Naif A. Ganadily Programming Assignment 2

EE P 596: Advanced Introduction to Machine Learning

Programming Assignment 2: Spam Classification + Kaggle Competition
Due January 14th, 2023, by 11:59 PM
Student: Naif A Ganadily
Instructor - Prof. Karthik Mohan
TA - Ayush Singh
Grader - Fatwir SM

Kaggle Competition: https://www.kaggle.com/competitions/spam-classification-winter-2023/leaderboard?

Guidelines for this Notebook:

- Dont run ALL the models due to the amount of computational power
- Learn the process and study the concepts

```python
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split

# Added Libraries
import nltk
from nltk.corpus import stopwords
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
nltk.download('punkt')
import string

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Email Dataset

Loading the dataset

```
#local_file="all_emails.csv"
#data_set =
pd.read_csv(local_file,sep=',',index_col=0,header=None,engine='python,er
ror_bad_lines=False)

data_set = pd.read_csv('all_emails.csv')
data_set.shape

(4260, 3)

test_set = pd.read_csv("eval_students_2.csv")
test_set.shape

(1468, 2)
```

Explanation (Loading the data set):

The code loads two data sets using pandas library's read_csv function, "all_emails.csv" and "eval_students_2.csv" and uses the shape attribute to display the number of rows and columns in each data set to ensure they were loaded correctly.

1) Inspecting the dataset

```
def data_inspecter1(data_set):
    # 1. Print a few lines (i.e. each line is an email and a label) from the data_set containing spam (use a pandas functionality - e.g. getting the top lines)
    return data_set.head(10)

data_inspecter1(data_set)

     id                                               text  spam
0  1235  Subject: naturally irresistible your corporate...     1
1  1236  Subject: the stock trading gunslinger  fanny i...     1
2  1238  Subject: 4 color printing special  request add...     1
3  1239  Subject: do not have money , get software cds ...     1
4  1240  Subject: great nnews  hello , welcome to medzo...     1
5  1242  Subject: save your money buy getting this thin...     1
6  1243  Subject: undeliverable : home based business f...     1
7  1244  Subject: save your money buy getting this thin...     1
8  1246  Subject: save your money buy getting this thin...     1
9  1247  Subject: brighten those teeth  get your  teeth...     1

data_set.columns

Index(['id', 'text', 'spam'], dtype='object')

def data_inspecter2(data_set):
  not_spam = data_set[data_set['spam'] == 0]
  print(not_spam.head(5))


    # 2. Print a few lines from data_set that are not spam
data_inspecter2(data_set)

        id                                               text  spam
1026  2603  Subject: hello guys ,  i ' m " bugging you " f...     0
1027  2604  Subject: sacramento weather station fyi  - - ...     0
1028  2605  Subject: from the enron india newsdesk - jan 1...     0
1029  2606  Subject: re : powerisk 2001 - your invitation ...     0
1030  2607  Subject: re : resco database and customer capt...     0
```

```python
def data_inspecter3(data_set):
  df = data_set[data_set['id'].between(5000, 5011)]
  return df

data_inspecter3(data_set)
    # 3. Print the emails between lines 5000 and 5010 in the data set
```

```
        id                                               text  spam
2790  5000  Subject: re : enron - resume interview of jame...     0
2791  5002  Subject: re : nj alliance  michael lassle is i...     0
2792  5003  Subject: contract summaries  attached are the ...     0
2793  5004  Subject: re : working with enron on catastroph...     0
2794  5006  Subject: maureen raymoin ' ds review  norma , ...     0
2795  5007  Subject: john sherriff ' s copper position  te...     0
2796  5008  Subject: is the supply rebound beginning ? an ...     0
2797  5009  Subject: re : resco database and customer capt...     0
```

Explanation (Inspecting the data set):

This code defines three functions:

1- data_inspecter1: prints the first 10 lines of the data_set where the 'spam' column is equal to 1.
2- data_inspecter2: prints the first 5 lines of the data_set where the 'spam' column is equal to 0.
3- data_inspecter3: filters the data_set to only include rows where the 'id' column is between 5000 and 5011, and returns the filtered dataframe. These functions are used to inspect the data and understand its characteristics. It's done to prepare the data for further analysis or modeling.

2) Data processing step for this HW:

Do the following process for all emails in your data set - 1) Tokenize into words 2) Remove stop/filler words and 3) Remove punctuations Below - We have it done for a sample sentence

Tokenizer

Apply a tokenizer to tokenize the sentences in your email - So your sentence gets broken down to words. We will use a tokenizer from the NLTK library (Natural Language Tool Kit) below for a single sentence.

```
# Example Sentence
from nltk.tokenize import word_tokenize
nltk.download('punkt')
sentence = """Subject: only our software is guaranteed 100 % legal .
name - brand software at low , low , low , low prices everything comes
to him who hustles while he waits . many would be cowards if they had
courage enough ."""
sentence_tokenized = word_tokenize(sentence)
print(sentence_tokenized)
print()
sentence_tokenized
#nltk.download('punkt')
```

```
['Subject', ':', 'only', 'our', 'software', 'is', 'guaranteed', '100',
'%', 'legal', '.', 'name', '-', 'brand', 'software', 'at', 'low', ',',
'low', ',', 'low', ',', 'low', 'prices', 'everything', 'comes', 'to',
'him', 'who', 'hustles', 'while', 'he', 'waits', '.', 'many', 'would',
'be', 'cowards', 'if', 'they', 'had', 'courage', 'enough', '.']
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!

['Subject',
 ':',
 'only',
 'our',
 'software',
 'is',
 'guaranteed',
 '100',
 '%',
 'legal',
 '.',
 'name',
 '-',
 'brand',
 'software',
 'at',
 'low',
 ',',
 'low',
 ',',
 'low',
 ',',
 'low',
 'prices',
 'everything',
 'comes',
 'to',
 'him',
 'who',
 'hustles',
 'while',
 'he',
 'waits',
 '.',
 'many',
 'would',
 'be',
 'cowards',
 'if',
 'they',
 'had',
 'courage',
 'enough',
 '.']
```

Stop Words: Remove Stop Words (or Filler words) using stop words list

```
from nltk.corpus import stopwords
nltk.download('stopwords')
filtered_words = [word for word in sentence_tokenized if word not in
stopwords.words('english')]
filtered_words
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
['Subject',
 ':',
 'software',
 'guaranteed',
 '100',
 '%',
 'legal',
 '.',
 'name',
 '-',
 'brand',
 'software',
 'low',
 ',',
 'low',
 ',',
 'low',
 ',',
 'low',
 'prices',
 'everything',
 'comes',
 'hustles',
 'waits',
 '.',
 'many',
 'would',
 'cowards',
 'courage',
 'enough',
 '.']
```

Punctuations: Remove punctuations and other special characters from tokens

3) Exercise:

Inspect the resulting list below for any of your emails - Does it look clean and ready to be used for the next step in spam detection? Any other pre-processing steps you can think of or may want to do before spam detection? How about including other NLP features like bi-grams and tri-grams?

```
new_words = [word for word in filtered_words if word.isalnum()]
new_words
```

```
['Subject',
 'software',
 'guaranteed',
 '100',
 'legal',
 'name',
 'brand',
 'software',
 'low',
 'low',
 'low',
 'low',
 'prices',
 'everything',
 'comes',
 'hustles',
 'waits',
 'many',
 'would',
 'cowards',
 'courage',
 'enough
```

3) Applying pre-processing to the entire dataset

```
# Removing any duplicates
# Checking any null values
print(data_set.shape)
print()
data_set.drop_duplicates(inplace = True)
print(data_set.shape)
print()
data_set.isnull().sum()
```

```
(4260, 3)

(4260, 3)


id      0
text    0
spam    0
dtype: int64
```

```
def pre_processor(data_set):
  no_p = [char for char in data_set if char not in string.punctuation]
  no_p = ''.join(no_p)
  words = [word for word in no_p.split() if word.lower() not in
stopwords.words('english')]
  return words

data_set['text'].head().apply(pre_processor)
```

```
0    [Subject, naturally, irresistible, corporate, ...
1    [Subject, stock, trading, gunslinger, fanny, m...
2    [Subject, 4, color, printing, special, request...
3    [Subject, money, get, software, cds, software,...
4    [Subject, great, nnews, hello, welcome, medzon...
Name: text, dtype: object
```

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(analyzer = pre_processor)
vectorizer.fit(data_set['text'])

emails = vectorizer.transform(data_set['text'])

df_emails = pd.DataFrame(emails.toarray())
print(df_emails.shape)
print()
print(data_set['spam'].shape)
df_emails.head()
```

```
(4260, 32462)

(4260,)

   0      1      2      3      4      5      6      7      8      9
...  \
0      0      0      0      0      0      0      0      0      0
0  ...
1      0      0      0      0      0      0      0      0      0
0  ...
2      0      0      0      0      0      0      0      0      0
0  ...
3      0      0      0      0      0      0      0      0      0
0  ...
4      0      0      0      0      0      0      0      0      0
0  ...

   32452  32453  32454  32455  32456  32457  32458  32459  32460
32461
0      0      0      0      0      0      0      0      0      0
0
1      0      0      0      0      0      0      0      0      0
0
2      0      0      0      0      0      0      0      0      0
0
3      0      0      0      0      0      0      0      0      0
```

```
0
4       0       0       0       0       0       0       0       0       0
0

[5 rows x 32462 columns]
```

```
# Applying pre-processing to test-set
# test_emails = CountVectorizer(analyzer =
pre_processor).fit_transform(test_set['text'])

test_emails = vectorizer.transform(test_set['text'])

df_emails_test = pd.DataFrame(test_emails.toarray())
print(df_emails_test.shape)
print()
print(test_set.shape)
df_emails_test.head()
```

```
(1468, 32462)

(1468, 2)

   0       1       2       3       4       5       6       7       8       9
...   \
0       0       0       0       0       0       0       0       0       0
0   ...
1       0       0       0       0       0       0       0       0       0
0   ...
2       0       0       0       0       0       0       0       0       0
0   ...
3       0       0       0       0       0       0       0       0       0
0   ...
4       0       0       0       0       0       0       0       0       0
0   ...

   32452   32453   32454   32455   32456   32457   32458   32459   32460
32461
0       0       0       0       0       0       0       0       0       0
0
1       0       0       0       0       0       0       0       0       0
0
2       0       0       0       0       0       0       0       0       0
0
3       0       0       0       0       0       0       0       0       0
0
4       0       0       0       0       0       0       0       0       0
0

[5 rows x 32462 columns]
```

Explanation (Applying pre-processing the entire dataset):

I defined a pre-processing function that cleans the data_set by removing punctuation and stopwords. Then, I applied this function to the 'text' column of the data_set using the apply() function. After that, I used the CountVectorizer class to convert the 'text' column of the data_set into a sparse matrix of token counts, which can be used as input for machine learning models. And, I repeated the same process for the test_set. The main purpose of this process is to prepare the data for machine learning models by cleaning it, and making it easier for the models to process.

4) Train/Validation Split

What we will do now is split the data set into train and test set - The train set can have 80% of the data (i.e. emails along with their labels) chosen at random - But with good representation from both spam and not-spam email classes. And the same goes for the test set - Which would have the remaining 20% of the data.

```
df_emails.head()
```

```
   0      1      2      3      4      5      6      7      8      9
...   \
0      0      0      0      0      0      0      0      0      0
0   ...
1      0      0      0      0      0      0      0      0      0
0   ...
2      0      0      0      0      0      0      0      0      0
0   ...
3      0      0      0      0      0      0      0      0      0
0   ...
4      0      0      0      0      0      0      0      0      0
0   ...

   32452  32453  32454  32455  32456  32457  32458  32459  32460
32461
0      0      0      0      0      0      0      0      0      0
0
1      0      0      0      0      0      0      0      0      0
0
2      0      0      0      0      0      0      0      0      0
0
3      0      0      0      0      0      0      0      0      0
0
4      0      0      0      0      0      0      0      0      0
0

[5 rows x 32462 columns]
```

```
X = df_emails
y = data_set['spam']

X = X.values
y = y.values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.20, random_state = 0)

print(type(X))
print(X.shape)
```

```
<class 'numpy.ndarray'>
(4260, 32462)
```

```
print(type(y))
print(y.shape)
print(y)
```

```
<class 'numpy.ndarray'>
(4260,)
[1 1 1 ... 0 0 0]
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(3408, 32462)
(3408,)
(852, 32462)
(852,)
```

Explanation (Train/Validation Split):

I assigned the 'spam' column of the data_set to the variable 'y' and the dataframe of emails to the variable 'X' and then I converted both 'X' and 'y' into numpy arrays. I used the train_test_split() function to split the data into training and testing sets with 20% of the data for testing, and the remaining 80% for training. The random_state parameter is set to 0 to ensure the data is split randomly. This process is essential to prepare the data for training and testing machine learning models, so that models can be trained and evaluated on separate sets of data, which helps to prevent overfitting.

5) Train your model and evaluate on Kaggle

Report your train/validation F1-score for your baseline model (starter LR model) and also your best LR model. Also report your insights on what worked and what did not on the Kaggle evaluation. How can your model be improved? Where does your model make mistakes?

Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression

def LR_model(X_train, X_test, y_train, y_test):
    # Apply logistic regression on the given dataset, and return the predictions in the val dataset.
    # lr_model is the fitted logistic regression model.
    lr_model = LogisticRegression()
    lr_model.fit(X_train, y_train)
    y_pred = lr_model.predict(X_test)
    return lr_model, y_pred


lr_model, y_pred = LR_model(X_train, X_test, y_train, y_test)

# Checking the predictions
print(lr_model.predict(X_train))

# Checking the actual values
print(y_train)
```

```
[1 0 1 ... 0 0 0]
[1 0 1 ... 0 0 0]
```

```
lr_model.predict(X_train).mean()
```

```
0.23767605633802816
```

```
y_train.mean()
```

```
0.23767605633802816
```

```
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

y_pred = lr_model.predict(X_train)
def metrics(y_train, y_pred):
  y_pred = lr_model.predict(X_train)
  print(classification_report(y_train, y_pred))
  print()
  print('Confusion Matrix: \n', confusion_matrix(y_train, y_pred))
  print()
  print('Accuracy: ', accuracy_score(y_train, y_pred))

  # y_true are the true labels given, and y_pred are the ones
predicted by the model.
  # Show the required metrics for the given predictions.

metrics(y_train, y_pred)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2598
           1       1.00      1.00      1.00       810

    accuracy                           1.00      3408
   macro avg       1.00      1.00      1.00      3408
weighted avg       1.00      1.00      1.00      3408


Confusion Matrix: 
 [[2598    0]
 [   0  810]]

Accuracy:  1.0
```

```
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

y_pred = lr_model.predict(X_test)
def metrics(y_test, y_pred):
  y_pred = lr_model.predict(X_test)
```

```
  print(classification_report(y_test, y_pred))
  print()
  print('Confusion Matrix: \n', confusion_matrix(y_test, y_pred))
  print()
  print('Accuracy: ', accuracy_score(y_test, y_pred))

  # y_true are the true labels given, and y_pred are the ones
predicted by the model.
  # Show the required metrics for the given predictions.

metrics(y_test, y_pred)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       636
           1       0.99      0.99      0.99       216

    accuracy                           1.00       852
   macro avg       0.99      0.99      0.99       852
weighted avg       1.00      1.00      1.00       852


Confusion Matrix:
 [[634   2]
 [  2 214]]

Accuracy:  0.9953051643192489
```

```
test_set.shape
```

```
(1468, 3)
```

```
df_emails_test.shape
```

```
(1468, 32462)
```

```
# Prediction on external test set
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

X_test2 = df_emails_test.values
y_pred2 = lr_model.predict(X_test2)

print(y_pred2.shape)
print(y_pred2)
```

```
(1468,)
[0 0 1 ... 0 0 0]
```

```python
# Adding the predictions to datasets for submission format
test_set['spam'] = y_pred2
test_set[['id','spam']].to_csv('submission_lr.csv', index=False)
```

Explanation (Logistic Regression Model):

I created a Logistic Regression model function that applies logistic regression on the data set and returns the fitted logistic regression model and the predictions in the validation dataset. I then evaluated the model's performance on the train and test set by calculating the mean of the predictions and the performance metrics, such as classification report, confusion matrix, and accuracy score. Finally, I applied the model on the external test set and calculated its performance metrics. The model's performance on the train set is very good with an accuracy of 1.0, however, the performance on the test set needs to be mentioned. This model gave me the best accuracy score in the Kaggle Competition: https://www.kaggle.com/competitions/spam-classification-winter-2023/leaderboard?

XGBoost Model (Extreme Gradient Boosting)

```
import xgboost as xgb
from xgboost import XGBClassifier
def xgb_model(X_train, X_test, y_train, y_test):

    xgb_model = XGBClassifier(objective='binary:logistic',
n_estimators=500, seed=42)
    xgb_model.fit(X_train, y_train)
    y_pred = xgb_model.predict(X_test)
    return xgb_model, y_pred


xgb_model, y_pred = xgb_model(X_train, X_test, y_train, y_test)

# Checking the predictions
print(xgb_model.predict(X_train))

# Checking the actual values
print(y_train)

[1 0 1 ... 0 0 0]
[1 0 1 ... 0 0 0]

xgb_model.predict(X_train).mean()

0.23943661971830985

y_train.mean()

0.23767605633802816

from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

y_pred = xgb_model.predict(X_train)
def metrics(y_train, y_pred):
  y_pred = xgb_model.predict(X_train)
  print(classification_report(y_train, y_pred))
  print()
  print('Confusion Matrix: \n', confusion_matrix(y_train, y_pred))
  print()
  print('Accuracy: ', accuracy_score(y_train, y_pred))

  # y_true are the true labels given, and y_pred are the ones
predicted by the model.
  # Show the required metrics for the given predictions.
```

```
metrics(y_train, y_pred)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2598
           1       0.99      1.00      1.00       810

    accuracy                           1.00      3408
   macro avg       1.00      1.00      1.00      3408
weighted avg       1.00      1.00      1.00      3408


Confusion Matrix:
 [[2592    6]
 [   0  810]]

Accuracy:  0.9982394366197183
```

```python
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

y_pred = xgb_model.predict(X_test)
def metrics(y_test, y_pred):
  y_pred = xgb_model.predict(X_test)
  print(classification_report(y_test, y_pred))
  print()
  print('Confusion Matrix: \n', confusion_matrix(y_test, y_pred))
  print()
  print('Accuracy: ', accuracy_score(y_test, y_pred))

  # y_true are the true labels given, and y_pred are the ones
predicted by the model.
  # Show the required metrics for the given predictions.

metrics(y_test, y_pred)
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       636
           1       0.95      0.99      0.97       216

    accuracy                           0.98       852
   macro avg       0.97      0.98      0.98       852
weighted avg       0.98      0.98      0.98       852


Confusion Matrix:
 [[625  11]
 [  3 213]]
```

```
Accuracy:  0.9835680751173709
```

```
test_set.shape
```

```
(1468, 3)
```

```
df_emails_test.shape
```

```
(1468, 32462)
```

```
# Prediction on external test set
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

X_test2 = df_emails_test.values
y_pred2 = xgb_model.predict(X_test2)

print(y_pred2.shape)
print(y_pred2)
```

```
(1468,)
[0 0 1 ... 0 0 0]
```

```
# Adding the predictions to datasets for submission format
test_set['spam'] = y_pred2
test_set[['id','spam']].to_csv('submission_xgboost.csv', index=False)
```

Explanation (XGBoost Model):

I created an Extreme Gradient Boosting (XGBoost) model function that applies the XGBoost model on the data set and returns the fitted XGBoost model and the predictions in the validation dataset. I then evaluated the model's performance on the train and test set by calculating the mean of the predictions and the performance metrics, such as classification report, confusion matrix, and accuracy score. Finally, I applied the model on the external test set and calculated its performance metrics. The model's performance on the train set is good with an accuracy of 0.998 and the performance on the test set is 0.984.

Best Model (Actually Logistic Regression is the best Model) based on the Kaggle Score.

```
from sklearn.ensemble import RandomForestClassifier


def best_model(X_train, X_test, y_train, y_test):
  # Apply any machine learning algorithm on the given dataset, and
return the predictions in the val dataset.
  # bt_model is the training data fitted model.
  bt_model = RandomForestClassifier(n_estimators = 800, random_state =
42)
  bt_model.fit(X_train, y_train)
  y_pred = bt_model.predict(X_test)
  return bt_model, y_pred


bt_model, y_pred = best_model(X_train, X_test, y_train, y_test)
```

```
# Checking the predictions
print(bt_model.predict(X_train))
# Checking the actual values
print(y_train)
```

```
[1 0 1 ... 0 0 0]
[1 0 1 ... 0 0 0]
```

```
bt_model.predict(X_train).mean()
```

```
0.23767605633802816
```

```
y_train.mean()
```

```
0.23767605633802816
```

```python
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

y_pred = bt_model.predict(X_train)
def metrics(y_train, y_pred):
  y_pred = bt_model.predict(X_train)
  print(classification_report(y_train, y_pred))
  print()
  print('Confusion Matrix: \n', confusion_matrix(y_train, y_pred))
  print()
  print('Accuracy: ', accuracy_score(y_train, y_pred))

  # y_true are the true labels given, and y_pred are the ones
predicted by the model.
  # Show the required metrics for the given predictions.

metrics(y_train, y_pred)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2598
           1       1.00      1.00      1.00       810

    accuracy                           1.00      3408
   macro avg       1.00      1.00      1.00      3408
weighted avg       1.00      1.00      1.00      3408


Confusion Matrix: 
 [[2598    0]
 [   0  810]]

Accuracy:  1.0
```

```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```python
y_pred = bt_model.predict(X_test)
def metrics(y_test, y_pred):
  y_pred = bt_model.predict(X_test)
  print(classification_report(y_test, y_pred))
  print()
  print('Confusion Matrix: \n', confusion_matrix(y_test, y_pred))
  print()
  print('Accuracy: ', accuracy_score(y_test, y_pred))

  # y_true are the true labels given, and y_pred are the ones predicted by the model.
  # Show the required metrics for the given predictions.

metrics(y_test, y_pred)
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       636
           1       1.00      0.94      0.97       216

    accuracy                           0.98       852
   macro avg       0.99      0.97      0.98       852
weighted avg       0.98      0.98      0.98       852


Confusion Matrix:
 [[636   0]
 [ 14 202]]

Accuracy:  0.9835680751173709
```

```python
test_set.shape
```

```
(1468, 3)
```

```python
df_emails_test.shape
```

```
(1468, 32462)
```

```python
# Prediction on external test set
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

X_test2 = df_emails_test.values
y_pred2 = bt_model.predict(X_test2)

print(y_pred2.shape)
print(y_pred2)
```

```
(1468,)
[0 0 1 ... 0 0 0]
```

```
test_set['spam'] = y_pred2
test_set[['id','spam']].to_csv('submission_best_model.csv',
index=False)
```

Explanation (Random Forest Classifier Model):

I implemented the Random Forest Classifier and set the number of estimators to 800 and the random state to 42. I did not set a maximum depth, allowing the model to find the optimal depth on its own. I then fit the model on the training data and made predictions on the test data. I evaluated the model by checking the predictions against the actual values and using classification report, confusion matrix and accuracy score. The results showed that the model had a precision, recall, f1-score and accuracy of 0.98, which is similar to Logistic regression.

Conclusion:

I implemented three machine learning models to classify emails as spam or non-spam. These models were Logistic Regression, XGBoost, and Random Forest. Logistic Regression was the best model as it achieved the highest Kaggle public score of 0.98807 and a private score of 0.99319, placing me in third place. However, it should be noted that the first and second-place competitors achieved the same score of 0.99319. Despite trying Random Forest, its Kaggle score was lower than Logistic Regression, with a score of 0.97359.

Kaggle Competition: https://www.kaggle.com/competitions/spam-classification-winter-2023/leaderboard?