EXPERIMENT: 11

(2K17/SE/79 PARV GUPTA)

AIM: Validate the results obtained in experiment 3 using 10-cross validation, hold out validation or leave one out cross-validation.

THEORY:

10-CROSS VALIDATION:

Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it. In k-fold cross-validation, the original sample is randomly partitioned into k equal size sub-samples. Of the k sub-samples, a single sub-sample is retained as the validation data for testing the model, and the remaining k-1 sub-samples are used as training data. The cross validation process is then repeated k times (the folds), with each of the k sub-samples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once. For classification problems, one typically uses stratified k-fold cross-validation, in which the folds are selected so that each fold contains roughly the same proportions of class labels. In repeated cross-validation, the cross-validation procedure is repeated n times, yielding n random partitions of the original sample. The n results are again averaged (or otherwise combined) to produce a single estimation.

PROCEDURE OF 10-FOLD CROSS VALIDATION:

1. Shuffle the dataset randomly.
2. Split the dataset into 10 groups
3. For each unique group:
    a. Take the group as a hold out or test data set
    b. Take the remaining groups as a training data set
    c. Fit a model on the training set and evaluate it on the test set
    d. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model 9 times.

HOLDOUT VALIDATION:

In the holdout method, we randomly assign data points to two sets d0 and d1, usually called the training set and the test set, respectively. The size of each of the sets is arbitrary although typically the test set is smaller than the training set. We then train (build a model) on d0 and test (evaluate its performance) on d1. In typical cross validation, results of multiple runs of model-testing are averaged together; in contrast, the holdout method, in isolation, involves a single run. It should be used with caution because without such averaging of multiple runs, one may achieve highly misleading results. One's indicator of predictive accuracy (F*) will tend to be unstable since it will not be smoothed out by multiple iterations (see below). Similarly, indicators of the specific role played by various predictor variables (e.g., values of regression coefficients) will tend to be unstable. While the holdout method can be framed as "the simplest kind of cross-validation", many sources instead classify holdout as a type of simple validation, rather than a simple or degenerate form of cross-validation.

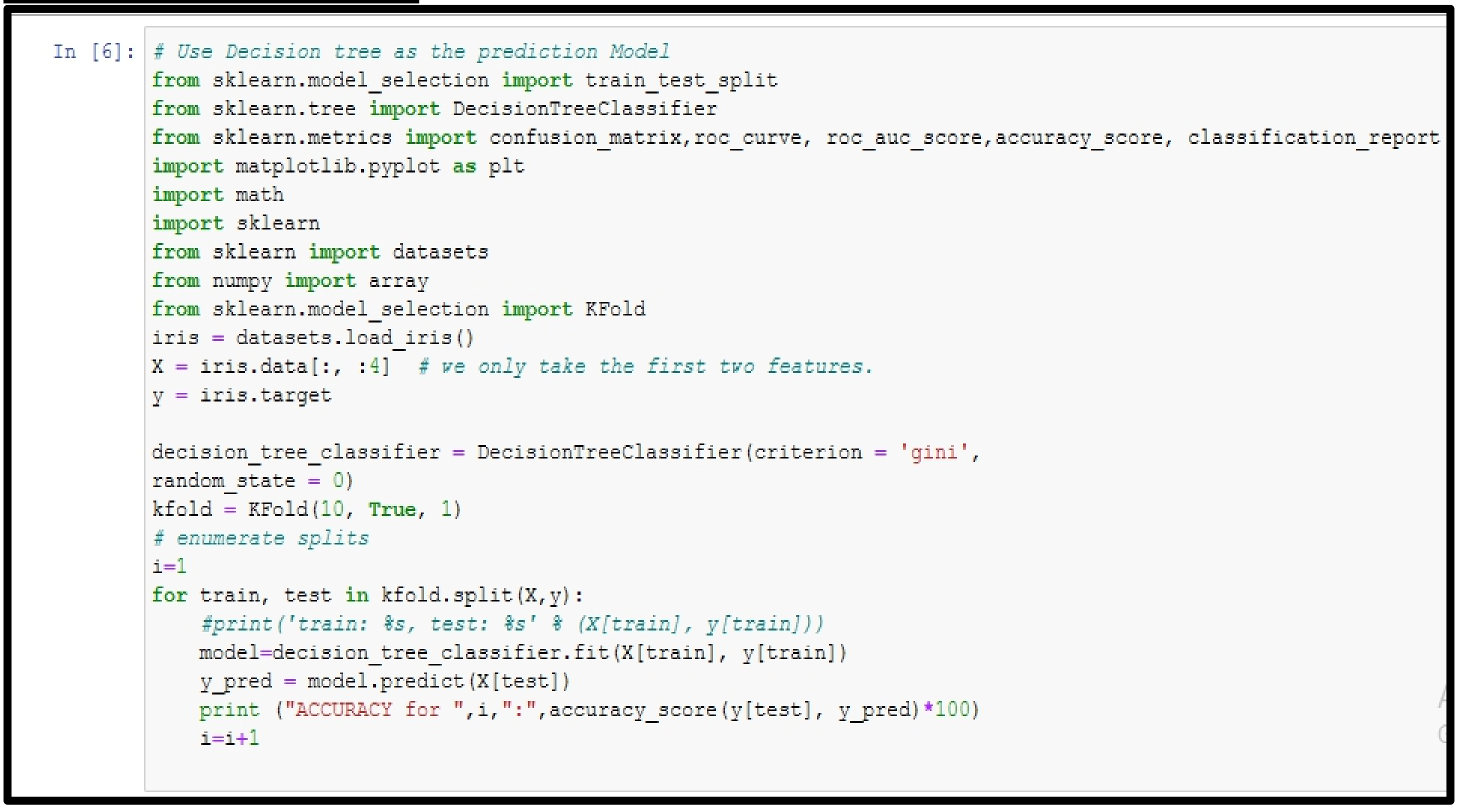
LEAVE-ONE-OUT CROSS VALIDATION:

Leave-one-out cross-validation, or LOOCV, is a configuration of k-fold cross validation where *k* is set to the number of examples in the dataset.

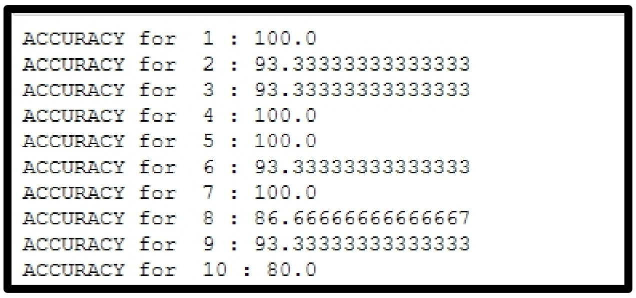
We then average ALL of these folds and build our model with the average. Because we would get a big number of training sets (equals to the number of samples), this method is very computationally expensive and should be used on small datasets. If the dataset is big, it would most likely be better to use a different method, like k-fold.

OUTPUT:

10 Fold Validation:

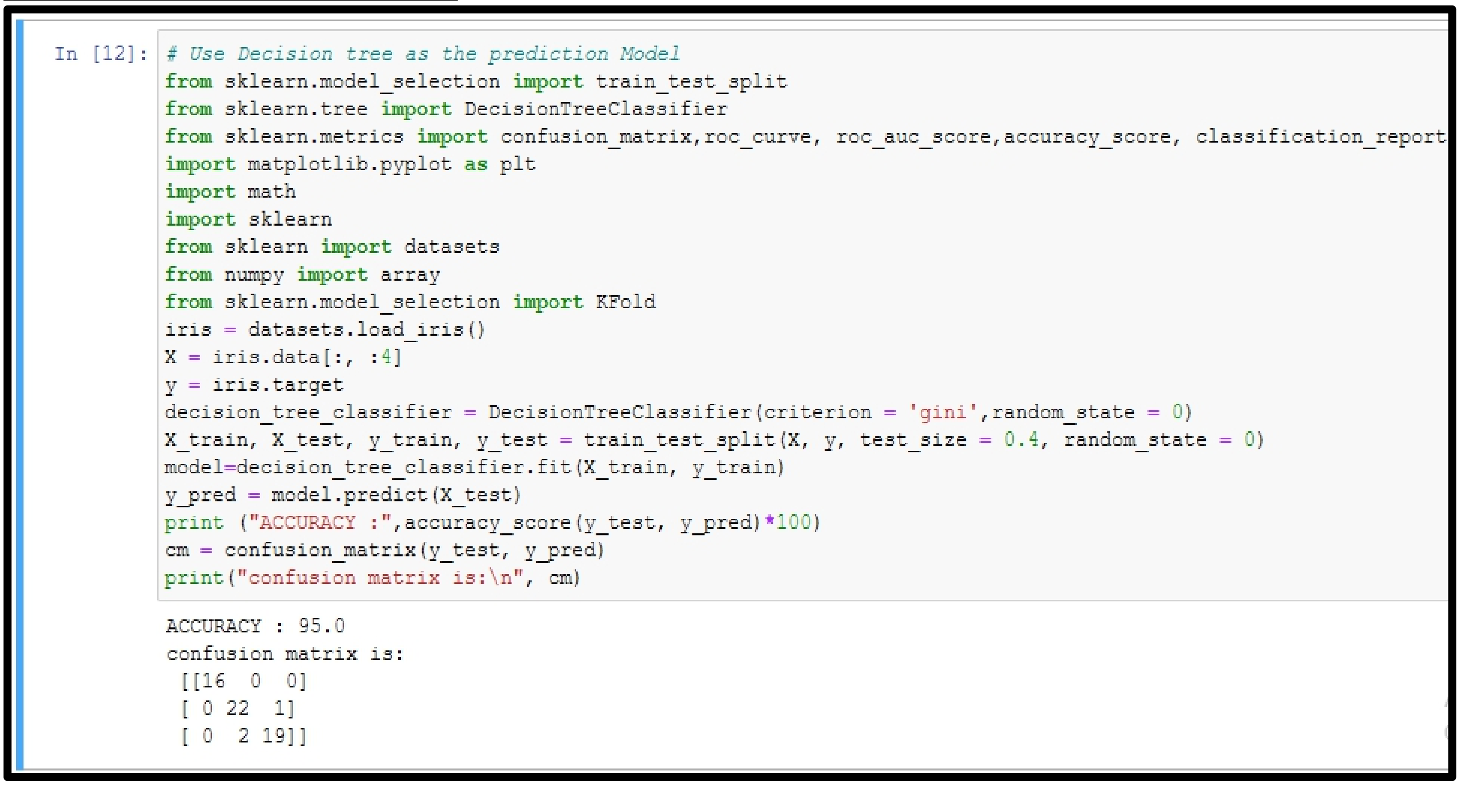


```
In [6]: # Use Decision tree as the prediction Model
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import confusion_matrix,roc_curve, roc_auc_score,accuracy_score, classification_report
        import matplotlib.pyplot as plt
        import math
        import sklearn
        from sklearn import datasets
        from numpy import array
        from sklearn.model_selection import KFold
        iris = datasets.load_iris()
        X = iris.data[:, :4]  # we only take the first two features.
        y = iris.target

        decision_tree_classifier = DecisionTreeClassifier(criterion = 'gini',
        random_state = 0)
        kfold = KFold(10, True, 1)
        # enumerate splits
        i=1
        for train, test in kfold.split(X,y):
            #print('train: %s, test: %s' % (X[train], y[train]))
            model=decision_tree_classifier.fit(X[train], y[train])
            y_pred = model.predict(X[test])
            print ("ACCURACY for ",i,":",accuracy_score(y[test], y_pred)*100)
            i=i+1
```



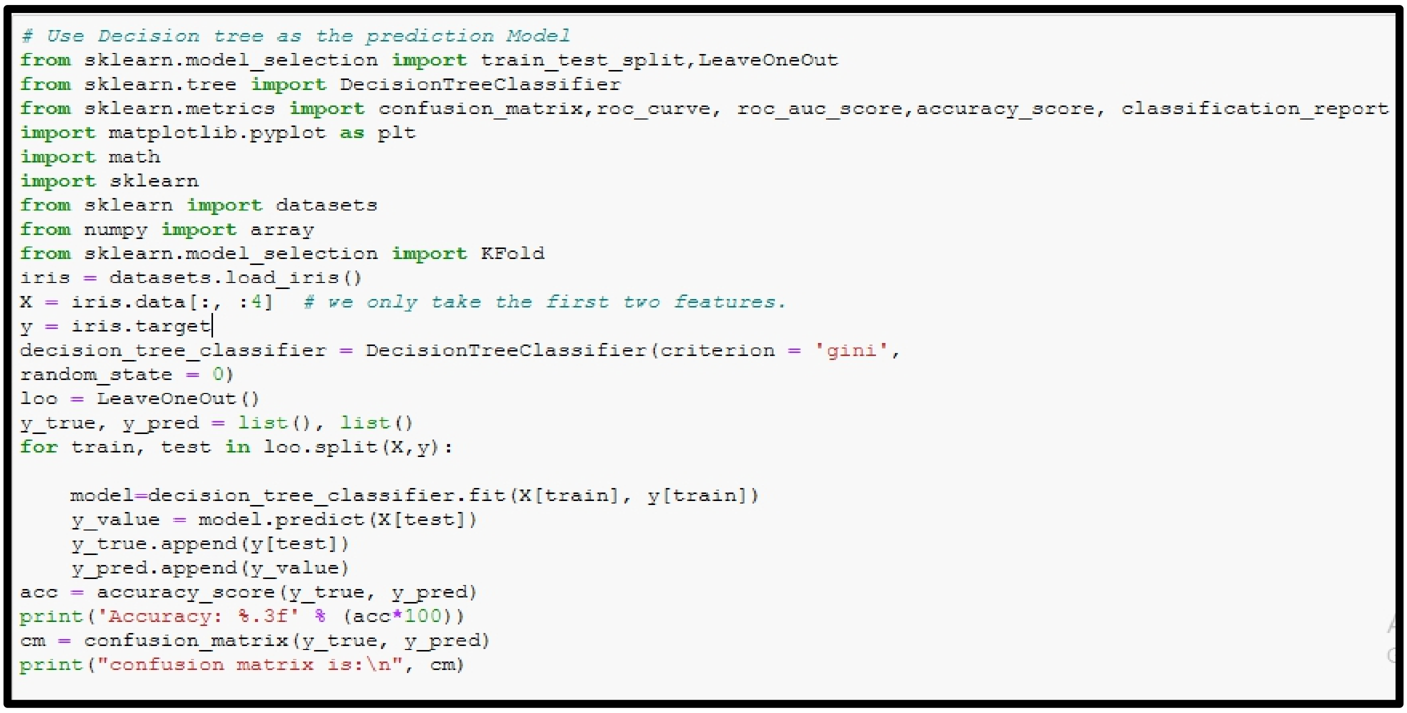
```
ACCURACY for  1 : 100.0
ACCURACY for  2 : 93.33333333333333
ACCURACY for  3 : 93.33333333333333
ACCURACY for  4 : 100.0
ACCURACY for  5 : 100.0
ACCURACY for  6 : 93.33333333333333
ACCURACY for  7 : 100.0
ACCURACY for  8 : 86.66666666666667
ACCURACY for  9 : 93.33333333333333
ACCURACY for  10 : 80.0
```

Hold Out Validation:

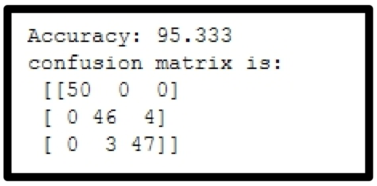


```
In [12]: # Use Decision tree as the prediction Model
         from sklearn.model_selection import train_test_split
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import confusion_matrix,roc_curve, roc_auc_score,accuracy_score, classification_report
         import matplotlib.pyplot as plt
         import math
         import sklearn
         from sklearn import datasets
         from numpy import array
         from sklearn.model_selection import KFold
         iris = datasets.load_iris()
         X = iris.data[:, :4]
         y = iris.target
         decision_tree_classifier = DecisionTreeClassifier(criterion = 'gini',random_state = 0)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4, random_state = 0)
         model=decision_tree_classifier.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         print ("ACCURACY :",accuracy_score(y_test, y_pred)*100)
         cm = confusion_matrix(y_test, y_pred)
         print("confusion matrix is:\n", cm)

         ACCURACY : 95.0
         confusion matrix is:
          [[16  0  0]
          [ 0 22  1]
          [ 0  2 19]]
```

Leave One Out Validation:



```
# Use Decision tree as the prediction Model
from sklearn.model_selection import train_test_split,LeaveOneOut
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix,roc_curve, roc_auc_score,accuracy_score, classification_report
import matplotlib.pyplot as plt
import math
import sklearn
from sklearn import datasets
from numpy import array
from sklearn.model_selection import KFold
iris = datasets.load_iris()
X = iris.data[:, :4]   # we only take the first two features.
y = iris.target
decision_tree_classifier = DecisionTreeClassifier(criterion = 'gini',
random_state = 0)
loo = LeaveOneOut()
y_true, y_pred = list(), list()
for train, test in loo.split(X,y):

    model=decision_tree_classifier.fit(X[train], y[train])
    y_value = model.predict(X[test])
    y_true.append(y[test])
    y_pred.append(y_value)
acc = accuracy_score(y_true, y_pred)
print('Accuracy: %.3f' % (acc*100))
cm = confusion_matrix(y_true, y_pred)
print("confusion matrix is:\n", cm)
```



```
Accuracy: 95.333
confusion matrix is:
 [[50  0  0]
 [ 0 46  4]
 [ 0  3 47]]
```

CONCLUSION:

The model has been validated by all methods.