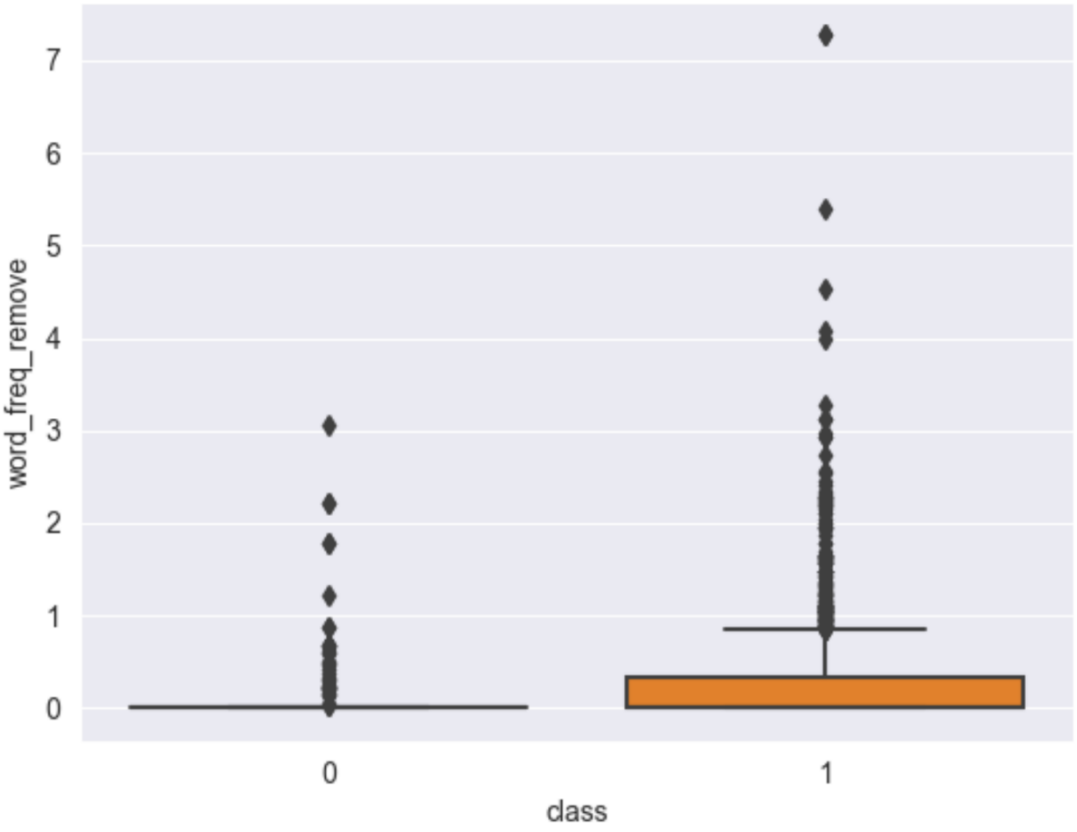


Figure frequency vs class distribution for word_remove

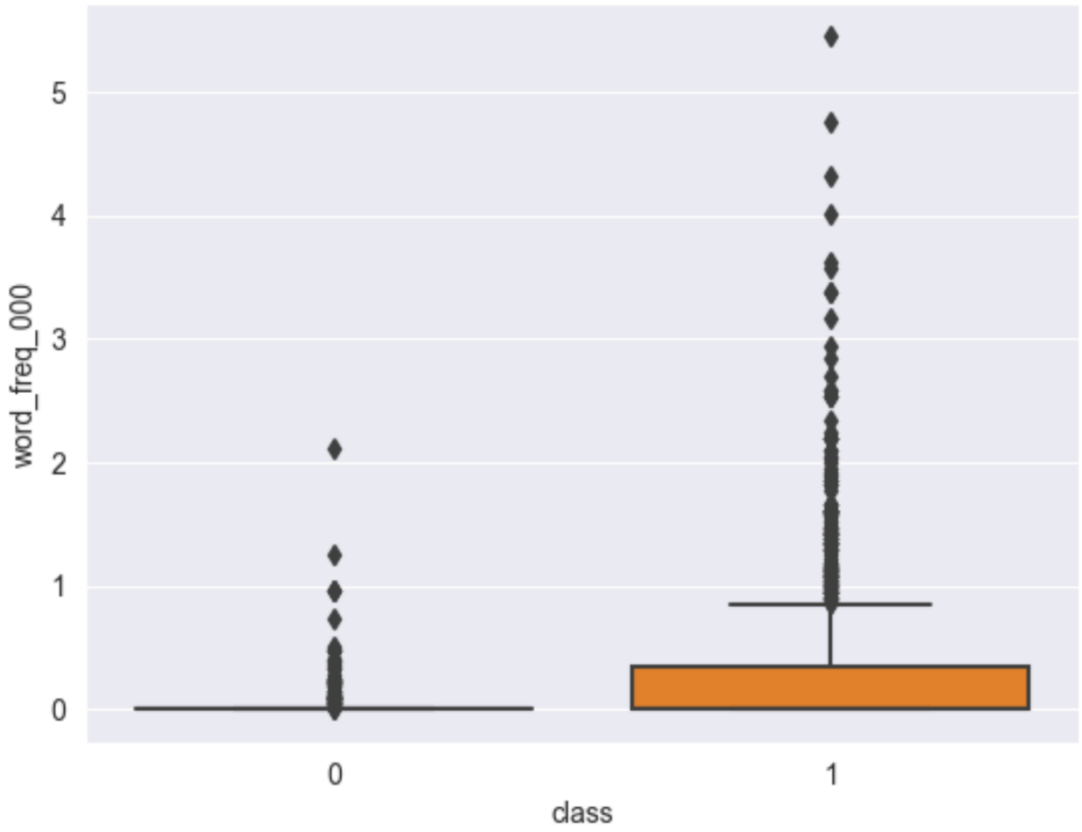


Figure frequency vs class distribution for word_000

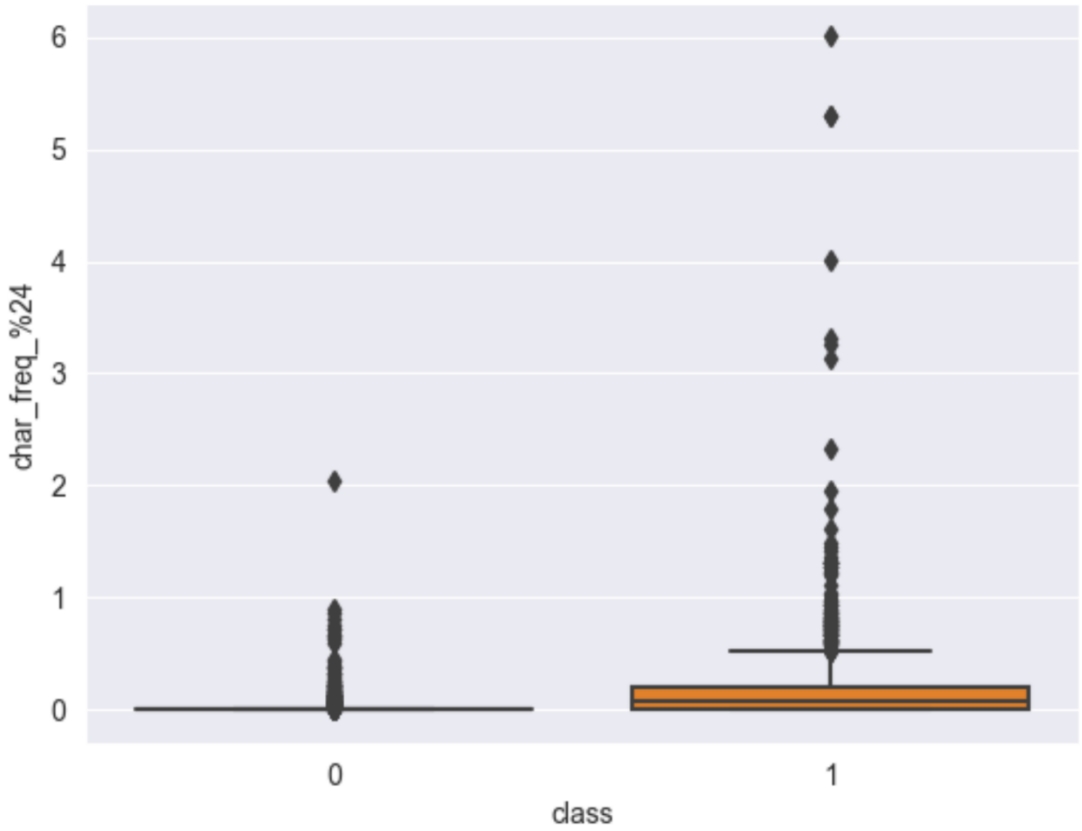


Figure frequency vs class distribution for char_%24

This step is done by using another package called seaborn:

```
import seaborn as sb
sb.set_style('darkgrid')

for i in vec:
    sb.boxplot(data = df, x = 'class', y = i)
    plt.show()
```

As we can visualize in the above four graphs, for the most frequent counts of the four variables, spam emails contribute to the most. Which could be one of the reasons these variables are the most important features. Also, the plots show that the upper quartile and upper bound for spam emails are always higher than normal emails.

Overall, I think I was being too generous on outlier and extreme value discover settings, because the most of frequency proportions for each variable are close to or round at least to the 0 to 1 range. I will need to do more research and come up with better settings later on to make the project more user-friendly to read and visualize and potentially help leverage the overall accuracy of the machine learning model.

2.4 Verify the data quality

In terms of verifying data quality, first we need to check if there is missing value for each data point. It can be easily done by Pyspark one line code:

```
from pyspark.sql.functions import isnan, when, count, col
dfs.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in dfs.columns]).show()
```

The result is as follow:

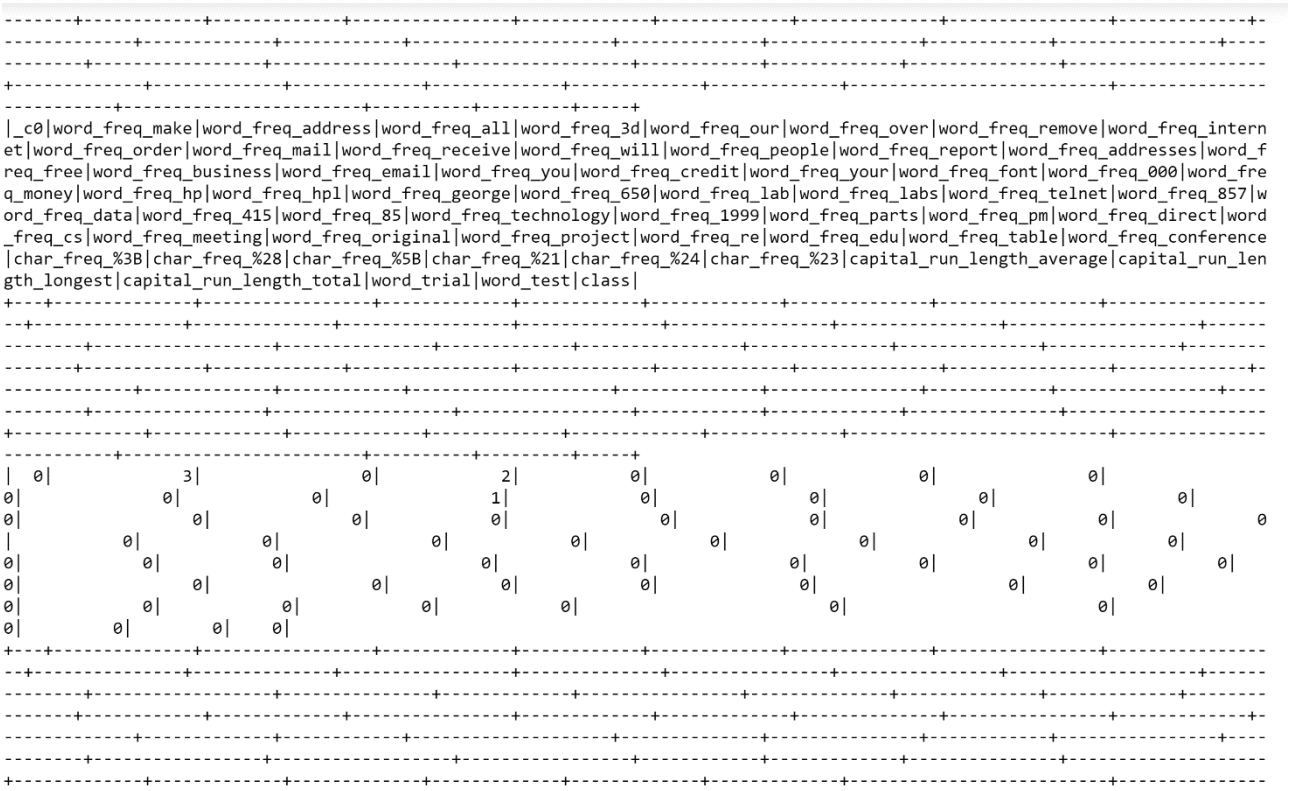


Figure missing value

The above screen shot looks pretty messy because I would like to check for all missing values for this dataset. Obviously, 3 instances miss the value for ‘word_freq_make’, 2 instances miss the value for ‘word_freq_all’, and 1 instance miss the value for ‘word_freq_receive’. Now I have the exact number of missing values, I can decide if I need to remove or do other preprocessing on them later if my learning model does not perform as expected. I will evaluate them, but theoretically, these missing values are too small compare to our overall number of instances, I doubt if these missing values would have a large impact on overall prediction. But I still need to deal with them carefully, so I used below functions to get the mean value for missing value columns and filled them with their mean value:

```
from pyspark.sql.functions import mean

mean_make = dfs.select(mean(dfs.word_freq_make)).collect()[0][0]
mean_receive = dfs.select(mean(dfs.word_freq_receive)).collect()[0][0]
mean_all = dfs.select(mean(dfs.word_freq_all)).collect()[0][0]

mean = {'word_freq_make': mean_make, 'word_freq_receive': mean_receive, 'word_freq_all': mean_all}
dfs = dfs.na.fill(mean)

dfs.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in dfs.columns]).show()
```

And now I check if missing values are still there:

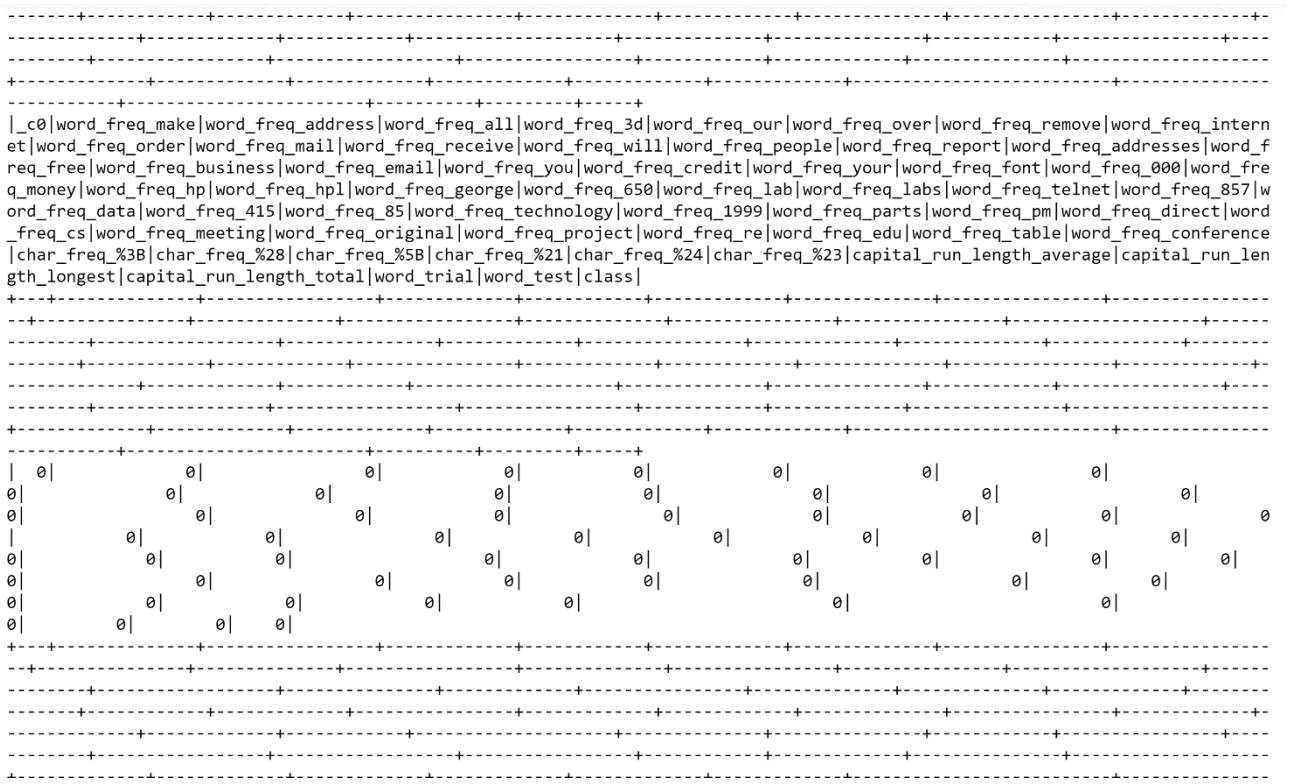


Figure check missing again

Now all the missing values are dealt with.

Next, I need to perform outlier and extreme value analysis. As discussed in section 2.3, the measure of outliers and extreme value need to be precise, otherwise the overall project quality will be compromised. To conduct precise analysis, one of many easy ways is to use statistical plots. I created four features and then test their outlier using Python functions.

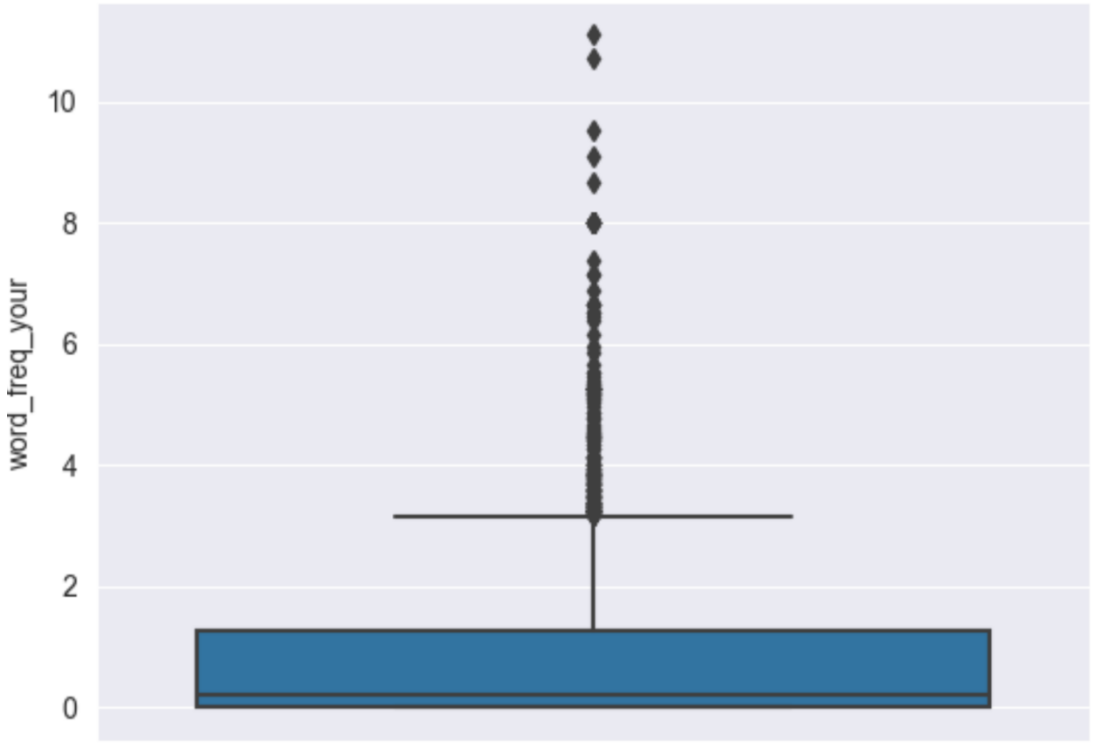


Figure distribution for word_your

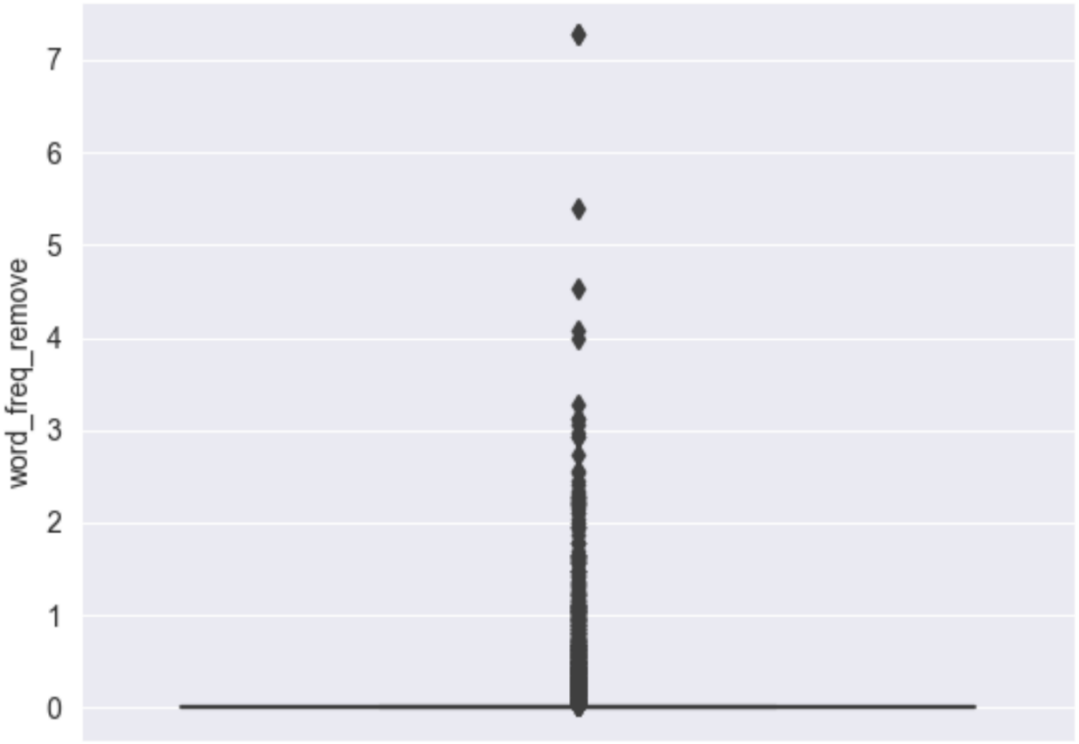


Figure distribution for word_remove

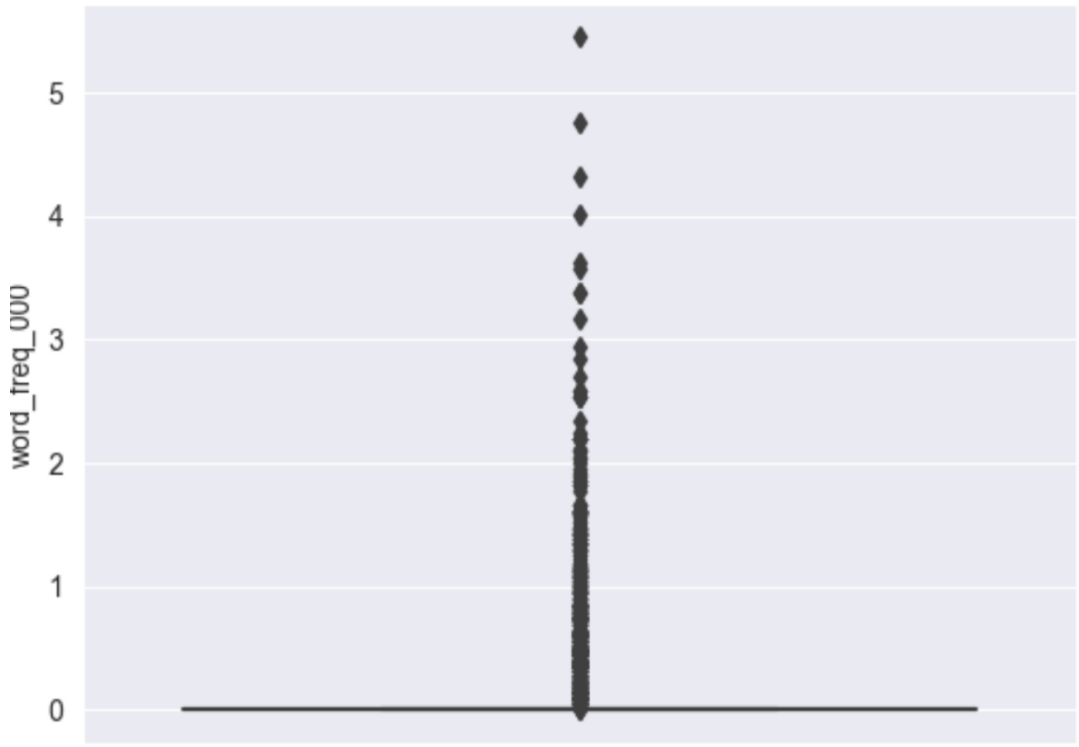


Figure distribution for word_000

Figure distribution for char_%24

After this step, I have a rough idea on the distribution of outliers and their extreme values, no negative value for these four distributions, and the distributions are centered in 0 to 1 range. Apart from word_your, other features have upper bound smaller than 1.

Now, before I perform outlier analysis, I need to deal with two string type columns because in pyspark, string and float are not convertible. Now as a trail, I use Pyspark functions to get the precise outlier bound for these features:

```
dfs_outlier = dfs

drop_list = ['word_trial', 'word_test']

dfs_outlier = dfs_outlier.select([column for column in df.columns if column not in drop_list])

# dfs_outlier.printSchema()

bounds = {
    c: dict(
        zip(["q1", "q3"], dfs_outlier.approxQuantile(c, [0.25, 0.75], 0))
    )
    for c in dfs_outlier.columns
}

for c in bounds:
    iqr = bounds[c]['q3'] - bounds[c]['q1']
    bounds[c]['lower'] = bounds[c]['q1'] - (iqr * 1.5)
    bounds[c]['upper'] = bounds[c]['q3'] + (iqr * 1.5)


for i in dfs_outlier.schema.names:
    print(i + ": " + str(bounds[i]))
    print("")
```

Figure outlier

Note that when I perform outlier detection, I created a new dataframe that excluded two string columns: word_trial and word_test, I will reformat them in the following section.

```
char_freq_%28: {'q3': 0.188, 'upper': 0.47000000000000003, 'lower': -0.28200000000000003, 'q1': 0.0}

char_freq_%5B: {'q3': 0.0, 'upper': 0.0, 'lower': 0.0, 'q1': 0.0}

char_freq_%21: {'q3': 0.315, 'upper': 0.7875000000000001, 'lower': -0.47250000000000003, 'q1': 0.0}

char_freq_%24: {'q3': 0.052, 'upper': 0.13, 'lower': -0.078, 'q1': 0.0}

char_freq_%23: {'q3': 0.0, 'upper': 0.0, 'lower': 0.0, 'q1': 0.0}

capital_run_length_average: {'q3': 3.707, 'upper': 6.8825, 'lower': -1.5855, 'q1': 1.59}

capital_run_length_longest: {'q3': 43.0, 'upper': 98.5, 'lower': -49.5, 'q1': 6.0}

capital_run_length_total: {'q3': 266.0, 'upper': 612.5, 'lower': -311.5, 'q1': 35.0}

class: {'q3': 1.0, 'upper': 2.5, 'lower': -1.5, 'q1': 0.0}
```

```
word_freq_free: {'q3': 0.1, 'upper': 0.25, 'lower': -0.15000000000000002, 'q1': 0.0}

word_freq_business: {'q3': 0.0, 'upper': 0.0, 'lower': 0.0, 'q1': 0.0}

word_freq_email: {'q3': 0.0, 'upper': 0.0, 'lower': 0.0, 'q1': 0.0}

word_freq_you: {'q3': 2.64, 'upper': 6.6, 'lower': -3.96, 'q1': 0.0}

word_freq_credit: {'q3': 0.0, 'upper': 0.0, 'lower': 0.0, 'q1': 0.0}

word_freq_your: {'q3': 1.27, 'upper': 3.175, 'lower': -1.905, 'q1': 0.0}

word_freq_font: {'q3': 0.0, 'upper': 0.0, 'lower': 0.0, 'q1': 0.0}

word_freq_000: {'q3': 0.0, 'upper': 0.0, 'lower': 0.0, 'q1': 0.0}

word_freq_money: {'q3': 0.0, 'upper': 0.0, 'lower': 0.0, 'q1': 0.0}

word_freq_hp: {'q3': 0.0, 'upper': 0.0, 'lower': 0.0, 'q1': 0.0}
```

As we can visualize above, for frequent words like ‘your’ or ‘000’, the upper and lower quartile are very close together, while as the three capital run length variables, are far more spread. There is no conclusion that these outliers should be discarded or not, we need to examine them in the coming section.

Finally, I should provide an overview of the dataset, which gives the relevant statistics for each feature in the dataset. The Figure below provides a direct overall quality check for the dataset:

```
dfs_outlier.describe('word_freq_make', 'word_freq_all', 'word_freq_receive', 'word_freq_you', 'char_freq_%24').show(2)
```

```
+-------+------------------+------------------+--------------------+------------------+--------------------+
|summary|     word_freq_make|      word_freq_all|   word_freq_receive|     word_freq_you|      char_freq_%24|
+-------+------------------+------------------+--------------------+------------------+--------------------+
|  count|              4601|              4601|                4601|              4601|                4601|
|   mean|0.1046215745976511|0.2807784300934978|0.059836956521739114|1.6620995435774817|0.07581069332753756|
+-------+------------------+------------------+--------------------+------------------+--------------------+
only showing top 2 rows
```

It contains count, mean, standard deviation, min, max, 25% quartile, 50% quartile and 75% quartile. We do not show all for page layout purpose. It is indicated by the above statistics that the surface data quality is ensured.

However, there is no guarantee that underlying data quality issue will not occur as the project progress deeper. Because I was not involved in dataset creating process. If future data quality issue is discovered, I will come back and log them in this section.

3 Data preparation

After the preliminary investigation on the overall data quality, the next step is to use data preprocessing methods to get the data format that is required for our data-mining tasks. This process

involves extremely large time-consuming steps, but it is a crucial step we need to take. As suggested in section 2.4, although there is no obvious missing value or error to be found, there still might be hidden redundancies or inconsistencies in the dataset that compromise the integrity of the set. If we build and train our model with a faulty dataset, chances are that the system will create biases and deviation and therefore lead to poor performance. Therefore, we need our dataset to be as "clean" as possible so that once we confirm our data-mining method and starting to build machine learning model(s), our outcome can be as accurate as possible.

3.1 Select the data

Before we perform any data preprocessing task, the data we collected needs to be well-structured in a sense. For our data-mining purpose, I need to collect enough trainable instances with common features, otherwise the model might be biased because there is simply not enough training samples. The original dataset that I used only contains 4000 instances, however our goal is to develop a model that can accurately detect spam emails. If the training samples are too small, some original spam emails that contain for instance "000" may only contribute to a small portion of total spam emails, so our model might classify this frequent word as normal, but in reality it often occurs in "wining a lottery with $5,000" kind of spam emails. Feature-wise speaking, in total we have 60 attributes, and the data contains not only frequent words count, but also detail features about the structure of the email: i.e. how many capital words are there in this email in total to indicate the length of the email, so we can focus on finding enough training samples at this stage. Luckily, the author of the original dataset that I used published a new version of this dataset, it contains more instances along with mixed class attributes. I made a new dataframe to store new dataset: After this step, I added more instances into the data (Figure 1), now we have in total 7702 records. From the dataset, each attribute it contains is very important, and the number of each record has a reference value, so I do not need to do other collections and integration.

```
print("Number of rows for version 1: ", dfs.count())
print("Number of rows for version 2: ", dfs1.count())

dfs_combine = dfs.union(dfs1)
print("Number of rows for combined dataframe: ", dfs_combine.count())
```

```
Number of rows for version 1:  4601
Number of rows for version 2:  3101
Number of rows for combined dataframe:  7702
```

Figure combining V1 and V2 together

3.2 Clean the data

After data selection, we need to clean the data. As previously mentioned in data understanding section, clean the data requires we deal with three main faulty data: noisy data, extreme data and outliers. The three types of data are similar but with differences. Extreme value describes observation with value at the boundaries of the domain, outliers describes observation which appears to be inconsistent with the remainder of that dataset. Noise on the other hand is any undesirable or unwanted signal or part of a signal in the dataset. For the data cleaning purpose, we will also need to deal with missing value first. Similar as in step 2.4, I used mean value to deal with missing value. Even though I think this step is unnecessary in our project as we only have less than ten missing values compare to the overall number of instances. Now we can perform outlier and extreme value analysis based on this finding.

```
from pyspark.sql.functions import isnan, when, count, col

dfs1 = spark.read.csv('dataset/spambaseV1.csv', inferSchema=True, header=True)
```

Again, we need to perform missing value check and fill them with mean values:

```
dfs1.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in dfs1.columns]).show()
```

```
+--------------+-----------------+-------------+-------------+-------------+--------------+-----------------+-------------------+-
--------------+--------------+-----------------+--------------+----------------+-----------------+--------------------+----------
----+-----------------+---------------+-----------------+-------------+----------------+---------------+-------------+--------------
---+-------------+-------------+-------------+-----------------+-------------+-------------+---------------+-------------+-----
---------+-------------+-------------+----------------------+----------------+---------------+-------------+-----------------+--------
----+-----------------+-----------------+-------------------+------------------+-------------+----------------+-----------------+---
----------+-------------+-------------+--------------+-------------+-------------+-------------------------+-------------------
-------+-------------------------+-----+
|word_freq_make|word_freq_address|word_freq_all|word_freq_3d|word_freq_our|word_freq_over|word_freq_remove|word_freq_internet|w
ord_freq_order|word_freq_mail|word_freq_receive|word_freq_will|word_freq_people|word_freq_report|word_freq_addresses|word_freq_
free|word_freq_business|word_freq_email|word_freq_you|word_freq_credit|word_freq_your|word_freq_font|word_freq_000|word_freq_mo
ney|word_freq_hp|word_freq_hpl|word_freq_george|word_freq_650|word_freq_lab|word_freq_labs|word_freq_telnet|word_freq_857|word_
freq_data|word_freq_415|word_freq_85|word_freq_technology|word_freq_1999|word_freq_parts|word_freq_pm|word_freq_direct|word_fre
q_cs|word_freq_meeting|word_freq_original|word_freq_project|word_freq_re|word_freq_edu|word_freq_table|word_freq_conference|cha
r_freq_%3B|char_freq_%28|char_freq_%5B|char_freq_%21|char_freq_%24|char_freq_%23|capital_run_length_average|capital_run_length_
longest|capital_run_length_total|class|
+--------------+-----------------+-------------+-------------+-------------+--------------+-----------------+-------------------+-
--------------+--------------+-----------------+--------------+----------------+-----------------+--------------------+----------
----+-----------------+---------------+-----------------+-------------+----------------+---------------+-------------+--------------
---+-------------+-------------+-------------+-----------------+-------------+-------------+---------------+-------------+-----
---------+-------------+-------------+----------------------+----------------+---------------+-------------+-----------------+--------
----+-----------------+-----------------+-------------------+------------------+-------------+----------------+-----------------+---
----------+-------------+-------------+--------------+-------------+-------------+-------------------------+-------------------
-------+-------------------------+-----+
|             2|                0|            0|           0|            0|             0|               0|                 0|
0|             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|                      0|              0|             0|               0|            0|           0|               0|
0|                 0|            0|             0|           0|             0|            0|                 0|
0|             0|            0|                      0|            0|            0|   0|
+--------------+-----------------+-------------+-------------+-------------+--------------+-----------------+-------------------+-
```

Figure missing value

```
from pyspark.sql.functions import mean

mean_make = dfs1.select(mean(dfs1.word_freq_make)).collect()[0][0]
mean_receive = dfs1.select(mean(dfs1.word_freq_receive)).collect()[0][0]

mean = {'word_freq_make': mean_make, 'word_freq_receive': mean_receive}
dfs1 = dfs1.na.fill(mean)

dfs1.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in dfs1.columns]).show()
```

```
--------+-----------------+-----------------+----------------+------------+------------+--------------+--------------------
+-------------+-------------+-------------+-------------+-------------+-------------+--------------------------+--------------
-----------+------------------------+----------+---------+-----+
|_c0|word_freq_make|word_freq_address|word_freq_all|word_freq_3d|word_freq_our|word_freq_over|word_freq_remove|word_freq_intern
et|word_freq_order|word_freq_mail|word_freq_receive|word_freq_will|word_freq_people|word_freq_report|word_freq_addresses|word_f
req_free|word_freq_business|word_freq_email|word_freq_you|word_freq_credit|word_freq_your|word_freq_font|word_freq_000|word_fre
q_money|word_freq_hp|word_freq_hpl|word_freq_george|word_freq_650|word_freq_lab|word_freq_labs|word_freq_telnet|word_freq_857|w
ord_freq_data|word_freq_415|word_freq_85|word_freq_technology|word_freq_1999|word_freq_parts|word_freq_pm|word_freq_direct|word
_freq_cs|word_freq_meeting|word_freq_original|word_freq_project|word_freq_re|word_freq_edu|word_freq_table|word_freq_conference
|char_freq_%3B|char_freq_%28|char_freq_%5B|char_freq_%21|char_freq_%24|char_freq_%23|capital_run_length_average|capital_run_len
gth_longest|capital_run_length_total|word_trial|word_test|class|
+---+--------------+----------------+------------+------------+-------------+--------------+---------------+----------------
--+--------------+-------------+----------------+-------------+---------------+---------------+-----------------+------
-------+-----------------+--------------+---------------+-------------+-------------+--------------+---------------+-------------+-------
-------+------------+-------------+---------------+-------------+-------------+--------------+---------------+-------------+-
-------------+-------------+------------+--------------------+--------------+--------------+------------+---------------+----
--------+-----------------+-----------------+----------------+------------+------------+--------------+--------------------
+-------------+-------------+-------------+-------------+-------------+-------------+--------------------------+--------------
-----------+------------------------+----------+---------+-----+
|  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|              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|                0|           0|
0|           0|           0|           0|           0|                        0|                       0|
0|        0|       0|    0|
+---+--------------+----------------+------------+------------+-------------+--------------+---------------+----------------
--+--------------+-------------+----------------+-------------+---------------+---------------+-----------------+------
-------+-----------------+--------------+---------------+-------------+-------------+--------------+---------------+-------------+-------
-------+------------+-------------+---------------+-------------+-------------+--------------+---------------+-------------+-
-------------+-------------+------------+--------------------+--------------+--------------+------------+---------------+----
```

(1) Outlier & extreme value analysis

As suggested in Data Understanding section, our dataset contains mainly numeric attributes: i.e. they are continuous. For frequent words count, the range is from 0 to 1, and since spam emails vary in forms, so we expect the frequency for a certain word in a email to be extremely low or at least not regular. As far as the result suggests, in total we have 7702 records and zero missing value, so we need to keep some recorded outliers and extreme values just in case they are key attributes to identify spam email.

Since we find outliers and extreme values are potentially useful in our case, we can now set a range to identify our change in proportion when we perform outliers and extreme value analysis. As circled in blue, once I set to discard extremes, these extreme values automatically changed to outliers. when we coerce outliers and discard the extreme values, extreme values were gone but outliers increased, and the increase in outliers is larger than the decrease in extreme values. We can conclude that for this dataset, the farther the attribute value distribution is from center, we get smaller number of samples with the

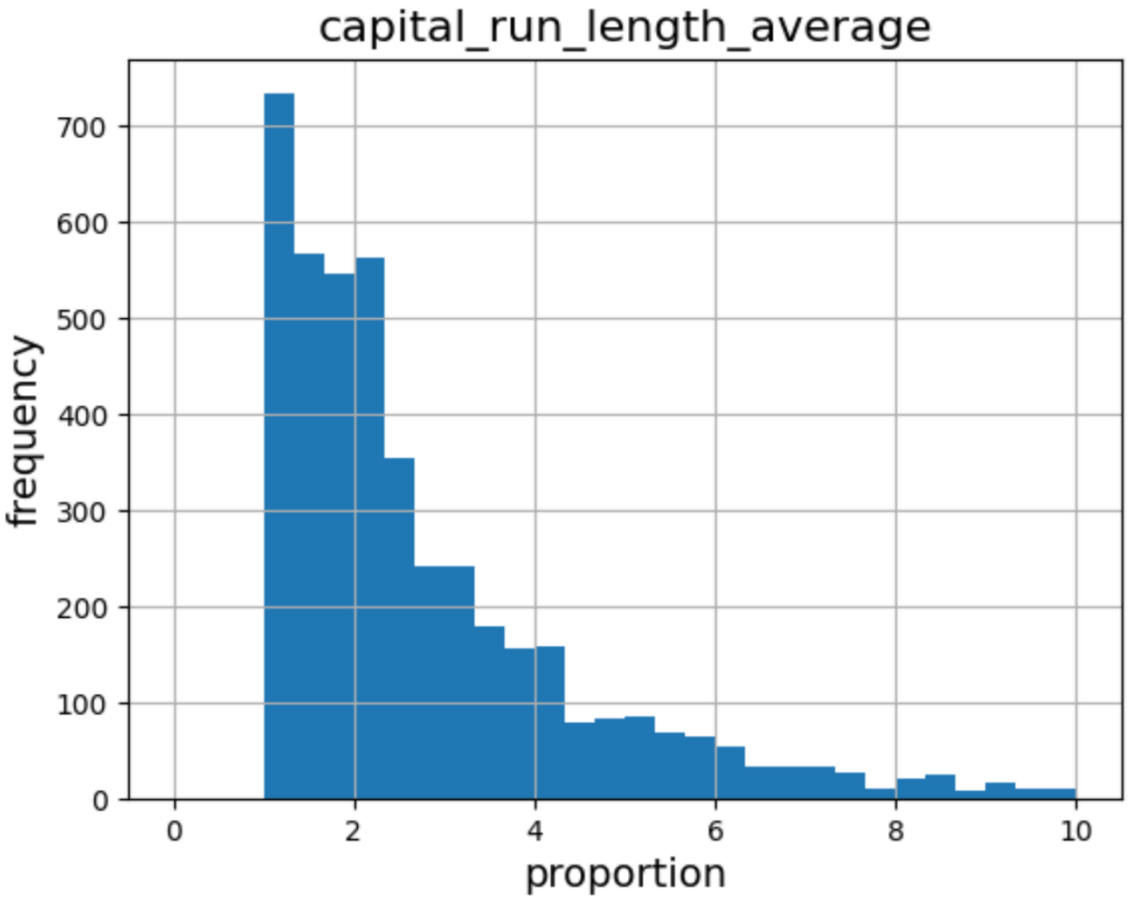
attribute value. However, these values do not disappear. Which indicate that some extreme values are normal for this dataset, the outliers and extreme values we got is highly unlikely to causing measurement trouble when we preform next steps. Because when I looked at the change in frequency distribution for frequent word “over” and “order”, once I changed their behaviour to coerce outliers and discard extremes, the attributes become normal in a sense: the word frequency attributes are calculated by: (number of appearance of this word in the email)/(total number of words in the email)*100, once I discarded outliers and extreme values, the range became only from 0 to 1, all the values that are not “centered” are gone. If we apply this to the dataset, the integrity of the dataset will be compromised. Take the frequent word 000 as an example, when apply outlier analysis on the combined dataset, there are 1300 outliers identified, whereas in total we have more than 7700 records. As discussed above, we are more than happy to keep these outliers for now to maintain data integrity, because after all our main range for this feature is lay somewhere between 0 to 1. The outliers and extreme values are normal in our dataset, so the raw data is reliable, we keep the original dataset.

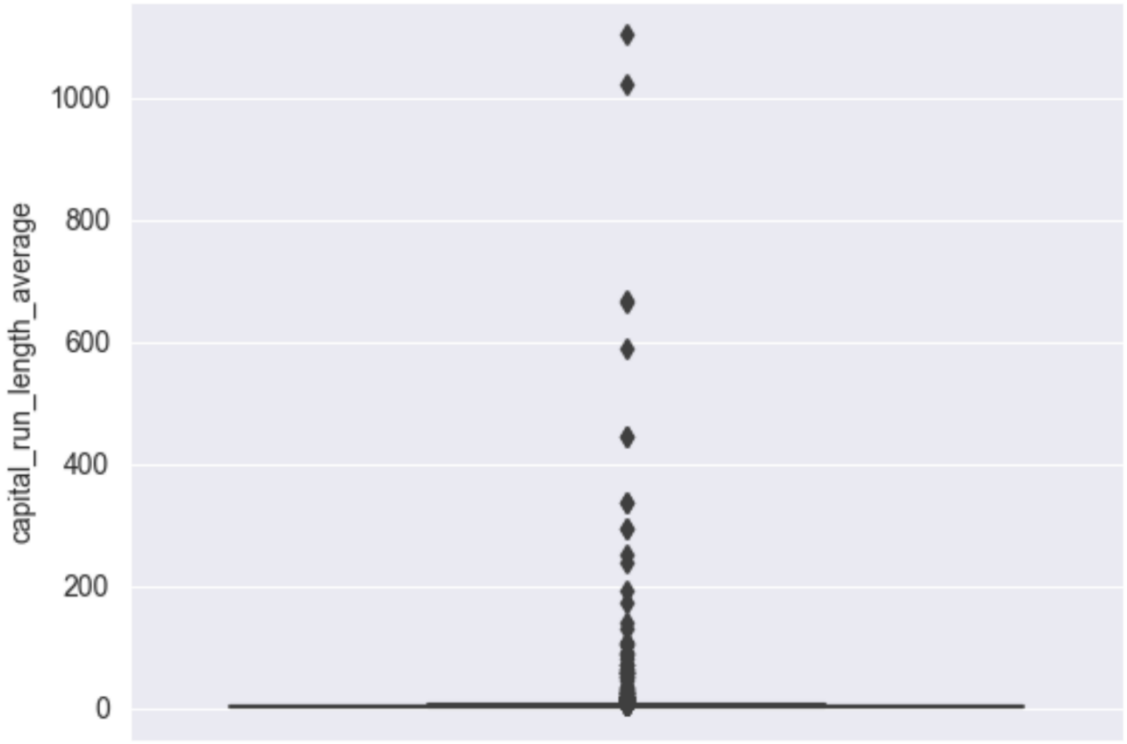
```
df = dfs_combine.toPandas()

out_list = []
out_list2 = []


d = df['word_freq_000'].astype(int)
for i in range(0, len(d)):
    out_list.append(df['word_freq_000'][i])
    out_list2.append(df['word_freq_000'][i])
out_list.sort()
# print(out_list)
q1, q3 = np.percentile(out_list, [25, 75])

iqr = q3 - q1
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)
print("lower bound: ", lower_bound)
print("upper bound: ", upper_bound)

outliers = []
for i in range(0, len(d)):
    if out_list2[i] > upper_bound or out_list2[i] < lower_bound:
        outliers.append(out_list2[i])
print("")
print("Number of outliers for '000': ", len(outliers))
```

```
lower bound:  0.0
upper bound:  0.0

Number of outliers for '000':  1320
```

3.3 Construct the data

For this dataset, our goal is to use 57 features to predict the class attribute, and all of the three main feature categories are labelled as continuous: word frequency; char frequency and capital length. If we try to re-construct the data, all we can do is to analyse the underlying relationship between features to class attribute, and try to discard or remove some irrelevant features, or form them together as a super node. After glancing through all the attributes, I discovered that for frequent words and frequent char, it is difficult to decide if a feature should be removed or not based on their statistical analysis, because their standard deviation and skewness are pretty much all similar. Individual words are not so common in an email letter after all. So I changed my direction to analysis email length related attributes: capital_run_length_average, capital_run_length_longest and capital_run_length_total. I discovered that for capital_run-length_average, the standard deviation is extremely low compare with other two capital features, this indicates that the average email length are somewhat similar, and therefore we can remove this attribute as it would not contribute much to our model, but might cause other issue like overfitting.



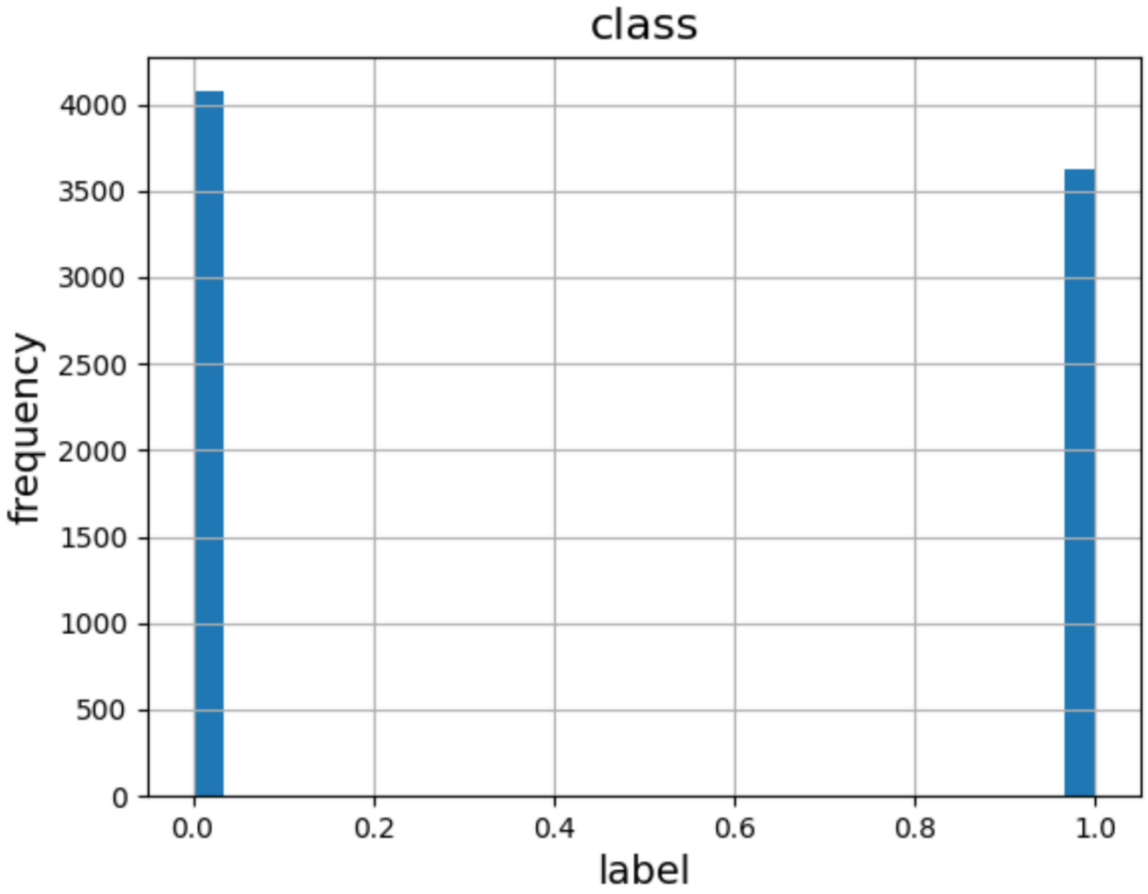


(There are in total 4600 1s and 1 1205, so we discard this feature.)

```
# dfs_combine.printSchema()

print("Before drop: ", len(dfs_combine.columns))

dfs_combine = dfs_combine.drop('capital_run_length_average')

print("After drop: ", len(dfs_combine.columns))
```

```
Before drop:  61
After drop:  60
```

As for our key classification attribute class, since the dataset is will organized and the author of the dataset has already given us the classification outcome: 1 for spam email and 0 for real email, we do not need to construct it and we can directly use it for later steps.



3.4 Integrate various data sources

As mentioned in section 3.1, originally I only have 4000 records as a whole, but this time I combined the newest version of the dataset along with the previous one, so I need to preform data integration. There are mainly two data integration methods available:

Merging: This method is required when data was collected from different sources. Although data types are similar, for instance they all describe behaviors of spam email, but their data structures are different. In particular, for text-mining tasks like this, I found various forms of data structures on Kaggle, some of them only contain two columns: one for classification and the other one for text content. If this is the case, I will need to preform text-mining related preprocessing methods to split the text content into individual word columns, and then use data merge methods to match datasets' columns. Often time it also involves dealing with NAs: for certain datasets, they do not have some attributes that the other datasets have, so we would need to know how to handle NAs such as use instance deletion or other methods like mean, median or closest value. The two datasets that I collected contain same attributes, the new one is just an upgrade version of the old one. So all the attributes are the same, therefore no need for data merge for this project.

Appending: This method involves integrating two datasets with same features but different records into one dataset. The general steps were demonstrated in section 3.1, where I used the newer version of the dataset appended in our original dataset, the result were also shown above. Before we only have 4601 data records, now by adding more instances into the dataset, we are confident that this will provide more predicting power to our later learning models.

```
print("Number of rows for version 1: ", dfs.count())
print("Number of rows for version 2: ", dfs1.count())

dfs_combine = dfs.union(dfs1)
print("Number of rows for combined dataframe: ", dfs_combine.count())
```

```
Number of rows for version 1:  4601
Number of rows for version 2:  3101
Number of rows for combined dataframe:  7702
```

3.5 Format the data as required

When taking class attribute into account, I discovered that the distribution of class attribute is not balanced. Apart from this, the remaining attributes are all in the correct type and no need to change their format. At this stage I am not sure if the imbalanced class attribute would affect our model outcome. However, when considering two columns ‘word_trial’ and ‘word_test’, we need to change their categorical type to numeric as it is required by the classifiers. I created a string indexer to solve this problem:

```
from pyspark.ml.feature import StringIndexer


indexer = StringIndexer(inputCol="word_trial", outputCol="trial_Index")
indexed = indexer.fit(dfs_combine).transform(dfs_combine)

indexer2 = StringIndexer(inputCol="word_test", outputCol="test_Index")
indexed = indexer2.fit(indexed).transform(indexed)

indexed = indexed.drop('word_test')
indexed = indexed.drop('word_trial')
indexed.describe('trial_Index', 'test_Index').show()
```

And now our two string columns are:

```
+-------+------------------+-------------------+
|summary|       trial_Index|         test_Index|
+-------+------------------+-------------------+
|  count|              7702|               7702|
|   mean|0.2455206439885744|0.25772526616463254|
| stddev|0.6281622210078537| 0.6443561320528145|
|    min|               0.0|                0.0|
|    max|               3.0|                3.0|
+-------+------------------+-------------------+
```

From this point, we can perform machine learning via pyspark.

4 Data transformation

After data preparation steps, we now come to data transformation step. This is another critical process we need to take before start to build machine learning model(s) because it can help us to focus on the important data points, preserve efficiency and save training time (vertical wise); it also enable our work to focus on the most interesting features throughout the training set (horizontal wise). It refers to either reducing training data via selecting the most representative subset; or reducing dimensionality of our chosen dataset. I will go into depth in section 4.1 as I will use both methods.

4.1 Data Reduction

As mentioned in the general description above, in this section we have two main tasks to perform: feature selection that choose the most important features and leave non-crucial features behind by a preset threshold. I performed this step by using the feature selection tool suggested in section 2.3, feature selection tool from sklearn package, this time however, I printed every feature to see their correlation with each other:

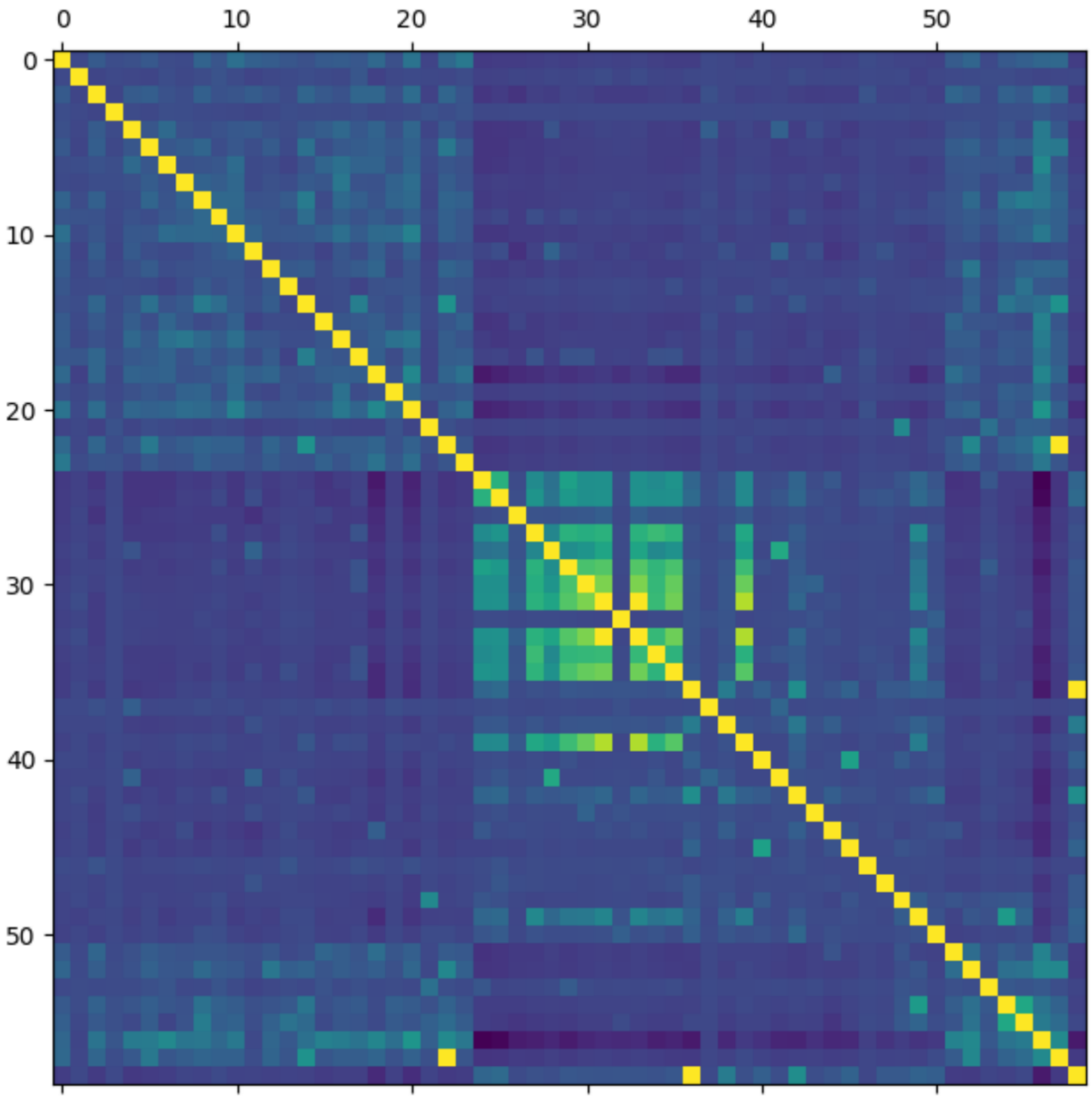


Figure feature correlations

To get more detailed idea, I used correlation index and made a matrix for it:

	_c0	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove
_c0	1	-0.101694	0.0300902	-0.155754	-0.036428	-0.219891	-0.184603	-0.273945
word_freq_make	-0.101694	1	-0.0111145	0.074058	0.013839	0.0210597	0.0642083	0.0036765
word_freq_address	0.0300902	-0.0111145	1	-0.0261412	-0.00779838	-0.017703	-0.0221901	0.0109912
word_freq_all	-0.155754	0.074058	-0.0261412	1	-0.0237493	0.0760231	0.0932362	0.0318528
word_freq_3d	-0.036428	0.013839	-0.00779838	-0.0237493	1	0.00108283	-0.0124234	0.017563
word_freq_our	-0.219891	0.0210597	-0.017703	0.0760231	0.00108283	1	0.0531339	0.147041
word_freq_over	-0.184603	0.0642083	-0.0221901	0.0932362	-0.0124234	0.0531339	1	0.0588029
word_freq_remove	-0.273945	0.0036765	0.0109912	0.0318528	0.017563	0.147041	0.0588029	1
word_freq_internet	-0.164886	-0.00366836	-0.0147633	0.0110633	0.00936395	0.0261208	0.0824289	0.0320056
word_freq_order	-0.18336	0.118941	0.00437534	0.100584	-0.00461698	0.012117	0.125953	0.0421473
word_freq_mail	-0.0904885	0.0468654	0.0367274	0.032101	-0.00651212	0.0317488	0.0121562	0.0557381

After I conduct my investigation, I found that for frequent word: font, it's correlation with others are somehow weaker than others:

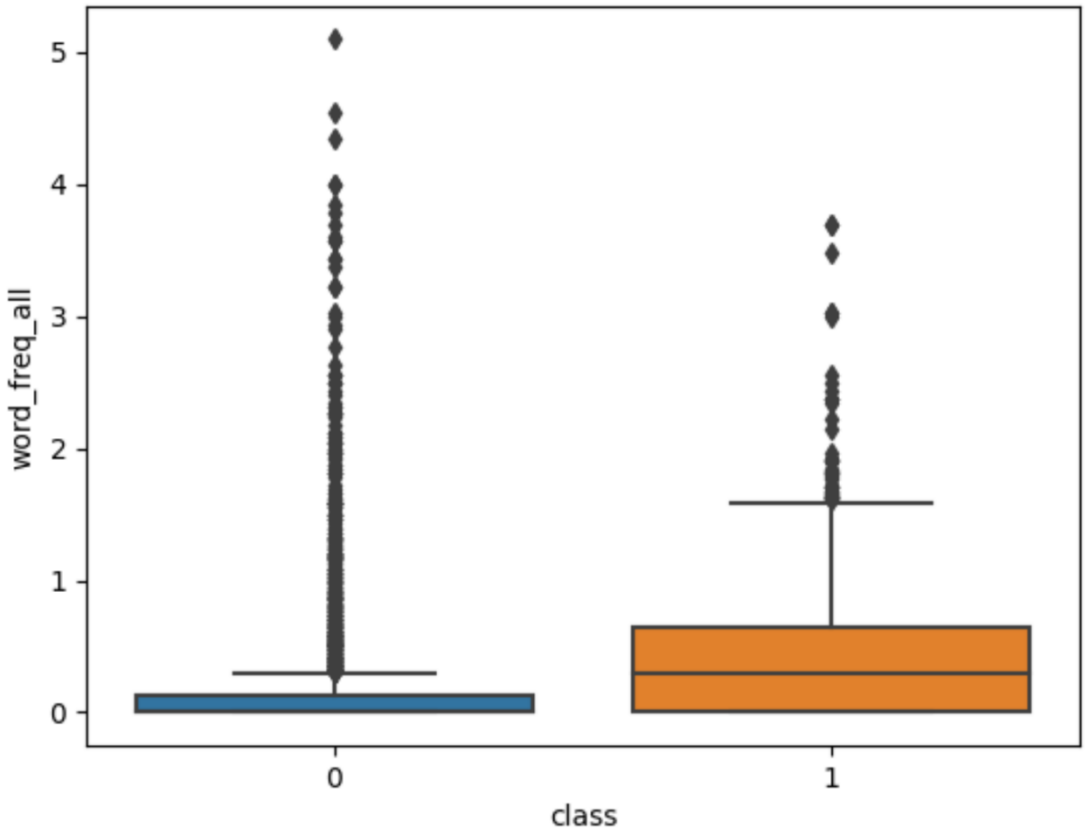
word_freq_credit	word_freq_your	word_freq_font	word_freq_000	word_freq_money	word_freq_hp	word_freq_hpl
-0.152093	-0.317582	-0.0593901	-0.264884	-0.161151	0.173751	0.175487
0.0208282	0.199738	-0.0275567	0.144329	0.221422	-0.0759496	-0.0659153
-0.0162676	-0.00513942	-0.0116015	-0.0198782	0.00839809	-0.0422992	-0.0371106
0.0286662	0.165231	-0.0401879	0.126513	0.042383	-0.0898537	-0.0641885
-0.00678823	0.00560116	0.0293947	0.00904265	0.0360652	-0.0162734	-0.0148748
0.0243944	0.133455	-0.022845	0.0623918	-0.00646671	-0.0923988	-0.0872614
0.0597364	0.105072	0.00964657	0.223871	0.0647811	-0.0933392	-0.0954748
0.0390431	0.122709	-0.00399656	0.0520574	0.0243447	-0.0965654	-0.0870891
0.111405	0.169165	-0.0176626	0.0869366	0.0338367	-0.0613168	-0.0501442
0.122824	0.153128	-0.0219954	0.122026	0.104108	-0.0756728	-0.055849
0.0303089	0.0956793	0.00157836	0.100262	0.0581107	-0.0374693	-0.0144739

Figure feature score for combined dataset

After screening key features, I discovered that for continuous feature types, it seems that the Overall feature scores for all the features have increased tremendously, however, the variance of features are high: from 1302 to 0.34. Therefore, this time we can discard some features that the feature selection predictor analysed to be irrelevant to preserve algorithm efficiency. We discard the frequent word font this time.

```
In [69]: indexed = indexed.drop('word_freq_font')
```

Also, we need to test on whether data quality was compromised from this process. I took some random features to visualise the correctness of this step:



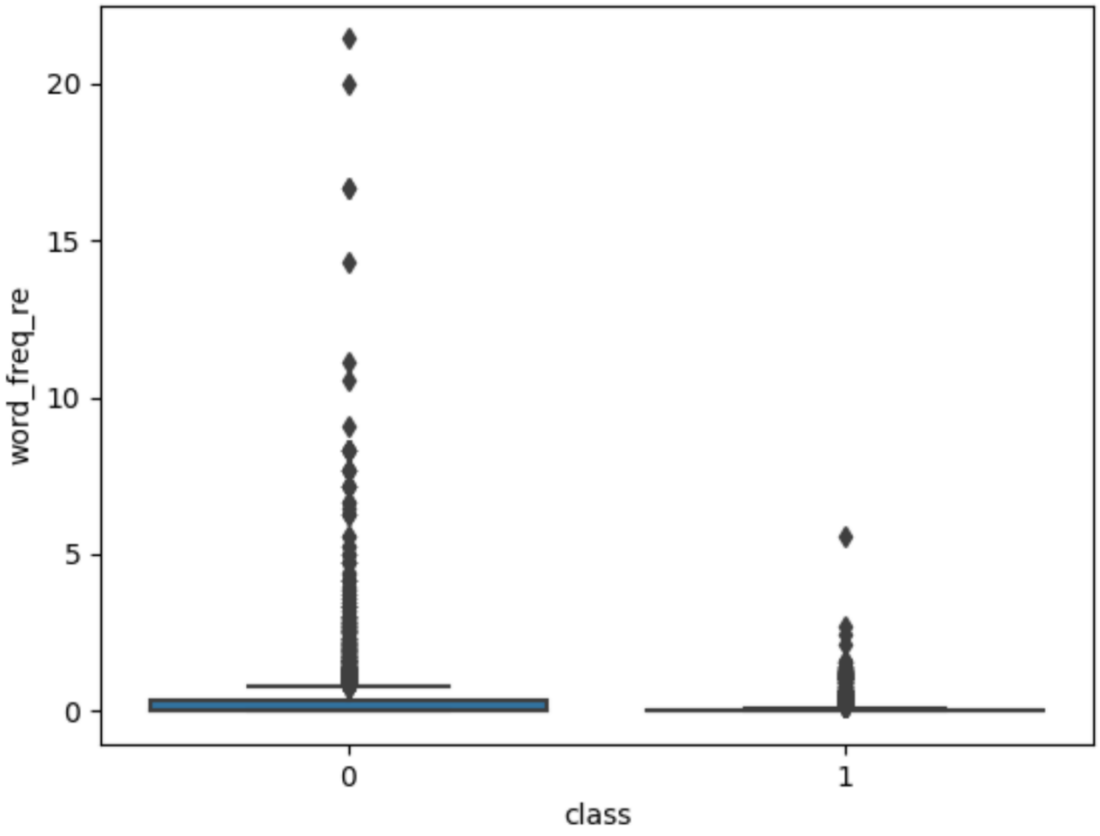


Figure class distribution for ‘re’ and ‘all’

Above are some random features selected from feature selection process, as we can see from the figure, word “all” when in range 0 to 1, real emails contributes to almost 60 percent of the whole proportion, while from range 1 to more, spam emails almost dominated every ranges. On the contrary, frequent word “re”s histogram shows that for every range of frequencies, it is more likely that if word “re” occurs repeatably, it indicates the email is likely to be a normal email. Both histograms show that the features are actually important when it comes to model building, so the feature selection process is completed with logic.
The next step is to balance the sample. Relating to previous discovery (section 3.3), a key factor we need to address is that the distribution of our class attribute is imbalanced. It is important to understand that improper accuracy scoring rules would lead to bogus models, because all the statistics such as receiver operating characteristic curve (ROC) or Area under the curve (AUC) might be affected by the scoring rule. An imbalanced class distribution (Figure 17) indicates the sample is skewed.
Although we have a seemingly imbalanced class distribution: 60% normal and 40% fraud emails, after I conducted a series of researches, I find that basically all the imbalance class handlers such as SMOTE and imbalance-learn are mainly focusing on reducing extremely imbalanced data: i.e. 1000 records in total, 980 normal and 20 abnormal. Therefore, if I did perform class balancing at this stage, I afraid it would compromise the overall data quality because the key for oversampling is that the algorithm copy data behaviors from the original dataset, and randomly generate data points that are similar to the original dataset. In such way, our model’s learning capability may be damaged. Therefore, I did not balance the class attribute at this point.

4.2 Project the data

For the projection purpose, I realised that I need to classify some attributes, and the attribute I selected as a example is the frequent char “%24” (utf8). Originally, the attribute distribution is in Figure 20, and I projected this attribute into a new categorical attribute, because it is difficult to identify the significant difference as the attribute is densely distributed within 0 to 0.5 range. In normal life, we do not put ‘$’ (encoding for char “%24”) very often in our daily emails. So I made five categories representing the frequency distribution: low, median, high, boundary and unknown (Figure 21) with function tool. It is done in pandas as pyspark dataframe does not allow this operation.

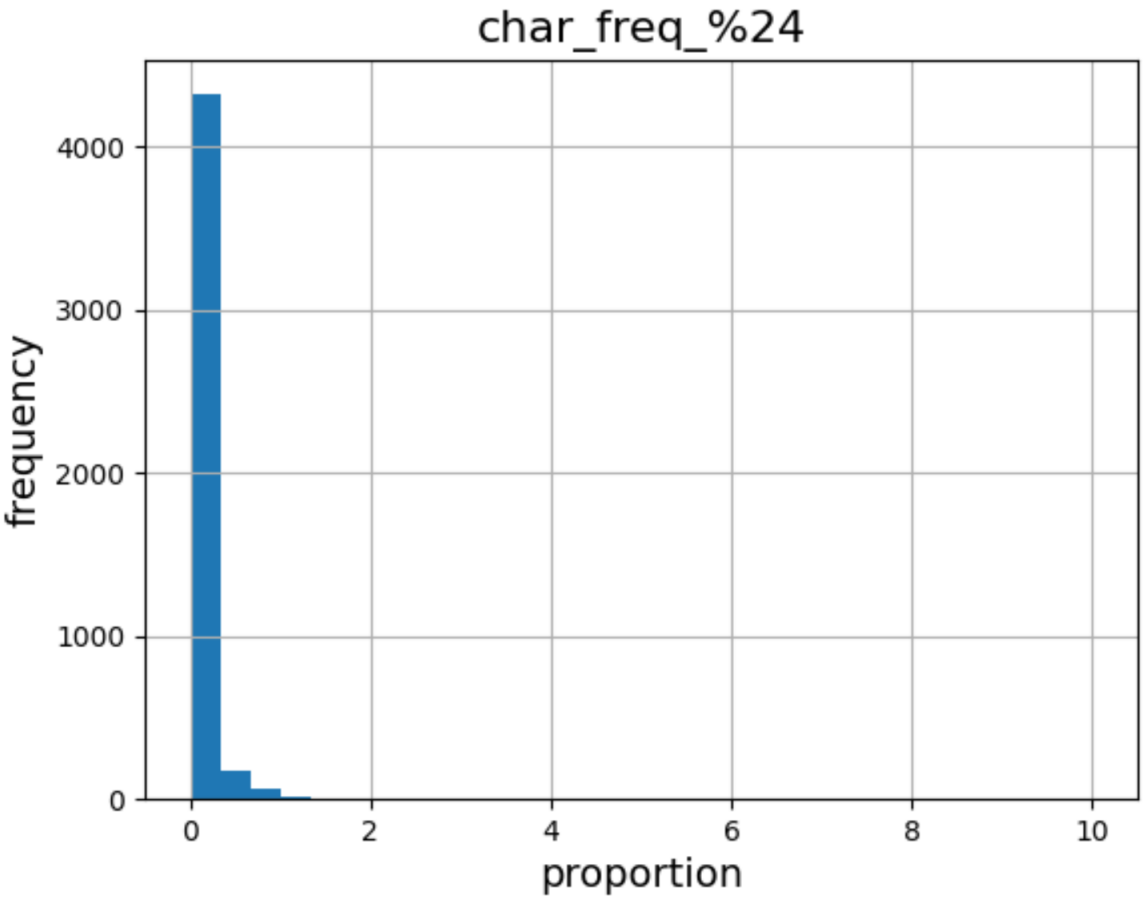


Figure original frequency distribution

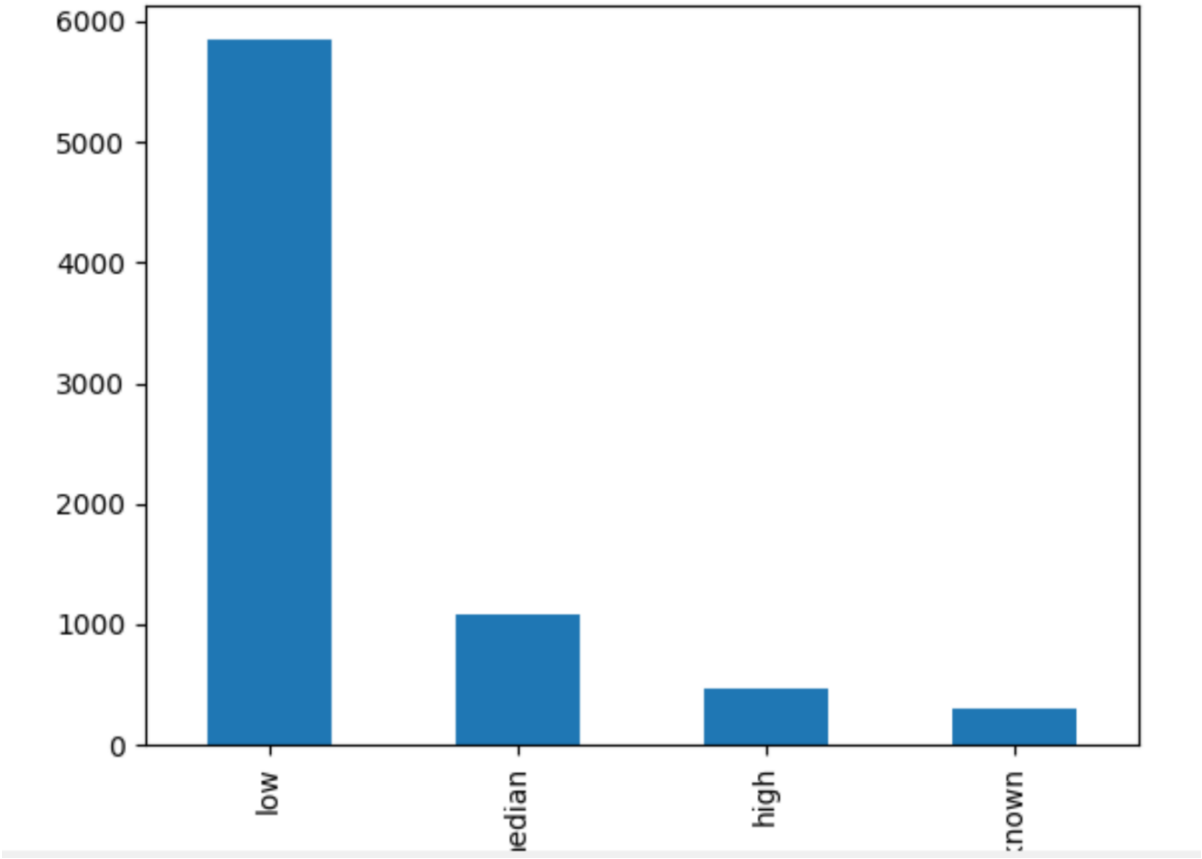


Figure more intuitive frequency for char '%24'

We can then tell that for normal emails (representing '1'), the frequency is very low, whereas for spam emails (representing '0'), when the frequency of this char is above 0.25, we get almost all emails are classified as spam. This also indicate the credibility of our dataset: it follows common logic, and all the source data are collected from actual emails. Therefore, this is a reasonably good grouping.

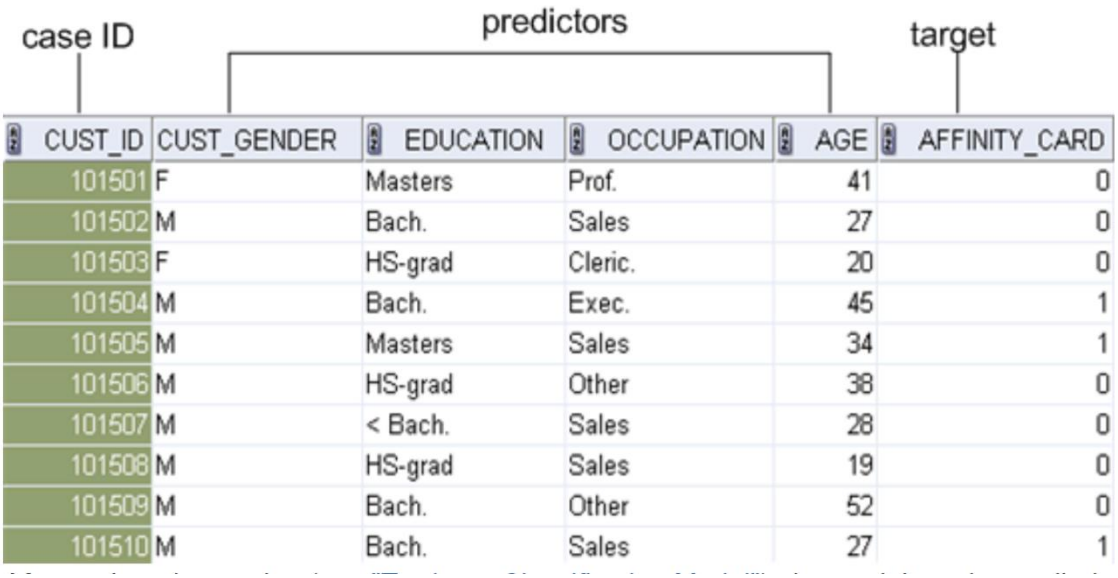
5 Data-mining method(s) selection

5.1 Match and discuss the objectives of data mining to data mining methods

In section 1.1, I have mentioned that our project objective is to first find frequently used words by spammers and build machine learning model(s) to detect if an certain email is spam email or not based on given word features. Given our business objectives, we also built our success criteria: find the best featured words that can identify spam email content and then our machine learning model(s) need to have average accuracy above preset threshold. After some research, I discovered that there are a number of data mining methods available for me to choose. I will briefly introduce them, discuss their pros and cons on this project and finally discuss why certain method was chosen.

- Classification: It is a data mining function that assigns items in a collection to target categories or classes. The ultimate outcome and final achievement it brings is that we create data mining or machine learning model(s) to accurately predict the target class for each input instance. It is a supervised data mining method, that is, we consider each input example or instance is a pair consisting of an input object or vector, and a desired output value. Eventually we have our final output as the measurement of success of our model. This method is discrete and do not imply order. Different types of attributes such as continuous, floats, booleans would indicate a numerical class attribute rather than a categorical target. In the model building process (or training process), classification models are designed to find relationships between values of predictors (features) and the target value (Figure 23). Pros of classification include: Easy to interpret and construct; it is able to adapt to complex hypothesis functions; many of classification algorithms build feature importance when setting its' parameters; and finally, some of the algorithms perform well on large datasets. Cons of classification include: In often cases it could cause the problem of overfitting (perform well on the training set but failed in test set, because the algorithm focus more on the training set, when there are some patterns changed in the test set, we lose accuracy); it is also not very robust, small changes in the training set can lead to major differences in algorithm's hypothesis function; and finally, single classification algorithms are sometimes not performing well (different data types

and various forms of data) compare to modern ensemble methods like xgboost and random forest. Typical classification algorithms include: Decision tree, Regression and Neural Network.



CUST_ID	CUST_GENDER	EDUCATION	OCCUPATION	AGE	AFFINITY_CARD
101501	F	Masters	Prof.	41	0
101502	M	Bach.	Sales	27	0
101503	F	HS-grad	Cleric.	20	0
101504	M	Bach.	Exec.	45	1
101505	M	Masters	Sales	34	1
101506	M	HS-grad	Other	38	0
101507	M	< Bach.	Sales	28	0
101508	M	HS-grad	Sales	19	0
101509	M	Bach.	Other	52	0
101510	M	Bach.	Sales	27	1

Figure 23

- Association rule mining: It is a if-then statement method to help to show the probability of relations data items within large datasets. Association rule mining at basic level involves the use of data mining models to analysis features' co-occurrence. It contains two major parts: an antecedent and a consequent. For instance, a supermarket has some customer transactions. 2% of total transactions include the purchase of diapers, 5% transactions include the purchase of beer. In the total transactions, 1.75% of them include both diapers and beer. In general, the occurrence of beer and diapers are very low, but in reality, if a person purchases beer, we have 87.5% change that he or she has also purchased diapers. We could potentially use association rules to identify possible word combinations for our dataset, and visualise the impact these association rules do to our class attribute. Pros of association rule mining include:the resulting rules are very intuitive and easy for human to understand; it does not require labeled data because its' purpose is to find strong associations among features, once we found associations between frequent words, we can then access them along with our class attribute to see if we have improved result; and finally, the algorithm itself is exhaustive, so it finds all the rules with specified support and confidence. Cons of association rules include: if our training data is too small, chances are our associations were found simply by chance; and it can be computationally expensive: because to calculating support we have to loop through the entire dataset. Typical association rule mining algorithms include: Apriori Algorithm and FP-growth Tree.

Market-Basket transactions

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Association Rules

{Diaper} → {Beer},
{Milk, Bread} → {Eggs,Coke},
{Beer, Bread} → {Milk},

Figure 24 Example of association rule mining

- Clustering/segmentation: This is another commonly used data mining method. The main idea is to group objects in the dataset into clusters according to their similarity. Similarity in clustering is defined in many ways, such as we calculate each data points' distance between each other, and if two data points are closer to each other compare to others, then we group them together, and then we do the same process for the rest of the dataset. For clustering, there is a common denominator: a group of data points, and then we use different clustering techniques to apply on them. It is an automatic method and therefore it is an unsupervised machine learning process. Pros of clustering include: it is relatively simple to implement, as there are many predefined clustering algorithms available out there; it can also scale to large data sets, and convergence is guaranteed (it will form clusters); it perform auto-correction on the choice of centroids' positions by iterations. Cons of clustering include: for datasets that contain noise and outliers, cluster's center can be affected by them and therefore lose accuracy; it also faces the problem of dimensionality: as the number of dimensions increases, the distance-based similarity will converge to a constant that makes no sense in terms of prediction. Typical clustering algorithms include: KNN, SVM.

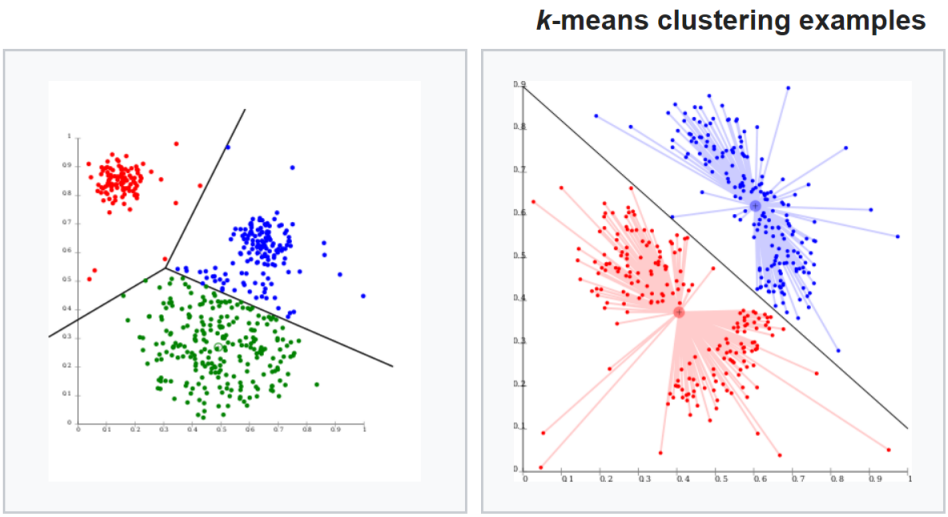


Figure 25 Clustering example

Now we have different types of methods to choose from, I will make my decision in the next section and clarify my reason of choosing it.

5.2 Select the appropriate data-mining method based on discussion

In order to serve the business objectives for this project, we mainly need to select data mining method and build data mining models based on correctly identify spam emails (accurately predict the class attribute based on instance's features). I would like to find which featured words or char or the overall length of an email have the most impact when the model predicts. Also, since we already have class attribute, then we consider our dataset to be more suitable for supervised learning. Therefore, in the above three data mining methods, I found classification to be the best fit to our business objectives. Because our dataset is structured to adapt supervised learning, we are trying to discover the relationship between all the attributes to our class attribute, nevertheless, we are able to build confusion matrix and get other statistical analysis such as Kappa statistic (measuring how random our result is); ROC and AUC to further expand our project depth. On the contrary, association rule mining is a form of unsupervised learning method, it mainly focusing on finding underlying associations between features, for instance if the frequent word "re" occurs, what is the probability that frequent char "%24" also occur. The rules found from this method can be used in our future studies such as studying word combinations and how they affects the behavior of spam email. However, in terms of our business objectives, since we are focusing on accurately predict class attribute, we save this method only for future study purpose. Similarly, for clustering method, it treat each data point as an individual, and grouping them according to predefined distance on a dimensional graph, it is also an

unsupervised learning method so we discard it. Therefore, I will be using classification method as our main data mining method and I am going to discuss the classification algorithm(s) I chose for this project in the next section.

6 Data-mining algorithm(s) selection

6.1 Conduct exploratory analysis and discuss

In the previous section, I discussed the most suitable data mining method for this project, that is classification. One thing that I need to point out is that in section 5.2, I discussed regression methodology is not suitable for this classification project. That left us with two choice: either decision tree or neural network. A traditional decision tree algorithm (Figure 26) is a data structure that built from one node and branches out by two. Each branch represents the outcome of a “test” on an attribute (if we get a frequent word “all”, the highest percentage in probability we get another frequent word “win”). Once the branching process is completed, each path from root to end leaf indicate one classification rule. The general logic flow is as follow: if rule 1 and rule 2 and rule 3 we can get classification attribute 1 (for instance). This algorithm itself is much easier to implement compare to neural network. It is also efficiency preserving and works well since it was developed decades ago but researchers and developers are still using it as one of the base classifiers in the real world. This algorithm also comes with some potential disadvantages that I need to address. For instance, if our target dataset contains too many features, the decision tree algorithm model may cause overfitting problem (too many branches but some may not be useful in the test set). Decision tree algorithm can be divided into two types: classification tree (for class attribute to be categorical) and regression tree (for class attribute to be continuous). The Cart (Classification and regression tree) algorithm is another form of decision tree, it combined the two types of decision trees and allow them to work on more complex datasets. It is a non-parametric decision tree solution that generate either of the two types of trees. It is then suitable for our project objectives. The mechanism behind this algorithm is to use information gain as key split factor. When a potential split provides the highest information gain among other candidates, it branches it. Each sub-sample defined by the previous split then get split again based on the next highest information gain. Finally, by iterations, until there is no potential split available, we get well-structured C5.0 tree, and the algorithm inspect from leaf nodes and trim the ones do not contribute enough to our C5.0 model. This process is called “post-pruning”. Besides, C5.0 can be used in categorical classification problems, so it is also a candidate for our algorithm selection. In general, the sklearn package provides a combined powerful decision tree classifier we can use.
Apart from single decision tree classifier, we can use some ensemble classifiers to increase the robustness of our learning model. Ensemble is a term used in statistics and machine

learning, which use multiple learning algorithms to obtain better prediction power or increase robustness for the overall prediction. All these learning algorithms can be used in machine learning by themselves. Typically, an ensemble machine learning approach consists a finite set of alternative models by providing more flexible structure to exist among these alternatives. The evaluation of an ensemble predictive power requires more computation than evaluating the predictive power of a single model. For instance, an ensemble system maybe more efficiency at increasing overall accuracy, computation, storage or communication resources compare to using individual learning algorithms along. Fast learning algorithms are often used such as decision trees in ensemble. Thus, I introduce an ensemble decision tree classifier: random forest. Random forest is a way of compiling multiple single decision trees altogether as a learning model. Each decision tree in the random forest spits out their own class predictions, and then the random forest uses a voting mechanism to vote out the most likely prediction by viewing each individual decision tree's predictive power or the majority votes (Figure random forest). However, there exists a potential problem for this ensemble algorithm: if all the decision trees were built in a similar fashion: for instance there are not many branch difference among decision trees, then all the trees would share similar predictive power (they will predict in the same way even if their prediction is wrong for some data points), and then the voting mechanism is no longer working as expected, and the random forest would perform like a single decision tree because all the trees are the same. Therefore, when constructing a random forest, there needs to be some actual signal in our features for better branch split, and the predictions or errors made by individual trees need to have low correlations with each other. Our dataset consists 60 features, according to our feature analysis, all of them have high feature scores to the class attribute, therefore use random forest as an ensemble learning approach is a viable way.

Another approach that pyspark provide is logistic regression classifier. In statistics, the logistic model can be applied for detecting the probability of a certain event existing. For instance, it can be used to detect pass or fail, win or lose, alive or dead, healthy or sick. Logistic regression is a statistical model that use logistic function to model a binary dependent variable. Binary dependent variable is the case where there are only two desired output from classification. For instance, if an image is a cat or dog; if an car is blue or red; or if a person is sick or not. Mathematically, a binary logistic model has a dependent variable with two possible values, these two values are labeled as 0 and 1, which is similar in our spam email case. Logistic regression is performed by apply logistic function to the binary class, and output the probability that one instance belong to certain class label. Thus, we can use this method as one of our base classifier since there is build-in package for it.

Therefore, through the above analysis, I have gained enough understanding on the algorithms that I am going to use. In the next section, I will try to decide which specific algorithm(s) I am going to use.

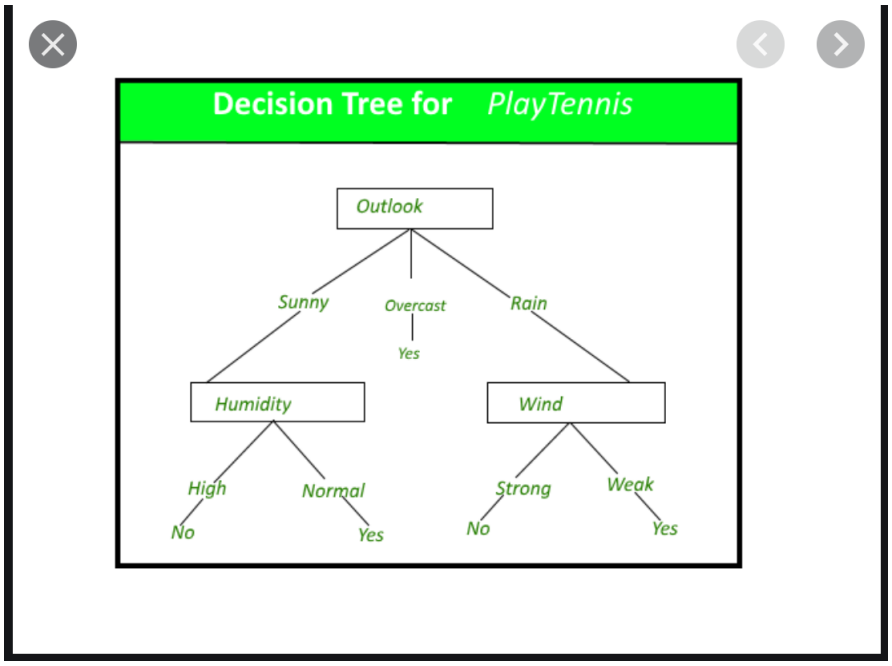


Figure 26 Example decision tree

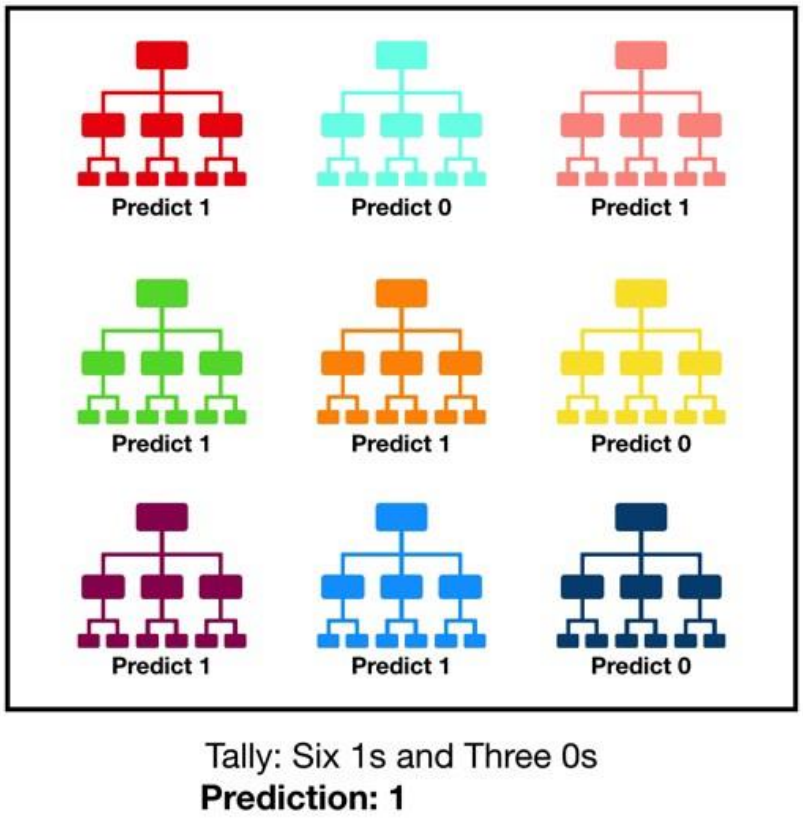


Figure random forest

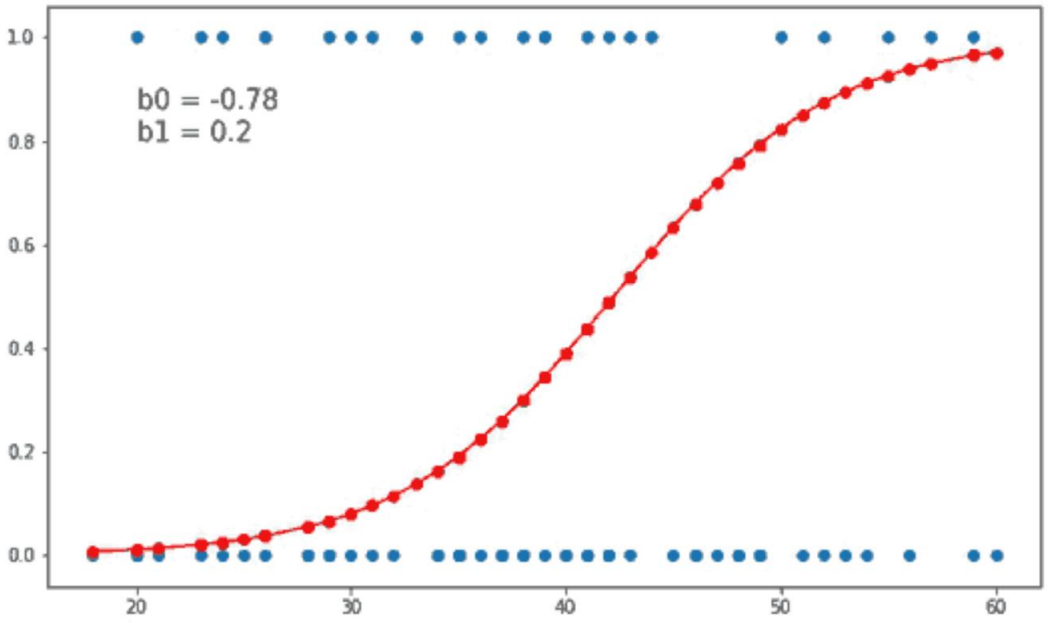


Figure 27 Example logistic regression

6.2 Select data-mining algorithms based on discussion

In section 6.1, three classification algorithms are identified to be suit for our project: decision tree and logistic regression. both are powerful data mining algorithms by default, I would like to try them individually on my dataset and see if there are differences in terms of performance. I believe both algorithms will perform well. However, one thing to note is that for fast developing data mining methodologies, where is a way to build even more powerful algorithm: random forest. In this case, we could not only get higher accuracy in terms of predicting, but also remain our model's stability. This process is called ensemble. Unlike a statistical ensemble in statistical world, ensemble in data mining is finite (only contain certain number of data mining models), while allow more flexible structure among other alternatives. Random forest as one of the most widely used ensemble mechanism, can be applied on our chosen algorithms to reduce discrepancies and provide a "voting" mechanism to avoid our models to be overfit, I have mentioned the potential overfitting problem our algorithms could bring in the previous section. Therefore, from this algorithm integration I can get a better overall prediction.

6.3 Build/Select appropriate model(s) and choose relevant parameter(s)

As discussed above, I will be mainly using decision tree, random forest and logistic regression as three of my core data mining algorithms. In this section I need to address their parameter settings since they are all supervised learning at this point. I decided to use more

than one parameter settings for all of the three algorithms because there will be differences in predictions when using different parameter settings, by using more sets of settings we can get more precise patterns on which pair or pairs of settings are the best fit to our data mining project. I will introduce them accordingly.
To begin with, for a single decision tree, by studying the pyspark build-in package, I can choose from how many branches I want for the tree structure. This parameter is called 'maxdepth', which is similar from sklearn package in python. This time, as I have enough prior knowledge, I would like to see how much number of branches matter to this machine learning problem does, so I set the max depth to 3 and 10 this time:

```
from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 3)
```

```
dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 10)
```

Also, I need to address the structure of decision tree output. In pyspark, the decision tree outputs three parameters, they are: prediction column, raw prediction column and probability column. The first one is double type, the other two are vectors. We need to use them to interpret our results later. I only focused on these parameters for now as they are the most relevant settings to be considered for this particular data mining project. As for the attribute side, they are all pretty standard, so we only focus on the factors mentioned above.

Similarly, pyspark also provides build-in random forest classifier. Since random forest is just an ensemble of individual decision trees, so apart from **criterion** controls the quality of a split. In other words, we can choose decision tree's split function (split according to what factor). We can either use Gini impurity or entropy for information gain as split factor. **max_depth** this parameter controls how deep does the decision tree grow. In other words: the height of decision tree. **featureSubsetStrategy** max features are also related to split phase; we can manually select how many features to be evaluated when doing a split. **random_state** also related to split, but only used when splitter is choosing to be "random", therefore, we disregard this setting since we want our learning model to not to be random. **numTrees** is where we choose the number of trees in the forest. Also, I need to address the structure of decision tree output. In pyspark, the decision tree outputs three parameters, they are: prediction column, raw prediction column and probability column. The first one is double type, the other two are vectors. We need to use them to interpret our results later. Therefore, I would like to use similar parameter settings compare to decision tree but tuning the number of estimators for random forest. For instance, I will set max depth as 10 if our

single decision tree is performing in maximum when its' depth is 10, but I will change number of trees to see if this parameter affects the classification outcome.

```
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label', numTrees = 10, maxDepth = 10)
```

```
In [130]: rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label', numTrees = 50, maxDepth = 10)
```

```
In [131]: rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label', numTrees = 100, maxDepth = 10)
```

Figure random forest settings

In this way, we can not only compare the impact number of estimators does to the whole ensemble, but also evaluate the overall performances of decision tree and random forest by comparing their results.

Finally, I also need to discuss the settings for logistic regression. For simplicity, in pyspark, we only need to set one hyperparameter: maxIter. This is how much time do we want to iterate the regression. It can be seen as fitting our data to the desired regression line, so it is a very important parameter. I will set the maximum iteration to 1, 5 and 10 to see if our classification results have much in difference.

```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)

lrModel = lr.fit(train)
```

```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)

lrModel = lr.fit(train)
```

Figure logistic regression settings

7 Data mining

7.1 Create and justify test design

Since we have decided the data mining algorithms we are going to use, now we have to construct a test design to evaluate our modelling result. In general, there is a commonly used terminology called cross validation in machine learning and data mining, where we randomly assign some portion of the whole dataset into training set, and the remaining part as test set. Our data mining model then gets trained on the training set to analysis instances' behaviors and feature importance, get them into model's "memory", and then we use our trained model on the test set: we basically let the model "guess" which attribute each instance belongs to judging from its' behaviors, finally we evaluate model accuracy and stability using tools such as confusion matrix and Kappa statistics. However, when split the dataset into training and test set, there are some building blocks that I need to address:

(1) I need to make sure that the most portion of data is assigned to the training set, because we need enough training samples to train our model, if we put reverse number of training samples say only 20% of the whole dataset, our model can only learn from limited number of instances, while other instances in the test set may contain other unseen features, and eventually causing our model to fail in the test process.
(2) Similarly, our test set cannot be set too small either, because if we only use say 5% of total data in the testing process, even if our model predicted all the instances correctly, we cannot say that our prediction result is statistically significant, because there is simply not enough statistical samples to make significant conclusion.
(3) We have to make sure our training and testing sets contain the same characteristics: same features, same logical distribution of the class attribute (they are from the same dataset so I do not need to worry this for now).

Through above building blocks, we understand that our training samples need to be large, but our test set cannot be too small, so I introduce the Pareto principle. This principle states that for many events, roughly 80% of the effects come from 20% of the causes. This concept is widely used in many areas of study: economics, sports, and of cause computing, data mining and machine learning. By evaluating the nature of this principle, I conclude that this ratio of split is suitable for this project, we can get a statistically significant result of our prediction.

```
train, test = dfs_for_model.randomSplit([0.8, 0.2], seed = 2018)

print("Training Dataset Count: " + str(train.count()))

print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 6164
Test Dataset Count: 1538
```

Figure training and test split

7.2 Conduct data mining

I now conduct data mining process on our dataset. I will use all the parameter settings I mentioned in section 6.3 and implement them accordingly. First, I will individually build models for Decision tree, neural network and random forest and try their parameter settings, secondly, I will use ensemble to combine them together and try the ensemble model's performance.

- Decision tree: In general, decision tree is a powerful algorithm in terms of classification. I have six different settings that vary from either the height of the tree or the maximum number of features used in split, the results are interesting and worth investigating: as we can see from below figures, no matter how many features used in split, when the max_depth (height) goes down, accuracy goes down as well. Which indicates that the number of splits is important in this data mining project, we need the decision tree to be somewhat "well-grew" to get higher predictive power. What is also interesting is, when consider more features while splitting, the accuracy dropped from 93.5% to 93.2%. This situation may cause by read in more irrelevant features actually decrease predictive power for decision tree.

```
from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 3)

dtModel = dt.fit(train)

dtPreds = dtModel.transform(test)

dtEval = BinaryClassificationEvaluator()

dtROC = dtEval.evaluate(dtPreds, {dtEval.metricName: "areaUnderROC"})

print("Test Area Under ROC decision tree max depth 3: " + str(dtROC))
```

```
Test Area Under ROC decision tree max depth 3: 0.7537666846576442
```

Figure decision tree 1

```
dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 5)

dtModel = dt.fit(train)

dtPreds = dtModel.transform(test)

dtEval = BinaryClassificationEvaluator()

dtROC = dtEval.evaluate(dtPreds, {dtEval.metricName: "areaUnderROC"})

print("Test Area Under ROC decision tree max depth 5: " + str(dtROC))
```

```
Test Area Under ROC decision tree max depth 5: 0.8160698907175014
```

Figure decision tree 2

```
dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 10)

dtModel = dt.fit(train)

dtPreds = dtModel.transform(test)

dtEval = BinaryClassificationEvaluator()

dtROC = dtEval.evaluate(dtPreds, {dtEval.metricName: "areaUnderROC"})

print("Test Area Under ROC decision tree max depth 10: " + str(dtROC))

print("")
```

```
Test Area Under ROC decision tree max depth 10: 0.917462866114981
```

Figure decision tree 3

- Random forest: For this algorithm, I also have six parameter settings. As we know, random forest as an ensemble method have two main strength: it is adaptive to change, in other words more robust to pattern change, it is also stabler often time compare to single algorithm. It is clearly shown below that all of the chosen parameters achieved higher than 95% accuracy with only minor differences. However, as the number of trees in the forest goes up, the accuracy dropped slightly. My estimation is that since our dataset is only a medium sized dataset, sometimes using less trees actually improve

overall predictive power. Therefore, parameter settings are dependent on data characteristics.

```
: from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label', numTrees = 10, maxDepth = 10)

rfModel = rf.fit(train)

rfPreds = rfModel.transform(test)

rfEval = BinaryClassificationEvaluator()

rfROC = rfEval.evaluate(rfPreds, {rfEval.metricName: "areaUnderROC"})

print("Test Area Under ROC: " + str(rfROC))
```

```
Test Area Under ROC for random forest 10: 0.9896630836986526
```

Figure random forest 1

```
0]: rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label', numTrees = 50, maxDepth = 10)

rfModel = rf.fit(train)

rfPreds = rfModel.transform(test)

rfEval = BinaryClassificationEvaluator()

rfROC = rfEval.evaluate(rfPreds, {rfEval.metricName: "areaUnderROC"})

print("Test Area Under ROC: " + str(rfROC))
```

```
Test Area Under ROC for random forest 50: 0.9912822716682717
```

Figure random forest 2

```
: rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label', numTrees = 100, maxDepth = 10)

rfModel = rf.fit(train)

rfPreds = rfModel.transform(test)

rfEval = BinaryClassificationEvaluator()

rfROC = rfEval.evaluate(rfPreds, {rfEval.metricName: "areaUnderROC"})

print("Test Area Under ROC: " + str(rfROC))
```

```
Test Area Under ROC for random forest 100: 0.9915715936565738
```

Figure random forest 3

- For logistic regression, I also have three different parameter settings. I will show their structure below:

```
from pyspark.ml.classification import LogisticRegression
import matplotlib.pyplot as plt
import numpy as np

lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)

lrModel = lr.fit(train)


print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))

print("")

from pyspark.ml.evaluation import BinaryClassificationEvaluator

lrEval = BinaryClassificationEvaluator()

print('Test Area Under ROC regression 10:', lrEval.evaluate(lrPreds))
```

```
Test Area Under ROC regression 10: 0.9624592030617393
```

Figure logistic regression

```
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=1)

lrModel = lr.fit(train)

lrPreds = lrModel.transform(test)

lrEval = BinaryClassificationEvaluator()

print('Test Area Under ROC regression 1:', lrEval.evaluate(lrPreds))
```

```
Test Area Under ROC regression 1: 0.937176521943637
```

Figure logistic regression

```
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=5)

lrModel = lr.fit(train)

lrPreds = lrModel.transform(test)

lrEval = BinaryClassificationEvaluator()

print('Test Area Under ROC regression 5:', lrEval.evaluate(lrPreds))
```

```
Test Area Under ROC regression 5: 0.9624592030617378
```

Figure logistic regression

I will explain my findings and patterns in the next section.

7.3 Search for patterns

After the modeling step, now I have to not only evaluate my models' performance, but also look deeply into my models' findings. To sum up their performance first, random forest achieved the highest performance in terms of accuracy, for my project, the model it built does not seem to be affected by how many folds I make, which indicates we got a reasonably good prediction. My ensemble model would perform better on unseen data (maybe the next version of this dataset) compare to decision tree itself. Neural network's accuracy is just slightly lower than random forest and sometimes decision tree, but when I actually run the individual model, it took a tremendous amount of time to run, I suspect that if we have more data, for instance 10 times more than our current dataset, neural network would be even slower. Cart has the lowest accuracy among others, but it is also in our acceptable range: as discussed in section 1.1: measure of success. Apart from general findings, I also did some research on how different algorithm select important features. My findings are attached in Figures below for decision tree and random forest. The general behaviors for both random forest and decision tree are similar: they recognize frequent word "%24", "000", "%21" and "remove" as one of the five most important predictors, which make sense as "%24" is the character symbol for $ sign, if an email contains too many $ signs, for instance: "click here to win $5,000 cash", then there is a high chance that this email is spam, and "000" often comes with money values such as "win $50,000 today", "remove" sometimes can be seen in virus warning spam emails: "Your PC is endanger, click here to remove virus". The predictor importance of neural network is somehow different, it is mainly because it uses different approach when calculating feature importance.

```
SparseVector(56, {0: 0.0043, 1: 0.0043, 2: 0.0011, 3: 0.0013, 4: 0.0137, 5: 0.0028, 6: 0.1162, 7: 0.0024, 8: 0.0037, 9: 0.0049,
10: 0.0042, 11: 0.0014, 12: 0.0015, 13: 0.0001, 14: 0.0054, 15: 0.0385, 16: 0.0163, 17: 0.0038, 18: 0.0055, 20: 0.0028, 21: 0.0
035, 22: 0.0405, 23: 0.1186, 25: 0.0236, 26: 0.0006, 27: 0.0007, 28: 0.0007, 31: 0.0008, 33: 0.0014, 34: 0.0038, 35: 0.0038, 3
7: 0.0028, 39: 0.0012, 40: 0.0101, 42: 0.0, 43: 0.0167, 44: 0.0289, 46: 0.0035, 47: 0.0015, 48: 0.0027, 50: 0.3965, 51: 0.0152,
53: 0.0876, 54: 0.001})
```

```
print("")

dtModel.featureImportances
```

Figure importance scores for the best decision tree

```
SparseVector(56, {0: 0.0019, 1: 0.0137, 2: 0.0047, 3: 0.0002, 4: 0.016, 5: 0.0005, 6: 0.0944, 7: 0.0071, 8: 0.0017, 9: 0.0045,
10: 0.0032, 11: 0.0051, 12: 0.0009, 13: 0.0003, 14: 0.0009, 15: 0.0444, 16: 0.0072, 17: 0.005, 18: 0.0154, 19: 0.0101, 20: 0.11
4, 21: 0.0281, 22: 0.0766, 23: 0.1063, 24: 0.0212, 25: 0.0404, 26: 0.0015, 27: 0.0005, 28: 0.0086, 29: 0.0007, 30: 0.001, 31:
0.0027, 32: 0.0007, 33: 0.0091, 34: 0.003, 35: 0.0121, 36: 0.0007, 37: 0.0019, 38: 0.0005, 39: 0.0019, 40: 0.0078, 41: 0.0013,
42: 0.0018, 43: 0.0069, 44: 0.0204, 45: 0.0004, 46: 0.0017, 47: 0.0028, 48: 0.0078, 49: 0.0013, 50: 0.0885, 51: 0.0516, 52: 0.0
018, 53: 0.1132, 54: 0.0097, 55: 0.0142})
```

```
print("Test Area Under ROC for random fore

rfModel.featureImportances
```

Figure importance scores for the best random forest

```
dfs.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- word_freq_make: double (nullable = true)
 |-- word_freq_address: double (nullable = true)
 |-- word_freq_all: double (nullable = true)
 |-- word_freq_3d: double (nullable = true)
 |-- word_freq_our: double (nullable = true)
 |-- word_freq_over: double (nullable = true)
 |-- word_freq_remove: double (nullable = true)
 |-- word_freq_internet: double (nullable = true)
 |-- word_freq_order: double (nullable = true)
 |-- word_freq_mail: double (nullable = true)
 |-- word_freq_receive: double (nullable = true)
 |-- word_freq_will: double (nullable = true)
 |-- word_freq_people: double (nullable = true)
 |-- word_freq_report: double (nullable = true)
 |-- word_freq_addresses: double (nullable = true)
 |-- word_freq_free: double (nullable = true)
 |-- word_freq_business: double (nullable = true)
 |-- word_freq_email: double (nullable = true)
 |-- word_freq_you: double (nullable = true)
 |-- word_freq_credit: double (nullable = true)
 |-- word_freq_your: double (nullable = true)
 |-- word_freq_font: double (nullable = true)
 |-- word_freq_000: double (nullable = true)
 |-- word_freq_money: double (nullable = true)
 |-- word_freq_hp: double (nullable = true)
 |-- word_freq_hpl: double (nullable = true)
 |-- word_freq_george: double (nullable = true)
 |-- word_freq_650: double (nullable = true)
 |-- word_freq_lab: double (nullable = true)
 |-- word_freq_labs: double (nullable = true)
 |-- word_freq_telnet: double (nullable = true)
```

```
|-- word_freq_lab: double (nullable = true)
|-- word_freq_labs: double (nullable = true)
|-- word_freq_telnet: double (nullable = true)
|-- word_freq_857: double (nullable = true)
|-- word_freq_data: double (nullable = true)
|-- word_freq_415: double (nullable = true)
|-- word_freq_85: double (nullable = true)
|-- word_freq_technology: double (nullable = true)
|-- word_freq_1999: double (nullable = true)
|-- word_freq_parts: double (nullable = true)
|-- word_freq_pm: double (nullable = true)
|-- word_freq_direct: double (nullable = true)
|-- word_freq_cs: double (nullable = true)
|-- word_freq_meeting: double (nullable = true)
|-- word_freq_original: double (nullable = true)
|-- word_freq_project: double (nullable = true)
|-- word_freq_re: double (nullable = true)
|-- word_freq_edu: double (nullable = true)
|-- word_freq_table: double (nullable = true)
|-- word_freq_conference: double (nullable = true)
|-- char_freq_%3B: double (nullable = true)
|-- char_freq_%28: double (nullable = true)
|-- char_freq_%5B: double (nullable = true)
|-- char_freq_%21: double (nullable = true)
|-- char_freq_%24: double (nullable = true)
|-- char_freq_%23: double (nullable = true)
|-- capital_run_length_longest: integer (nullable = true)
|-- capital_run_length_total: integer (nullable = true)
|-- class: integer (nullable = true)
```

Figure all selected features, the scores are listed in this order

Also, comparing the end results, I discovered that as number of max depth goes up, the decision tree accuracy goes up, which means that the branch is a critical factor that affecting decision tree learning outcome:

```
Test Area Under ROC decision tree max depth 3: 0.7537666846576442
```

```
Test Area Under ROC decision tree max depth 5: 0.8160698907175014
```

```
Test Area Under ROC decision tree max depth 10: 0.917462866114981
```

Similarly, for random forest, although the number of trees does not seem to have a huge impact on the overall accuracy because it is already predicting in high accuracy, we can still visualise minor increase when we change number of trees:

```
Test Area Under ROC for random forest 10: 0.9896630836986526
```

```
Test Area Under ROC for random forest 50: 0.9912822716682717
```

```
Test Area Under ROC for random forest 100: 0.9915715936565738
```

Besides, for logistic regression, our key parameter: max iteration increases, and so as the overall accuracy:

```
Test Area Under ROC regression 10: 0.9624592030617393
```

```
Test Area Under ROC regression 1: 0.937176521943637
```

```
Test Area Under ROC regression 5: 0.9624592030617378
```

Thus, we can conclude that our chosen parameter settings do have impact on overall accuracy.

8 Interpretation

8.1 Study and discuss the mined patterns

In this section, we will discuss what our models or model combination explored from the previous steps and we will also need to evaluate if our business objectives can be successfully reached through our mining process. From our mining result from section 7, we discovered that random forest algorithm achieved the highest accuracy in classifying the class attribute, which was 95.9% in the test set. The business objectives for this project is to build a data mining model that achieve 80% accuracy and above, therefore, all our learning models have fulfilled the objectives. I will use the top three most important predictors random forest selected as the deciding factors of this dataset: word_freq_remove, char_freq_%24 and char_freq_%21. However, we cannot make any conclusion on whether these three predictors are common deciding factors for this dataset or there exists other combinations of predictors that may lead to even higher accuracy. I will try to make some statistical analysis on the discovered data patterns to observe more accurate information. I can try some trial tests on the model, use it for field predictions (categorical attributes that I created in section 3) and evaluate it. A better model may thrive through continuous verification and trial tests (Maybe applied in datasets with different conditions such as imbalanced dataset).

8.2 Visualise the data, results, models, and patterns

In this section, we need to look back to our prior steps and re-evaluate data, results, models, and patterns through this process. First, we need to plot the raw class distribution from section 4, where originally, we have an slightly imbalanced class attribute distribution (Figure 49). We can visualise that real emails are more than spam emails in terms of number of instances. The data transformation approach that I chose involved in increase the number of overall instances. Regarding data points, all the instances are considered as completed, as I mentioned in section 2.4, when we performed data quality verification, there are only in total 5 NA values exist for the whole dataset, and we used the dropna() function in Python to deal with them. Both Figures provide some preliminary understanding of the dataset.

```
print("Number of rows for version 1: ", dfs.count())
print("Number of rows for version 2: ", dfs1.count())

dfs_combine = dfs.union(dfs1)
print("Number of rows for combined dataframe: ", dfs_combine.count())
```

```
Number of rows for version 1:  4601
Number of rows for version 2:  3101
Number of rows for combined dataframe:  7702
```

Figure combined dataset total records

```
from pyspark.sql.functions import mean

mean_make = dfs1.select(mean(dfs1.word_freq_make)).collect()[0][0]
mean_receive = dfs1.select(mean(dfs1.word_freq_receive)).collect()[0][0]

mean = {'word_freq_make': mean_make, 'word_freq_receive': mean_receive}
dfs1 = dfs1.na.fill(mean)

dfs1.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in dfs1.columns]).show()
```

Figure check missing value after preprocessing

In the data exploration step (section 2.3), I analysed some features' relationship with the class attribute. Those features were selected from Python, after we finished our whole data mining process above, we got three different important features: word_freq_remove, word_freq_hp and word_freq_%21. Since all of the three features are continuous, I made some modifications on them to let them become categorical type so we can have a more intuitive understanding of them (because the key range for these ratio type of features are from 0 to 1, and we have some outliers that with extreme values such as 1000, so we cannot visualize the direct impact these three features make). The processed feature analysis for the three features are attached as Figure 51, 52, and 53 below. As we can see the transformed

data types of the three features, the majority group is low (where the ratio of frequent word/total word count * 100) is in range 0 to 0.005. In this range, all of the three features contain both real and spam emails, which indicate it is a normal range. This is also a key conclusion we can make from spam attack, if the misleading words only occur in the total email with no larger than 0.5%, it means us human and computer would not recognize a potential spam attack, because if we use our conclusion, for a specific key word, it appeared in a normal range. As the ratio goes up, we see that for all of the three frequent words, there is nearly no or very small number of real email can be found. Therefore, we can reach a conclusion and patterns that as the number of repeated words goes up in an email, the chance of us getting a spam attack also goes up. This is probably because spammers tend to use auto tools to randomly fill in an email, they set the key target words themselves and make software to generate the rest of the content. Again, this step is done by to_pandas() since pyspark does not support direct visualisation.

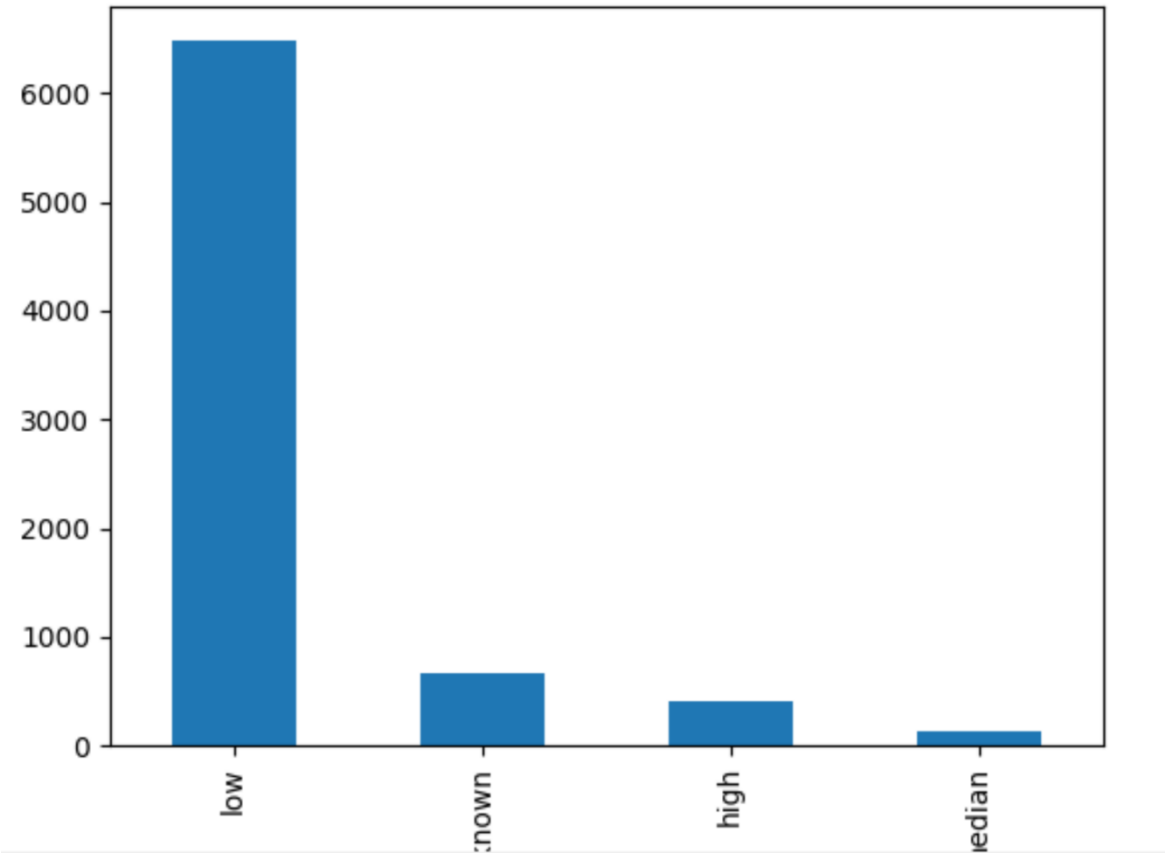


Figure regrouping for "000"

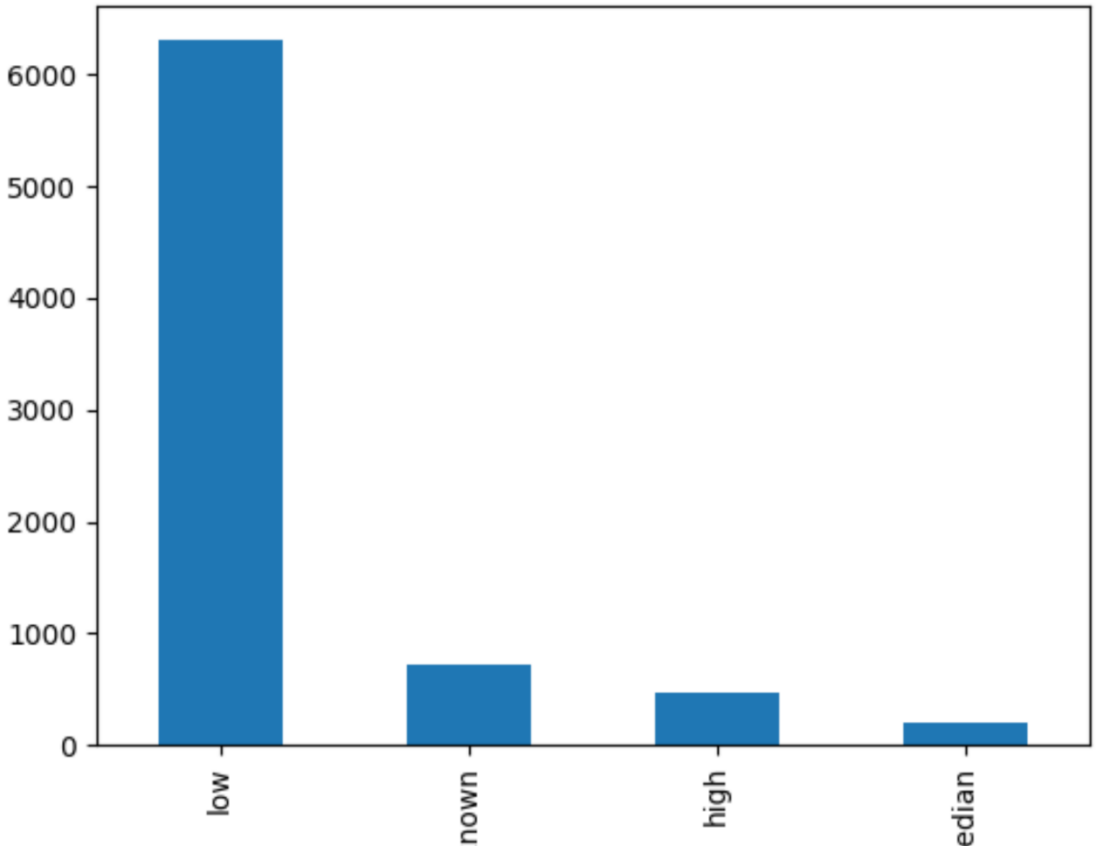


Figure regrouping for "remove"

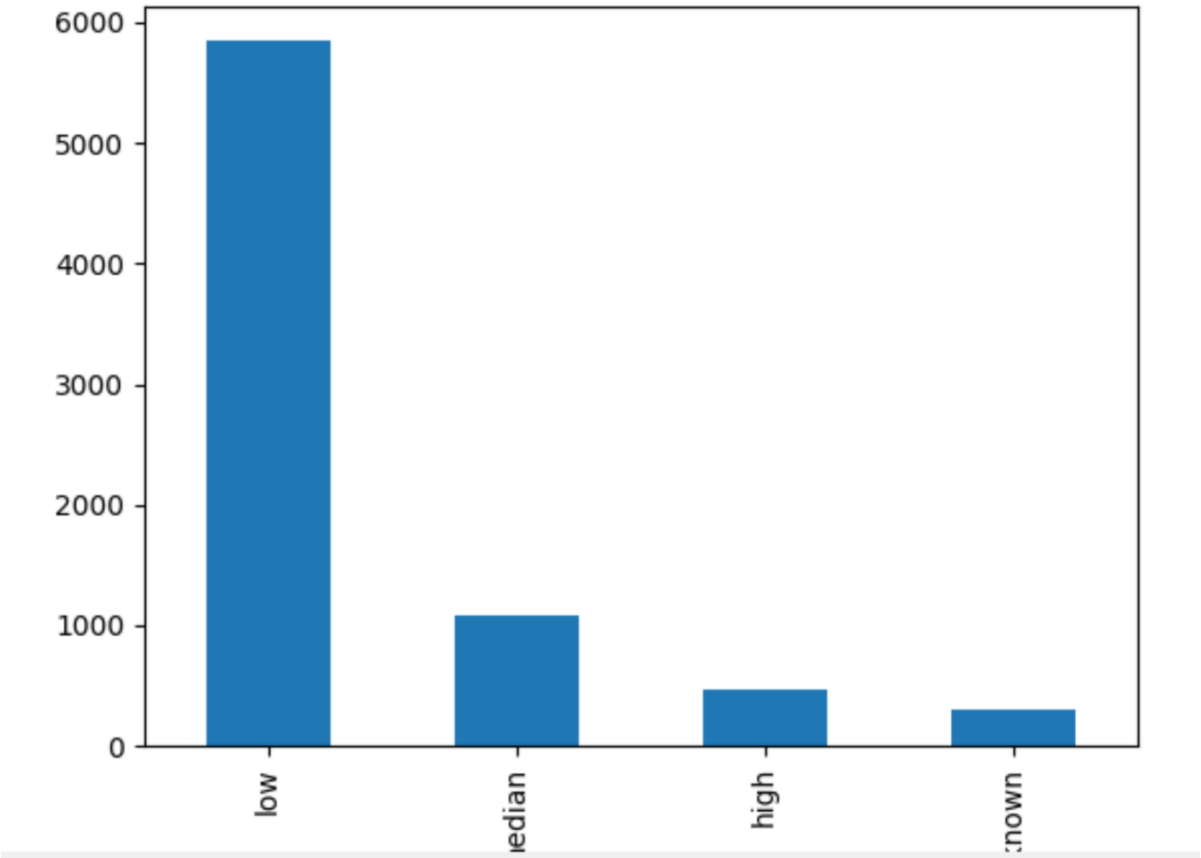


Figure regrouping for "%24"

Finally, we will need to visualize our models to gain more detailed information about the data. First, I present the tree structures for decision tree algorithm. To be specific, I will plot the tree structure for max_depth (height) for decision tree as 2 and plot the tree structure for decision tree which was set to have 10 max_depth. Because by changing the parameters, the algorithm will build tree models according to predefined parameter settings. For instance, if we select the max_depth or other changes in settings, we will have totally different structures. Therefore, it is essential for me to visualize my models by using my chosen parameter settings.

```
print(dtModel.toDebugString)
```

```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4366a7fea5c959b64c13) of depth 10 with 253 nodes
  If (feature 50 <= 0.073)
   If (feature 6 <= 0.04)
    If (feature 22 <= 0.0)
     If (feature 15 <= 0.1)
      If (feature 51 <= 0.173)
       If (feature 23 <= 0.0)
        If (feature 53 <= 10.0)
         If (feature 10 <= 0.21)
          If (feature 54 in {1.0,2.0})
           If (feature 0 <= 0.0)
            Predict: 1.0
           Else (feature 0 > 0.0)
            Predict: 0.0
          Else (feature 54 not in {1.0,2.0})
           If (feature 20 <= 0.51)
            Predict: 0.0
           Else (feature 20 > 0.51)
            Predict: 0.0
```

Figure for max_depth 10 tree

```
print(dtModel.toDebugString)
```

```
Test Area Under ROC decision tree max depth 3: 0.7537666846576442
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4cff80116db5e28991ad) of depth 3 with 15 nodes
  If (feature 50 <= 0.073)
   If (feature 6 <= 0.04)
    If (feature 22 <= 0.0)
     Predict: 0.0
    Else (feature 22 > 0.0)
     Predict: 1.0
   Else (feature 6 > 0.04)
    If (feature 25 <= 0.0)
     Predict: 1.0
    Else (feature 25 > 0.0)
     Predict: 0.0
  Else (feature 50 > 0.073)
   If (feature 23 <= 0.17)
    If (feature 53 <= 10.0)
     Predict: 0.0
    Else (feature 53 > 10.0)
     Predict: 1.0
   Else (feature 23 > 0.17)
    If (feature 50 <= 0.692)
     Predict: 0.0
    Else (feature 50 > 0.692)
     Predict: 1.0
```

Figure for max_depth 3 tree

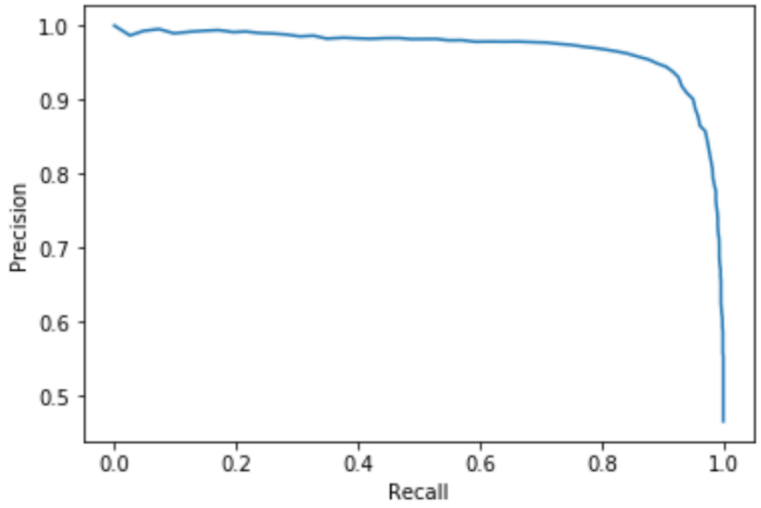
From above figures we can see that when choosing different parameter settings, the structure changed totally, since the second tree only have 3 splits, predictive power drop significantly. Also, as my random forest settings are like my best decision tree settings, there is no need to plot individual trees from the forest as we already have a sample above.

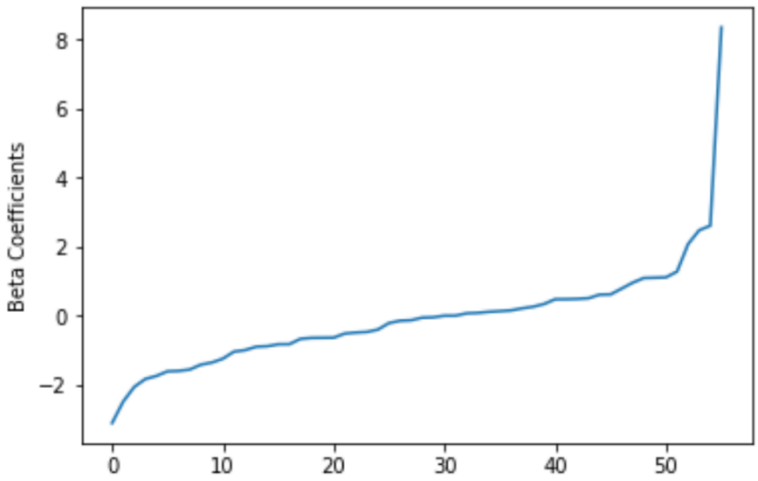
We can also visualise the direct prediction our models make. For instance, we can show 10 of our logistic regression prediction made by our model:

```
+-------------+------------+---------------+------------+--------------------+----------+--------------------+
|word_freq_make|word_freq_000|word_freq_remove|char_freq_%24|       rawPrediction|prediction|         probability|
+-------------+------------+---------------+------------+--------------------+----------+--------------------+
|         0.04|         0.0|            0.0|         0.0|[0.96062592591377...|       0.0|[0.72324710841663...|
|         0.02|         0.0|            0.0|       0.004|[3.74072887117859...|       0.0|[0.97681357566056...|
|         0.08|         0.0|           0.04|         0.0|[3.65738656550759...|       0.0|[0.97484904009949...|
|         0.09|         0.0|            0.0|       0.041|[3.70919457729772...|       0.0|[0.97608851945426...|
|         0.11|         0.0|            0.0|       0.029|[3.32749242312963...|       0.0|[0.96536001397990...|
|          0.2|         0.0|            0.0|       0.018|[5.90884306283747...|       0.0|[0.99729202649927...|
|         0.31|         0.0|            0.0|         0.0|[3.95949751852324...|       0.0|[0.98128426397495...|
|         0.27|         0.0|            0.0|         0.0|[3.00723594796779...|       0.0|[0.95289995433906...|
|         0.23|        0.23|            0.0|       0.205|[-4.0152461779508...|       1.0|[0.01771889087087...|
|         0.15|        0.05|            0.0|         0.0|[1.89411290148605...|       0.0|[0.86922377013320...|
+-------------+------------+---------------+------------+--------------------+----------+--------------------+
only showing top 10 rows
```

```
lrPreds = lrModel.transform(test)
lrPreds.select('word_freq_make', 'word_freq_000',
               'word_freq_remove', 'char_freq_%24',
               'rawPrediction', 'prediction', 'probability').show(10)
```

We can also plot the beta coefficients and precision and recall for our logistic regression model:






```
print('Test Area Under ROC regression 10:', lrEval.evaluate(lrPreds))

pr = trainingSummary.pr.toPandas()
plt.plot(pr['recall'],pr['precision'])
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.show()

beta = np.sort(lrModel.coefficients)
plt.plot(beta)
plt.ylabel('Beta Coefficients')
plt.show()
```

8.3 Interpret the results, models, and patterns

Through the data and results visualization process, I have identified some of the deciding factors that add to the success for the business objectives of this project as well as the class attribute distribution for the three important predictors. Since they are all continuous type data, I used the derive function in Python to restructure them, made them more accessible

and more understandable to us end users. From section 8.2, we can see some direct impact of these three predictors. The occurrence of spam emails is closely related to the word count ratio of these words or char. If the ratio of word occurrence is above my predefined threshold, which is 0.0005, then this data point or this email is almost always identified as spam email according to our class attribute. This is one of the key reasons these three predictors are considered as deciding factors for the selected algorithms. Since almost every data point in our dataset follow certain logic flow, for instance not many noisy data in our dataset, the classification algorithms can perform well in this situation.
On the model side, I also plotted the structures for each chosen algorithm. As I mentioned in section 8.2, if we chose different parameter settings, we will get totally different model structures even though we are analysis the same dataset. This is because supervised learning algorithms are very dependent to parameter settings, especially for tree type algorithms, by only change a small portion of node threshold can cause a butterfly effect on the whole model. By tuning the right parameters, we get both our decision tree and random forest to have the same root node and same first split: word_freq_remove and word_freq_000. If we set the parameter too large, the tree would have a shape of skewing to left or right as the algorithm cannot split there is simply no feature satisfy the minimum requirement for one split. For neural network, contains 10 layers, each neuron is assigned to all the selected features from the algorithm itself, and for some neurons, certain words have positive correlation to it while other words have negative correlations to it. Other neurons may take words other neurons consider as negative correlated to be a positive correlation. This is because the principle of reassigning weight, when we train our neural network, we will compare the output each neuron generated with the actual class value, if a neuron guessed a certain input wrong, then as a penalty, the neuron is reassigned in weight. So each neuron may have their own reaction when come across different input.

8.4 Assess and evaluate results, models, and patterns

In Chapter 7 we have assessed the results from different algorithms along with their combined bagging model. By selecting the analysis node from output bar, we get all the accuracy we needed, which are shown in section 7.2. I chose random forest algorithm as it provides highest prediction accuracy and it is very efficient compare to neural network. Then I estimated the most important predictors these algorithms generated. I can use more statistic tools to help better understand our results and models. For instance, I can use sklearn to plot ROC for the algorithms (Figures below), which represent ROC for decision tree, random forest and logistic regression.

```
Out[70]: [<matplotlib.lines.Line2D at 0x7fc88ac379b0>]
```

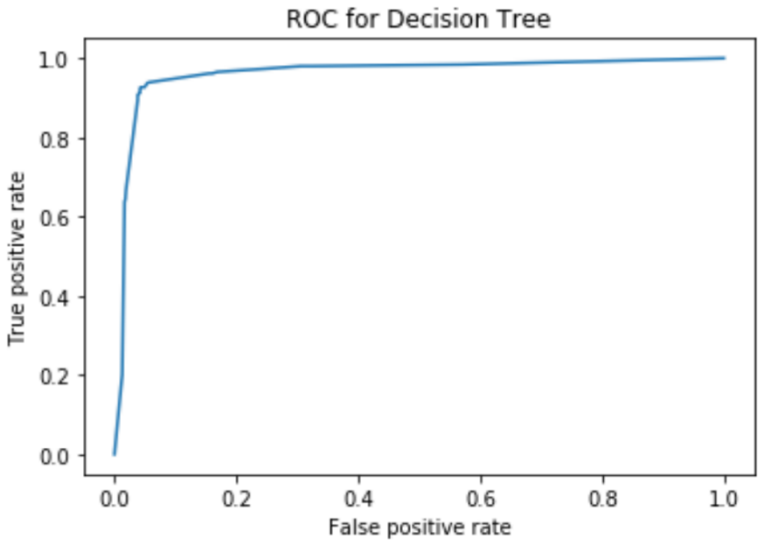


Figure ROC for decision tree

```
Out[71]: [<matplotlib.lines.Line2D at 0x7fc88b05e0f0>]
```

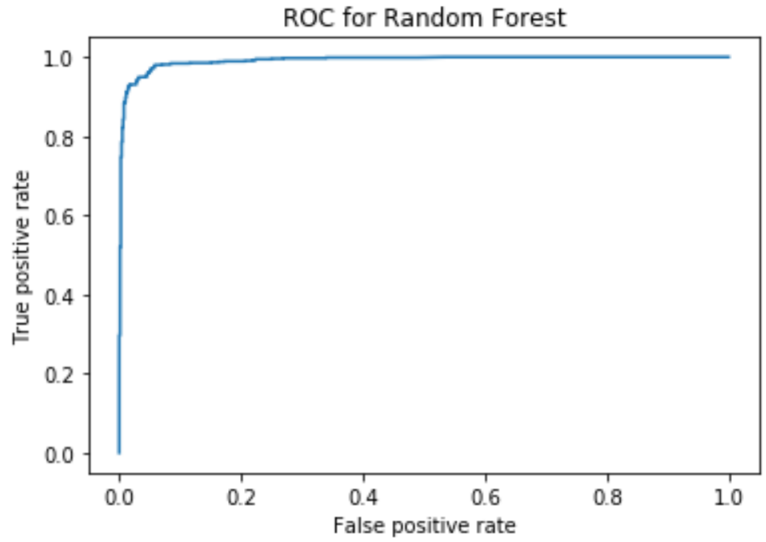


Figure ROC for random forest

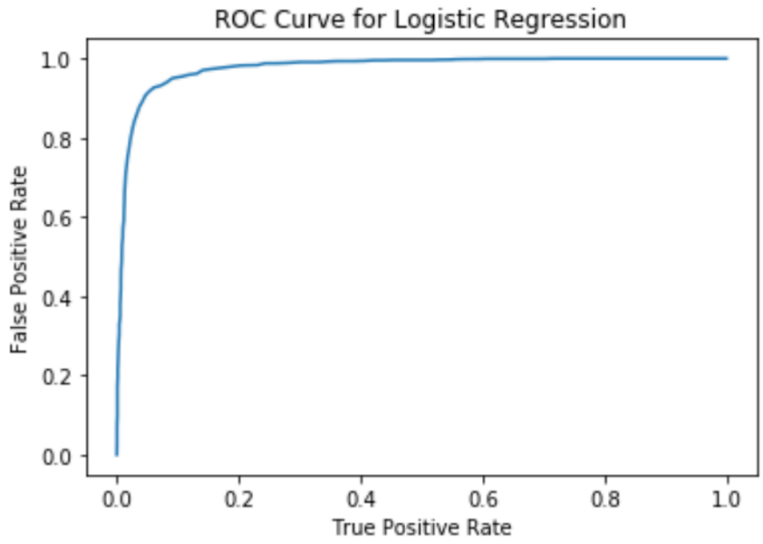


Figure ROC for logistic regression

Note that I used two different methods when plotting ROC for decision tree, random forest and logistic regression. They are:

```
class CurveMetrics(BinaryClassificationMetrics):
    def __init__(self, *args):
        super(CurveMetrics, self).__init__(*args)

    def _to_list(self, rdd):
        points = []

        for row in rdd.collect():
            # Results are returned as type scala.Tuple2,
            # which doesn't appear to have a py4j mapping
            points += [(float(row._1()), float(row._2()))]
        return points

    def get_curve(self, method):
        rdd = getattr(self._java_model, method)().toJavaRDD()
        return self._to_list(rdd)
```

```
results = dtPreds.select(['probability', 'label'])

results_collect = results.collect()

preds = results.rdd.map(lambda row: (float(row['probability'][1]), float(row['label'])))
points = CurveMetrics(preds).get_curve('roc')

plt.figure()
x_val = [x[0] for x in points]
y_val = [x[1] for x in points]
plt.title('ROC for Decision Tree')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot(x_val, y_val)
```

```
results = rfPreds.select(['probability', 'label'])

results_collect = results.collect()

preds = results.rdd.map(lambda row: (float(row['probability'][1]), float(row['label'])))
points = CurveMetrics(preds).get_curve('roc')

plt.figure()
x_val = [x[0] for x in points]
y_val = [x[1] for x in points]
plt.title('ROC for Random Forest')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot(x_val, y_val)
```

Figure plotting ROC for decision tree and random forest

```
trainingSummary = lrModel.summary
lrROC = trainingSummary.roc.toPandas()

plt.plot(lrROC['FPR'],lrROC['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression')
plt.show()
```

Figure plotting ROC for logistic regression

I used different approaches because unlike logistic regression, decision tree and random forest does not have training summary, so we need to manually extract information out.

The explanation for ROC is that if the blue curve is close to the diagonal line, it indicates that this model is not doing well in terms of accuracy. On the contrary, when look at the ROC curve for decision tree, we can see that the blue curve is closer to the top left corner compare to the other two algorithms, it is because when the curve is as far away from the diagonal, the model performs better.
As we can see from the three ROC curves, they are all performing well, though random forest curve covered the most areas, therefore we can use random forest as base learning algorithm for future research.
We can also visualise the confusion matrix for the three machine learning algorithms:

```
Confusion matrix for decision tree:
[[754.  33.]
 [ 57. 694.]]
```

```
Confusion matrix for random forest:
[[773.  14.]
 [ 53. 698.]]


Confusion matrix for logistic regression:
[[747.  40.]
 [ 68. 683.]]
```

These confusion matrices are used to detect type 0 and type 1 error so we have a more in depth understanding of our classifier. This step is done by:

```
import pyspark.sql.functions as F
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.sql.types import FloatType

# #important: need to cast to float type, and order by prediction, else it won't work
preds_and_labels = lrPreds.select(['prediction','label']).withColumn('label', F.col('label').cast(FloatType())).orderBy('predicti

#select only prediction and label columns
preds_and_labels = preds_and_labels.select(['prediction','label'])

metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))

print("Confusion matrix for logistic regression:")

print(metrics.confusionMatrix().toArray())
```

After changing the above parameters, I built a boosting ensemble model that mainly focusing on improving accuracy. I would like to visualize if there is any actual improvement boosting method makes, and what features it select as important differ from our previous models.
However, when I compare the new model with previous model, they have the same accuracy as well as the same important features. I will not attach the result of new model as there is no visible difference between the two models. I will keep the most important features that decision selected: word_freq_remove, word_freq_hp and word_freq_%21, and I will also conclude that we have built a well performed model(s): random forest.

8.5 Iterate prior steps (1 to 7) as required

- In the first section, I mainly decided to choose predicting spam emails among normal emails as my project topic. Then I analyzed the current business situation, I also evaluated potential risks and other related key factors that would potentially affect my data mining project. Next, I set a standard to determine if my later work fit my business objectives or not, I will iterate the whole business process if the predefined business objectives are not met regarding data mining. Finally, I created a gante graph to distribute workload for my whole project and I will process my workflow according to my project schedule.

- The next section, I made preliminary investigation on the type of data I need, then search and collect initial data. I provided an introduction on the raw data that I collected in plain language such as the structure of the dataset, and how do our data mining model would work on my dataset (by using values in features to analyze each class attribute's behaviour) and made some preliminary research on selecting important features by just investigate the data on the surface. I also made some plots to visualize some obvious relationships between features and class attribute.

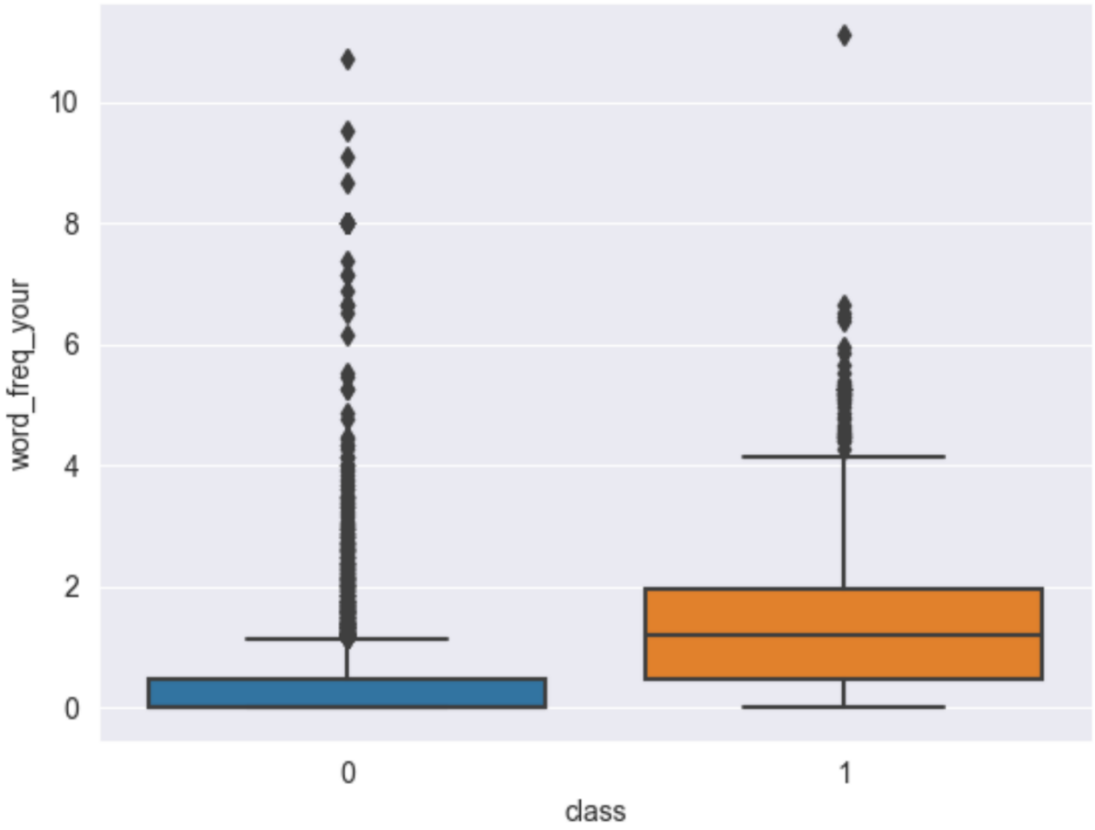


Figure frequency vs class distribution for word_your

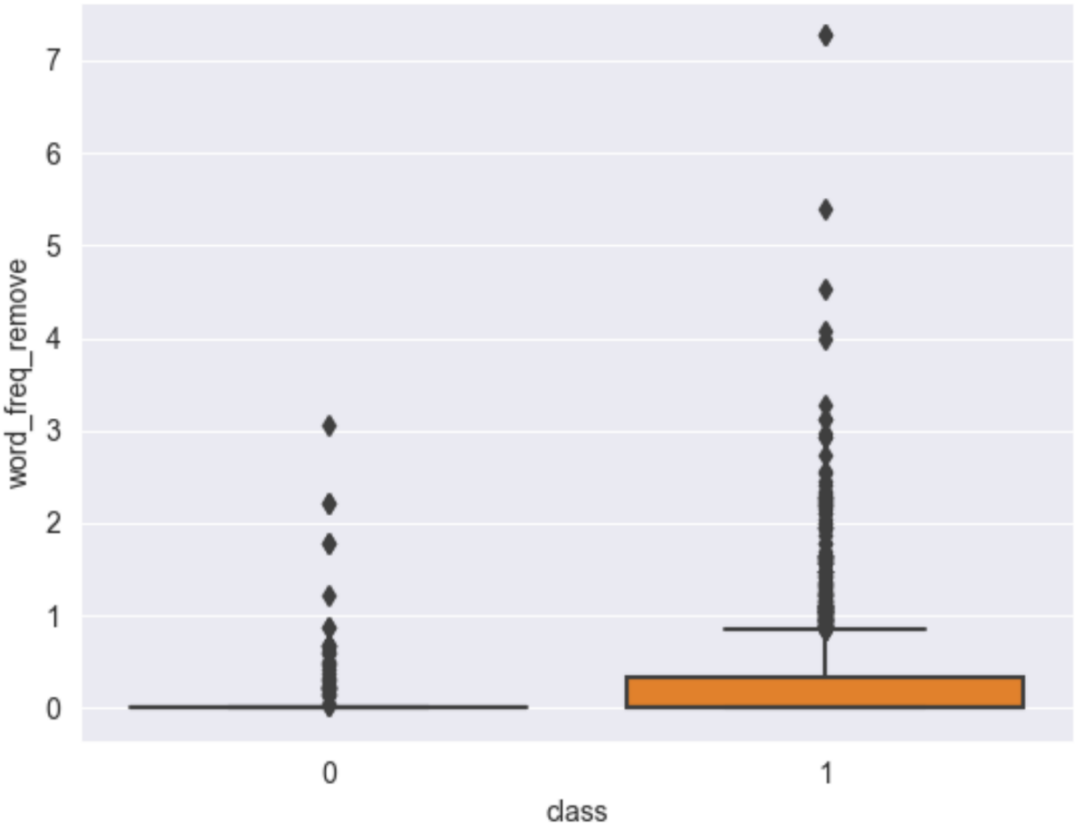


Figure frequency vs class distribution for word_remove

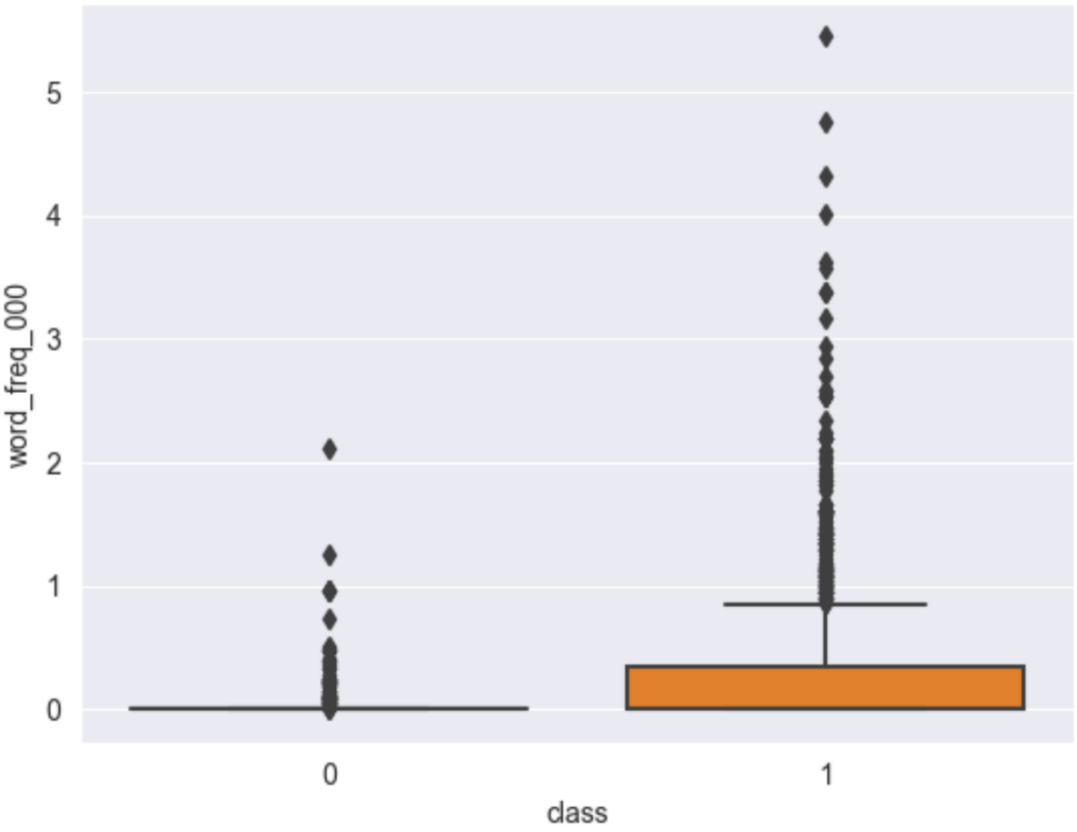


Figure frequency vs class distribution for word_000

Finally, I made some brief investigations on data quality to see if our dataset needs any data preprocessing before we go into the next phase.

- At the beginning of the third section, I evaluated our preliminary result of our raw dataset and suggested that the dataset is already complete. So by selecting the data, we directly use our raw data, and we performed data cleaning by noise analysis and outlier and extreme value removal step.

```
from pyspark.sql.functions import mean

mean_make = dfs1.select(mean(dfs1.word_freq_make)).collect()[0][0]
mean_receive = dfs1.select(mean(dfs1.word_freq_receive)).collect()[0][0]

mean = {'word_freq_make': mean_make, 'word_freq_receive': mean_receive}
dfs1 = dfs1.na.fill(mean)

dfs1.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in dfs1.columns]).show()
```

```
--------+-----------------+------------------+-----------------+------------+-------------+---------------+--------------------
+-------------+-------------+-------------+-------------+-------------+-------------+-------------------------+-------------
-----------+------------------------+----------+---------+-----+
|_c0|word_freq_make|word_freq_address|word_freq_all|word_freq_3d|word_freq_our|word_freq_over|word_freq_remove|word_freq_intern
et|word_freq_order|word_freq_mail|word_freq_receive|word_freq_will|word_freq_people|word_freq_report|word_freq_addresses|word_f
req_free|word_freq_business|word_freq_email|word_freq_you|word_freq_credit|word_freq_your|word_freq_font|word_freq_000|word_fre
q_money|word_freq_hp|word_freq_hpl|word_freq_george|word_freq_650|word_freq_lab|word_freq_labs|word_freq_telnet|word_freq_857|w
ord_freq_data|word_freq_415|word_freq_85|word_freq_technology|word_freq_1999|word_freq_parts|word_freq_pm|word_freq_direct|word
_freq_cs|word_freq_meeting|word_freq_original|word_freq_project|word_freq_re|word_freq_edu|word_freq_table|word_freq_conference
|char_freq_%3B|char_freq_%28|char_freq_%5B|char_freq_%21|char_freq_%24|char_freq_%23|capital_run_length_average|capital_run_len
gth_longest|capital_run_length_total|word_trial|word_test|class|
+---+--------------+-----------------+-------------+------------+-------------+--------------+----------------+----------------
--+---------------+--------------+-----------------+--------------+----------------+----------------+-------------------+------
--------+------------------+---------------+-------------+----------------+--------------+--------------+-------------+-------
-------+------------+-------------+----------------+-------------+-------------+--------------+----------------+-------------+-
------------+-------------+------------+--------------------+--------------+---------------+------------+----------------+----
--------+-----------------+------------------+-----------------+------------+-------------+---------------+--------------------
+-------------+-------------+-------------+-------------+-------------+-------------+-------------------------+-------------
-----------+------------------------+----------+---------+-----+
|  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|               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|            0|            0|                       0|                    
0|         0|        0|    0|
+---+--------------+-----------------+-------------+------------+-------------+--------------+----------------+----------------
--+---------------+--------------+-----------------+--------------+----------------+----------------+-------------------+------
--------+------------------+---------------+-------------+----------------+--------------+--------------+-------------+-------
-------+------------+-------------+----------------+-------------+-------------+--------------+----------------+-------------+-
------------+-------------+------------+--------------------+--------------+---------------+------------+----------------+----
```

I then tried to reconstruct some of the data format to better fit our data mining goal. Finally, I evaluated that there are not enough training instances and our dataset is imbalanced, so I added the new version of our current dataset as our data preprocessing final step.

- In this section, I first dealt with missing value, although missing values in this dataset are not significant, we still need to deal with them for better predictive power. We then evaluated feature by evaluating their correlations to each other.

	_c0	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove
_c0	1	-0.101694	0.0300902	-0.155754	-0.036428	-0.219891	-0.184603	-0.273945
word_freq_make	-0.101694	1	-0.0111145	0.074058	0.013839	0.0210597	0.0642083	0.0036765
word_freq_address	0.0300902	-0.0111145	1	-0.0261412	-0.00779838	-0.017703	-0.0221901	0.0109912
word_freq_all	-0.155754	0.074058	-0.0261412	1	-0.0237493	0.0760231	0.0932362	0.0318528
word_freq_3d	-0.036428	0.013839	-0.00779838	-0.0237493	1	0.00108283	-0.0124234	0.017563
word_freq_our	-0.219891	0.0210597	-0.017703	0.0760231	0.00108283	1	0.0531339	0.147041
word_freq_over	-0.184603	0.0642083	-0.0221901	0.0932362	-0.0124234	0.0531339	1	0.0588029
word_freq_remove	-0.273945	0.0036765	0.0109912	0.0318528	0.017563	0.147041	0.0588029	1
word_freq_internet	-0.164886	-0.00366836	-0.0147633	0.0110633	0.00936395	0.0261208	0.0824289	0.0320056
word_freq_order	-0.18336	0.118941	0.00437534	0.100584	-0.00461698	0.012117	0.125953	0.0421473
word_freq_mail	-0.0904885	0.0468654	0.0367274	0.032101	-0.00651212	0.0317488	0.0121562	0.0557381

Finally, I chose one of the most important features and project it into new classification distribution.

- In the fifth section, I briefly introduced some data mining and machine learning methods for our data mining project, all the three methods can be applied as our model building method. After compared the data mining methods, I selected classification as our data mining method for this study.
- In this section, as we decided to use classification as our data mining method, we determined three classification algorithms: decision tree, random forest and neural network along with combined ensemble bagging as our model building algorithms. I also discussed the parameter settings for the above algorithms.
- Finally, we step into the data mining process. In the beginning, I set a test design, divided the whole dataset into training set and test set, by Pareto's scheme, we divided 80% of total dataset as training set and 20% of total dataset as test set.

```
train, test = dfs_for_model.randomSplit([0.8, 0.2], seed = 2018)

print("Training Dataset Count: " + str(train.count()))

print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 6164
Test Dataset Count: 1538
```

Next, I combined our previous design on all three data mining algorithms, by using our predefined parameter settings we got the following result accuracy: 91% for decision tree on test set, 97% for regression on test set and nearly 99% for both random forest models.

```
Test Area Under ROC decision tree max depth 10: 0.917462866114981
```

```
Test Area Under ROC for random forest 100: 0.9915715936565738
```

```
Test Area Under ROC regression 10: 0.9624592030617393
```

After we got our data mining results, I interpreted data mining result by analyse different models' predictor importance. By comparing them together, I found that word_freq_remove, word_freq_hp and word_freq_000 as three most important features, and I confirmed our data mining results by conducted further investigation.

(Word count: 13218)