**DS 7333; Case Study #3: Email Parsing and Spam Classification.**

Allen Miller, Vijayasrikanth Kaniti, Venkata ‘MG’ Vanga.

October 2, 2021

Professor: Dr. R. D. Slater

# Introduction

The case study focuses on the email parsing and spam / not spam classification of the parsed emails. The dataset consists of emails categorized into five folders [total: 9353 emails] with the number of emails in each folder shown:   
[‘easy\_ham’: 5052; ‘easy\_ham\_2’: 1401; ‘hard\_ham’: 501; ‘spam’: 1001 and ‘spam\_2’: 1398]. The first three folders are folders that contain non spam email messages [total: 6954] and the last two folders ‘spam’ and ‘spam\_2’ are the folders that contain spam messages [total: 2399]. The email messages are stored as ‘text/plain’, ‘text/html, ‘text/multipart’ and ‘multipart/alternative’. The case study uses Naïve Bayes approach (Eq. 1) for classification of the emails into Spam and Non Spam.

# Methods

The emails in five folders are parsed using regular text parser and html parser based on the email type. Only four emails two in Spam [msg 00467, msg 0478] and two in Spam 2 [msg 00204, msg 01214] were not parsed from the total email data set, these belonged to the ‘multipart/alternative’ category.

Python Natural Language Toolkit used to remove the stop words from the parsed dataset. A train test split of 65%/35% is considered for the classification.

Sklearn countvectorizer, Tfidfvectorizer and Naïve Bayes modules are used to classify the final parsed dataset without stop words.

# Results

The email classification accuracy using the Count Vectorizer is 98% and using the Term Frequency Inverse Document Frequency (TFIDF) is 89% and are shown in Figure 1 and Figure 2 respectively. The confusion matrix heat maps for Count Vectorizer and TFIDF are shown in Figure 3.

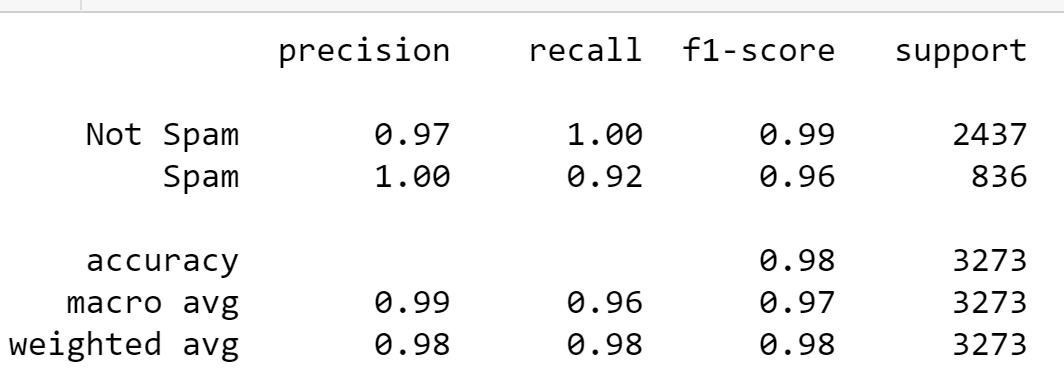


Figure 1: Classification Report using Count-Vectorizer.

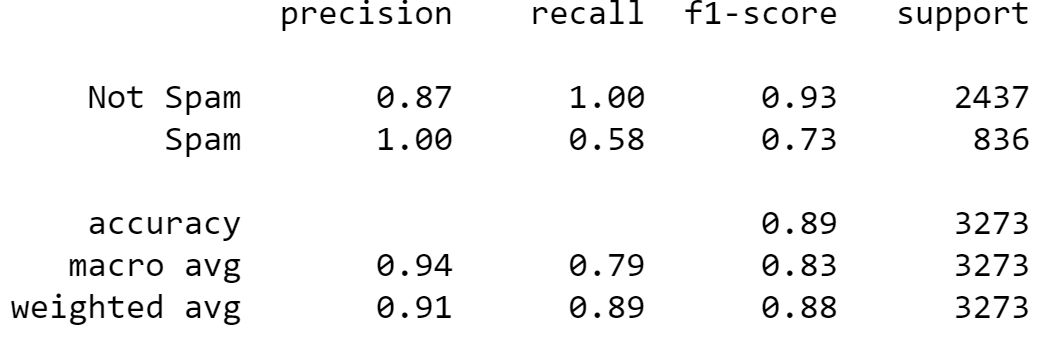


Figure : Classification Report using TFIDF.

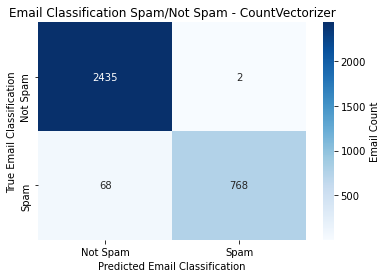
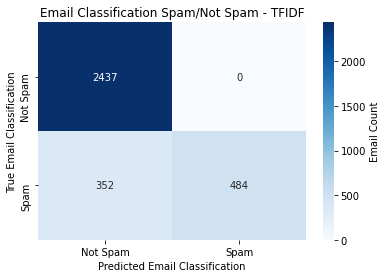
 

Figure 3: Confusion Matrix Heatmap using Count Vectorizer and TFIDF.

# Conclusion

The creation of a spam filter was done using two different types of tokenizers. The first, a count vectorizer, performed much better than the second, a term frequency inverse document frequency (TFIDF), with an overall accuracy of 98% and 89% respectively. Each email’s subject and body were parsed to allow the tokenizers to look at the content of each email after removing stop words, and then fit using a Naïve Bayes model. Further refinement allowing the model to perform a continuous fit as new emails come in; an update to this model using the partial fit parameter could prove beneficial. Since the accuracy depends upon finding the best feature using randomness, we can also try different classifiers such as random forest, svm, decision trees

# Code

# Email parsing and Classification (Case Study #3: DS7331)

SMU Student collaboration:

- Allen Miller

- Vijay Kaniti

- Venkata 'MG' Vanga

```python

import os

import re

```

```python

import numpy as np

import pandas as pd

import email

from html.parser import HTMLParser

from bs4 import BeautifulSoup

```

Checking all the directories in the folder of jupiter notebook.

```python

os.listdir()

```

Separating to see how many files and directories are there.

```python

basepath = '.'

for fname in os.listdir(basepath):

path = os.path.join(basepath, fname)

if os.path.isdir(path):

continue

else:

print(fname)

```

```python

files = [f for f in os.listdir('.') if (re.match(r'easy', f)) or (re.match(r'hard', f)) or (re.match(r'spam', f))]

files

```

```python

DIR = []

for i in range(len(files)):

DIR = files[i]

print(DIR,':',len([name for name in os.listdir(DIR) if os.path.isfile(os.path.join(DIR, name))]))

```

Creating function for stripping the header test.

```python

def strip\_head(text):

tmp=[]

flag = False

for i in text:

if i=='':

flag = True

if flag == True:

tmp.append(i)

return tmp

```

Creating function for multipart/text email parsing.

```python

def return\_text(mail):

if mail.is\_multipart():

tmp = mail.get\_payload()[0]

return return\_text(tmp)

elif mail.is\_multipart() == False:

return mail

```

'easy\_ham' parsing

```python

targets = []

data = []

count = 0

not\_spamList = os.listdir("./easy\_ham/")

for i in not\_spamList:

with open("./easy\_ham/"+i, "r",encoding= 'utf-8', errors='ignore') as f:

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

x = return\_text(x)

mailType = x.get\_content\_type()

if mailType == "text/plain":

tmp = x.get\_payload()

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

tmp = tmp.replace("-"," ")

data.append(tmp)

targets.append(0)

elif mailType == "text/html":

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

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

tmp = tmp.replace("-"," ")

data.append(tmp)

targets.append(0)

else:

print(f"msg {i} not parsed")

print(x.get\_content\_type())

print('Number of emails parsed in easy\_ham:',len(data))

print('Number of targets for easy\_ham:',len(targets))

```

'easy\_ham\_2' parsing

```python

not\_spamList1 = os.listdir("./easy\_ham\_2/")

for i in not\_spamList1:

with open("./easy\_ham\_2/"+i, "r",encoding= 'utf-8', errors='ignore') as f:

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

x = return\_text(x)

mailType = x.get\_content\_type()

if mailType == "text/plain":

tmp = x.get\_payload()

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

tmp = tmp.replace("-"," ")

data.append(tmp)

targets.append(0)

elif mailType == "text/html":

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

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

data.append(tmp)

targets.append(0)

else:

print(f"msg {i} not parsed")

print(x.get\_content\_type())

print('Number of emails parsed in easy\_ham + easy\_ham\_2:',len(data))

print('Number of targets for easy\_ham + easy\_ham\_2:',len(targets))

```

'hard\_ham' parsing

```python

not\_spamList2 = os.listdir("./hard\_ham/")

for i in not\_spamList2:

with open("./hard\_ham/"+i, "r",encoding= 'utf-8', errors='ignore') as f:

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

x = return\_text(x)

mailType = x.get\_content\_type()

if mailType == "text/plain":

tmp = x.get\_payload()

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

tmp = tmp.replace("-"," ")

data.append(tmp)

targets.append(0)

elif mailType == "text/html":

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

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

data.append(tmp)

targets.append(0)

else:

print(f"msg {i} not parsed")

print(x.get\_content\_type())

print('Number of emails parsed in easy\_ham + easy\_ham\_2 + hard\_ham:',len(data))

print('Number of targets for easy\_ham + easy\_ham\_2 + hard\_ham:',len(targets))

```

'spam' parsing

```python

spamList = os.listdir("./spam/")

for i in spamList:

with open("./spam/"+i, "r", encoding = 'utf-8' or 'us-ascii' or 'windows-1252' or 'iso-8859-1', errors='ignore') as f:

try:

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

except:

print(f"msg {i} not read")

x = return\_text(x)

mailType = x.get\_content\_type()

textType = x.get\_content\_charset()

if mailType == "text/plain":

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

targets.append(1)

elif mailType == "text/html":

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

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

data.append(tmp)

targets.append(1)

else:

print(f"msg {i} not parsed")

print(x.get\_content\_type())

print(textType)

print('Number of emails parsed in easy\_ham + easy\_ham\_2 + hard\_ham + spam:',len(data))

print('Number of targets for easy\_ham + easy\_ham\_2 + hard\_ham + spam:',len(targets))

```

'spam\_2' parsing

```python

spamList1 = os.listdir("./spam\_2/")

for i in spamList1:

with open("./spam\_2/"+i, "r", encoding = 'utf-8' or 'us-ascii' or 'windows-1252' or 'iso-8859-1', errors='ignore') as f:

try:

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

except:

print(f"msg {i} not read")

x = return\_text(x)

mailType = x.get\_content\_type()

textType = x.get\_content\_charset()

if mailType == "text/plain":

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

targets.append(1)

elif mailType == "text/html":

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

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

data.append(tmp)

targets.append(1)

else:

print(f"msg {i} not parsed")

print(x.get\_content\_type())

print(textType)

print('Number of emails parsed in easy\_ham + easy\_ham\_2 + hard\_ham + spam\_2:',len(data))

print('Number of targets for easy\_ham + easy\_ham\_2 + hard\_ham + spam\_2:',len(targets))

```

```python

print(len(data))

```

```python

len(data) == len(targets)

```

```python

import nltk

#nltk.download('stopwords')

#nltk.download('punkt')

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

stopset = set(stopwords.words('english'))

def remove\_stopwords(dt):

output\_array=[]

for sen in dt:

temp\_list=[]

for word in sen.split():

if word.lower() not in stopset:

temp\_list.append(word)

output\_array.append(' '.join(temp\_list))

return output\_array

a = remove\_stopwords(data)

print("data",len(data))

print("a",len(a))

```

```python

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(a, targets,test\_size=0.35, shuffle=True, random\_state=42)

```

```python

from sklearn.feature\_extraction.text import CountVectorizer

```

```python

cv = CountVectorizer()

cv\_data\_train = cv.fit\_transform(X\_train)

cv\_data\_test = cv.transform(X\_test)

```

```python

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

```

```python

model.fit(cv\_data\_train, y\_train)

```

```python

cv\_preds = model.predict(cv\_data\_test)

```

```python

from sklearn.metrics import accuracy\_score

```

```python

accuracy\_score(cv\_preds, y\_test)

```

```python

from sklearn.metrics import classification\_report

print(classification\_report(y\_test,cv\_preds,target\_names=['Not Spam', 'Spam']))

```

```python

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

cm0 = confusion\_matrix(y\_test,cv\_preds)

x\_axis\_labels = ['Not Spam', 'Spam']

y\_axis\_labels = ['Not Spam', 'Spam']

ax = plt.axes()

sns.heatmap(cm0, cmap='Blues', annot=True, fmt='d',xticklabels=x\_axis\_labels,

yticklabels=y\_axis\_labels, ax=ax, cbar\_kws={'label': 'Email Count',})

ax.set\_title('Email Classification Spam/Not Spam - CountVectorizer')

plt.xlabel("Predicted Email Classification")

plt.ylabel("True Email Classification")

plt.show

```

#### TFIDF

```python

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()

```

```python

tf\_data\_train = tfidf.fit\_transform(X\_train)

tf\_data\_test = tfidf.transform(X\_test)

```

```python

y\_train = np.array(y\_train)

```

```python

y\_test = np.array(y\_test)

```

```python

model.fit(tf\_data\_train, y\_train)

```

```python

preds = model.predict(tf\_data\_test)

```

```python

accuracy\_score(preds, y\_test)

```

```python

from sklearn.metrics import classification\_report

print(classification\_report(y\_test,preds,target\_names=['Not Spam', 'Spam']))

```

```python

cm0 = confusion\_matrix(y\_test,preds)

x\_axis\_labels = ['Not Spam', 'Spam']

y\_axis\_labels = ['Not Spam', 'Spam']

ax = plt.axes()

sns.heatmap(cm0, cmap='Blues', annot=True, fmt='d',xticklabels=x\_axis\_labels,

yticklabels=y\_axis\_labels, ax=ax, cbar\_kws={'label': 'Email Count',})

ax.set\_title('Email Classification Spam/Not Spam - TFIDF')

plt.xlabel("Predicted Email Classification")

plt.ylabel("True Email Classification")

plt.show

```