*Balaji Avvaru, Apurv Mittal, Ravi Sivaraman*

*Quantifying The World | SMU | DR. SLATER*

*Case study 3*

Classification/Clustering of Emails

# Abstract

The objective of this study is to classify emails as spam or not spam. In addition to that, cluster emails to observe any hidden group characteristics. The emails have several metadata, email body text, and sometimes attachments. The texts in the emails are parsed and analyzed for clustering and for the classification of spam emails.

# Introduction

The dataset consists of five directories, two of which are spam, and the rest are not spam. Each directory contains several files of spam or not spam email and each email is stored as a single file. The format of the file is email header and body text.

To create the dataset, this study parses each file, gets the body text and various headers, and creates a pandas dataframe. The emails in the files are of the following types:

|  |  |
| --- | --- |
| Email Type | Count |
| text/plain | 7413 |
| text/html | 1193 |
| multipart/alternative | 326 |
| multipart/signed | 180 |
| multipart/mixed | 179 |
| multipart/related | 56 |
| multipart/report | 5 |
| text/plain charset=us-ascii | 1 |

Table 1-Email types

Each file type must be parsed differently.

### text/plain

They are simple text and can be parsed as strings

### text/html

The HTML text is parsed as HTML.

### multipart

Multipart emails are emails that are large and sent in multiple parts. To get the entire message, each multipart of the message must be parsed and then appended to one message. Sometimes, the messages contain mixed multipart messages, a combination of *HTML* and *plaintext* messages. In such cases, the type of each multipart needs to be checked and then parsed appropriately.

### Attachments

The email headers are parsed and if *Content-Disposition* has any value attachment, then the message has attachment, else none.

### Target

All the email messages are parsed and then stored in the dataframe. The target is the message is spam or not (1 spam/0 not-spam) is computed based on folder name contains text “spam”, is treated as spam, else not spam.

|  |  |  |  |
| --- | --- | --- | --- |
| Column | Non-Null | Count | Dtype |
| data | 9353 | non-null | object |
| target | 9353 | non-null | int64 |
| Attachments | 9353 | non-null | int64 |

Table 2 - Dataframe fields

# 2. Methods

The email texts are parsed and stored as lower text to catch spammers who use either mixed texts or upper case to defeat spam filters. Converting all of them to lower case does lose information but reduces the number of features and is better at catching based on texts. However, sometimes mixed cases or unnatural all lower case or upper case may signify spam.

### Missing Data & Imputation

There are no missing data, and there was no imputation done.

## Target

Spam or not spam is the target variable, and the distribution of spam vs not-spam is as follows; the graph indicates the target variable is heavily imbalanced.

Chart, pie chart

Description automatically generated

Figure 1- Count of Spam (1) vs Non-Spam (0)

The graph below shows the distribution of attachments in Spam/Not-Spam emails. The attachments are imbalanced, the non-spam emails contain proportionately higher attachments compared to spam emails. The KMeans clustering and Naïve Bayes classification doesn’t required the data to be balanced.

Chart, bar chart

Description automatically generated

Figure 2 - Attachments in Spam vs Non-Spam (target- 0: non-spam/1: spam)

### Vectorization

The body of email texts may contain a lot of smaller sized, but very common words like the, of, etc. are stripped. Some common email texts like spamassasin (part of email footer, for example) are also removed.

The extra stop words that are added to corpus including:

* Spamassassin
* Email
* Message
* nbsp
* font
* exhm
* subject
* list
* URL
* net
* http

This model uses a Tfidf vectorizer, to eliminate the most common words.

## Clustering Models

### KMeans Clustering

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable k. The algorithm works iteratively to assign each data point to one of the k groups based on the features that are provided. Data points are clustered based on feature similarity.

The graph below shows the sum of squares at various values of k (clusters), with elbow at 15, which indicates there may be 15 optimal clusters, which this study will use.

Chart, line chart

Description automatically generated

Figure 2-KMeans Range of k Values (Elbow at 15)

### Visualize Clusters

This study uses a technique called t-SNE (t-distributed Stochastic Neighbor Embedding) to generate a 2-dimensional representation of this dataset, which provides an intuitive view of clusters.

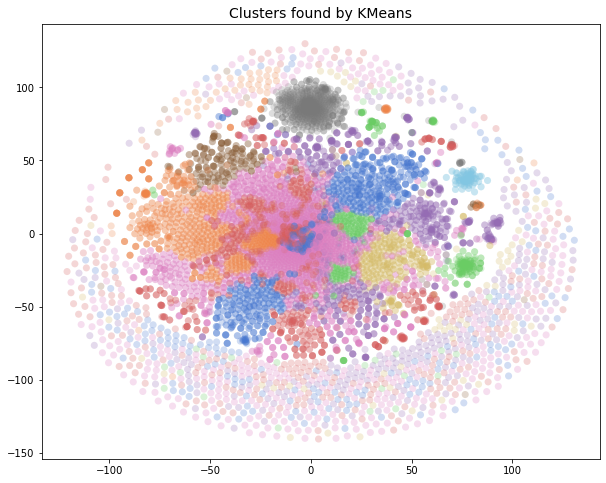


Figure 3-Two-dimensional cluster of emails

### Word Cloud

Each picture is a cluster, two sample clusters are shown here below. The cluster shows the emails are grouped that are related by texts (or theme of texts). The texts in the same cluster means the KMeans distance between them are low compared to the texts that are not in same cluster.

Text

Description automatically generated

Text

Description automatically generated

Figure 3-Sample Word Cloud of Email Texts

## Naïve Bayes Classification

It is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. This study uses NB Classification to classify spam emails.

# 3. Results

Naïve-Bayes algorithm is run with Grid search with several parameters, to find the optimal results.

|  |  |  |  |
| --- | --- | --- | --- |
| Mean | Std-dev | Alpha | Fit Prior |
| 0.985567 | -0.002797 | 0.0001 | TRUE |
| 0.986957 | -0.002473 | 0.0001 | FALSE |
| 0.98717 | -0.002339 | 0.001 | TRUE |
| 0.988774 | -0.002253 | 0.001 | FALSE |
| 0.987064 | -0.002159 | 0.01 | TRUE |
| 0.989202 | -0.002763 | 0.01 | FALSE |
| 0.980862 | -0.00194 | 0.1 | TRUE |
| 0.983642 | -0.002909 | 0.1 | FALSE |
| 0.837806 | -0.008439 | 1 | TRUE |
| 0.882925 | -0.007082 | 1 | FALSE |
| 0.743505 | -0.000318 | 10 | TRUE |
| 0.747675 | -0.003525 | 10 | FALSE |
| 0.743505 | -0.000318 | 100 | TRUE |
| 0.744039 | -0.000549 | 100 | FALSE |
| 0.743505 | -0.000318 | 1000 | TRUE |

Table 3-NB Classifier Grid Search Results (Green highest accuracy)

|  |
| --- |
| Best Accuracy with Grid Search: 0.989 |

### Training data Metrics

|  |  |
| --- | --- |
| Metrics | Value |
| average accuracy | 0.993 |
| average precision | 0.997 |
| average recall | 0.977 |

### Test data Metrics

|  |  |
| --- | --- |
| Metrics | Value |
| average accuracy | 0.989 |
| average precision | 0.990 |
| average recall | 0.968 |

### Classification report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-Score | Support |
| 1 (Spam) | 0.99 | 0.97 | 0.98 | 2399 |
| 0 (Not Spam) | 0.99 | 1.00 | 0.99 | 6954 |
| Accuracy |  |  | 0.99 | 9353 |
| Macro avg | 0.99 | 0.98 | 0.99 | 9353 |
| Weighted avg | 0.99 | 0.99 | 0.99 | 9353 |

### ROC Curve

Chart, line chart

Description automatically generated

Figure 4:ROC Curve (True Positive vs False Positive)

The ROC curve for the NB classifier showed a mean area under the curve AUC of 1.00.

The accuracy, precision, and f-score are all very high and close to 1.00. The ROC curve also indicates there are no false positives compared to the true positive rate. The area under the curve indicates the results cover the entire dataset.

# Conclusion

The features (texts) indicate that spam vs non-spam is: The positive feature\_coef with high values corresponds to a higher incidence of spams vs feature\_coef (Negative) with lowest scores corresponding to least incidence of spams.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| features | feature\_coef (Positive) | | features | | feature\_coef (Negative) | |
| sightings | 8.086723 | date | | -10.818954 | |
| please | 7.802528 | com | | -9.38241 | |
| click | 6.755935 | newsisfree | | -8.952799 | |
| remove | 6.451776 | wrote | | -8.711082 | |
| free | 6.294921 | rpm | | -7.929359 | |
| money | 6.189775 | supplied | | -6.946106 | |
| removed | 5.814065 | said | | -6.220962 | |
| visit | 4.46381 | cnet | | -5.064446 | |
| receive | 4.329541 | use | | -4.781836 | |
| offer | 4.261584 | exmh | | 4.603416 | |

### 

Figure 5-Feature Importance based on co-efficient

The above table displays the features with their coefficient levels. These are the top 10 (positive & negative) coefficients after the L2 penalty. The table shows the common words in spam (money, free, offer, click) do tend to appear more on spam messages. On the contrary, the messages with negative coef words like cnet, use, date do tend to appear more on non-spam messages.

This model is recommended to use as it produces higher accuracy and precision (with a high f-score).

# Appendix – Code

Nbviewer link: <https://nbviewer.org/github/ravisiv/SpamClassificationML/blob/main/Case%20Study%203%20Balaji%20-%20Apurv%20-%20Ravi.ipynb>

## Python Jupyter Notebook

### Case Study 3: Spam classifier

Submitted by:

* Ravi Sivaraman
* Balaji Avvaru
* Apurv Mittal

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import os

from os.path import isfile

import email

#import BeautifulSoup

from bs4 import BeautifulSoup

import re

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import silhouette\_score

import hdbscan

from sklearn.cluster import KMeans

from sklearn.model\_selection import cross\_val\_predict

from sklearn.metrics import classification\_report

from sklearn.model\_selection import cross\_validate

from sklearn.linear\_model import LogisticRegression

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.model\_selection import GridSearchCV

import seaborn as sns

from sklearn.preprocessing import label\_binarize

from sklearn import metrics as mt

from sklearn.feature\_extraction import text

from wordcloud import WordCloud

from scipy.sparse import hstack

import warnings

warnings.filterwarnings("ignore")

# location of emails

data\_path = "/Users/ravis/Downloads/SpamAssassinMessages"

# get all sub folders

sub\_folders = [x[0] for x in os.walk(data\_path) if x[0] != data\_path]

%%time

# read all emails from all sub folders

mail\_ty = []

text\_ty = []

data = []

target = []

email\_attachment = []

attachment = False

for folder in sub\_folders:

files = [f for f in os.listdir(folder) if os.path.isfile(os.path.join(folder, f))]

for file in files:

with open(f"{folder}/{file}", encoding="latin1") as f:

# with open(f"{folder}/{file}","r") as f:

x = email.message\_from\_file(f)

# print(x)

# if (file != 'cmds'):

# mail\_data.append(lines)

mail\_type = x.get\_content\_type()

text\_type = x.get\_content\_charset()

mail\_ty.append(mail\_type)

text\_ty.append(text\_type)

if re.search("spam", folder):

target.append(1)

else:

target.append(0)

if mail\_type == "text/html":

if not (isinstance(x.get\_payload(), str)) and x.get\_payload().get('Content-Disposition'):

dispositions = x.get\_payload().get("Content-Disposition", None).strip().split(";")

if bool(dispositions[0].lower() == "attachment"):

attachment = True

else:

attachment = False

tmp = BeautifulSoup(x.get\_payload(), 'html.parser')

tmp = tmp.text.replace("\n", " ")

data.append(tmp)

elif "multipart" in mail\_type:

attachment = False

multipart\_data = []

for text in x.get\_payload():

if not isinstance(text, str):

if text.get('Content-Disposition'):

dispositions = text.get("Content-Disposition", None).strip().split(";")

if bool(dispositions[0].lower() == "attachment"):

attachment = True

if text.get\_content\_type() == "text/html":

tmp = BeautifulSoup(text.get\_payload(), 'html.parser')

tmp = tmp.text.replace("\n", " ")

multipart\_data.append(tmp)

elif text.get\_content\_type() == "text/plain":

multipart\_data.append(text.get\_payload())

multipart\_email = [''.join(str(item)) for item in multipart\_data]

data.append(multipart\_email)

else:

if not (isinstance(x.get\_payload(), str)) and x.get\_payload().get('Content-Disposition'):

dispositions = x.get\_payload().get("Content-Disposition", None).strip().split(";")

if bool(dispositions[0].lower() == "attachment"):

attachment = True

else:

attachment = False

data.append(x.get\_payload())

if attachment:

email\_attachment.append(1)

else:

email\_attachment.append(0)

# Reference: https://gaurav.kuwar.us/index.php/2017/10/09/extracting-files-from-raw-email-with-python/

df = pd.DataFrame()

df["mail\_types"] = mail\_ty

df["text\_types"] = text\_ty

# Count of mail types

df["mail\_types"].value\_counts()

# Count of text types

df["text\_types"].value\_counts()

# Create a data frame with email text and target (whether mail is spam or not, 1 for spam and 0 for not a spam)

email\_df = pd.DataFrame()

email\_df["data"] = data

#email\_df["mail\_type"] = mail\_ty

#email\_df["text\_type"] = text\_ty

email\_df["target"] = target

email\_df["Attachments"] = email\_attachment

email\_df['target'].value\_counts()

email\_df.info()

email\_df.loc[1].data

email\_df["data\_new"] = [''.join(str(item).lower()) for item in email\_df.data]

print(email\_df["data\_new"][0])

# get the instanc of TfidfVectorizer

#my\_stop\_words = text.ENGLISH\_STOP\_WORDS.union(["spamassassin", "email", "message", "\n", "nbsp", "font","exhm", "subject", "list", "url", "net"])

from nltk.corpus import stopwords

stop = list(stopwords.words('english'))

stop.extend("spamassassin email message \n nbsp font exhm subject list url net http www org html linux 2002 font e2 c2 div 0d c2 0a xa0 8c 2ffont e2 3e sourceforge spamassasin 01 yahoo 1440 a0".split())

tf\_vectorizer = TfidfVectorizer(analyzer = 'word',stop\_words=set(stop))

# tf\_vectorizer = TfidfVectorizer(ngram\_range=(1,2), stop\_words=text.ENGLISH\_STOP\_WORDS)

#tf\_vectorizer = TfidfVectorizer()

# fit and transform email data

new\_vectors = tf\_vectorizer.fit\_transform(email\_df.data\_new)

# Pie chart

plt.figure(figsize=(5,4))

email\_df.target.value\_counts().plot.pie(autopct = "%.1f%%")

plt.title("Proportion of Target Value")

plt.show()

email\_df['Attachments'].value\_counts()

plt.figure(figsize=(5,4))

sns.countplot(x ="Attachments", data = email\_df)

plt.title("Distribution of Attachments")

plt.show()

plt.figure(figsize=(5,4))

sns.countplot(x ="Attachments", hue = "target", data = email\_df)

plt.title("Attachments in Spam (1) vs Not Spam (0)")

plt.show()

new\_vectors = hstack((new\_vectors,np.array(email\_attachment)[:,None]))

Clustering

KMeans Clustering

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity

KMeans Clustering with default parameters

wcss = []

score = []

K = range(2,30)

for k in K:

km = KMeans(n\_clusters=k, random\_state=1234, init = 'k-means++')

km = km.fit(new\_vectors)

labels = km.predict(new\_vectors)

wcss.append(km.inertia\_)

sc = silhouette\_score(new\_vectors, labels)

score.append(sc)

plt.rcParams['figure.figsize'] = (15, 5)

#plt.subplot(1,1,1)

plt.plot(K, wcss, 'bx-')

plt.xlabel('Number of centroids')

plt.ylabel('Within-Cluster-Sum-of-Squares')

plt.title('Elbow Method For Optimal k')

Visualize Clusters

We will use a technique called t-SNE (t-distributed Stochastic Neighbor Embedding) to generate a 2 dimensional representation of our dataset, in order to have a more intuitive understanding of how the clustering looks.

First let's look at an un-clustered version of this 2D projection.

%%time

from sklearn.manifold import TSNE

import sklearn.cluster as cluster

newdims = (12, 8)

plt.subplots(1, 1, figsize=newdims)

plt.subplot(1, 1, 1)

plot\_kwds = {'alpha' : 0.25, 's' : 40, 'linewidths':0}

projection = TSNE().fit\_transform(new\_vectors)

plt.scatter(\*projection.T, \*\*plot\_kwds)

plt.title("")

plt.show()

Now look at clustered version of this 2D projection with various clustering techniques

%%time

import seaborn as sns

# This function will run a given clustering algorithm and plot the clusters on the same 2D TSNE projection as above

def plot\_clusters(data, algorithm, args, kwds):

labels = algorithm(\*args, \*\*kwds).fit\_predict(data)

palette = sns.color\_palette('muted', np.unique(labels).max() + 1)

colors = [palette[x] if x >= 0 else (.5, .5, .5) for x in labels]

plt.scatter(\*projection.T, s=50, linewidth=0, c=colors, alpha=0.25)

frame = plt.gca()

frame.axes.get\_xaxis().set\_visible(True)

frame.axes.get\_yaxis().set\_visible(True)

#plot\_kwds = {'alpha' : 0.25, 's' : 40, 'linewidths':0}

plt.title('Clusters found by {}'.format(str(algorithm.\_\_name\_\_)), fontsize=14)

best\_k = 15

# plot the clusters

newdims = (10, 8)

plt.subplots(1, 1, figsize=newdims)

plt.subplot(1, 1, 1)

plot\_clusters(new\_vectors, cluster.KMeans, (), {'n\_clusters':best\_k})

Word Cloud

# word cloud with best K

km = KMeans(n\_clusters=best\_k, init = 'k-means++')

km = km.fit(new\_vectors)

labels = km.predict(new\_vectors)

clusters = list(labels)

kmeans\_result={'cluster':clusters,'reviews':email\_df.data\_new}

kmeans\_result=pd.DataFrame(kmeans\_result)

for k in range(0,12):

s=kmeans\_result[kmeans\_result.cluster==k]

text=s['reviews'].str.cat(sep=' ')

text=text.lower()

text=' '.join([word for word in text.split()])

wordcloud = WordCloud(max\_font\_size=50, max\_words=100, background\_color="white").generate(text)

plt.figure()

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis("off")

plt.show()

Naive Bayes Classification

It is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

def displayModel\_metrics(best\_model, grid\_model, features, target, cv):

metrics = cross\_validate(best\_model, features, y=target, cv=cv,

scoring=['accuracy','precision','recall'], return\_train\_score=True)

y\_predict = cross\_val\_predict(best\_model, features, target, cv=cv)

print('\nBest Accuracy with Grid Search : {:.3f}'.format(grid\_model.best\_score\_))

print('\nTraining data Metrics')

print('\n The average accuraccy : {:.3f}'.format(metrics['train\_accuracy'].mean()))

print(' The average precision : {:.3f}'.format(metrics['train\_precision'].mean()))

print(' The average recall : {:.3f}'.format(metrics['train\_recall'].mean()))

print('\nTest data Metrics')

print('\n The average accuracy : {:.3f}'.format(metrics['test\_accuracy'].mean()))

print(' The average precision : {:.3f}'.format(metrics['test\_precision'].mean()))

print(' The average recall : {:.3f}'.format(metrics['test\_recall'].mean()))

matrix = classification\_report(target, y\_predict, labels=[1,0])

print('\nClassification report\n')

print(matrix)

# Reference https://github.com/jakemdrew/DataMiningNotebooks/blob/master/06.%20Classification.ipynb

# ROC curve plot

def roc\_curve\_plot(model\_fit, features, target):

sns.set\_palette("dark")

yhat\_score = model\_fit.predict\_proba(features)

# Compute ROC curve for a subset of interesting classes

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in np.unique(target):

fpr[i], tpr[i], \_ = mt.roc\_curve(target, yhat\_score[:, i], pos\_label=i)

roc\_auc[i] = mt.auc(fpr[i], tpr[i])

for i in np.unique(target):

plt.plot(fpr[i], tpr[i], label= ('class %d (area = %0.2f)' % (i, roc\_auc[i])))

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.legend(loc="lower right")

plt.title('Receiver operating characteristic')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

#Create Cross Validation Procedure

cv = StratifiedKFold(n\_splits=10, random\_state=1234, shuffle=True)

# Naive Bayes (NB) classifier

clf = MultinomialNB().fit(new\_vectors,email\_df['target'])

# define parameters

C\_nb = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

nb\_prior=[True, False]

nb\_clf = MultinomialNB()

# define grid search

param\_grid\_nb = dict(alpha=C\_nb, fit\_prior=nb\_prior)

grid\_search\_nb = GridSearchCV(estimator=nb\_clf, param\_grid=param\_grid\_nb, n\_jobs=-1, cv=cv,

scoring='accuracy',error\_score=0)

%%time

grid\_result\_nb = grid\_search\_nb.fit(new\_vectors,email\_df['target'])

# summarize results

print("Best: %f using %s" % (grid\_result\_nb.best\_score\_, grid\_result\_nb.best\_params\_))

means = grid\_result\_nb.cv\_results\_['mean\_test\_score']

stds = grid\_result\_nb.cv\_results\_['std\_test\_score']

params = grid\_result\_nb.cv\_results\_['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))

# The GridSearch algorithm determined the following optimal parameters

best\_Estimator\_nb =grid\_result\_nb.best\_estimator\_

best\_Estimator\_nb

# Display model metrics

displayModel\_metrics(best\_Estimator\_nb, grid\_result\_nb, new\_vectors,email\_df['target'], cv)

# Plot ROC curve

roc\_curve\_plot(grid\_result\_nb, new\_vectors, email\_df['target'])

Naive Bayes Classification with clusters as feature

# add clusters as feature

new\_vectors = hstack((new\_vectors,np.array(clusters)[:,None]))

new\_vectors

%%time

grid\_result\_nb = grid\_search\_nb.fit(new\_vectors,email\_df['target'])

# summarize results

print("Best: %f using %s" % (grid\_result\_nb.best\_score\_, grid\_result\_nb.best\_params\_))

means = grid\_result\_nb.cv\_results\_['mean\_test\_score']

stds = grid\_result\_nb.cv\_results\_['std\_test\_score']

params = grid\_result\_nb.cv\_results\_['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))

# The GridSearch algorithm determined the following optimal parameters

best\_Estimator\_nb =grid\_result\_nb.best\_estimator\_

best\_Estimator\_nb

# Display model metrics

displayModel\_metrics(best\_Estimator\_nb, grid\_result\_nb, new\_vectors,email\_df['target'], cv)

# Plot ROC curve

roc\_curve\_plot(grid\_result\_nb, new\_vectors, email\_df['target'])

Feature importance with Logistic regression

from sklearn.model\_selection import GridSearchCV

from sklearn.linear\_model import LogisticRegression

LR = LogisticRegression()

# define parameters

penalty\_LR = ['l1', 'l2', 'elasticnet', 'none']

#penalty\_LR = [ 'l1', 'l2']

C\_LR = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

#C\_LR = [0.001,10, 100]

max\_iter\_LR = [500]

#max\_iter\_LR = [500]

class\_weight\_LR = ['balanced']

#solver\_LR = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

solver\_LR = ['lbfgs', 'liblinear']

# define grid search

param\_grid\_LR = dict(penalty=penalty\_LR, C=C\_LR, max\_iter=max\_iter\_LR, class\_weight=class\_weight\_LR, solver=solver\_LR)

grid\_search\_LR = GridSearchCV(estimator=LR, param\_grid=param\_grid\_LR, n\_jobs=-1, cv=cv,

scoring='accuracy',error\_score=0)

%%time

grid\_result\_LR = grid\_search\_LR.fit(new\_vectors,email\_df['target'])

# summarize results

print("Best: %f using %s" % (grid\_result\_LR.best\_score\_, grid\_result\_LR.best\_params\_))

means = grid\_result\_LR.cv\_results\_['mean\_test\_score']

stds = grid\_result\_LR.cv\_results\_['std\_test\_score']

params = grid\_result\_LR.cv\_results\_['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))

# The GridSearch algorithm determined the following optimal parameters

best\_Estimator\_LR =grid\_result\_LR.best\_estimator\_

best\_Estimator\_LR

features = tf\_vectorizer.get\_feature\_names()

features.append('email\_attachment')

features.append('clusters')

feature\_importance\_df = pd.DataFrame(features, columns=['features'])

feature\_importance\_df['feature\_coef'] = best\_Estimator\_LR.coef\_[0]

feature\_importance\_df.head()

feature\_importance\_df = feature\_importance\_df.sort\_values(by=['feature\_coef'])

feature\_importance\_df.tail(20)

feature\_importance\_df.head(20)

Bottom of Form