**Ex.no:8 DHAMODARAN.B**

**Date:26.09.2020** **1832016**

**HYPER PARAMETER TUNING**

**Problem Statement:**

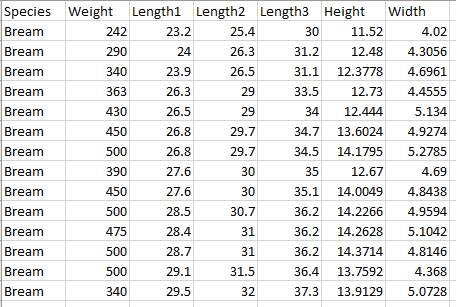
Fishery is one of the industries which contributes much to the Indian GDP. Ensuring the quality of fish is necessary. Since the fishes are priced on their weight and how rare the species is, detecting the species of the fish is necessary to determine the price. We will try to predict the species of the fish using the KNN classifier. The KNN model built with different hyper-parameters vary in their performance. So, determining the optimal values for hyper-parameters for which the model has the best performance score becomes necessary. Our motto is to fine tune the model and decide the optimal values for the hyper-parameters. This can be implemented using the Grid-search or Random-Search techniques.

**Problem Description:**

The dependent variable or the target variable that we want to predict is ‘species’ and the independent variables are the ‘weight’, ‘length1’, ’length2’, ’lenght3’, ’height’ and ‘width’. All the independent features were continuous in nature. The dataset had information on 7 species but we have taken only two species i.e. ‘Bream’ and ‘Perch’. The information on two species ‘Bream’ and ‘Perch’ had about 91 data instances.

Initially, the data was checked for null values. The data are scaled using the standard scaling technique and later it was split into train and test in the ratio 85:15. The K-nearest neighbor classifier model is implemented using the scikit’s learn’ s KNeighborsClassifier class. Later the model’s hyper-parameters are fined tuned using the grid search and random search techniques. The performance of the fined tuned model and the untuned model are compared under different metrics.

**Sample Dataset:**



**Code:**

**Normal Model without Hyper-parameter Tuning:**

import pandas as pd

import numpy as np

dataset=pd.read\_csv('Fish.csv')

print("The Different Species are :",list(dataset['Species'].unique()))

print("The data for the species Bream and Perch are :")

dataframe=pd.DataFrame(dataset[dataset['Species'].isin(['Bream','Perch'])])

dataframe.index=range(len(dataframe))

dataframe.isnull()

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

scaler=StandardScaler()

dataframe.iloc[:,1:]=scaler.fit\_transform(dataframe.iloc[:,1:])

xtrain,xtest,ytrain,ytest=train\_test\_split(dataframe.iloc[:,1:].values,dataframe['Species'].values,test\_size=0.15)

from sklearn.neighbors import KNeighborsClassifier

untuned\_model = KNeighborsClassifier(n\_jobs=-1,n\_neighbors=7,metric='manhattan')

untuned\_model.fit(xtrain,ytrain)

print('Untuned Model \n',untuned\_model)

print("Training R2 score : ",untuned\_model.score(xtrain,ytrain))

ypred=untuned\_model.predict(xtest)

print("Predictions \n",ypred)

print("True Values \n",ytest)

print("Untuned Model results\n\n")

conf\_mat=confusion\_matrix(ytest,ypred)

print("Confusion Matrix \n",conf\_mat)

print("\nClassification Report : \n",classification\_report(ytest,ypred))

untuned\_accuracy = (conf\_mat[0][0]+conf\_mat[1][1])/len(ytest)

print("Accuracy : " ,untuned\_accuracy )

probs=untuned\_model.predict\_proba(xtest)

probs=probs[:,1]

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelBinarizer

binarizer=LabelBinarizer()

ytest\_b=binarizer.fit\_transform(ytest)

fpr,tpr,\_=roc\_curve(ytest\_b,probs)

random\_probs = [0 for \_ in range(len(ytest))]

p\_fpr,p\_tpr,\_ = roc\_curve(ytest\_b,random\_probs)

auc\_score=roc\_auc\_score(ytest,probs)

print("AUC SCORE : " ,auc\_score)

plt.plot(p\_fpr, p\_tpr, linestyle='--')

plt.plot(fpr, tpr, marker='.', label='UNTUNED KNN (area=%0.2f)'% auc\_score)

plt.xlabel('False Positive Rate')

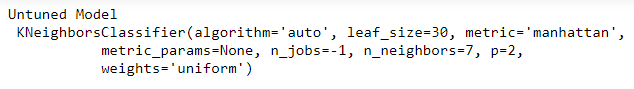
plt.ylabel('True Positive Rate')

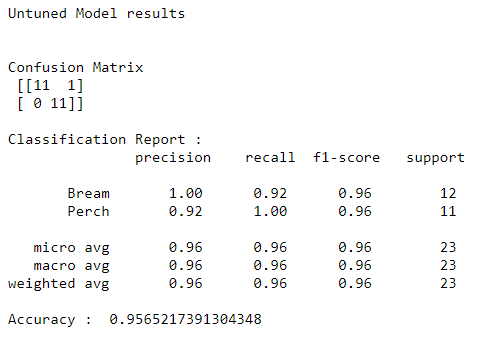
plt.title("ROC-AUC CURVE for UNTUNED KNN Classifier")

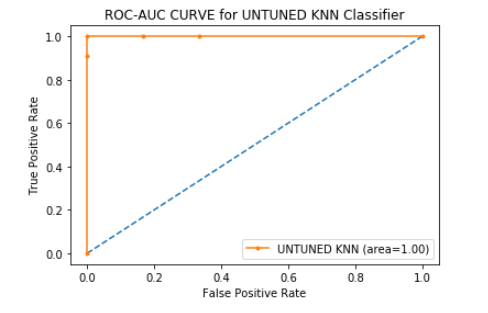
plt.legend()

plt.show()

**Output:**



x



**Inference of the Untuned model:**

The untuned model has a good 100% precision for the Bream class i.e. all the instances classified as Bream are actually Bream in nature while the Perch class has only 92% precision. The untuned model has an accuracy of 95%. The AUC score is 1.00 which explains that the model is capable to distinguish the classes very well.

**Model with the fine-tuned hyper-parameters:**

**Code:**

from sklearn.model\_selection import GridSearchCV,RandomizedSearchCV

ytrain\_b=binarizer.fit\_transform(ytrain).flatten()

from sklearn.metrics import scorer

from time import time

knn = KNeighborsClassifier()

params = {'n\_neighbors':range(2,11),'metric':['minkowski','manhattan','euclidean']}

grid\_search = GridSearchCV(knn,param\_grid = params,scoring='precision',cv=2)

start = time()

grid\_search.fit(xtrain,ytrain\_b)

end = time()

print("Grid Search Results\n \n")

print("Best Params : ",grid\_search.best\_params\_)

print("Best Precision : ",grid\_search.best\_score\_)

print("Best Model: \n",grid\_search.best\_estimator\_)

print("Time Taken : ",float(end - start))

knn = KNeighborsClassifier()

params = {'n\_neighbors':np.arange(2,11,step=1),'metric':['minkowski','manhattan','euclidean']}

random\_search = RandomizedSearchCV(knn,params,scoring='precision' ,cv=2)

start = time()

random\_search.fit(xtrain,ytrain\_b)

end = time()

print("Random Search Results")

print("Best Params : ",random\_search.best\_params\_)

print("Best Precision : ",random\_search.best\_score\_)

print("Best Model: \n",random\_search.best\_estimator\_)

print("Time Taken : ",float(end - start))

tuned\_model = random\_search.best\_estimator\_

ypred = tuned\_model.predict(xtest).flatten()

print("Fine Tuned Model results\n\n")

conf\_mat=confusion\_matrix(ytest\_b,ypred)

print("Confusion Matrix \n",conf\_mat)

print("\nClassification Report : \n",classification\_report(ytest\_b,ypred))

tuned\_accuracy = (conf\_mat[0][0]+conf\_mat[1][1])/len(ytest)

print("Accuracy : " ,tuned\_accuracy )

probs=tuned\_model.predict\_proba(xtest)

probs=probs[:,1]

fpr,tpr,\_=roc\_curve(ytest\_b,probs)

random\_probs = [0 for \_ in range(len(ytest))]

p\_fpr,p\_tpr,\_ = roc\_curve(ytest\_b,random\_probs)

auc\_score=roc\_auc\_score(ytest,probs)

print("AUC SCORE : " ,auc\_score)

plt.plot(p\_fpr, p\_tpr, linestyle='--')

plt.plot(fpr, tpr, marker='.', label='TUNED KNN (area=%0.2f)'% auc\_score)

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title("ROC-AUC CURVE for TUNED KNN Classifier")

plt.legend()

plt.show()

accuracy = {'Untuned model':untuned\_accuracy,'Fine-tuned model':tuned\_accuracy}

plt.scatter('Untuned model',accuracy['Untuned model'],label='Untuned Model')

plt.scatter('Fine-tuned model',accuracy['Fine-tuned model'],label='Fine-tuned Model')

plt.ylim(0.8,1.05)

plt.xlabel('Models')

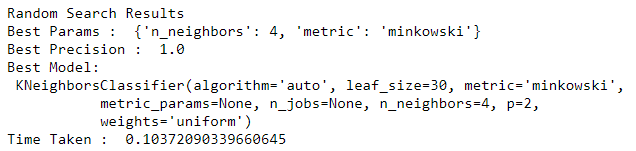
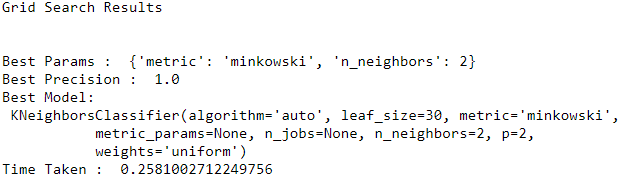
plt.ylabel('Accuracy')

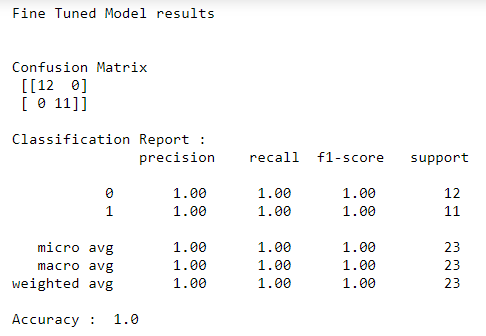
plt.legend()

plt.title("Comparision of fine-tuned and untuned models")

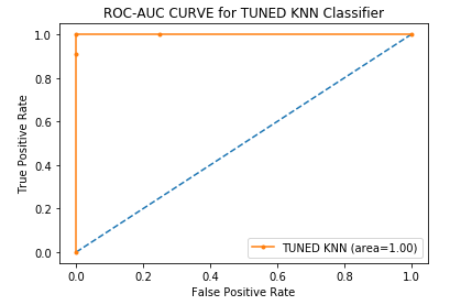
plt.show()

**Output:**







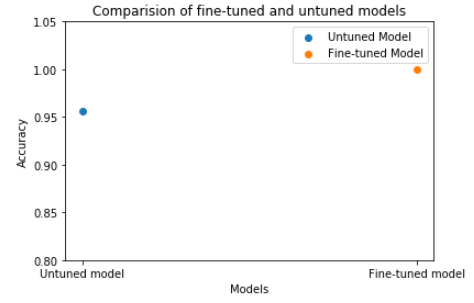


**Inference of the Fine-tuned Model:**

The hyper-parameters of the KNN model are fine tuned using the techniques Grid search and the Random Search. Both the techniques fined tuned the model and came with the same accuracy. Random search took 0.103 seconds to fine tune while the grid search comparatively more time of 0.258 seconds. So, the fine tuned model from the random search is considered as the best model and it is used for making further predictions. The values of the hyper-parameters ‘n\_neighbors’ and ‘metric’ obtained after the fine-tuning are 4 and ‘minkowski’.

The fine-tuned model had 100% precision for both the classes. Accuracy of 100% for the test class shows the model more stunning. The AUC score of 1.0 explains that the model is very well capable of distinguishing between the classes.

**Comparison of the fine-tuned and untuned model:**



The untuned model had the hyper parameters value for ‘n\_neighbors’ and ‘metric’ as 7 and ‘manhattan’ while the fine-tuned model had 4 and ‘minkowski’. The fine-tuned model also had better precision in making predictions than the untuned model. The fine-tuned model’s accuracy of 100% is comparatively better than the untuned model’s accuracy of 95%.Both the models had the same AUC score of 1.0 and both are very well capable of distinguishing the classes. so, considering all the metrics we can conclude that the fine-tuned model with hyper-parameters values for ‘n\_neighbors’ and ‘metric’ as 4 and ‘minkowsi’ performs well than the untuned one. Thus, the hyper-parameter tuning has resulted in improving the model performance.