Portfolio Assignment: Text Classification

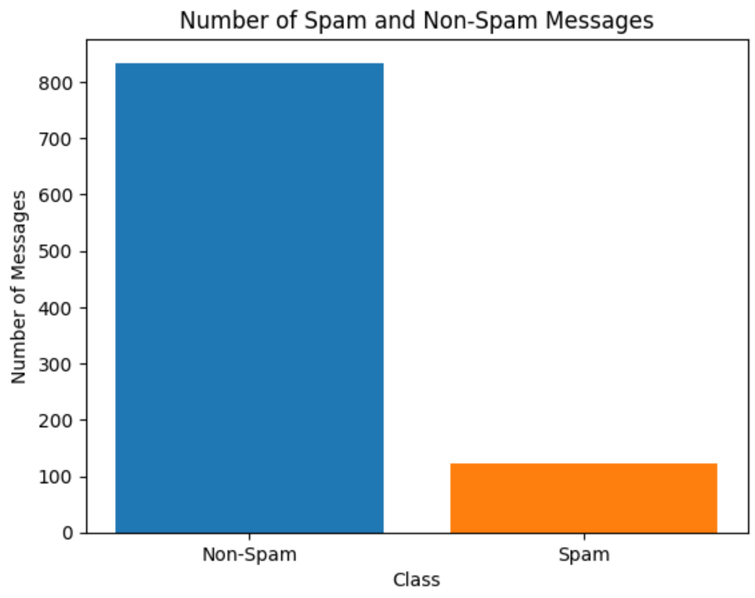
by: Ahmed Iqbal(axi180011)

The dataset I selected is on SMS text messages and whether they are spam or not. The model should be able to predict whether a message is spam or not.

Naive Bayes Approach

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
import matplotlib.pyplot as plt

import nltk
#nltk.download('stopwords')


df = pd.read_csv("https://raw.githubusercontent.com/ahmxdiqbal/CS4395/main/SMS_train.csv", header=1, usecols=[1,2], encoding = 'latin1')

stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=list(stopwords))

# set up X and y
X = df.iloc[:,0]
y = df.iloc[:,1]


for i in range(len(y)):
  if y[i] == "Non-Spam":
    y[i] = 0
  else:
    y[i] = 1


y = y.astype(int)


# Count the number of occurrences of each class
counts = y.value_counts()

# Plot the counts
fig, ax = plt.subplots()
ax.bar(["Non-Spam", "Spam"], counts, color=["#1f77b4", "#ff7f0e"])
ax.set_xlabel("Class")
ax.set_ylabel("Number of Messages")
ax.set_title("Number of Spam and Non-Spam Messages")
plt.show()

print("Number of Spam messages: ", counts[1])
print("Number of Non-Spam messages: ", counts[0])

print()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

X_train.shape

# apply tfidf vectorizer
X_train = vectorizer.fit_transform(X_train)  # fit and transform the train data
X_test = vectorizer.transform(X_test)        # transform only the test data

from sklearn.naive_bayes import MultinomialNB

naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

# priors
import math
prior_p = sum(y_train == 1)/len(y_train)
print('prior spam:', prior_p, '\nlog of prior:', math.log(prior_p))
```

```
# the model prior matches the prior calculated above
naive_bayes.class_log_prior_[1]

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# make predictions on the test data
pred = naive_bayes.predict(X_test)

# print confusion matrix
#print(confusion_matrix(y_test, pred))

print('\naccuracy score: ', accuracy_score(y_test, pred))

print('\nprecision score (not spam): ', precision_score(y_test, pred, pos_label=0))
print('precision score (spam): ', precision_score(y_test, pred))

print('\nrecall score: (not spam)', recall_score(y_test, pred, pos_label=0))
print('recall score: (spam)', recall_score(y_test, pred))

print('\nf1 score: ', f1_score(y_test, pred))
```



```
Number of Spam messages:  122
Number of Non-Spam messages:  834

prior spam: 0.12827225130890052
log of prior: -2.0536003104959484

accuracy score:  0.921875

precision score (not spam):  0.9180327868852459
precision score (spam):  1.0

recall score: (not spam) 1.0
recall score: (spam) 0.375

f1 score:  0.5454545454545454
```

Logistic Regression Approach

```
# all the imports for the next code block

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

# vectorizer
vectorizer = TfidfVectorizer(binary=True)
X_train = vectorizer.fit_transform(X_train)  # fit and transform the train data
X_test = vectorizer.transform(X_test)        # transform only the test data
```

```
#train
classifier = LogisticRegression(solver='lbfgs', class_weight='balanced')
classifier.fit(X_train, y_train)

# evaluate
pred = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
probs = classifier.predict_proba(X_test)
print('log loss: ', log_loss(y_test, probs))
```

```
accuracy score:  0.9791666666666666
precision score:  1.0
recall score:  0.8333333333333334
f1 score:  0.9090909090909091
log loss:  0.25870177450781123
```

Neural Networks approach

```
vectorizernn = TfidfVectorizer(stop_words=list(stopwords), binary=True)

# set up X and y
X = vectorizernn.fit_transform(df.iloc[:,0])
y = df.iloc[:,1]

y = y.astype(int)  # convert y to integer type


# divide into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=(15, 2), random_state=1)
classifier.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score
pred = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.953125
precision score:  1.0
recall score:  0.625
f1 score:  0.7692307692307693
```

Analysis

Between the three approaches, the logistic regression had the highest accuracy. With regards to the naive bayes classifier, this may be since the logistic regression approach has fewer prior assumptions about how the data is distributed, and so it gets a higher score. Also, since the target has only two answer choices, logistic regression is well suited for this kind of data. With regards to the neural network approach, the relatively small size of the dataset might have been an issue. Neural networks need larger datasets to be effective. Another issue is the neural network might be overfitting.

From the f1 score we can see the pattern repeating. Logistic regression is the most accurate, followwed by the neural network model, and finally by the naive bayes approach. From this, we can satte pretty confidently that igvent he dataset, the logistic regression model models this data best.

One of the things that pops out to me is that the precision score is 1 for all three approaches. This strikes me as odd, but maybe this is because there is just a lot of spam messages in the dataset. Another explantion could be that it is overfitting the data.

✓ 0s completed at 9:10 PM