Document Classification

In ...

```
It can be useful to be able to classify new "test" documents using already classified "training" documents.
A common example is using a corpus of labeled spam and ham (non-spam) e-mails to predict whether or not a new document is
Here is one example of such data:  UCI Machine Learning Repository: Spambase Data Set For this project, you can either us
the above dataset to predict the class of new documents (either withheld from the training dataset or from another source
as your own spam folder). For more adventurous students, you are welcome (encouraged!) to come up a different set of docu
(including scraped web pages!?) that have already been classified (e.g. tagged), then analyze these documents to predict
how new documents should be classified. This assignment is due end of day on Sunday.
```

I.

```
#### Spambase Data Set


Abstract: Classifying Email as Spam or Non-Spam

Source:

Creators:

Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt Hewlett-Packard Labs, 1501 Page Mill Rd., Palo Alto, CA 94304

Donor: George Forman (gforman at nospam hpl.hp.com) 650-857-7835

Data Set Information:

The "spam" concept is diverse: advertisements for products/web sites, make money fast schemes, chain letters, pornography...

Our collection of spam e-mails came from our postmaster and individuals who had filed spam. Our collection of non-spam e-ma

For background on spam:

Cranor, Lorrie F., LaMacchia, Brian A. Spam!
Communications of the ACM, 41(8):74-83, 1998.

(a) Hewlett-Packard Internal-only Technical Report. External forthcoming.
(b) Determine whether a given email is spam or not.
(c) ~7% misclassification error. False positives (marking good mail as spam) are very undesirable.If we insist on zero false


Attribute Information:

The last column of 'spambase.data' denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercia
```

```
48 continuous real [0,100] attributes of type word_freq_WORD
= percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total nur

6 continuous real [0,100] attributes of type char_freq_CHAR]
= percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-ma:

1 continuous real [1,...] attribute of type capital_run_length_average
= average length of uninterrupted sequences of capital letters

1 continuous integer [1,...] attribute of type capital_run_length_longest
= length of longest uninterrupted sequence of capital letters

1 continuous integer [1,...] attribute of type capital_run_length_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.

http://archive.ics.uci.edu/ml/datasets/Spambase
```

In [106]:

```
#!pip install sklearn
#!pip install seaborn
#!pip install statsmodels
```

In [116]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from nltk.probability import FreqDist
from nltk import sent_tokenize
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression #logistic regression
from sklearn.naive_bayes import BernoulliNB #naive-bayes
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
import statsmodels.api as sm

import seaborn as sns
from sklearn.model_selection import cross_validate
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
```

```
#from nltk.corpus import stopwords
from nltk.text import Text
from collections import Counter #
import re
import nltk
import string
from nltk.util import bigrams
from nltk.corpus import names
import random
#https://github.com/nltk/nltk/blob/develop/nltk/book.py
#https://www.datacamp.com/tutorial/understanding-logistic-regression-python
```

Let's load the data

In [25]:

```
df1 = pd.read_csv('C:\\Users\\owner\\Downloads\\spambase.csv',encoding='latin')
#with pd.option_context('display.max_rows', None, 'display.max_columns', None):
#    display(df1)
df1.head(5)
```

O..

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_int
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	

5 rows × 58 columns

In [ ]:

In [ ]:

```
### 58 columns is a lot, I am curious about the values distribution. The readme says spam =1 and ham(non-spam) = 0
```

In [23]:

```
# Checking for any missing values
df1.isnull().sum()
```

Out[23]:
```
word_freq_make              0
word_freq_address           0
word_freq_all               0
word_freq_3d                0
word_freq_our               0
word_freq_over              0
word_freq_remove            0
word_freq_internet          0
word_freq_order             0
word_freq_mail              0
word_freq_receive           0
word_freq_will              0
word_freq_people            0
word_freq_report            0
word_freq_addresses         0
word_freq_free              0
word_freq_business          0
word_freq_email             0
word_freq_you               0
word_freq_credit            0
word_freq_your              0
word_freq_font              0
word_freq_000               0
word_freq_money             0
word_freq_hp                0
word_freq_hpl               0
word_freq_george            0
word_freq_650               0
word_freq_lab               0
word_freq_labs              0
word_freq_telnet            0
word_freq_857               0
word_freq_data              0
word_freq_415               0
word_freq_85                0
word_freq_technology        0
word_freq_1999              0
word_freq_parts             0
word_freq_pm                0
word_freq_direct            0
word_freq_cs                0
word_freq_meeting           0
word_freq_original          0
word_freq_project           0
word_freq_re                0
```

```
word_freq_edu                 0
word_freq_table               0
word_freq_conference          0
char_freq_%3B                 0
char_freq_%28                 0
char_freq_%5B                 0
char_freq_%21                 0
char_freq_%24                 0
char_freq_%23                 0
capital_run_length_average    0
capital_run_length_longest    0
capital_run_length_total      0
class                         0
dtype: int64
```

In [56]:

```
# I see there is a class column which contains the target value (spam and ham)
#rename column
#df.rename(columns = {'old_col1':'new_col1', 'old_col2':'new_col2'}, inplace = True)
df1.rename(columns = {'class':'target'}, inplace = True)
df1.target.value_counts()
#df1.iloc[:,[57]].value_counts()
```

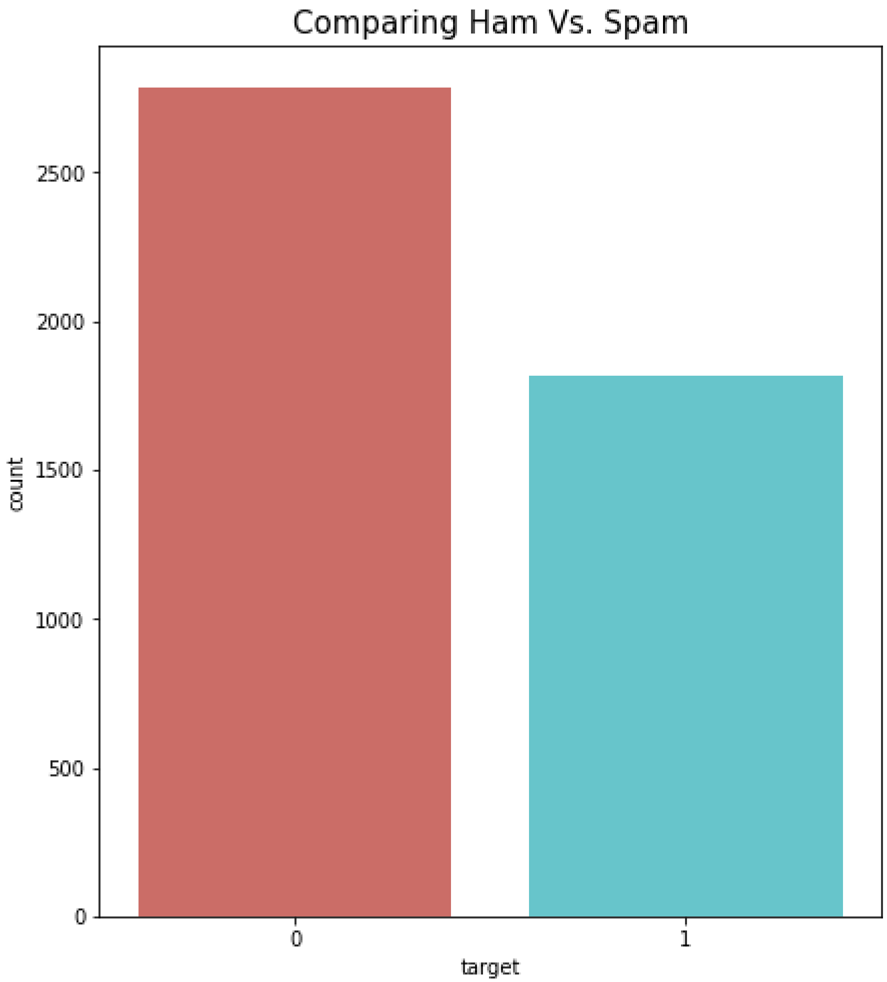
Out[56]:

```
0    2788
1    1813
Name: target, dtype: int64
```

In [79]:

```
fig = plt.figure(figsize = (7, 8))
plt.title(label="Comparing Ham Vs. Spam",fontsize=15,color="black")
#df1.iloc[:,[57]].hist()
sns.countplot(x = 'target', data = df1, palette = 'hls')
plt.show()
#df1['class'].plot(kind="bar", color=['green', 'red'])

#plt.bar(df1.class)
#df1.class.hist()
```

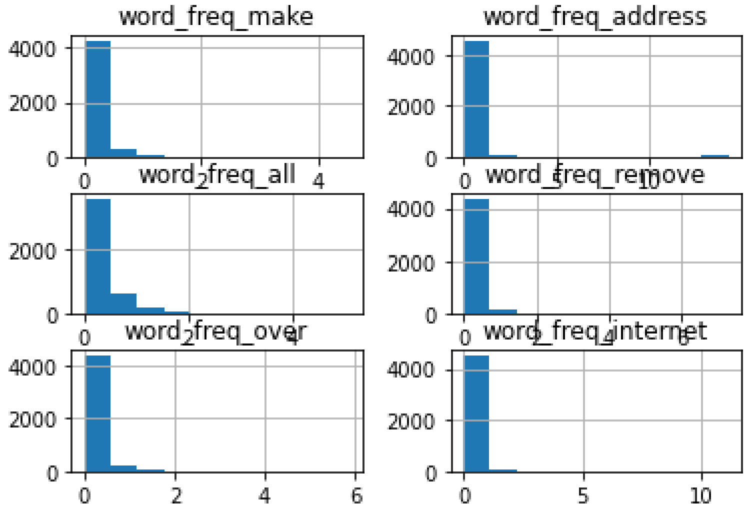


In [77]:

```
#df1.word_freq_make.hist()
#df1.word_freq_address.hist()
#df1.hist()
df1.iloc[:,[0,1,2,6,5, 7]].hist()
```

Out[77]:

```
array([[<AxesSubplot:title={'center':'word_freq_make'}>,
        <AxesSubplot:title={'center':'word_freq_address'}>],
       [<AxesSubplot:title={'center':'word_freq_all'}>,
        <AxesSubplot:title={'center':'word_freq_remove'}>],
       [<AxesSubplot:title={'center':'word_freq_over'}>,
        <AxesSubplot:title={'center':'word_freq_internet'}>]],
      dtype=object)
```



In [ ]:
```
The target variable is has binary values. So, we can use logistic regression to classify the spam email

The supervised algorigthm is used to display relationship among variables.
Under supervised machine, there is classification which predicts based on set of features.
classification requires discrete and finite outputs called classes or categories.
```

In [ ]:
```
##### Modeling
## Let's split the data into test and train
```

In [80]:
```
list(df1)
```

Out[80]:
```
['word_freq_make',
 'word_freq_address',
 'word_freq_all',
 'word_freq_3d',
 'word_freq_our',
 'word_freq_over',
 'word_freq_remove',
 'word_freq_internet',
 'word_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_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',
         '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_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_length_longest',
         'capital_run_length_total',
         'target']
```

In [...

```
colnames = ['word_freq_make', 'word_freq_address', 'word_freq_all', 'word_freq_3d', 'word_freq_our', 'word_freq_over', '
 'word_freq_internet', 'word_freq_order', 'word_freq_mail', 'word_freq_receive', 'word_freq_will', 'word_freq_people', '
 'word_freq_addresses', 'word_freq_free', 'word_freq_business', 'word_freq_email', 'word_freq_you', 'word_freq_credit',
 'word_freq_font', 'word_freq_000', 'word_freq_money', 'word_freq_hp', 'word_freq_hpl', 'word_freq_george', 'word_freq_6
 'word_freq_labs', 'word_freq_telnet', 'word_freq_857', 'word_freq_data', 'word_freq_415', 'word_freq_85', 'word_freq_te
 'word_freq_1999', 'word_freq_parts', 'word_freq_pm', 'word_freq_direct', 'word_freq_cs', 'word_freq_meeting', 'word_fre
 'word_freq_project', 'word_freq_re', 'word_freq_edu', 'word_freq_table', 'word_freq_conference', 'char_freq_%3B', 'char
 'char_freq_%5B', 'char_freq_%21', 'char_freq_%24', 'char_freq_%23', 'capital_run_length_average', 'capital_run_length_l
 'capital_run_length_total']

x = df1[colnames]
y = df1.target
```

In [82]:

```
# Splitting df1 in training set and testing sets
trainX,testX,trainY,testY=train_test_split(x,y,test_size=0.2,random_state=0)

#Now, we have splitted the data in training and testing ,...we need to scale up the input
#scale = StandardScaler() # calling the function
#trainX = scale.fit_transform(trainX)
#testX = scale.fit_transform(testX)
```

In [83]:

```
## Calling logistic function for our model ...solver='liblinear',
model = LogisticRegression()
model.fit(x, y)

train, test = train_test_split(df, test_size=0.2)
```

```
C:\Users\owner\opencv\lib\site-packages\sklearn\linear_model\_logistic.py:444: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Fitting our model with 80% training data

In [84]:

```
model.fit(trainX,trainY)
```

```
C:\Users\owner\opencv\lib\site-packages\sklearn\linear_model\_logistic.py:444: ConvergenceWarning: lbfgs failed to converge
(status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[84]:LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [86]:

```
model_pred=model.predict(testX)
#model_pred
```

Evaluating our model Performance

In [100]:

```
####### Model Accuracy
model.score(testX,testY)
```

Out[100]: 0.9218241042345277

92.2% accuracy on performance test is good.wonder if there is any overfitting...we can verify by checking accuracy on training model.score(trainX,trainY)....0.9214673913043478

In [88]:

Out[88]: 0.9214673913043478

In [9...

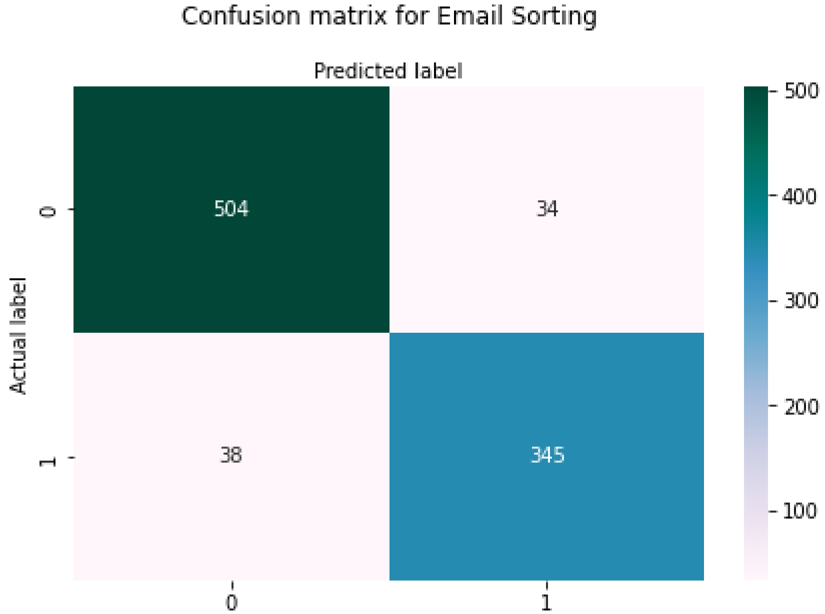
```
#A confusion matrix is a table that is used to evaluate the performance of a classification model.
#You can also visualize the performance of an algorithm. The fundamental of a confusion matrix is the number of correct
#and incorrect predictions
confusion_matrix = metrics.confusion_matrix(testY, model_pred)
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)


sns.heatmap(pd.DataFrame(confusion_matrix), annot=True, cmap="PuBuGn" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix for Email Sorting', y=1.1)
```

```
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
```

Out[96]:

```
Text(0.5, 257.44, 'Predicted label')
```



In [...

```
According to the confusion matrix, our model correctly predicted 504 email to be non-spam (ham) and 345 email to be spam.
38 email labeled as spam ones were incorrectly classified as non-spam email ...Oooops somebody got more email to go throu
34 email labeled as non-spam ones were incorrectly classified as spam email....Oooops somebody got some email lost
```

In [102]:

```
print(classification_report(testY, model_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       538
           1       0.91      0.90      0.91       383

    accuracy                           0.92       921
   macro avg       0.92      0.92      0.92       921
weighted avg       0.92      0.92      0.92       921
```

We already saw that the model was 92.2 % accurate ...this time, we can see that the model will be precised in predictiing non-spam email 93% of the time and 91% for spam email. This is considered good as no model is perfect. fishers and hackers always have an advance on tricking good systems...Good systems are reactive most of the time.

There are a time where a good company will tell you to look for their email in the spam folder. This might sound unbelievable but in the world of technology, it can makes sense and it is possible. If one filter/firewall system behind the others, the likelyhood for suc error to occur is possible. So, it is good to check the spam folder sometimes.

Precision and accuracy often bring confusion. A good way to apprehend the difference is to think about a mechanic system like mechanic watch, relays, switches, etc. Most of the time accuracy on these systems means that if the relay is supposed to react to a fault within let's say 1min setting, the actual response might occur at 1.05 min or 1.03min...that is approximately within +5% tolerance...that is how accurate the relay performs. On the other hands, this relay is supposed to trip whenever it senses the fault and it does each time regardless...that is how the relay is precised...Such system have higher expectation in the range of 99% if not 100% because the consequence might be a disaster. Our model says that we can count on it to be effective 91% in capturing spam email...that leave us with 9% of expected bad mail in the mail folder to go through.

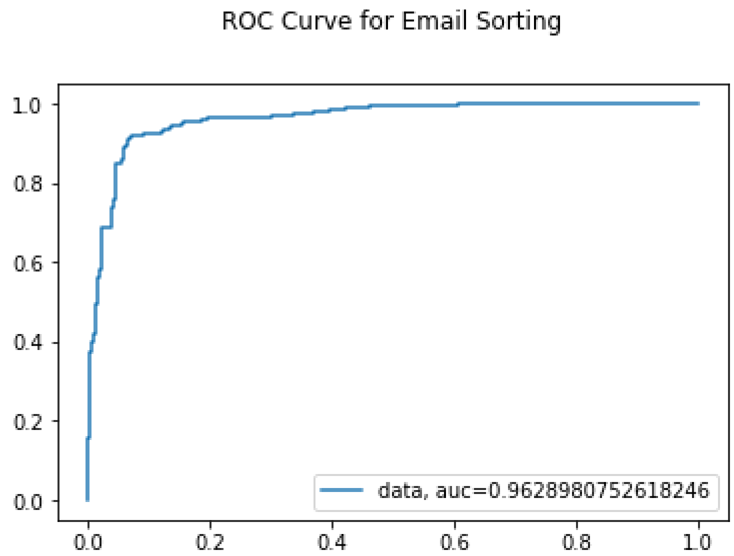
This performance is based on non-spam 2788 spam 1813

A company/individual receiving 1813 spam can be huge if it is like weekly...Something need to be done. I get minimum of 1000 email per week at work. I never actually pay attention to count the spam/junk folder.

In [120]:

```
## ROC Curve is sometime used to tell how good a classifier is.

model_pred_prob = model.predict_proba(testX)[::,1]
fpr, tpr, _ = metrics.roc_curve(testY,  model_pred_prob)  #True Positive Rate (trp), False Positive Rate(fpr)
auc = metrics.roc_auc_score(testY, model_pred_prob)
plt.title('ROC Curve for Email Sorting', y=1.1)
plt.plot(fpr,tpr,label="data, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



The ROC Curve is telling us that based on the AUC(area under curve) score 0.96...our model is close to perfect. Let's say the model did not perform well...meaning we got like 0.60 AUC score. What could we do to improve our model performance? One way is to check the significance of each independent variable in relationship with the target variable.

In [110]:
```
model_logit=sm.Logit(y,x)
model_fit=model_logit.fit()
print(model_fit.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.212842
         Iterations 15
                              Results: Logit
============================================================================
Model:                 Logit                Pseudo R-squared:    0.683
Dependent Variable:    target               AIC:                 2072.5739
Date:                  2022-07-18 16:11     BIC:                 2439.3136
No. Observations:      4601                 Log-Likelihood:      -979.29
Df Model:              56                   LL-Null:             -3085.1
Df Residuals:          4544                 LLR p-value:         0.0000
Converged:             1.0000               Scale:               1.0000
No. Iterations:        15.0000
----------------------------------------------------------------------------
                            Coef.   Std.Err.    z     P>|z|   [0.025   0.975]
----------------------------------------------------------------------------
word_freq_make             -0.4925   0.2040 -2.4141 0.0158  -0.8924 -0.0926
word_freq_address          -0.2455   0.0659 -3.7283 0.0002  -0.3746 -0.1165
```

```
word_freq_all             -0.0881   0.1086 -0.8110 0.4174  -0.3009  0.1248
word_freq_3d               2.0840   1.4406  1.4467 0.1480  -0.7394  4.9075
word_freq_our              0.3643   0.0990  3.6788 0.0002   0.1702  0.5584
word_freq_over             0.5346   0.2399  2.2286 0.0258   0.0644  1.0048
word_freq_remove           2.1275   0.3273  6.5006 0.0000   1.4861  2.7690
word_freq_internet         0.4187   0.1444  2.9008 0.0037   0.1358  0.7017
word_freq_order            0.5012   0.2695  1.8597 0.0629  -0.0270  1.0294
word_freq_mail             0.0377   0.0680  0.5553 0.5787  -0.0955  0.1709
word_freq_receive         -0.1094   0.2911 -0.3757 0.7071  -0.6799  0.4612
word_freq_will            -0.3316   0.0718 -4.6149 0.0000  -0.4724 -0.1907
word_freq_people          -0.3511   0.2158 -1.6273 0.1037  -0.7740  0.0718
word_freq_report          -0.0540   0.1352 -0.3995 0.6895  -0.3191  0.2110
word_freq_addresses        1.4374   0.7500  1.9165 0.0553  -0.0326  2.9074
word_freq_free             0.8798   0.1431  6.1494 0.0000   0.5994  1.1603
word_freq_business         0.8914   0.2254  3.9556 0.0001   0.4497  1.3332
word_freq_email           -0.0107   0.1102 -0.0972 0.9226  -0.2268  0.2053
word_freq_you             -0.0830   0.0318 -2.6123 0.0090  -0.1453 -0.0207
word_freq_credit           1.1846   0.5539  2.1386 0.0325   0.0990  2.2703
word_freq_your             0.1524   0.0510  2.9859 0.0028   0.0524  0.2525
word_freq_font             0.1957   0.1856  1.0545 0.2917  -0.1681  0.5596
word_freq_000              2.1029   0.4621  4.5506 0.0000   1.1972  3.0086
word_freq_money            0.4031   0.1661  2.4263 0.0153   0.0775  0.7286
word_freq_hp              -2.2495   0.3190 -7.0516 0.0000  -2.8748 -1.6243
word_freq_hpl             -1.1445   0.4358 -2.6259 0.0086  -1.9987 -0.2903
word_freq_george          -8.8352   1.6491 -5.3576 0.0000 -12.0673 -5.6030
word_freq_650              0.4550   0.2028  2.2436 0.0249   0.0575  0.8524
word_freq_lab             -3.5130   1.7222 -2.0399 0.0414  -6.8885 -0.1376
word_freq_labs            -0.4937   0.3218 -1.5345 0.1249  -1.1244  0.1369
word_freq_telnet          -0.3485   0.6544 -0.5325 0.5944  -1.6310  0.9341
word_freq_857              1.4011   3.1060  0.4511 0.6519  -4.6865  7.4887
word_freq_data            -1.0208   0.3052 -3.3451 0.0008  -1.6189 -0.4227
word_freq_415              0.2151   1.5568  0.1382 0.8901  -2.8362  3.2664
word_freq_85              -2.3772   0.7568 -3.1412 0.0017  -3.8604 -0.8939
word_freq_technology       0.3976   0.3047  1.3050 0.1919  -0.1995  0.9947
word_freq_1999            -0.1732   0.1774 -0.9764 0.3289  -0.5208  0.1744
word_freq_parts           -0.7428   0.4404 -1.6864 0.0917  -1.6060  0.1205
word_freq_pm              -0.9949   0.3723 -2.6721 0.0075  -1.7247 -0.2652
word_freq_direct          -0.5242   0.3530 -1.4850 0.1375  -1.2161  0.1677
word_freq_cs             -35.6540  22.5596 -1.5804 0.1140 -79.8699  8.5619
word_freq_meeting         -2.9040   0.8308 -3.4955 0.0005  -4.5323 -1.2757
word_freq_original        -1.7695   0.8759 -2.0203 0.0434  -3.4863 -0.0528
word_freq_project         -2.1610   0.5600 -3.8592 0.0001  -3.2585 -1.0635
word_freq_re              -0.9973   0.1497 -6.6623 0.0000  -1.2907 -0.7039
word_freq_edu             -1.9000   0.2868 -6.6252 0.0000  -2.4621 -1.3379
word_freq_table           -2.4466   1.7914 -1.3658 0.1720  -5.9577  1.0644
```

```
word_freq_conference        -3.5800   1.2071 -2.9657 0.0030  -5.9459 -1.2140
char_freq_%3B               -1.6146   0.5125 -3.1502 0.0016  -2.6191 -0.6100
char_freq_%28               -1.2256   0.3365 -3.6419 0.0003  -1.8852 -0.5660
char_freq_%5B               -1.5855   0.9863 -1.6076 0.1079  -3.5186  0.3475
char_freq_%21                0.2349   0.0555  4.2310 0.0000   0.1261  0.3437
char_freq_%24                4.9602   0.7040  7.0457 0.0000   3.5804  6.3400
char_freq_%23                2.2275   1.1314  1.9689 0.0490   0.0101  4.4449
capital_run_length_average  -0.0142   0.0091 -1.5602 0.1187  -0.0321  0.0036
capital_run_length_longest   0.0092   0.0021  4.2802 0.0000   0.0050  0.0134
capital_run_length_total     0.0004   0.0002  2.0834 0.0372   0.0000  0.0008
============================================================================
```

The above result contains p-value for the model... A p-value is a statistical measurement used to validate a hypothesis against observed data. A p-value measures the probability of obtaining the observed results, assuming that the null hypothesis is true. The lower the p-value, the greater the statistical significance of the observed difference.

Based on p-values greater than 0.05...the variable should be removed, thereafter, running the new model to see if the performance has improved. This approach normally improve the model performance because even if Logistic regregression is data hungry, having way too many independence variables affect the model...that is one issue that dimensionality deals with.

Another approach in improving the model performance is to try another classifier. For binary classification, Naive Bayes is another good classifier.

https://www.investopedia.com/terms/p/p-value.asp#:~:text=A%20p%2Dvalue%20is%20a,significance%20of%20the%20observed%20difference.

In [117]:

```
# Let's call the naive-bayes function...
model2 = BernoulliNB()

# training the model
model2.fit(trainX,trainY)

# testing the model
model2_pred = model2.predict(testX)
print(accuracy_score(model2_pred, testY))
```

```
0.8773072747014115
```

the model actually dropped in performance, meaning , logistic regression is better for this email classification

In [ ]: