Case Study 3: Spam – Email Data

Tim Cabaza

June 10, 2024

**Abstract**

The case study is an exercise in building a classifier using Naïve Bayes to predict a binary class target variable (spam or not spam). The following is a list of steps that are covered in the case study: Parsing and Data Cleansing, , Multinomial Naïve Bayes, and KFold Cross Validation.

**1 Introduction**

The stakeholder in this study is an IT department. Given the significant amount of Spam, the stakeholder has specified that a filter that can detect and classify spam emails is required and the data provided is set of 5 text files that contain spam emails and “ham” emails. The specificity of the filter as far as over-filtering or under-filtering spam email is left at the researcher’s digression.

**2 Methods**

**Exploratory Data Analysis:** The data has contains both spam and non-spam emails (aka ham emails which is what they will be referred to going forward) were provided in 5 files with a mix of text types.

A computer screen with white text

Description automatically generated

**Figure 1:** Email Data Sample from stakeholder file

The data is in a format that is not workable, the data was thus loaded and the types detected using the a counter and the idea of recursion to pluck out the types (whether multipart or not) to determine the level and type of parsing needed to accurately capture enough of the text.

**Table 1:** Data Types to be parsed

|  |  |
| --- | --- |
| **Type** | **Count** |
| Text/plain | 7417 |
| Multipart/alternative | 326 |
| Text/html | 1193 |
| Multipart/mixed | 179 |
| Multipart/related | 56 |
| Multipart/signed | 180 |
| Text/plain | 1 |
| Multipart/report | 5 |

The target variable is set to Spam or No Spam binary classifier (0/1) with 9357 targets. After parsing the text using the beautiful soup package for HTML data and email package for email based multi-part data, it would be ideal to stay near or about the number of targets to return an acceptable model. The data is transformed using counter package and recursive methods to go through each file in the directory to convert the text data to vectors.

**Table 2:** Number of targets

|  |  |
| --- | --- |
| **Target** | **Count** |
| Spam / Not Spam | 9357 |

**3 Results**

Using the MulitnomialNB Classifier model and the cross-validation metric from SciKit Learn the following results were returned.

**Table 3:** MultinomialNB Results

|  |
| --- |
| **Cross-Val Accuracy Score** |
| 0.90918803 |
| 0.98824786 |
| 0.98770711 |
| 0.99358632 |
| 0.98931053 |

Cross validation was performed on 5 folds, where the model is trained on 4 folds and the remain fold is repeated 5 times to return an average score (Table 4). The log probabilities of each class were also captured and recorded. The classification report and confusion matrix provide evidence of positive results with the model performing well at 97% accuracy (expected to be high).

**Table 4:** Results Table

|  |  |
| --- | --- |
| **Log Probabilities by Class:** | **Class0:** 0.3100, **Class1**: 0.69 |
| **Best Cross-validation Mean Accuracy Score:** | 0.973607970508024 |
| **Classification Report:** |  |
| **Confusion Matrix Report** |  |

**4 Conclusions**

The case study assessed an unstructured set of various types of text files requiring unique parsing methods to properly extract the text data need to build a spam filter. The HTML data and muli-part data required extra attention, overall the level of parsing appears to be well considering the score the Multinomial Naïve Bayes model returned (97% accuracy). To illustrate the effectiveness of the filter method, see the visual below in Figure 2 to visualize the clusters of spam and ham.

A blue and green dot diagram

Description automatically generated

**Figure 2:** Clusters -- Spam (1), Not Spam (0)

**Appendix:** Code

import os

import email

from sklearn.feature\_extraction.text import CountVectorizer

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

import numpy as np

# Set the directory path

directory = "/Users/tmc/Desktop/MS\_SMU\_Admin/05\_2024Summer/QTW/Data\_Science/Case\_study3/"

# Change the current working directory

os.chdir(directory)

os.listdir("/Users/tmc/Desktop/MS\_SMU\_Admin/05\_2024Summer/QTW/Data\_Science/Case\_study3/SpamAssassinMessages/")

# # join email and file together

# # you need to parse the HTML email and get to it

# # encoding had to be set

# # introduce some error handling with try and except

# # DERP is a placeholder for the error handling

# # counting different codec types

# # Counter a fast way to count things - boiler plate

# #

from collections import Counter

types = Counter()

targets=[]

text=[]

for root,dirs,files in os.walk("."):

for file in files:

if "ipynb" in file:

pass

else:

with open(os.path.join(root,file), "r",encoding='latin-1') as f:

try:

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

type\_ = tmp.get\_content\_type()

if type\_== 'text/html':

print("Need to parse HTML")

types[type\_]+=1

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

except:

pass

tmp = "DERP"

text.append(tmp)

if "spam" in root:

targets.append(1)

else:

targets.append(0)

print(root, file, types)

import email

from bs4 import BeautifulSoup

from collections import Counter

from sklearn.feature\_extraction.text import CountVectorizer

types = Counter()

targets = []

text\_data = []

# Function to extract text from email parts

def extract\_text\_from\_email(msg):

text\_parts = []

if msg.is\_multipart():

for part in msg.get\_payload():

text\_parts.extend(extract\_text\_from\_email(part))

else:

content\_type = msg.get\_content\_type()

if content\_type == 'text/plain':

text\_parts.append(msg.get\_payload(decode=True).decode('latin-1'))

elif content\_type == 'text/html':

html\_content = msg.get\_payload(decode=True).decode('latin-1')

soup = BeautifulSoup(html\_content, 'html.parser')

text\_parts.append(soup.get\_text())

return text\_parts

for root, dirs, files in os.walk("."):

for file in files:

if "ipynb" in file:

continue

try:

with open(os.path.join(root, file), "r", encoding='latin-1') as f:

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

type\_ = msg.get\_content\_type()

types[type\_] += 1

text\_parts = extract\_text\_from\_email(msg)

combined\_text = ' '.join(text\_parts).replace("\n", " ")

text\_data.append(combined\_text)

if "spam" in root:

targets.append(1)

else:

targets.append(0)

except Exception as e:

print(f"Error processing file {file}: {e}")

text\_data.append("DERP")

targets.append(0)

# Ensure the combined text length

processed\_text\_data = []

for text in text\_data:

if len(text) > 9200:

processed\_text\_data.append(text[:9200])

else:

processed\_text\_data.append(text.ljust(9200))

vectorizer = CountVectorizer()

X\_features = vectorizer.fit\_transform(processed\_text\_data)

print(X\_features.toarray())

print(vectorizer.get\_feature\_names\_out())

# # Select a row index to inspect

# row\_index = 0 # Change this to the row index you want to inspect

# print("Email Content:")

# print(text\_data[row\_index])

# print("\nFeatures and Frequencies:")

# for feature\_name, frequency in zip(vectorizer.get\_feature\_names\_out(), X\_features.toarray()[row\_index]):

# if frequency > 0:

# print(f"{feature\_name}: {frequency}")

print(len(targets))

types

from sklearn.naive\_bayes import MultinomialNB

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

from sklearn.manifold import TSNE

import matplotlib.pyplot as plt

# !pip install seaborn

import seaborn as sns

from sklearn.metrics import classification\_report, ConfusionMatrixDisplay

model = MultinomialNB(alpha=1.0)

cv\_scores = cross\_val\_score(model, X\_features, targets, cv=5)

print(f"Cross-validation scores: {cv\_scores}")

print(f"Mean cross-validation score: {cv\_scores.mean()}")

# # Get prediction probabilities using cross-validation

# model.fit(X\_features, targets)

# model.predict\_log\_proba(X\_features)

# Generate cross-validated predictions

predictions = cross\_val\_predict(model, X\_features, targets, cv=5)

# Print the classification report

print(classification\_report(targets, predictions))

# Plot the confusion matrix

ConfusionMatrixDisplay.from\_predictions(targets, predictions, cmap='Greens')

plt.show()

# Get prediction probabilities using cross-validation

# Fit the model on the entire dataset

model.fit(X\_features, targets)

# Get the log probabilities for each class

log\_proba = model.predict\_log\_proba(X\_features)

print(log\_proba)

# Get the predicted class

predicted\_class = model.predict(X\_features)

predicted\_class

proba = np.exp(log\_proba)

# Print the converted probabilities

print(proba)

# Example of inspecting a few instances

for i in range(10): # Change the range if you want to inspect more instances

print(f"Email {i+1}:")

print(f"Text: {processed\_text\_data[i][:200]}...") # Show the first 200 characters

print(f"Actual target: {'Spam' if targets[i] == 1 else 'Not Spam'}")

print(f"Predicted probabilities: Not Spam: {proba[i][0]:.4f}, Spam: {proba[i][1]:.4f}")

print("\n")

# Normalize probabilities to sum up to 1

proba /= np.sum(proba)

# Print the probabilities

for i, prob in enumerate(proba):

print(f"Class {i}: {prob:.4f}")

from sklearn.manifold import TSNE

import matplotlib.pyplot as plt

# !pip install seaborn

import seaborn as sns

from sklearn.metrics import classification\_report, ConfusionMatrixDisplay

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

# Reduce dimensionality using t-SNE for visualization

tsne = TSNE(n\_components=2, random\_state=42)

X\_embedded = tsne.fit\_transform(X\_features.toarray())

# Plot the t-SNE results

plt.figure(figsize=(10, 6))

sns.scatterplot(x=X\_embedded[:, 0], y=X\_embedded[:, 1], hue=targets, palette='viridis')

plt.title('t-SNE Visualization of Email Data')

plt.xlabel('t-SNE Feature 1')

plt.ylabel('t-SNE Feature 2')

plt.show()

# Additional logging for verification

print("\nSample parsed email content:")

for i, (t, txt) in enumerate(zip(targets, processed\_text\_data)):

if i < 5: # Print the first 5 samples for inspection

print(f"\nTarget: {t}")

print(f"Content:\n{txt[:500]}...") # Print the first 500 characters of the text

else:

break

# Print the counter to verify email types

print("\nCounter of email types:")

print(types)