20bsit154-ass-3

March 30, 2023

[3]:
```
# Assignment - III
# ID : 20BSIT154

#Q-1 Read the seeds.csv file
import pandas as pd

df = pd.read_csv("D:/sem6/dma/practical/seeds.csv")
```

[4]:
```
print(df)
```

```
      Area  Perimeter  Compactness  Kernel.Length  Kernel.Width  \
0    15.26      14.84       0.8710          5.763         3.312
1    14.88      14.57       0.8811          5.554         3.333
2    14.29      14.09       0.9050          5.291         3.337
3    13.84      13.94       0.8955          5.324         3.379
4    16.14      14.99       0.9034          5.658         3.562
..     …         …           …              …             …
194  12.19      13.20       0.8783          5.137         2.981
195  11.23      12.88       0.8511          5.140         2.795
196  13.20      13.66       0.8883          5.236         3.232
197  11.84      13.21       0.8521          5.175         2.836
198  12.30      13.34       0.8684          5.243         2.974

     Asymmetry.Coeff  Kernel.Groove  Type
0              2.221          5.220     1
1              1.018          4.956     1
2              2.699          4.825     1
3              2.259          4.805     1
4              1.355          5.175     1
..               …              …   …
194            3.631          4.870     3
195            4.325          5.003     3
196            8.315          5.056     3
197            3.598          5.044     3
198            5.637          5.063     3

[199 rows x 8 columns]
```

[5]:
```
#Q-2 Perform all data pre-processing techniques.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 199 entries, 0 to 198
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Area             199 non-null    float64
 1   Perimeter        199 non-null    float64
 2   Compactness      199 non-null    float64
 3   Kernel.Length    199 non-null    float64
 4   Kernel.Width     199 non-null    float64
 5   Asymmetry.Coeff  199 non-null    float64
 6   Kernel.Groove    199 non-null    float64
 7   Type             199 non-null    int64
dtypes: float64(7), int64(1)
memory usage: 12.6 KB
```

[6]:
```
df.isnull().sum()
```

[6]:
```
Area              0
Perimeter         0
Compactness       0
Kernel.Length     0
Kernel.Width      0
Asymmetry.Coeff   0
Kernel.Groove     0
Type              0
dtype: int64
```

[7]:
```
#Q-3 Perform one-hot encoding and normalization.

df2 = pd.get_dummies(df,columns=['Type'])
```

[8]:
```
print(df2)
```

```
      Area  Perimeter  Compactness  Kernel.Length  Kernel.Width  \
0    15.26      14.84       0.8710          5.763         3.312
1    14.88      14.57       0.8811          5.554         3.333
2    14.29      14.09       0.9050          5.291         3.337
3    13.84      13.94       0.8955          5.324         3.379
4    16.14      14.99       0.9034          5.658         3.562
..     …          …           …              …             …
194  12.19      13.20       0.8783          5.137         2.981
195  11.23      12.88       0.8511          5.140         2.795
196  13.20      13.66       0.8883          5.236         3.232
197  11.84      13.21       0.8521          5.175         2.836
```

```
198  12.30      13.34       0.8684          5.243         2.974

     Asymmetry.Coeff  Kernel.Groove  Type_1  Type_2  Type_3
0              2.221          5.220       1       0       0
1              1.018          4.956       1       0       0
2              2.699          4.825       1       0       0
3              2.259          4.805       1       0       0
4              1.355          5.175       1       0       0
..               …              …     …     …     …
194            3.631          4.870       0       0       1
195            4.325          5.003       0       0       1
196            8.315          5.056       0       0       1
197            3.598          5.044       0       0       1
198            5.637          5.063       0       0       1

[199 rows x 10 columns]
```

[9]:
```
from sklearn import preprocessing
df3 = preprocessing.normalize(df2)
print(df3)
```

```
[[0.66209382 0.64387105 0.03779055 … 0.04338754 0.         0.        ]
 [0.66344682 0.64962501 0.03928515 … 0.04458648 0.         0.        ]
 [0.6561907  0.64700678 0.04155721 … 0.04591957 0.         0.        ]
 …
 [0.59318291 0.61385444 0.03991851 … 0.         0.         0.0449381 ]
 [0.59982874 0.6692346  0.04316842 … 0.         0.         0.05066121]
 [0.59681799 0.64728065 0.04213632 … 0.         0.         0.04852179]]
```

[10]:
```
x=df.values[:,0:6]
print(x)
```

```
[[15.26   14.84    0.871   5.763   3.312   2.221 ]
 [14.88   14.57    0.8811  5.554   3.333   1.018 ]
 [14.29   14.09    0.905   5.291   3.337   2.699 ]
 …
 [13.2    13.66    0.8883  5.236   3.232   8.315 ]
 [11.84   13.21    0.8521  5.175   2.836   3.598 ]
 [12.3    13.34    0.8684  5.243   2.974   5.637 ]]
```

[11]:
```
y=df.values[:,7]
print(y)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
```

```
2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
3. 3. 3. 3. 3. 3. 3.]
```

[12]:
```
#Q-4 Apply decision tree algorithm and display classification report.
from sklearn.model_selection import train_test_split
```

[13]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
↪2,random_state=100)
```

[14]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,
                                  max_depth=3, min_samples_leaf=5)
clf_gini.fit(x_train, y_train)
```

[14]:
```
DecisionTreeClassifier(max_depth=3, min_samples_leaf=5, random_state=100)
```

[15]:
```
y_pred = clf_gini.predict(x_test)
y_pred
```

[15]:
```
array([2., 2., 2., 2., 2., 3., 2., 1., 1., 2., 3., 1., 3., 3., 2., 3., 3.,
       3., 3., 1., 1., 2., 1., 2., 2., 2., 3., 2., 3., 2., 2., 3., 3., 3.,
       3., 2., 1., 3., 1., 2.])
```

[16]:
```
print("Accuracy is ", accuracy_score(y_test,y_pred)*100)
```

```
Accuracy is  87.5
```

[17]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[ 5  1  1]
 [ 2 16  0]
 [ 1  0 14]]
```

[18]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.62      0.71      0.67         7
         2.0       0.94      0.89      0.91        18
         3.0       0.93      0.93      0.93        15

    accuracy                           0.88        40
```

```
   macro avg       0.83      0.85      0.84        40
weighted avg       0.88      0.88      0.88        40
```

[19]:
```
clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,
 max_depth=3, min_samples_leaf=5)
clf_entropy.fit(x_train, y_train)
```

[19]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=5,
                       random_state=100)
```

[20]:
```
y_pred = clf_entropy.predict(x_test)
y_pred
```

[20]:
```
array([2., 2., 2., 2., 2., 3., 2., 2., 1., 2., 3., 2., 3., 3., 2., 3., 3.,
       3., 3., 1., 1., 2., 1., 1., 2., 2., 3., 2., 3., 2., 2., 3., 3., 3.,
       3., 2., 1., 3., 3., 2.])
```

[21]:
```
print("Accuracy is ", accuracy_score(y_test,y_pred)*100)
```

```
Accuracy is  97.5
```

[22]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[ 6  0  1]
 [ 0 18  0]
 [ 0  0 15]]
```

[23]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       1.00      0.86      0.92         7
         2.0       1.00      1.00      1.00        18
         3.0       0.94      1.00      0.97        15

    accuracy                           0.97        40
   macro avg       0.98      0.95      0.96        40
weighted avg       0.98      0.97      0.97        40
```

[24]:
```
#Q -5 Apply naïve-byes algorithm and display classification report.
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
# Model-training
model.fit(x_train, y_train)
```

```
y_pred = model.predict(x_test)
y_pred
```

[24]:
```
array([2., 2., 2., 2., 2., 3., 2., 2., 1., 2., 3., 2., 3., 3., 2., 3., 3.,
       3., 3., 1., 1., 2., 1., 1., 2., 2., 3., 2., 3., 2., 2., 3., 3., 3.,
       3., 2., 1., 3., 3., 2.])
```

[25]:
```
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    f1_score,
)

print("Accuracy is ", accuracy_score(y_test,y_pred)*100)

f1 = f1_score(y_pred, y_test, average="weighted")
print("F1 Score:", f1)
```

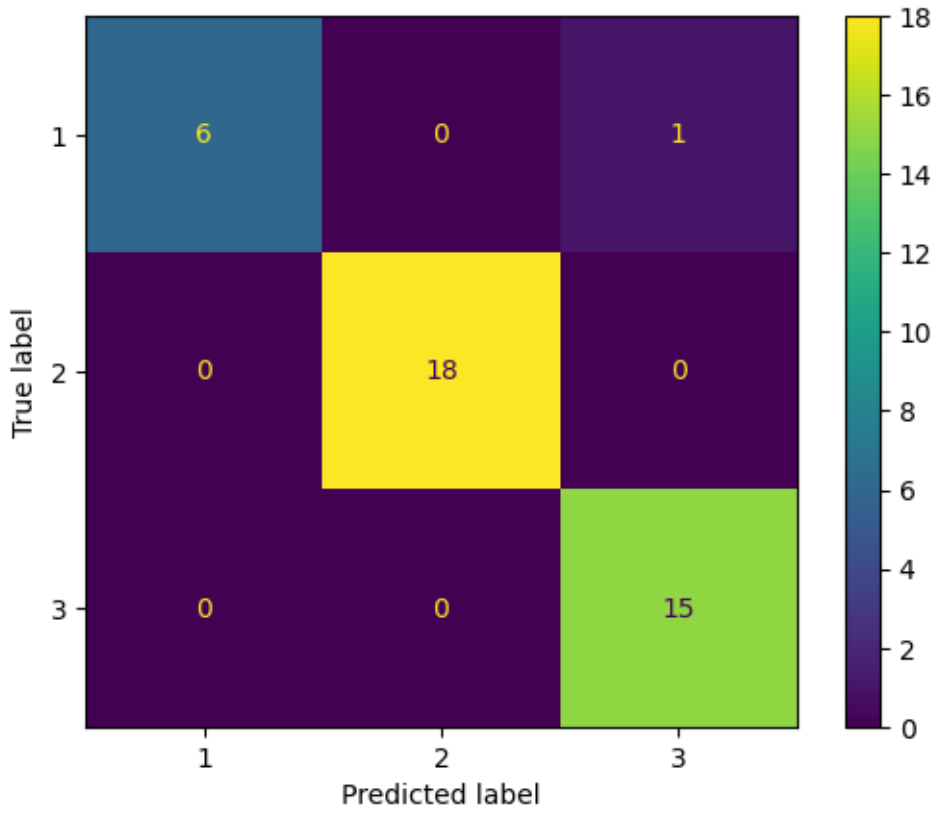
```
Accuracy is  97.5
F1 Score: 0.9755583126550869
```

[26]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[ 6  0  1]
 [ 0 18  0]
 [ 0  0 15]]
```

[27]:
```
labels = [1,2,3]
cm = confusion_matrix(y_test, y_pred, labels=labels)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot();
```



[28]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       1.00      0.86      0.92         7
         2.0       1.00      1.00      1.00        18
         3.0       0.94      1.00      0.97        15

    accuracy                           0.97        40
   macro avg       0.98      0.95      0.96        40
weighted avg       0.98      0.97      0.97        40
```

[ ]: