Program 24

October 25, 2022

1 PROGRAM 24

Aim : Program to implement k-NN classification using the datasets (Breastcancer.csv,Telco-Customer-Churn.csv) and find the accuracy of the algorithm

Date : 16/09/2022

By : Anu C Scharia

[212]:
```
import numpy as np
import pandas as pd
!pip install scikit-learn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scikit-learn in
/opt/anaconda3/lib/python3.9/site-packages (1.0.2)
Requirement already satisfied: numpy>=1.14.6 in
/opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.21.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: joblib>=0.11 in
/opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.1.0)
Requirement already satisfied: scipy>=1.1.0 in
/opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.7.3)
```

[213]:
```
#Step 1.load dataset
import pandas as pd
df=pd.read_csv('BreastCancer.csv')
#print(df)
```

[214]:
```
df.shape
```

[214]:
```
(569, 33)
```

[215]:
```
cols=df.columns
print(cols)
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

[216]:
```
df.value_counts("diagnosis")
```

[216]:
```
diagnosis
B    357
M    212
dtype: int64
```

[217]:
```
# Step 2.Separating the dependent and independent variable
y=df['diagnosis'] ##DEpendent variable
df.drop('diagnosis',axis=1,inplace=True)
print(y)
```

```
0      M
1      M
2      M
3      M
4      M
      ..
564    M
565    M
566    M
567    M
568    B
Name: diagnosis, Length: 569, dtype: object
```

[218]:
```
#Step 3. Removing Unimportant features and features with most of the values are␣
 ↪null
df.drop('Unnamed: 32', axis = 1,inplace=True)
df.drop('id', axis = 1,inplace=True)
cols=df.columns
print(cols)
x=df
#print(x)
```

```
Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
```

```
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

[219]:
```
df.describe()
```

[219]:

	radius_mean	texture_mean	perimeter_mean	area_mean	\
count	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	
std	3.524049	4.301036	24.298981	351.914129	
min	6.981000	9.710000	43.790000	143.500000	
25%	11.700000	16.170000	75.170000	420.300000	
50%	13.370000	18.840000	86.240000	551.100000	
75%	15.780000	21.800000	104.100000	782.700000	
max	28.110000	39.280000	188.500000	2501.000000	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
count	569.000000	569.000000	569.000000	569.000000	
mean	0.096360	0.104341	0.088799	0.048919	
std	0.014064	0.052813	0.079720	0.038803	
min	0.052630	0.019380	0.000000	0.000000	
25%	0.086370	0.064920	0.029560	0.020310	
50%	0.095870	0.092630	0.061540	0.033500	
75%	0.105300	0.130400	0.130700	0.074000	
max	0.163400	0.345400	0.426800	0.201200	

	symmetry_mean	fractal_dimension_mean	…	radius_worst	\
count	569.000000	569.000000	…	569.000000	
mean	0.181162	0.062798	…	16.269190	
std	0.027414	0.007060	…	4.833242	
min	0.106000	0.049960	…	7.930000	
25%	0.161900	0.057700	…	13.010000	
50%	0.179200	0.061540	…	14.970000	
75%	0.195700	0.066120	…	18.790000	
max	0.304000	0.097440	…	36.040000	

	texture_worst	perimeter_worst	area_worst	smoothness_worst	\
count	569.000000	569.000000	569.000000	569.000000	
mean	25.677223	107.261213	880.583128	0.132369	
std	6.146258	33.602542	569.356993	0.022832	
min	12.020000	50.410000	185.200000	0.071170	
25%	21.080000	84.110000	515.300000	0.116600	
50%	25.410000	97.660000	686.500000	0.131300	

```
75%        29.720000       125.400000  1084.000000          0.146000
max        49.540000       251.200000  4254.000000          0.222600

       compactness_worst  concavity_worst  concave points_worst  \
count         569.000000       569.000000            569.000000
mean            0.254265         0.272188              0.114606
std             0.157336         0.208624              0.065732
min             0.027290         0.000000              0.000000
25%             0.147200         0.114500              0.064930
50%             0.211900         0.226700              0.099930
75%             0.339100         0.382900              0.161400
max             1.058000         1.252000              0.291000

       symmetry_worst  fractal_dimension_worst
count      569.000000               569.000000
mean         0.290076                 0.083946
std          0.061867                 0.018061
min          0.156500                 0.055040
25%          0.250400                 0.071460
50%          0.282200                 0.080040
75%          0.317900                 0.092080
max          0.663800                 0.207500

[8 rows x 30 columns]
```

[221]:
```
df.isnull().sum()
```

[221]:
```
radius_mean               0
texture_mean              0
perimeter_mean            0
area_mean                 0
smoothness_mean           0
compactness_mean          0
concavity_mean            0
concave points_mean       0
symmetry_mean             0
fractal_dimension_mean    0
radius_se                 0
texture_se                0
perimeter_se              0
area_se                   0
smoothness_se             0
compactness_se            0
concavity_se              0
concave points_se         0
symmetry_se               0
fractal_dimension_se      0
```

```
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

[222]:
```
#Finding Missing Data,Encoding Categorical Data not need in this problem
# Step 4:Splitting the data into training and testing data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x, y,test_size= 0.
 ↪25,random_state=0)
```

[223]:
```
# Step 5:Feature Scaling(MinMaxScaler)
from sklearn.preprocessing import MinMaxScaler
st_x=MinMaxScaler()
x_train=st_x.fit_transform(x_train)
x_test=st_x.fit_transform(x_test)
print(x_train)
```

```
[[0.23044157 0.32157676 0.21940433 … 0.31484671 0.30277942 0.09858323]
 [0.20062473 0.42116183 0.19452699 … 0.06965208 0.34042973 0.06677161]
 [0.62232003 0.76929461 0.60403566 … 0.56079917 0.19850187 0.07431457]
 …
 [0.11619102 0.35726141 0.11077327 … 0.17402687 0.17524147 0.17263545]
 [0.12963226 0.35311203 0.11706171 … 0.         0.06780997 0.06919848]
 [0.21434995 0.59004149 0.21235575 … 0.33251808 0.10782574 0.21172767]]
```

[187]:
```
# step 6: Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)
```

[187]:
```
KNeighborsClassifier()
```

[188]:
```
# step 7: Prdicting the test result
y_pred=classifier.predict(x_test)
print(y_pred)
```

```
['M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'M'
 'M' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'M'
 'B' 'M' 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'M' 'M' 'B'
 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'B' 'M' 'M'
```

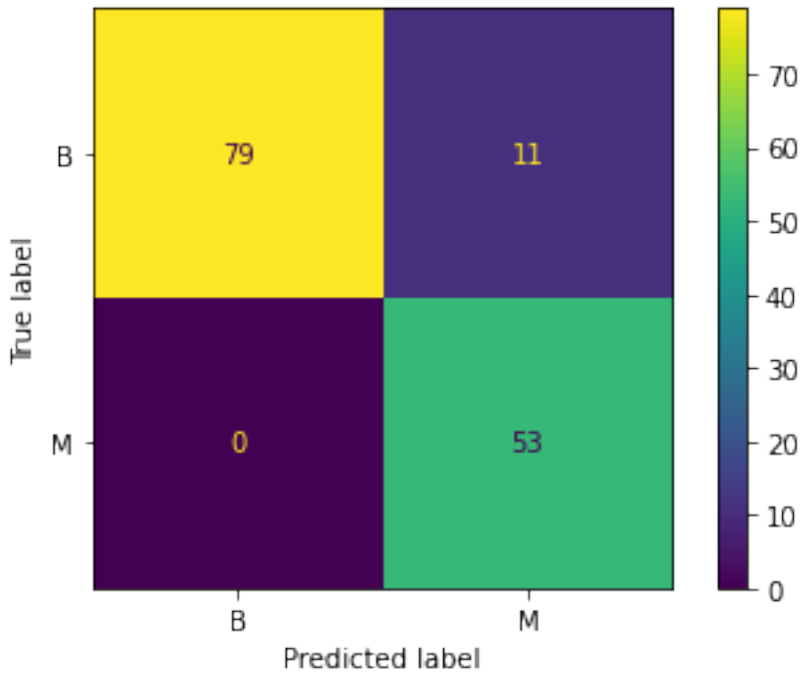
```
 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'M'
 'B' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'B'
 'M' 'M' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M'
 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M']
```

[189]:
```
# Step 8 : Creating the Confusion matrix to see the accuracy of the classifier.
from sklearn.metrics import confusion_matrix
cm=confusion_matrix (y_test,y_pred,labels=classifier.classes_)
print(cm)
```

```
[[79 11]
 [ 0 53]]
```

[190]:
```
# Confusion Matrix Display
from sklearn.metrics import ConfusionMatrixDisplay
disp=ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=classifier.
 ↪classes_)
disp.plot()
```

[190]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f44f4041250>
```



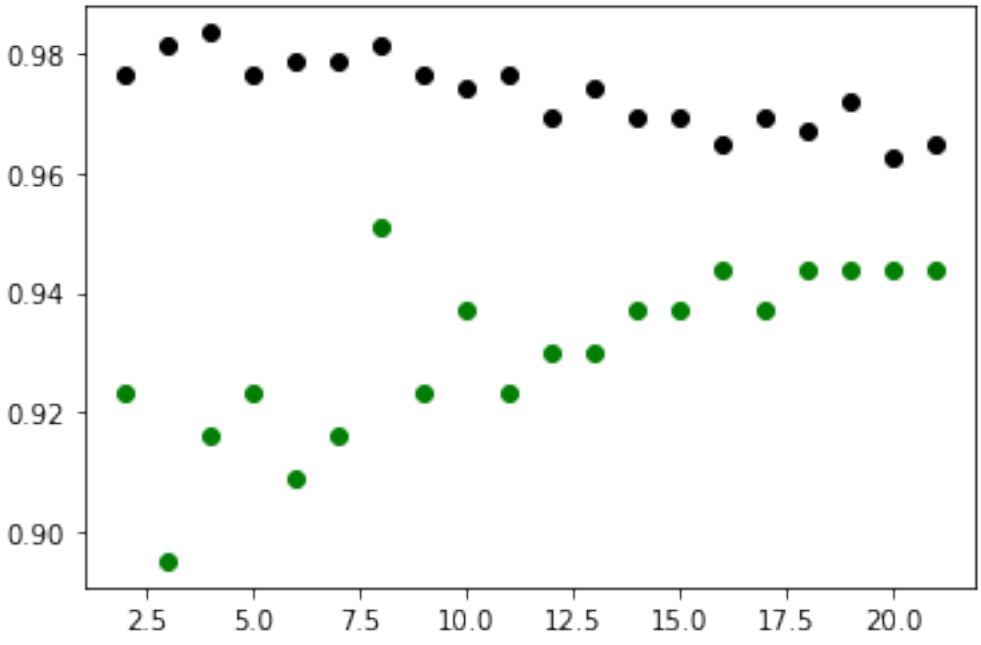
[191]:
```
training_score=classifier.score(x_train, y_train)
test_score=classifier.score(x_test, y_test)
print("Training Accuracy :",training_score)
```

```
print("Testing Accuracy :",test_score)
```

```
Training Accuracy : 0.9765258215962441
Testing Accuracy : 0.9230769230769231
```

[192]:
```
## With different values of k.
K=[]
training=[]
test=[]
scores={}
for k in range(2, 22):
    clf=KNeighborsClassifier(n_neighbors=k)
    clf.fit(x_train,y_train)
    training_score=clf.score(x_train, y_train)
    test_score=clf.score(x_test, y_test)
    K.append(k)
    training.append(training_score)
    test.append(test_score)
    scores[k]=[training_score, test_score]
for keys, values in scores.items():
    print(keys,':',values)
# visualization
import matplotlib.pyplot as plt
plt.scatter(K,training,color='k')
plt.scatter(K,test,color='g')
plt.show()
```

```
2 : [0.9765258215962441, 0.9230769230769231]
3 : [0.9812206572769953, 0.8951048951048951]
4 : [0.9835680751173709, 0.916083916083916]
5 : [0.9765258215962441, 0.9230769230769231]
6 : [0.9788732394366197, 0.9090909090909091]
7 : [0.9788732394366197, 0.916083916083916]
8 : [0.9812206572769953, 0.951048951048951]
9 : [0.9765258215962441, 0.9230769230769231]
10 : [0.9741784037558685, 0.9370629370629371]
11 : [0.9765258215962441, 0.9230769230769231]
12 : [0.9694835680751174, 0.9300699300699301]
13 : [0.9741784037558685, 0.9300699300699301]
14 : [0.9694835680751174, 0.9370629370629371]
15 : [0.9694835680751174, 0.9370629370629371]
16 : [0.9647887323943662, 0.9440559440559441]
17 : [0.9694835680751174, 0.9370629370629371]
18 : [0.9671361502347418, 0.9440559440559441]
19 : [0.971830985915493, 0.9440559440559441]
20 : [0.9624413145539906, 0.9440559440559441]
21 : [0.9647887323943662, 0.9440559440559441]
```


0.98
0.96
0.94
0.92
0.90
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0