pratical-6

April 17, 2024

1 Data Analytics III

1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

[ ]:

```
# # Import the required libraries
# import pandas as pd
# import numpy as np
# import matplotlib.pyplot as plt
# import seaborn as sns
# from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import LabelEncoder
# from sklearn.naive_bayes import GaussianNB
# from sklearn.metrics import confusion_matrix, classification_report

# # Load iris.csv file into a Pandas Data_Frame
# df = pd.read_csv('Iris.csv')
# df.head(10)

# X = df.iloc[:,1:5]
# y = df.iloc[:,-1]

# X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.
 ↪8,random_state=1)
# X_test

# la_object = LabelEncoder()
# y = la_object.fit_transform(y)

# model = GaussianNB()
# model.fit(X_train, y_train)

# y_predicted = model.predict(X_test)
# y_predicted
# model.score(X_test, y_test)

# cm = confusion_matrix(y_test, y_predicted)
# cm

# # classification report for precision, recall, f1-score and accuracy
# c1_report = classification_report(y_test, y_predicted)
# c1_report

# #                     precision    recall  f1-score   support\n\n
# # Iris-setosa          1.00      1.00      1.00        11\n
# # Iris-versicolor      1.00      0.92      0.96        13\n
# # Iris-virginica       0.86      1.00      0.92         6\n
# # accuracy                                 0.97        30\n
# # macro avg            0.95      0.97      0.96        30\n
# # weighted avg         0.97      0.97      0.97        30\n

# # Creating a dataframe for a array-formatted confusion matrix, so it will be easy for plotting
# cm_df = pd.DataFrame(cm,
#                      index= ['SETOSA', 'VERSICOLOR', 'VIRGINICA'],
#                      columns= ['SETOSA', 'VERSICOLOR', 'VIRGINICA'])

# # Plotting the confusion matrix
# plt.figure(figsize=(5,4))
# sns.heatmap(cm_df, annot=True)
# plt.title("Confusion Matrix")
# plt.xlabel("ACtual Values")
# plt.ylabel("Predicted Values")
# plt.show()

# def accuracy_cm(tp, fn, fp, tn):
#     return (tp+tn)/(tp+fp+tn+fn)
# def precision_cm(tp, fn, fp, tn):
#     return tp/(tp+fp)
# def recall_cm(tp, fn, fp, tn):
#     return tp/(tp+fn)
# def f1_score(tp, fn, fp, tn):
#     return 2/((1/recall_cm(tp, fn, fp, tn)+(1/precision_cm(tp, fn, fp, tn))))
# def error_rate_cm(tp, fn, fp, tn):
#               return 1-accuracy_cm(tp, fn, fp, tn)

# # For Virginica
# tp = cm[2][2]
# fn = cm[2][0]+cm[2][1]
# fp = cm[0][2]+cm[1][2]
# tn = cm[0][0]+cm[0][1]+cm[1][1]
# print("For Virginica \n")
# print(f"Accuracy   : {accuracy_cm(tp, fn, fp, tn)}")
# print(f"Precision  : {precision_cm(tp, fn, fp, tn)}")
# print(f"Recall     : {recall_cm(tp, fn, fp, tn)}")
# print(f"F1-Score   : {f1_score(tp, fn, fp, tn)}")
# print(f"Error rate   : {error_rate_cm(tp, fn, fp, tn)}")

# # For Versicolor
# tp = cm[1][1]
# fn = cm[1][0] + cm[1][2]
# fp = cm[0][1] + cm[2][1]
# tn = cm[0][0] + cm[0][2] + cm[2][0]
# print("For Versicolor \n")
# print(f"Accuracy   : {accuracy_cm(tp, fn, fp, tn)}")
# print(f"Precision  : {precision_cm(tp, fn, fp, tn)}")
# print(f"Recall     : {recall_cm(tp, fn, fp, tn)}")
# print(f"F1-Score   : {f1_score(tp, fn, fp, tn)}")
# print(f"Error rate   : {error_rate_cm(tp, fn, fp, tn)}")

# # For Setosa
# tp = cm[0][0]
# fn = cm[0][1] + cm[0][2]
# fp = cm[1][0] + cm[2][0]
# tn = cm[1][1] + cm[1][2] + cm[2][1]
# print("For Setosa \n")
# print(f"Accuracy   : {accuracy_cm(tp, fn, fp, tn)}")
# print(f"Precision  : {precision_cm(tp, fn, fp, tn)}")
# print(f"Recall     : {recall_cm(tp, fn, fp, tn)}")
# print(f"F1-Score   : {f1_score(tp, fn, fp, tn)}")
# print(f"Error rate   : {error_rate_cm(tp, fn, fp, tn)}")
```

[1]:

```
# Import the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report
```

[2]:

```
# Load iris.csv file into a Pandas Data_Frame
df = pd.read_csv('Iris.csv')
df.head(10)
```

```
[2]:    Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
     0   1            5.1           3.5            1.4           0.2  Iris-setosa
     1   2            4.9           3.0            1.4           0.2  Iris-setosa
     2   3            4.7           3.2            1.3           0.2  Iris-setosa
     3   4            4.6           3.1            1.5           0.2  Iris-setosa
     4   5            5.0           3.6            1.4           0.2  Iris-setosa
     5   6            5.4           3.9            1.7           0.4  Iris-setosa
     6   7            4.6           3.4            1.4           0.3  Iris-setosa
     7   8            5.0           3.4            1.5           0.2  Iris-setosa
     8   9            4.4           2.9            1.4           0.2  Iris-setosa
     9  10            4.9           3.1            1.5           0.1  Iris-setosa
```

```
[3]: X = df.iloc[:,1:5]
     y = df.iloc[:,-1]
```

```
[4]: X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.
       ↪8,random_state=1)
     X_test
```

```
[4]:      SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
     14             5.8           4.0            1.2           0.2
     98             5.1           2.5            3.0           1.1
     75             6.6           3.0            4.4           1.4
     16             5.4           3.9            1.3           0.4
     131            7.9           3.8            6.4           2.0
     56             6.3           3.3            4.7           1.6
     141            6.9           3.1            5.1           2.3
     44             5.1           3.8            1.9           0.4
     29             4.7           3.2            1.6           0.2
     120            6.9           3.2            5.7           2.3
     94             5.6           2.7            4.2           1.3
     5              5.4           3.9            1.7           0.4
     102            7.1           3.0            5.9           2.1
     51             6.4           3.2            4.5           1.5
     78             6.0           2.9            4.5           1.5
     42             4.4           3.2            1.3           0.2
     92             5.8           2.6            4.0           1.2
     66             5.6           3.0            4.5           1.5
     31             5.4           3.4            1.5           0.4
     35             5.0           3.2            1.2           0.2
     90             5.5           2.6            4.4           1.2
     84             5.4           3.0            4.5           1.5
     77             6.7           3.0            5.0           1.7
     40             5.0           3.5            1.3           0.3
     125            7.2           3.2            6.0           1.8
     99             5.7           2.8            4.1           1.3
     33             5.5           4.2            1.4           0.2
```

```
19             5.1           3.8            1.5           0.3
73             6.1           2.8            4.7           1.2
146            6.3           2.5            5.0           1.9
```

[5]:

```python
la_object = LabelEncoder()
y = la_object.fit_transform(y)
```

[6]:

```python
model = GaussianNB()
model.fit(X_train, y_train)
```

[6]:

```
GaussianNB()
```

[7]:

```python
y_predicted = model.predict(X_test)
y_predicted
```

[7]:

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-virginica', 'Iris-versicolor',
       'Iris-setosa', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-virginica'], dtype='<U15')
```

[8]:

```python
model.score(X_test, y_test)
```

[8]:

```
0.9666666666666667
```

[9]:

```python
cm = confusion_matrix(y_test, y_predicted)
cm
```

[9]:

```
array([[11,  0,  0],
       [ 0, 12,  1],
       [ 0,  0,  6]], dtype=int64)
```

[10]:

```python
# classification report for precision, recall, f1-score and accuracy
c1_report = classification_report(y_test, y_predicted)
c1_report
```

[10]:

```
'                 precision    recall  f1-score   support\n\n    Iris-setosa
1.00      1.00      1.00        11\nIris-versicolor       1.00      0.92
0.96        13\n Iris-virginica       0.86      1.00      0.92         6\n\n
accuracy                           0.97        30\n      macro avg       0.95
0.97      0.96        30\n   weighted avg       0.97      0.97      0.97
30\n'
```

[11]:

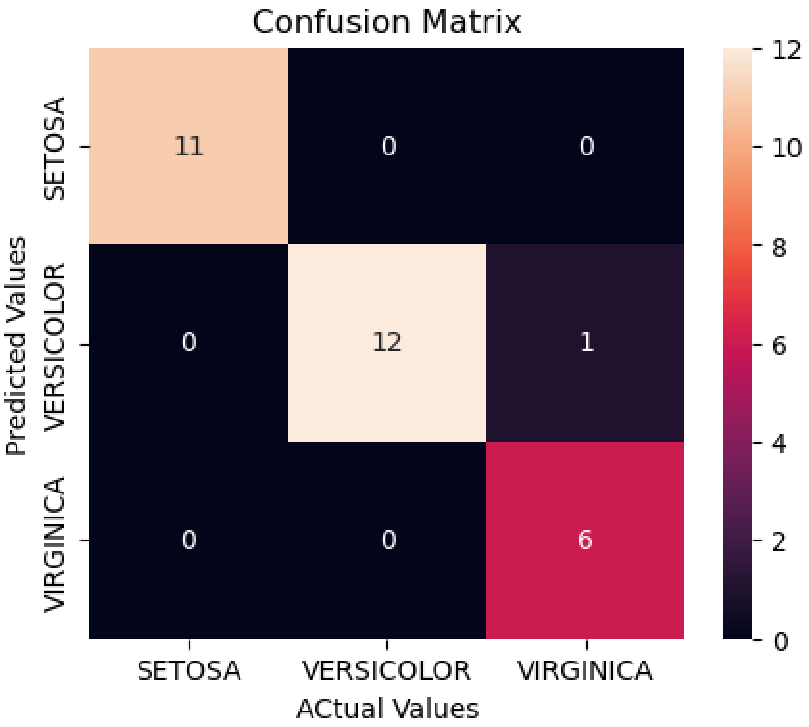
```
#                       precision    recall  f1-score   support\n\n
# Iris-setosa             1.00      1.00      1.00        11\n
# Iris-versicolor         1.00      0.92      0.96        13\n
# Iris-virginica          0.86      1.00      0.92         6\n
# accuracy                                    0.97        30\n
# macro avg               0.95      0.97      0.96        30\n
# weighted avg            0.97      0.97      0.97        30\n
```

[12]:

```
# Creating a dataframe for a array-formatted confusion matrix, so it will be␣
 ↪easy for plotting
cm_df = pd.DataFrame(cm,
                    index= ['SETOSA', 'VERSICOLOR', 'VIRGINICA'],
                    columns= ['SETOSA', 'VERSICOLOR', 'VIRGINICA'])
```

[13]:

```
# Plotting the confusion matrix
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title("Confusion Matrix")
plt.xlabel("ACtual Values")
plt.ylabel("Predicted Values")
plt.show()
```



[14]:

```
def accuracy_cm(tp, fn, fp, tn):
    return (tp+tn)/(tp+fp+tn+fn)

def precision_cm(tp, fn, fp, tn):
    return tp/(tp+fp)

def recall_cm(tp, fn, fp, tn):
    return tp/(tp+fn)

def f1_score(tp, fn, fp, tn):
    return 2/((1/recall_cm(tp, fn, fp, tn)+(1/precision_cm(tp, fn, fp, tn))))

def error_rate_cm(tp, fn, fp, tn):
              return 1-accuracy_cm(tp, fn, fp, tn)
```

[15]:

```
# For Virginica
tp = cm[2][2]
fn = cm[2][0]+cm[2][1]
fp = cm[0][2]+cm[1][2]
tn = cm[0][0]+cm[0][1]+cm[1][1]

print("For Virginica \n")
print(f"Accuracy   : {accuracy_cm(tp, fn, fp, tn)}")
print(f"Precision  : {precision_cm(tp, fn, fp, tn)}")
print(f"Recall     : {recall_cm(tp, fn, fp, tn)}")
print(f"F1-Score   : {f1_score(tp, fn, fp, tn)}")
print(f"Error rate   : {error_rate_cm(tp, fn, fp, tn)}")
```

```
For Virginica

Accuracy   : 0.9666666666666667
Precision  : 0.8571428571428571
Recall     : 1.0
F1-Score   : 0.923076923076923
Error rate   : 0.033333333333333326
```

[16]:

```
# For Versicolor
tp = cm[1][1]
fn = cm[1][0] + cm[1][2]
fp = cm[0][1] + cm[2][1]
tn = cm[0][0] + cm[0][2] + cm[2][0]

print("For Versicolor \n")
print(f"Accuracy   : {accuracy_cm(tp, fn, fp, tn)}")
print(f"Precision  : {precision_cm(tp, fn, fp, tn)}")
print(f"Recall     : {recall_cm(tp, fn, fp, tn)}")
print(f"F1-Score   : {f1_score(tp, fn, fp, tn)}")
print(f"Error rate   : {error_rate_cm(tp, fn, fp, tn)}")
```

```
For Versicolor

Accuracy   : 0.9583333333333334
Precision  : 1.0
Recall     : 0.9230769230769231
F1-Score   : 0.9600000000000002
Error rate   : 0.04166666666666663
```

[17]:

```
# For Setosa
tp = cm[0][0]
fn = cm[0][1] + cm[0][2]
fp = cm[1][0] + cm[2][0]
tn = cm[1][1] + cm[1][2] + cm[2][1]

print("For Setosa \n")
print(f"Accuracy   : {accuracy_cm(tp, fn, fp, tn)}")
print(f"Precision  : {precision_cm(tp, fn, fp, tn)}")
print(f"Recall     : {recall_cm(tp, fn, fp, tn)}")
print(f"F1-Score   : {f1_score(tp, fn, fp, tn)}")
print(f"Error rate   : {error_rate_cm(tp, fn, fp, tn)}")
```

```
For Setosa

Accuracy   : 1.0
Precision  : 1.0
Recall     : 1.0
F1-Score   : 1.0
Error rate   : 0.0
```

[ ]: