Case Study III

Randy Kim, Kati Schuerger, Will Sherman

October 3, 2022

1 Introduction

Objective

The objective of this case study was to build a dependable email (antispam) filter: a model that can classify an email as spam or not-spam based on the contents of the email. The intended use of this filter is to assist with reducing the number of useful emails that get lost in the shuffle due to a high volume of non-relevant (spam) emails.

An important part of this case study was pre-processing of the data. Prior to model development, we needed to first transform the data in the form it was provided (email content) into a data object that could be used to build a model. We then used natural language processing (NLP) and classification methods to generate the antispam filter.

Dataset

The dataset was provided by an IT client in the form of the actual emails themselves, rather than in a pre-existing dataframe object or spreadsheet. Therefore, an important part of this case study was pre-processing of the data, given that we needed to generate our own dataset object that could be used to develop the spam filter model.

2 Methods

Data pre-processing

The data was provided in the form of email text files in five different folders including two spam folders. To have a usable dataset to model, we conducted data preprocessing steps to reduce the data to pure text form—removing headers, footers, attachments, and similar items. The goal was to use the email body content to guide the model.

We then extracted the body of emails, identified mail, and text types, and assigned the target to binary variables, 0 which represents 'not a spam email' and 1 represents 'spam email'.

For 'text/HTML,' we used Beautiful Soup function to HTML parsing and for multipart, we broke it down to 'text/plain' and 'text/HTML' and combined them in a string. Based on text encoding and other issues we stripped out tab and newline objects that may have appeared as string literals, thus reducing the number of identified "words" when utilizing the NLP methods.

Following this, the body of each email was dissected to identify some commonly used words and the ratio of spam and non-spam emails by Mail Type. The highest proportion of spam emails occurred in Multipart/Alternative, Multipart/Mixed, and HTML text mail; although a not insignificant amount also occurred in ‘text/plain’ mail (Figure 1). Once we had our dataset transformed to text, we began the NLP process of creating the feature space that would allow us to use Multinomial Naïve Bayes and other classifiers.

Count vectorizer

One common NLP method is the Count Vectorizer (bag of words model). The Count Vectorizer works by taking the number of times a given word appears in the instance, and then treating those counts as features. This allows us to transform the data to numerical values that can be used to build a model.

We employed Count Vectorizer from SKLEARN, fitting the vectorizer to the dataset, which created a vocabulary for our counts. This is essentially one hot encoding our data.

TF-IDF (Term frequency, inverse document frequency)

One limitation of the Count Vectorizer is that if certain words occur in everything, they will have a high occurrence count across the board, which does not actually tell us much about the dataset. To help address this limitation, and better enable the model to distinguish between document types (spam or not-spam), we investigated inverse document frequency: TF-IDF (Term frequency, inverse document frequency).

The TF-IDF, or term frequency, inverse document frequency, typically shows an improvement over the count vectorizer in that it helps establish features that are words represented as numbers. Those numbers are counts, and then those counts are normalized to how often they appear per documents. The key here is that it helps with words that occur commonly, because it lessens the importance of those word-values when we multiply by the inverse of the document frequency.

However, we opted not to use this method after full model inspection as it worsened the precision for the final models.

Classification Methods

We explored several classification methods for correctly distinguish email classes. SKLEARN’s K-Nearest Neighbors (KNN) provided a baseline method to model the outcomes. We also looked at Random Forest Classification which relies on many decision tree classifiers “voting” on sub-sampled data and classifying output on the max votes.

The two models we submit for consideration are those which performed best at letting the fewest spam mails enter the inbox but filtering more real messages (lower false negatives / higher false positives) and not classifying real email messages as spam (low false positives / higher false negatives)—multi-layer perceptron and multinomial naïve bayes, respectively.

Train/Test/Validate split

Before we began modeling, we randomized the arrangement of the data samples and separated the data into 2 groups: a training dataset for model fitting and a validation dataset (holdout group of

20%) that was used to measure model performance on unseen data—evaluating whether the model generalizes well.

Scoring metric

We used a confusion matrix and a classification report to evaluate and compare each model. Both the matrix and the report provided measuring of Recall, Precision, and Accuracy. Depending on the models, it could be difficult to compare models with high precision and low recall or vice versa, so the classification report also provides F1-score (the harmonic mean of precision and recall) to measure them at the same time.

OOB (Out of bagging score)

Out of bagging score means that the samples that were not used to build the model were actually run through as a pseudo test set. This is similar to cross-validation for assessing model accuracy, with one drawback: that it is a biased score. It is biased, because it is using the same data to train and test each time, meaning the model becomes more familiar with and better at predicting the test data, much like when a model is overfit.

3 Results

The distribution of spam messages across the entire series of messages can primarily be attributed to text/plain messages and text/html. They make up the largest proportion in multipart/alternative, multipart/mixed, and text/html message classes (Figure 1).

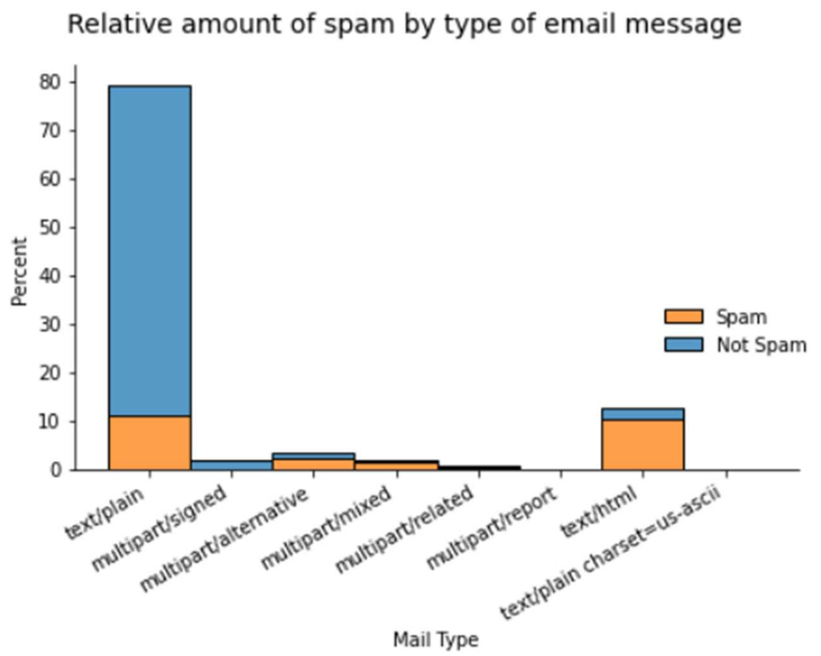


Figure 1. Percent distribution of spam messages by type of content in each email message.

In the initial comparison of models, KNN Classifier and Random Forest Classifier were evaluated for establishing a baseline. The initial findings showed an F1 of 0.91 for KNN and 0.95 for Random Forest. Therefore, we continued with the Random Forest baseline (Figures 2).

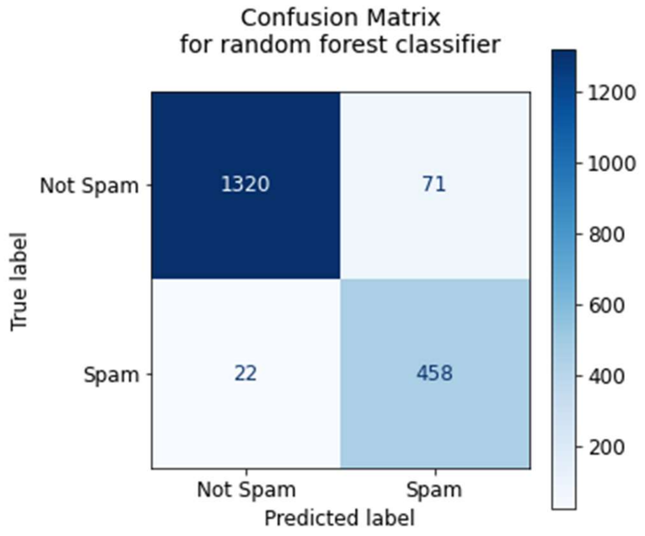


Figure 2. This confusion matrix shows the distribution of the assignments of spam and not spam mail messages on testing data for the Random Forest model.

The Multinomial Naïve Bayes model had comparable F1 of 0.94 across the 1871 test samples. However, investigating the confusion matrix and classification reports showed that the precision for predicting spam was 0.989, and the false positive rate (*i.e.*, not spam classified as spam) was 0.997 (aggregate metrics found in Appendix – Table 1). Therefore, this model has the highest fidelity of real email messages reaching users (Figure 3).

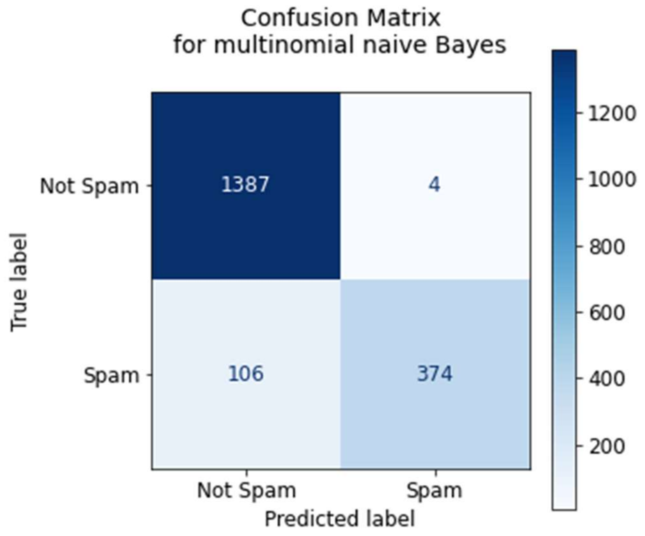


Figure 3. This confusion matrix shows the distribution of the assignments to spam and not spam mail messages on testing data for multinomial naive Bayes.

A competing model in the opposite direction of Multinomial Naïve Bayes is the Multilayer Perceptron (Figure 4). In practice, this had the fewest miscategorized emails: 79 misclassifications as opposed to 110 for Naïve Bayes and 93 for Random Forest. This resulted in the highest F1-accuracy at 0.96. However, it does misclassify real emails as spam at a higher rate than baseline and the Naïve Bayes model.

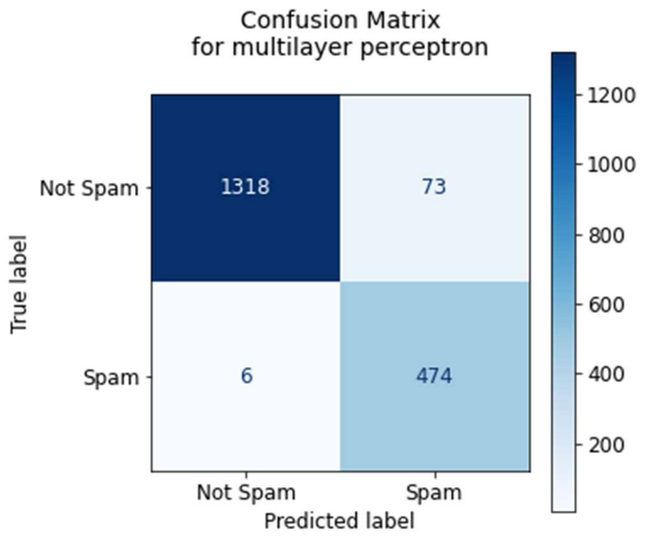


Figure 4. This This confusion matrix shows the distribution of the assignments to spam and not spam mail messages on testing data for the multilayer perceptron.

4 Conclusion

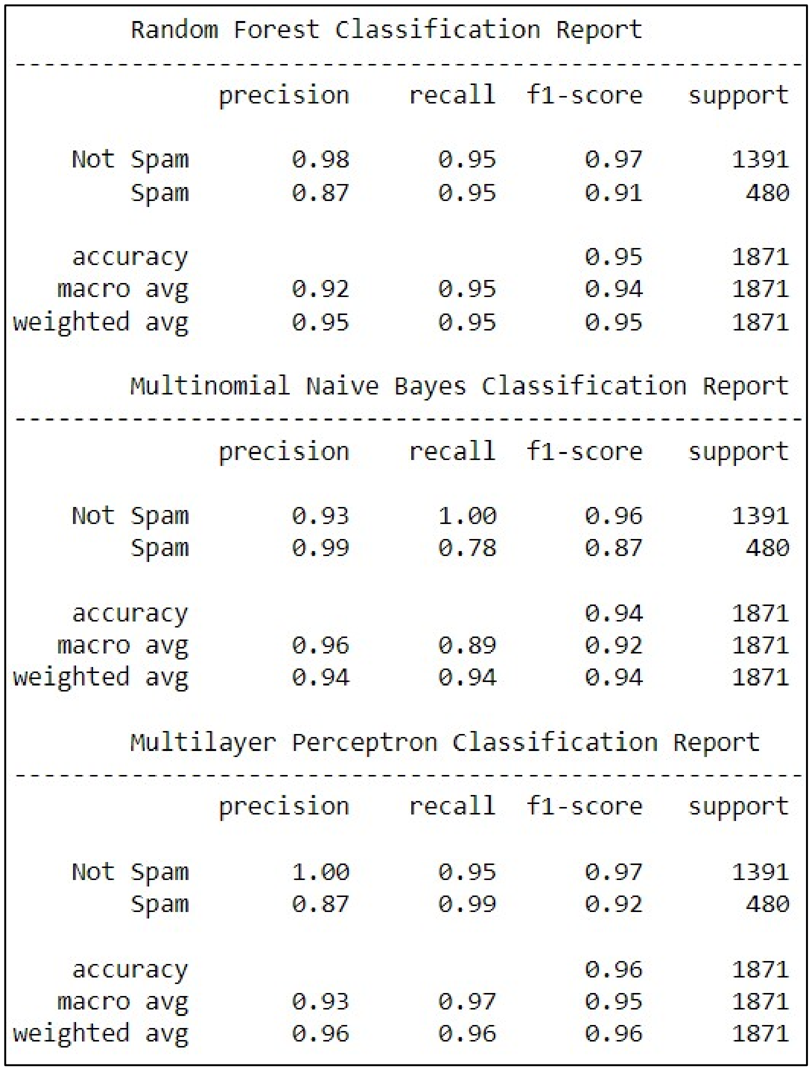
Based on the models we looked at, we believed that Multinomial Native Bayes and Multilayer Perceptron are the best performing single models. The possible advancement from these single models could be an ensemble (similar to the Random Forest ensemble). However, this would be tailored based on the best performing classifiers: multinomial naïve Bayes and multilayer perceptron.

A further question is whether the IT team would like us to continue with equal weight given to the precision and recall (F1 score), or if we should more heavily weight one of these (F_β). Should the model or stacked model prioritize high fidelity of real mail reaching the end user or prioritize filtering spam and possibly misclassify more real messages?

Future inquiries into this topic should also seek to break the spam feature out into benign advertising messages and malicious fishing messages. This would drastically change the question of interest as well as how we assign our evaluation in terms of precision and recall.

Appendix

Table 1.



```
        Random Forest Classification Report
-------------------------------------------------------
              precision    recall  f1-score   support

    Not Spam       0.98      0.95      0.97      1391
        Spam       0.87      0.95      0.91       480

    accuracy                           0.95      1871
   macro avg       0.92      0.95      0.94      1871
weighted avg       0.95      0.95      0.95      1871

        Multinomial Naive Bayes Classification Report
-------------------------------------------------------
              precision    recall  f1-score   support

    Not Spam       0.93      1.00      0.96      1391
        Spam       0.99      0.78      0.87       480

    accuracy                           0.94      1871
   macro avg       0.96      0.89      0.92      1871
weighted avg       0.94      0.94      0.94      1871

        Multilayer Perceptron Classification Report
-------------------------------------------------------
              precision    recall  f1-score   support

    Not Spam       1.00      0.95      0.97      1391
        Spam       0.87      0.99      0.92       480

    accuracy                           0.96      1871
   macro avg       0.93      0.97      0.95      1871
weighted avg       0.96      0.96      0.96      1871
```

Table 1. Depicts the aggregated scoring across multiple metrics by the models selected in the report.

Code

```
#!/usr/bin/env python
# coding: utf-8

import os
import re
import email
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from bs4 import BeautifulSoup
from collections import Counter
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer
# from sklearn.feature_extraction.text import TfidfVectorizer

base_dir = r"C:\Users\sherm\Documents\Grad School - Classes\MSDS - 7333 - Quantifying the
World\Case Study 3"
# base_dir = "D:/Project/Quantifying the World/SpamAssassinMessages"
folders = []

# Read in the data using os.walk to go through multiple directories
for i in os.walk(base_dir):
    if i[0] != base_dir:
        folders.append(i[0])

# os.walk was easier than glob in this case

for folder in folders:
    print(folder) # visualize the folders all the emails live in

def getFiles(folder):
    files = []

    for file in os.listdir(folder):
        files.append(file)

    return files

for folder in folders:
    if re.search("spam", folder):
        print(folder)

mailType = []
textType = []
body = []
target = []

for folder in folders:
    for file in getFiles(folder):
        with open(f"{folder}/{file}", encoding = 'ISO-8859-1') as file_handle: #encoding
handles unusual characters
            x = email.message_from_file(file_handle) # getting email object

        if re.search("spam", folder):
            target.append(1)
        else:
            target.append(0)

        mail = x.get_content_type()
        text = x.get_content_charset()

        mailType.append(mail)
        textType.append(text)

        if mail == 'text/plain':
            body.append(x.get_payload())   #   getting   payload   of   all   text/plain
charactertypes

        elif mail == 'text/html':
            tmp  =  BeautifulSoup(x.get_payload(),  'html.parser')  #  getting  payload  of
text/html
            tmp = tmp.text.replace('\n', ' ')
            body.append(tmp)

        elif x.is_multipart(): # Randy's code to iterate through multipart messages and
obtain payload
```

```
            combine = []

            for i in x.get_payload():
                mtype = i.get_content_type()
                ttype = i.get_content_charset()

                if mtype == 'text/plain':
                    combine.append(i.get_payload())

                elif mtype == 'text/html':
                    tmp = BeautifulSoup(i.get_payload(), 'html.parser')
                    tmp = tmp.text.replace('\n', ' ')
                    combine.append(tmp)

            body.append([' '.join(str(i)) for i in combine])

        else:
            body.append(x.get_payload())

### The above code is credited to Dr. Robert Slater with minor edits

maildf = pd.DataFrame() # creating dataframe of payloads
maildf["Body"] = body
maildf["Mail Type"] = mailType
maildf["Text Type"] = textType
maildf["Target"] = target
maildf

for i in range(len(maildf)):
    if type(maildf.iloc[i,0]) == list:
        maildf.iloc[i,0] = ''.join(maildf.iloc[i,0])

maildf

# visualization of distribution of spam payloads by mail type
g  =  sns.displot(maildf,  x='Mail  Type',  hue='Target',  multiple='stack',stat='percent',
legend=False)
g.fig.suptitle('Relative amount of spam by type of email message', fontsize=14)
g.set_xticklabels(rotation=30, horizontalalignment='right')
g.add_legend(title='', labels=['Spam', 'Not Spam'])
plt.tight_layout()


import re
word_counts = Counter()

mail_data_df = maildf.copy()

for i in range(maildf['Body'].shape[0]):
    tmp = maildf.iloc[i,0].lower() # casting messages to lowercase
    tmp = tmp.replace('\r',' ').replace('\n',' ').replace('\t',' ').replace('<br>',' ')
#removing string newlines, tabs, breaks
    tmp = tmp.split(" ") # splitting string sentences into a list of "words"
    for token in tmp:
        if r'\n' in token:
            token.replace(r'\n',' ') # catching string literal newline
        if '\t' in token:
            token.replace(r'\t',' ') # catching string literal tab
        # count vectorized words
        word_counts[token.strip()] += 1
    # re-creating a dataframe for with text stripped of special characters
    mail_data_df.iloc[i,0] = ' '.join([word for word in tmp])

extra = Counter({"<pad>":10000001,"<unk>":1000000})
word_counts = word_counts + extra
```

```
mail_data_df

print(len(word_counts))
word_counts.most_common(40000)

for word in list(word_counts): # remove single letter words
    if len(word) < 2:
        del word_counts[word]

print(len(word_counts))
word_counts.most_common(13000) # this represents about 10% of the data

# ### Generating Training and Testing Data

x_data = mail_data_df[['Body','Mail Type', 'Text Type']].copy()
y_data = mail_data_df[['Target']].copy() # where 0 is not spam and 1 is spam

# #### Train/Test split for OOB Model Evaluation

from sklearn.model_selection import train_test_split
x_train,  x_test,  y_train,  y_test  =  train_test_split(x_data,  y_data,  stratify=y_data,
test_size=0.20, random_state=42)

c_vector = CountVectorizer()

# x_fit represents the data transformation utilizing the count vectorizer method of word
tokenization
x_fit = c_vector.fit_transform(x_train['Body'])

class_labels = ['Not Spam','Spam']

# ### KNN
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(x_fit.toarray(), y_train.values.ravel())

# x_new is the test data transformed using the count_vectorized method
x_new = c_vector.transform(x_test['Body'])
y_preds_knn = knn.predict(x_new.toarray())

cm = confusion_matrix(y_test['Target'], y_preds_knn)

font = {'size'   : 12}
plt.rc('font', **font)

c_disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(5,5))
plt.title('Confusion Matrix\nfor K neighbors classifier\n', fontsize=14)
c_disp.plot(cmap=plt.cm.Blues, ax=ax)
ax.xaxis.set_ticklabels(class_labels); ax.yaxis.set_ticklabels(class_labels)

from sklearn.metrics import classification_report
print(classification_report(y_test['Target'], y_preds_knn, target_names=class_labels))

# ### Random Forest
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(x_fit.toarray(), y_train.values.ravel())

y_preds_rf = rf.predict(x_new.toarray())

cm = confusion_matrix(y_test['Target'], y_preds_rf)
```

```
font = {'size'   : 12}
plt.rc('font', **font)

c_disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(5,5))
plt.title('Confusion Matrix\nfor random forest classifier\n', fontsize=14)
c_disp.plot(cmap=plt.cm.Blues, ax=ax)
ax.xaxis.set_ticklabels(class_labels); ax.yaxis.set_ticklabels(class_labels)

print(classification_report(y_test['Target'], y_preds_rf, target_names=class_labels))

# ### Naïve Bayes
from sklearn.naive_bayes import MultinomialNB

nb = MultinomialNB()
nb.fit(x_fit.toarray(), y_train.values.ravel())

y_preds_nb = nb.predict(x_new.toarray())

cm = confusion_matrix(y_test['Target'], y_preds_nb)

font = {'size'   : 12}
plt.rc('font', **font)

c_disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(5,5))
plt.title('Confusion Matrix\nfor multinomial naive Bayes\n', fontsize=14)
c_disp.plot(cmap=plt.cm.Blues, ax=ax)
ax.xaxis.set_ticklabels(class_labels); ax.yaxis.set_ticklabels(class_labels)

print(classification_report(y_test['Target'], y_preds_nb, target_names=class_labels))

# ### Multilayer Perceptron
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier()
mlp.fit(x_fit.toarray(), y_train.values.ravel())

y_preds_mlp = mlp.predict(x_new.toarray())

cm = confusion_matrix(y_test['Target'], y_preds_mlp)

font = {'size'   : 12}
plt.rc('font', **font)

c_disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(5,5))
plt.title('Confusion Matrix\nfor multilayer perceptron\n', fontsize=14)
c_disp.plot(cmap=plt.cm.Blues, ax=ax)
ax.xaxis.set_ticklabels(class_labels); ax.yaxis.set_ticklabels(class_labels)

print(classification_report(y_test['Target'], y_preds_mlp, target_names=class_labels))

# ### Generate final “table”
print("\tRandom Forest Classification Report\n---------------------------------------------
-----------")
print(classification_report(y_test['Target'], y_preds_rf, target_names=class_labels))
print("\tMultinomial Naive Bayes Classification Report\n-----------------------------------
---------------------")
print(classification_report(y_test['Target'], y_preds_nb, target_names=class_labels))
print("\tMultilayer Perceptron Classification Report\n-------------------------------------
--------------------")
print(classification_report(y_test['Target'], y_preds_mlp, target_names=class_labels))
```