Assignment 9

The following will be going through running Naive Bayes, Logistic Regression, and Neural Networks on a dataset over gender by name (https://archive.ics.uci.edu/ml/datasets/Gender+by+Name) using sklearn. This dataset was altered by trimming it down to 5000 entries, removing the probability and count column, as well as changing the M/F column to be 0/1 respectively.

All of these algorithms are trying to identify male names. The data set is comprised of 41% male names total. Therefore, if the algorithms identify more than 41%, that means they are learning.

General Preprocessing

First things first though, we must process the data.

In [4]:
```
import pandas as pd
df = pd.read_csv('name_gender_dataset_edited.csv', header=0, usecols=[0,1], er
print('rows and columns:', df.shape)
print(df.head())
```

```
rows and columns: (4999, 2)
      Name  Gender
0    James       0
1     John       0
2   Robert       0
3  Michael       0
4  William       0
```

In [7]:
```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

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

# set up X and y
X = df.Name
y = df.Gender

# take a peek at X
X.head()
```

Out[7]:
```
0      James
1       John
2     Robert
3    Michael
4    William
Name: Name, dtype: object
```

In [8]:
```
# look at y
y[:10]
```

Out[8]:
```
0    0
1    0
2    0
3    0
4    0
5    1
6    0
7    0
8    0
9    0
Name: Gender, dtype: int64
```

Naive Bayes

The following will go over modelling Naive Bayes over the dataset.

In [9]:
```
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

X_train.shape
```

Out[9]:
```
(3999,)
```

In [10]:
```
# apply tfidf vectorizer
X_train = vectorizer.fit_transform(X_train)  # fit and transform the train dat
X_test = vectorizer.transform(X_test)        # transform only the test data

# take a peek at the data
# this is a very sparse matrix because most of the 8613 words don't occur in e

print('train size:', X_train.shape)
print(X_train.toarray()[:5])

print('\ntest size:', X_test.shape)
print(X_test.toarray()[:5])
```

```
train size: (3999, 3787)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

test size: (1000, 3787)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

In [11]:
```
from sklearn.naive_bayes import MultinomialNB

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

Out[11]:
```
MultinomialNB()
```

In [12]:
```
# priors
import math
prior_p = sum(y_train == 1)/len(y_train)
print('prior spam:', prior_p, 'log of prior:', math.log(prior_p))

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

```
prior spam: 0.5926481620405101 log of prior: -0.5231543747176416
```

Out[12]:
```
-0.5231543747176417
```

In [13]:
```
naive_bayes.feature_log_prob_
```

Out[13]:
```
array([[-7.90359629, -8.59674347, -7.90359629, ..., -8.59674347,
        -8.59674347, -8.59674347],
       [-8.72518249, -8.03203531, -8.72518249, ..., -8.03203531,
        -8.03203531, -8.03203531]])
```

In [14]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_

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

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

```
[[  0 413]
 [ 52 535]]
```

In [15]:
```
print('accuracy score: ', accuracy_score(y_test, pred))

print('\nprecision score (not spam): ', precision_score(y_test, pred, pos_labe
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))
```

```
accuracy score:  0.535

precision score (not spam):  0.0
precision score (spam):  0.5643459915611815

recall score: (not spam) 0.0
recall score: (spam) 0.9114139693356048

f1 score:  0.6970684039087949
```

In [16]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       413
           1       0.56      0.91      0.70       587

    accuracy                           0.54      1000
   macro avg       0.28      0.46      0.35      1000
weighted avg       0.33      0.54      0.41      1000
```

The accuracy of the f1-score here is shown to be 54%, and because this is greater than the 41% of the original distribution, we can see that the Naive Bayes model actually learned and improved on its own, being able to guess 13% more accurately from what it learned.

Logistic Regression

The following will go over modelling Logistic Regression over the dataset.

In [33]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_

#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:  0.535
precision score:  0.5643459915611815
recall score:  0.9114139693356048
f1 score:  0.6970684039087949
log loss:  0.7096596852017873
```

The accuracy is shown to be about 53% here, which is greater than the 41% of the original dataset. Because there is a 12% improvement, we can surmise that the logistic regression model learned and improved and was able to determine more about the dataset on its own.

Neural Network

The following will go over modelling Neural Networking over the dataset.

In [28]:
```
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)
```

Out[28]:
```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(15, 2), random_state=1,
              solver='lbfgs')
```

In [29]:
```
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:  0.587
precision score:  0.587
recall score:  1.0
f1 score:  0.7397605545053559
```

Finally, we have the neural network algorithm, and here we can see an even greater improvement from the past two models. This time, we have about 59%, which is a 18% improvement from the original datasets 41%. This shows that neural networking is perhaps the best way to learn and improve the accuracy for this particular dataset.

Conclusion

After looking at all the results, I think its clear to see that the neural network algorithm worked the best here, though it sinteresting to see how the different algorithms worked and how they preformed differently. They all have their own unique uses and overall purposes, and none of them fully succeeded at the given task, however, it is clear that the neural network did the best at improving in overall accuracy, having a 18% improvement in comparision to a 13% or 12% improvement of Naive Bayes and Logistic Regression respectively.