Experiment_7

March 10, 2024

[ ]:
```
EX NO 7 - Performance Analysis on KNN Classification Technique
Date:26-02-2024
HARIHARAN K - URK22AI1048
```

[ ]:
```
Description

    K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm␣
 ↪which can be
used for both classification as well as regression predictive problems.␣
 ↪However, it is mainly
used for classification predictive problems in industry. The following two␣
 ↪properties would
define KNN well:
    - Lazy learning algorithm: KNN is a lazy learning algorithm because it does␣
 ↪not have a
    specialized training phase and uses all the data for training while␣
 ↪classification.
    - Non-parametric learning algorithm: KNN is also a non-parametric learning␣
 ↪algorithm
    because it doesn't assume anything about the underlying data.

Working of KNN Algorithm
    K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict␣
 ↪the values of new
datapoints which further means that the new data point will be assigned a value␣
 ↪based on how
closely it matches the points in the training set. We can understand its␣
 ↪working with the help
of following steps:
    Step1: For implementing any algorithm, we need dataset. So, during the␣
 ↪first step of KNN,
load the training as well as test data.
    Step2: Choose the value of K i.e. the nearest data points. K can be any␣
 ↪integer.
    Step3: For each point in the test data do the following:
        3.1: Calculate the distance between test data and each row of training␣
 ↪data with the help of any
```

```
        of the method namely: Euclidean, Manhattan or Hamming distance. The
 ↪most commonly used
        method to calculate distance is Euclidean.
        3.2: Now, based on the distance value, sort them in ascending order.
        3.3: Next, it will choose the top K rows from the sorted array.
        3.4: Now, it will assign a class to the test point based on most
 ↪frequent class of these rows.
    Step4: End

Performance Metrics for Classification Problems
    Various performance metrics that can be used to evaluate predictions for
 ↪classification
problems are given below

Confusion Matrix
    It is the easiest way to measure the performance of a classification
 ↪problem where the output
can be of two or more type of classes. A confusion matrix is nothing but a
 ↪table with two
dimensions viz."Actual" and "Predicted" and furthermore, both the dimensions
 ↪have "True

Positives (TP)", "True Negatives (TN)", "False Positives (FP)", "False
 ↪Negatives (FN)"
• True Positives (TP): It is the case when both actual class & predicted class
 ↪of data point
is 1.
• True Negatives (TN): It is the case when both actual class & predicted class
 ↪of data
point is 0.
• False Positives (FP): It is the case when actual class of data point is 0 &
 ↪predicted class
of data point is 1.
• False Negatives (FN): It is the case when actual class of data point is 1 &
 ↪predicted class
of data point is 0.

confusion_matrix function of sklearn.metrics to compute Confusion Matrix
of our classification model.

Classification Accuracy
    It is most common performance metric for classification algorithms. It may
 ↪be defined as the
number of correct predictions made as a ratio of all predictions made. [][] + [][]

Classification Report
```

```
    This report consists of the scores of Precisions, Recall, F1 and Support.␣
 ↪They
are Precision
    Precision, used in document retrievals, may be defined as the number of
correct documents returned by classification model.
    Support may be defined as the number of samples of the true response that␣
 ↪lies in each
class of target values.

F1 Score
    This score will give us the harmonic mean of precision and recall.␣
 ↪Mathematically, F1
score is the weighted average of the precision and recall. The best value of F1␣
 ↪would be 1 and
    F1 score is having equal relative contribution of precision and recall.
classification_report function of sklearn.metrics is used to get the␣
 ↪classification report
of classification model.

AUC (Area Under ROC curve)
    AUC (Area Under Curve)-ROC (Receiver Operating Characteristic) is a␣
 ↪performance metric, based on varying threshold values, for classification␣
 ↪problems. As name suggests,
ROC is a probability curve and AUC measure the separability. In simple words,␣
 ↪AUC-ROC
metric will tell us about the capability of model in distinguishing the classes.
 ↪ Higher the AUC,
better the model. Mathematically, it can be created by plotting TPR (True␣
 ↪Positive Rate) i.e.
Sensitivity or recall vs FPR (False Positive Rate) i.e. 1-Specificity, at␣
 ↪various threshold values.
roc_auc_score function of sklearn.metrics is used to compute AUC-ROC.
```

[1]:
```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt

df = pd.read_csv("sn.csv")
df.shape
```

[1]:
```
(569, 12)
```

[2]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

[18]:
```
#extract data and store in var X
#URK22AI1048
X=df.iloc[:,3:10]
X.head()
```

[18]:

	perimeter_mean	area_mean	smoothness_mean	compactness_mean	\
0	122.80	1001.0	0.11840	0.27760	
1	132.90	1326.0	0.08474	0.07864	
2	130.00	1203.0	0.10960	0.15990	
3	77.58	386.1	0.14250	0.28390	
4	135.10	1297.0	0.10030	0.13280	

	concavity_mean	concave points_mean	symmetry_mean
0	0.3001	0.14710	0.2419
1	0.0869	0.07017	0.1812
2	0.1974	0.12790	0.2069
3	0.2414	0.10520	0.2597
4	0.1980	0.10430	0.1809

[19]:
```
#extract data and store in var Y
#URK22AI1048
Y=df.iloc[:,11]
Y.head()
```

[19]:
```
0    M
1    M
2    M
3    M
4    M
Name: diagnosis, dtype: object
```

[20]:
```
#Encode data into 1s and 0s
#URK22AI1048
le=LabelEncoder()
Y = le.fit_transform(Y)
Y
```

[20]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 1, 1, 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, 1, 1, 1, 1, 1, 1, 0])
```

[21]:
```
#splitting DataSet into training and testing set
#URK22AI1048
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.
↪25,random_state=1)
```

[29]:
```
#Memorize the data
#URK22AI1048
knn=KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
```

[29]:
```
KNeighborsClassifier(n_neighbors=7)
```

[23]:
```
#Predicting the result with testing dataset
#URK22AI1048
y_pred=knn.predict(X_test)
```

[24]:
```
conf_matrix=confusion_matrix(Y_test,y_pred)
conf_matrix
```

[24]:
```
array([[80,  8],
       [13, 42]])
```

[25]:
```
print("Accuracy",accuracy_score(Y_test,y_pred))
print("Recall",recall_score(Y_test,y_pred,pos_label=1))
print("Specificity",recall_score(Y_test,y_pred,pos_label=0))
print("Precision",precision_score(Y_test,y_pred))
print("Fscore",f1_score(Y_test,y_pred))
```

```
Accuracy 0.8531468531468531
Recall 0.7636363636363637
Specificity 0.9090909090909091
Precision 0.84
Fscore 0.8
```

[26]:
```
fpr, tpr, _ = roc_curve(Y_test, y_pred)
print(fpr)
print(tpr)
```

```
[0.         0.09090909 1.        ]
[0.         0.76363636 1.        ]
```

[27]:
```
auc_score=roc_auc_score(Y_test,y_pred)
print("auc",auc_score)
```

```
auc 0.8363636363636363
```

[ ]:
```
Result:
        The Above Program were Created and Executed Successfully
```