Machine_Learning_Programming_Assignment#2

November 19, 2021

In this homework we are using Gaussian Naive Bayes to classify the Spambase data.

Step1: Import the necessary libraries

[4]:
```
import random
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
```

Step2: Load the training data

[5]:
```
training_data = pd.read_csv("spambase.data", header=None, dtype=float);
training_data
```

[5]:
```
         0     1     2    3     4     5     6     7     8     9  …     48  \
0     0.00  0.64  0.64  0.0  0.32  0.00  0.00  0.00  0.00  0.00  …  0.000
1     0.21  0.28  0.50  0.0  0.14  0.28  0.21  0.07  0.00  0.94  …  0.000
2     0.06  0.00  0.71  0.0  1.23  0.19  0.19  0.12  0.64  0.25  …  0.010
3     0.00  0.00  0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  …  0.000
4     0.00  0.00  0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  …  0.000
…        …     …     …   …     …     …     …     …     …     …  …      …
4596  0.31  0.00  0.62  0.0  0.00  0.31  0.00  0.00  0.00  0.00  …  0.000
4597  0.00  0.00  0.00  0.0  0.00  0.00  0.00  0.00  0.00  0.00  …  0.000
4598  0.30  0.00  0.30  0.0  0.00  0.00  0.00  0.00  0.00  0.00  …  0.102
4599  0.96  0.00  0.00  0.0  0.32  0.00  0.00  0.00  0.00  0.00  …  0.000
4600  0.00  0.00  0.65  0.0  0.00  0.00  0.00  0.00  0.00  0.00  …  0.000

         49   50     51     52     53     54     55      56   57
0     0.000  0.0  0.778  0.000  0.000  3.756   61.0   278.0  1.0
1     0.132  0.0  0.372  0.180  0.048  5.114  101.0  1028.0  1.0
2     0.143  0.0  0.276  0.184  0.010  9.821  485.0  2259.0  1.0
3     0.137  0.0  0.137  0.000  0.000  3.537   40.0   191.0  1.0
4     0.135  0.0  0.135  0.000  0.000  3.537   40.0   191.0  1.0
…         …    …      …      …      …      …      …       …    …
4596  0.232  0.0  0.000  0.000  0.000  1.142    3.0    88.0  0.0
4597  0.000  0.0  0.353  0.000  0.000  1.555    4.0    14.0  0.0
4598  0.718  0.0  0.000  0.000  0.000  1.404    6.0   118.0  0.0
4599  0.057  0.0  0.000  0.000  0.000  1.147    5.0    78.0  0.0
4600  0.000  0.0  0.125  0.000  0.000  1.250    5.0    40.0  0.0
```

```
[4601 rows x 58 columns]
```

[6]:
```
np_data = training_data.to_numpy();
```

[7]:
```
spam = np_data[:1813,:]
notspam = np_data[1813:,:]
```

[8]:
```
it = np.arange(spam.shape[0])
np.random.shuffle(it)
```

[9]:
```
it1 = np.arange(notspam.shape[0])
np.random.shuffle(it1)
```

Step3: Initialise the values

[10]:
```
countsp = 0.0
countnsp =0.0
```

[11]:
```
true_positive = 0.0
true_negative = 0.0
false_positive = 0.0
false_negative = 0.0
```

[12]:
```
classifcation = 0
```

Step4: Split the data such that it has 40% spam and 60% notspam

[13]:
```
#Splitting training data
train_data_spam = spam[:906,:]
train_data_notspam = notspam[:1394,:]
```

[14]:
```
#Splitting test data
test_data_spam = spam[906:,:]
test_data_notspam = notspam[1394:,:]
```

Final train and test data

[15]:
```
final_train_data = np.concatenate((train_data_spam,train_data_notspam),axis=0)
final_train_target = final_train_data[:,57]
```

[16]:
```
final_test_data = np.concatenate((test_data_spam,test_data_notspam),axis=0)
final_test_target = final_test_data[:,57]
```

Step5: A function to implement Naive Bayes Classifier

[17]:
```
#function to implement Naive Bayes classifcation to classify the test dataset

def formula(mean,std,a):
        np.seterr(divide='ignore')
        part1 = float(1 / (std * (np.sqrt(2 * np.pi))))
        part2 = float(np.exp(-1 * (np.square(a - mean))/(2 * np.
 ↪square(float(std * std)))))
        res = part1 * part2
        return res
```

[18]:

```
for i in range(0,final_train_data.shape[0]):
    if(final_train_data[i,57] == 1):
        countsp += 1
    else:
        countnsp += 1
```

Calculate and print prior_spam and prior_notspam

[19]:

```
prior_spam = countsp / len(final_train_data);
print(prior_spam)
prior_notspam = countnsp / len(final_train_data);
print(prior_notspam)
```

```
0.3939130434782609
0.6060869565217392
```

[20]:

```
sp_mean = []
sp_sd = []
nsp_mean = []
nsp_sd = []
```

[21]:

```
for i in range(0,final_train_data.shape[1]):
    spam_array = []
    notspam_array = []

    for j in range(final_train_data.shape[0]):
        if (final_train_data[j][-1] == 1):
            spam_array.append(final_train_data[j][i])
        else:
            notspam_array.append(final_train_data[j][i])

    sp_mean.append(np.mean(spam_array))
    sp_sd.append(np.std(spam_array))
    nsp_mean.append(np.mean(notspam_array))
    nsp_sd.append(np.std(notspam_array))
```

[22]:

```
for k in range(len(sp_sd)):
    if (sp_sd[k] == 0):
        sp_sd[k] = 0.0001

    if (nsp_sd[k] == 0):
        nsp_sd[k] = 0.0001
```

[23]:
```
#classification result after Gaussian Naïve Bayes calculation

classification_result = []
```

[24]:
```
# classify the test datatset using Gaussian Naïve Bayes formula

for i in range(final_test_data.shape[0]):
    temp1 = np.log(prior_spam)
    temp2 = np.log(prior_notspam)

    for j in range(0,57):
        a = final_test_data[i][j]
        temp1 += np.log(formula(sp_mean[j], sp_sd[j], a))
        temp2 += np.log(formula(nsp_mean[j], nsp_sd[j], a))

    classification = np.argmax([temp2, temp1])
    classification_result.append(classification)
```

Step6: Confusion Matrix

[25]:
```
#confusion matrix using target values of test dataset and the classification␣
 ↪result

confusion_matrix = confusion_matrix(final_test_target, classification_result)
print("\nConfusion matrix\n",confusion_matrix)

for i in range(len(classification_result)):
    if (classification_result[i] == 1 and final_test_target[i] == 1):
        true_positive += 1

    elif (classification_result[i] == 0 and final_test_target[i] == 0):
        true_negative += 1

    elif (classification_result[i] == 1 and final_test_target[i] == 0):
        false_positive += 1

    else:
        false_negative += 1
```

```
Confusion matrix
 [[1339   55]
 [ 317  590]]
```

Step7: Calculate and print accuracy,precission and recall

[26]:
```
# calculating accuracy, precision and recall
```

```
accuracy = float(true_positive + true_negative) / (true_positive + true_negative + false_negative + false_positive)

precision = float(true_positive) / (true_positive + false_positive)

recall = float(true_positive) / (true_positive + false_negative)


print("\nAccuracy: ",accuracy)
print("Precision: ",precision)
print("Recall:",recall)
```

```
Accuracy:  0.8383311603650587
Precision:  0.9147286821705426
Recall: 0.6504961411245865
```