NOAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

Department of Computer Science & Telecommunication Engineering

Couse Title : High Performance Computing System Lab

Course Code : CSTE 4108

Submitted To:

Dr, Md. Kamal Uddin

Associate Professor

Department of CSTE

Submitted BY:

Name: Antu Chowdhury

Roll No: ASH1801009M

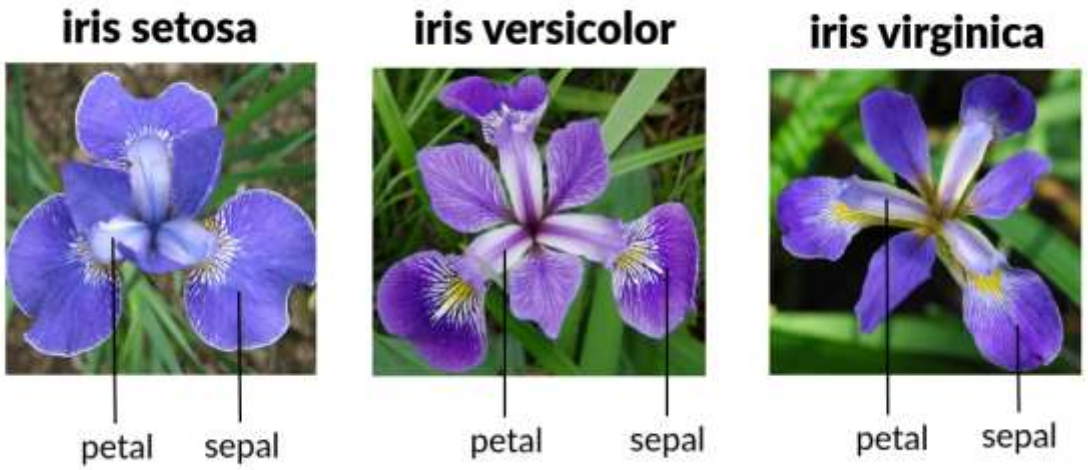
Year: 4 Term: 1

Date of Submission: 19.04.2022

Apply different machine learning algorithm on Iris Data Set to build classification models and evaluate their prediction accuracy.

About Iris Dataset:
The dataset contains the following data:

- 50 samples of 3 different species of iris (150 samples total)
- Measurements: sepal length, sepal width, petal length, petal width
- The format for the data: (sepal length, sepal width, petal length, petal width)



Data Preprocessing:

- Include Libraries
- Import Dataset

I. Include Libraries:

Importing pandas, numpy, matplotlib and Seaborn module.

To perform any computations, Numpy will be needed. For data visualization, Matplotlib and seaborn will be used. Pandas is required for data manipulation, data loading, and analysis.

```
# Import Packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

II. Import Dataset:

- Using pd.read csv(), we load the data and set the column name to reflect the iris data information.
- Pd.read csv reads CSV(comma separated value) files.
- Only the top 5 rows of the data set table are displayed by df.head().

```
# Load the data
#IRIS_Dataset.csv
columns = ['Sepal_length', 'Sepal_width', 'Petal_length', 'Petal_width', 'Class_labels']
df = pd.read_csv('C:/Users/user/Downloads/iris.data', names=columns)
df.head()
```

	Sepal_length	Sepal_width	Petal_length	Petal_width	Class_labels
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Analyze & Data Visualization:

- Statistical Analysis
- Pair Plot
- Hear map
- Bar Plot

I. Statistical analysis:

With “df.describe()” function we get some numerical information like Total data-points count, mean value, standard deviation value, 50 percentile value etc. for each numeric feature in our dataset. This will help us to understand the some basic Statistic analysis of data.

```
#statistical analysis about the data
df.describe()
```

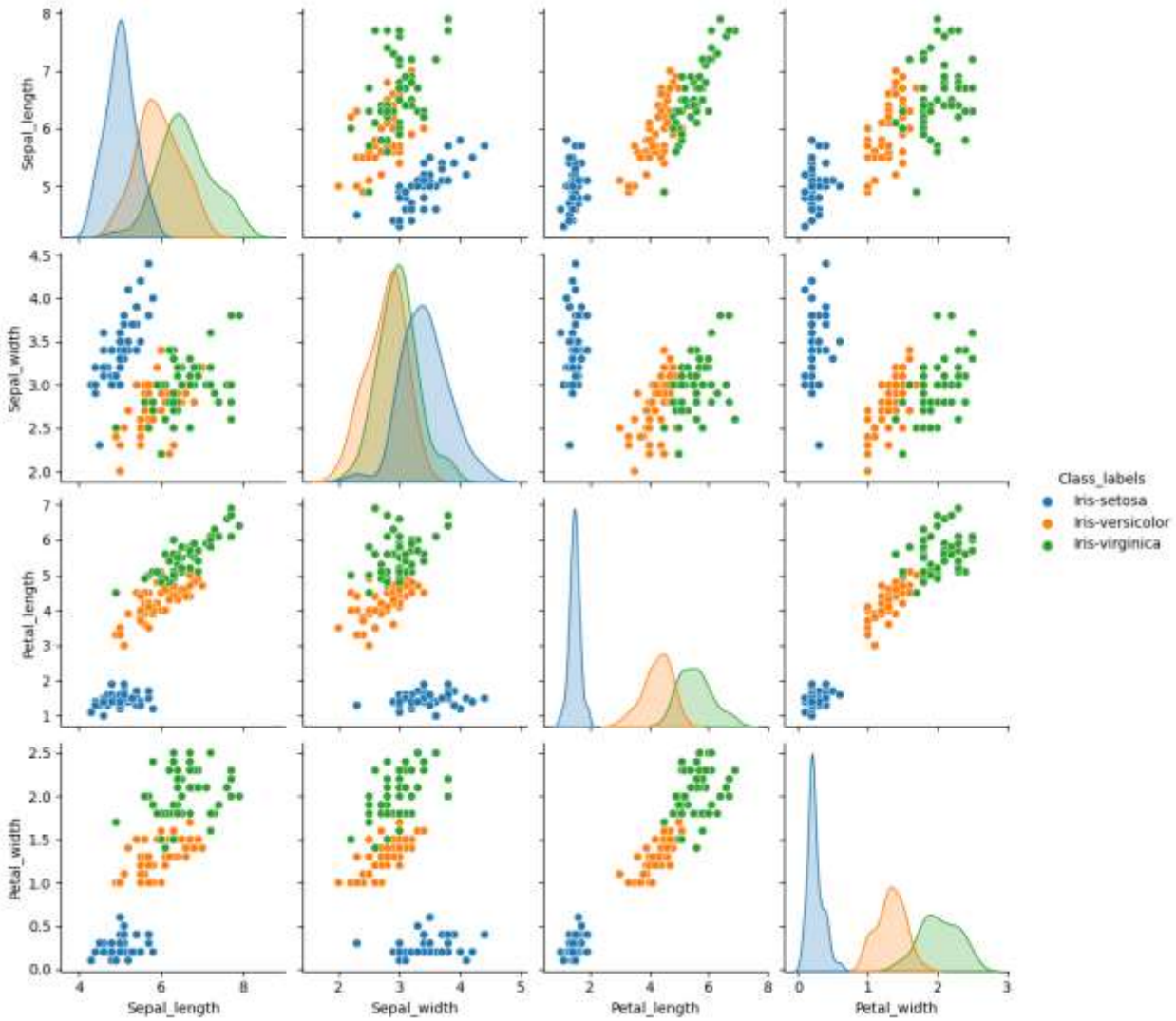
	Sepal_length	Sepal_width	Petal_length	Petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

II. Pair Plot:

By looking the result of pair plot us sure that all blue points are well separated with other two classes. But Versicolor and virginica are partially overlapping with each other.

In pair plot we saw that there are some feature combination which has very less overlapping between Versicolor and virginica, that's means those feature are very important for our classification task purpose.

```
#Visualize the whole dataset
sns.pairplot(df, hue='Class_labels')
plt.show()
```

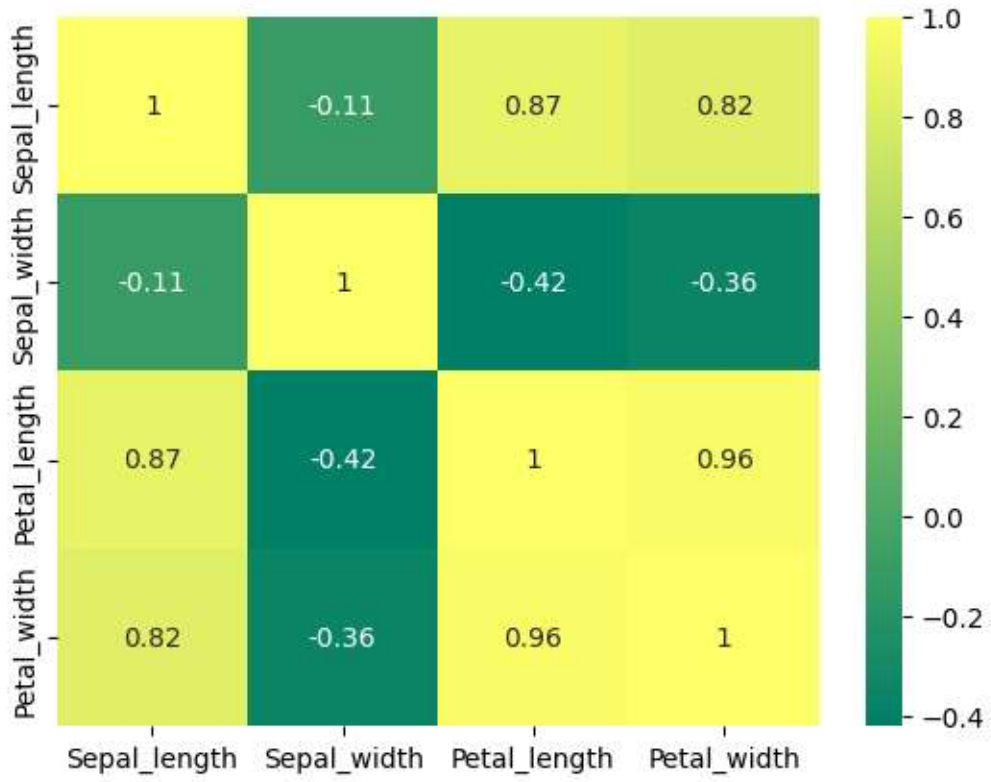


III. Heat Map:

Heat Map is used to find out the correlation between different features in the dataset. High positive or negative value shows that the features have high correlation. This helps us to select the parameters for machine learning.

```
#plotting heatmap
sns.heatmap(df.corr(),annot=True,cmap='summer')
plt.show()
```

There is a high correlation between:
Sepal Length & Petal Length, Sepal Length & Petal Width, and Petal Length & Petal Width.



IV. Bar Plot:

Now let's plot the average of each feature of each class. Here we separated the features from the target value.

Iris-data contain total 6 features in which 4 features (Sepal-length, Sepal-Width, Petal-Length, and Petal-width) are independent features and 1 feature (Species) is dependent or target variable. And Id column is like serial number for each data points. All Independent features has not-null float values and target variable has class labels (Iris-setosa, Iris-versicolor, Iris-virginica)

```
#Separating features from the target variable
data = df.values
X = data[:,0:4]
Y = data[:,4]
```

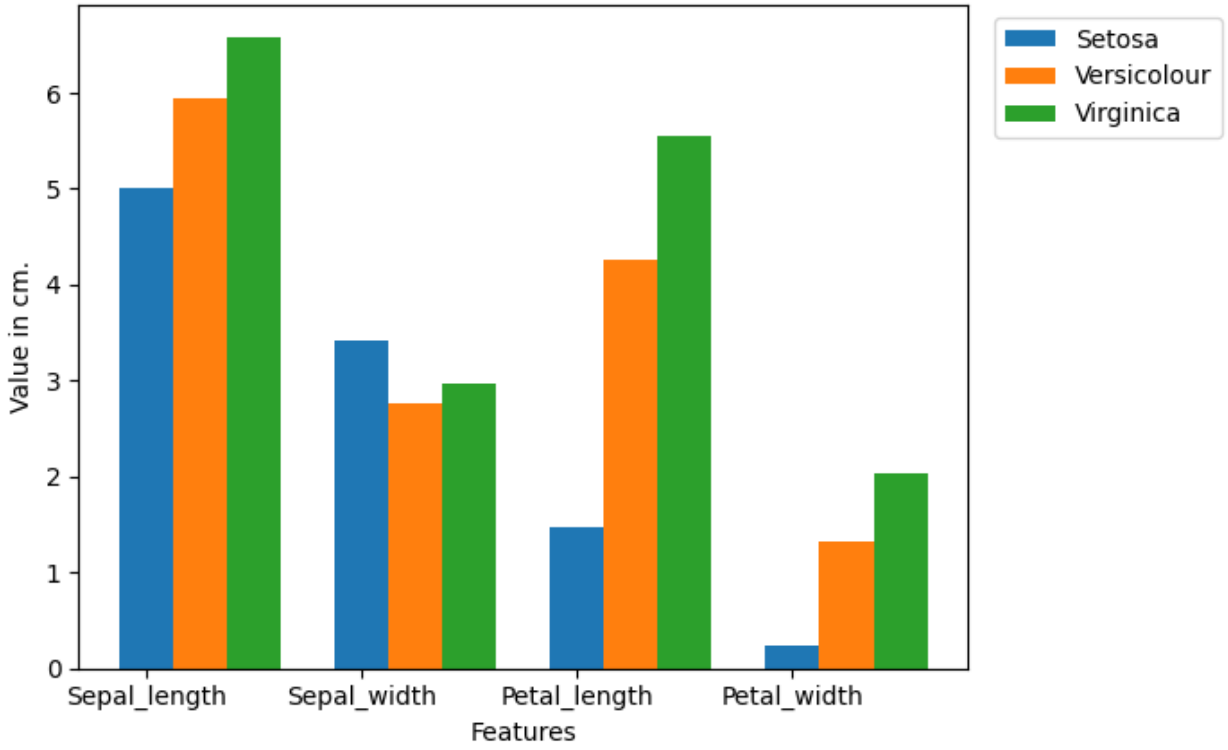
- Np.average calculates the average from an array.
- Here we used two for loops inside a list. This is known as list comprehension.
- List comprehension helps to reduce the number of lines of code.
- The Y_Data is a 1D array, but we have 4 features for every 3 classes. So we reshaped Y_Data to a (4, 3) shaped array.
- Then we change the axis of the reshaped matrix.

```
# Calculate average of each features for all classes
Y_Data = np.array([np.average(X[:, i][Y==j].astype('float32')) for i in range (X.shape[1])
 for j in (np.unique(Y))])
Y_Data_reshaped = Y_Data.reshape(4, 3)
Y_Data_reshaped = np.swapaxes(Y_Data_reshaped, 0, 1)
X_axis = np.arange(len(columns)-1)
width = 0.25
```

- To display the averages in a bar plot, we used matplotlib.

```
# Plot the average
plt.bar(X_axis, Y_Data_reshaped[0], width, label = 'Setosa')
plt.bar(X_axis+width, Y_Data_reshaped[1], width, label = 'Versicolour')
plt.bar(X_axis+width*2, Y_Data_reshaped[2], width, label = 'Virginica')
plt.xticks(X_axis, columns[:4])
plt.xlabel("Features")
plt.ylabel("Value in cm.")
plt.legend(bbox_to_anchor=(1.3,1))
plt.show()
```

- Here, it is obvious that the verginica flower is the longest and the setosa flower is the shortest.



Dividing the data for training and testing:

The training of a model using the methods may begin after we have a clear understanding of the dataset's contents. Some of the most popular machine learning algorithms will be put into practice in this part. Using a few of the data, let's begin by training our model. In order to divide our data set into a 60:40 ratio, we will be utilizing an internal library named "train test split."

```
# Split the data to train and test dataset.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.4)
```

Different Machine learning algorithms we used

We will be implementing these algorithms to compare:

1. Support Vector Machine (SVM)
2. k nearest neighbor (KNN)
3. Random Forest
4. Decision Trees
5. Gaussian Naive Bayes classifier
6. Logistic Regression

Let us start building our model and predicting accuracy of every algorithm used. For this purpose, we need to follow three steps in every machine learning algorithm we have used:

- Model training
- Model Evaluation.
- Testing the model.

1. Support Vector Machine (SVM):

Support vector machines are a group of supervised learning techniques for classifying data, doing regression analysis, and identifying outliers. Making a straight line between two classes is how a straightforward linear SVM classifier functions. In other words, the data points on one side of the line will all be assigned to one category, while the data points on the other side of the line will be assigned to a different category. This implies that the number of possible lines is unlimited.

Model training:
- Here we imported a support vector classifier from the scikit-learn support vector machine.
- Then, we created an object and named it svn.
- After that, we feed the training dataset into the algorithm by using the svn.fit() method.
- calculate model building time.

SVM

```
In [498]: # Support vector machine algorithm
          from sklearn.svm import SVC
          svn = SVC()
```

```
In [499]: #calculate model building time
          from time import time
          t0 = time()
          svn.fit(X_train, y_train)
          print ("training time:", round(time()-t0, 3), "s")

          training time: 0.014 s
```

Training time Output: 0.014 s.

Model Evaluation:
Now we predict the classes from the test dataset using our trained model. Then we check the accuracy score of the predicted classes. accuracy_score() takes true values and predicted values and returns the percentage of accuracy.

```
In [500]: # Predict from the test dataset
          predictions = svn.predict(X_test)
          # Calculate the accuracy
          from sklearn.metrics import accuracy_score
          accuracy_score(y_test, predictions)

Out[500]: 0.9166666666666666
```

Accuracy Output: 91.6 %

The classification report gives a detailed report of the prediction:
- Precision defines the ratio of true positives to the sum of true positive and false positives.
- Recall defines the ratio of true positive to the sum of true positive and false negative.
- F1-score is the mean of precision and recall value.
- Support is the number of actual occurrences of the class in the specified dataset.

```
In [501]: # A detailed classification report
          from sklearn.metrics import classification_report
          print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	17
Iris-versicolor	0.83	0.95	0.89	21
Iris-virginica	0.95	0.82	0.88	22
accuracy			0.92	60
macro avg	0.93	0.92	0.92	60
weighted avg	0.92	0.92	0.92	60

Testing the model:

Here we take some random values based on the average plot to see if the model can predict accurately.

```
X_new = np.array([[3, 2, 1, 0.2], [  4.9, 2.2, 3.8, 1.1 ], [  6.4, 2, 4.6, 4]])
#Prediction of the species from the input vector
prediction = svn.predict(X_new)
print("Prediction of Species: {}".format(prediction))
```

```
Prediction of Species: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

2. K nearest neighbor (KNN):

The supervised learning technique K-nearest neighbors (KNN) is used for both regression and classification. By computing the distance between the test data and all of the training points, KNN tries to predict the proper class for the test data. Then choose the K spots that are closest to the test data.
The KNN method determines which classes of the "K" training data the test data will belong to, and the class with the highest probability is chosen. The value in a regression situation is the average of the 'K' chosen training points.

Model training:

- Here we imported a K neighbor's classifier from the scikit-learn neighbours.
- Then, we created an object and named it knn.
- After that, we feed the training dataset into the algorithm by using the knn.fit() method.
- calculate model building time.

KNN

```
In [503]: from sklearn.neighbors import KNeighborsClassifier
          from sklearn import metrics
```

```
In [504]: knn = KNeighborsClassifier(n_neighbors=5)
          knn.fit(X_train, y_train)
```

```
Out[504]: KNeighborsClassifier
          KNeighborsClassifier()
```

```
In [505]: #calculate model building time
          from time import time
          t0 = time()
          knn.fit(X_train, y_train)
          print ("training time:", round(time()-t0, 3), "s")
```

```
training time: 0.001 s
```

Training time Output: 0.001 s.

Model Evaluation:

Now we predict the classes from the test dataset using our trained model. Then we check the accuracy score of the predicted classes. accuracy_score() takes true values and predicted values and returns the percentage of accuracy.

```
In [506]: # Predict from the test dataset
          y_pred = knn.predict(X_test)
          from sklearn.metrics import accuracy_score
          print(metrics.accuracy_score(y_test, y_pred))
```

```
0.9666666666666667
```

Accuracy Output: 96.67 %

The classification report gives a detailed report of the prediction:
- Precision defines the ratio of true positives to the sum of true positive and false positives.
- Recall defines the ratio of true positive to the sum of true positive and false negative.
- F1-score is the mean of precision and recall value.
- Support is the number of actual occurrences of the class in the specified dataset.

```
In [507]: # A detailed classification report
          from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

      Iris-setosa       1.00      1.00      1.00        17
  Iris-versicolor       0.91      1.00      0.95        21
   Iris-virginica       1.00      0.91      0.95        22

        accuracy                           0.97        60
       macro avg       0.97      0.97      0.97        60
    weighted avg       0.97      0.97      0.97        60
```

Testing the model:

Here we take some random values based on the average plot to see if the model can predict accurately.

```
X_new = np.array([[3, 2, 1, 0.2], [  4.9, 2.2, 3.8, 1.1 ], [  6.4, 2, 4.6, 4]])
#Prediction of the species from the input vector
prediction = svn.predict(X_new)
print("Prediction of Species: {}".format(prediction))
```

```
Prediction of Species: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

3. Random Forest:

Random forest is a supervised learning algorithm. The “forest” it builds is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. Random forest is a flexible, easy-to-use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most-used algorithms, due to its simplicity and diversity (it can be used for both classification and regression tasks).

Model training:
- Here we imported a Random forest classifier from the scikit-learn ensemble.
- Then, we created an object and named it clf.
- After that, we feed the training dataset into the algorithm by using the clf.fit() method.
- calculate model building time.

RANDOM FOREST

```
In [509]: from sklearn.ensemble import RandomForestClassifier
```

```
In [510]: #Create a Gaussian Classifier
          clf=RandomForestClassifier(n_estimators=100)
          #Train the model using the training sets
          clf.fit(X_train,y_train)
```

```
Out[510]: ▾ RandomForestClassifier
          RandomForestClassifier()
```

```
In [511]: #calculate model building time
          from time import time
          t0 = time()
          clf.fit(X_train, y_train)
          print ("training time:", round(time()-t0, 3), "s")
```

```
training time: 0.112 s
```

Training time Output: 0.112 s.

Model Evaluation:

Now we predict the classes from the test dataset using our trained model. Then we check the accuracy score of the predicted classes. accuracy_score() takes true values and predicted values and returns the percentage of accuracy.

```
In [512]: # prediction on test set
          y_pred=clf.predict(X_test)
          #Import scikit-learn metrics module for accuracy calculation
          from sklearn import metrics
          # Model Accuracy, how often is the classifier correct?
          print(accuracy_score(y_test, y_pred))
```

```
0.9166666666666666
```

Accuracy Output: 91.67 %

The classification report gives a detailed report of the prediction:

- Precision defines the ratio of true positives to the sum of true positive and false positives.
- Recall defines the ratio of true positive to the sum of true positive and false negative.
- F1-score is the mean of precision and recall value.
- Support is the number of actual occurrences of the class in the specified dataset

```
In [513]: # A detailed classification report
          from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

     Iris-setosa       1.00      1.00      1.00        17
 Iris-versicolor       0.83      0.95      0.89        21
  Iris-virginica       0.95      0.82      0.88        22

        accuracy                           0.92        60
       macro avg       0.93      0.92      0.92        60
    weighted avg       0.92      0.92      0.92        60
```

Testing the model:

Here we take some random values based on the average plot to see if the model can predict accurately.

```
In [514]: X_new = np.array([[3, 2, 1, 0.2], [  4.9, 2.2, 3.8, 1.1 ], [  6.4, 2, 4.6, 4]])
          #Prediction of the species from the input vector
          predictions = clf.predict(X_new)
          print("Prediction of Species: {}".format(predictions))
```

```
Prediction of Species: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

4. Decision Trees:

The Decision Tree algorithm is a member of the supervised learning algorithm family.
The decision tree technique, in contrast to other supervised learning methods, is capable of handling both classification and regression issues. By learning straightforward decision rules derived from previous data, a Decision Tree is used to build a training model that may be used to predict the class or value of the target variable (training data.) In decision trees, we begin at the tree's root when anticipating a record's class label. We contrast the root attribute's values with that of the attribute on the record. We follow the branch that corresponds to that value and go on to the next node based on the comparison.

Model training:
- Here we imported a Decision tree classifier from the scikit-learn tree.
- Then, we created an object and named it classifier.
- After that, we feed the training dataset into the algorithm by using the classifier.fit() method.
- calculate model building time.

DECISION TREE CLASSIFIER

In [515]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [516]:
```
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

In [517]:
```
#calculate model building time
from time import time
t0 = time()
classifier.fit(X_train, y_train)
print ("training time:", round(time()-t0, 3), "s")
```

```
training time: 0.001 s
```

Training time Output: 0.001 s.

Model Evaluation:

Now we predict the classes from the test dataset using our trained model. Then we check the accuracy score of the predicted classes. Accuracy_score() takes true values and predicted values and returns the percentage of accuracy.

In [518]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_pred))
```

```
0.9166666666666666
```

Accuracy Output: 91.67 %

The classification report gives a detailed report of the prediction:
- Precision defines the ratio of true positives to the sum of true positive and false positives.
- Recall defines the ratio of true positive to the sum of true positive and false negative.
- F1-score is the mean of precision and recall value.
- Support is the number of actual occurrences of the class in the specified dataset.

```
In [519]: # A detailed classification report
          from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

     Iris-setosa       1.00      1.00      1.00        17
 Iris-versicolor       0.83      0.95      0.89        21
  Iris-virginica       0.95      0.82      0.88        22

        accuracy                           0.92        60
       macro avg       0.93      0.92      0.92        60
    weighted avg       0.92      0.92      0.92        60
```

Testing the model:

Here we take some random values based on the average plot to see if the model can predict accurately.

```
In [520]: X_new = np.array([[3, 2, 1, 0.2], [  4.9, 2.2, 3.8, 1.1 ], [  6.4, 2, 4.6, 4]])
          #Prediction of the species from the input vector
          predictions = classifier.predict(X_new)
          print("Prediction of Species: {}".format(predictions))
```

```
Prediction of Species: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

5. Gaussian Naive Bayes classifier:

Gaussian Naive Bayes is a probabilistic classification algorithm based on applying Bayes' theorem with strong independence assumptions. In the context of classification, independence refers to the idea that the presence of one value of a feature does not influence the presence of another (unlike independence in probability theory). Naive refers to the use of an assumption that the features of an object are independent of one another. In the context of machine learning, naive Bayes classifiers are known to be highly expressive, scalable, and reasonably accurate, but their performance deteriorates rapidly with the growth of the training set. A number of features contribute to the success of naive Bayes classifiers. Most notably, they do not require any tuning of the parameters of the classification model, they scale well with the size of the training data set, and they can easily handle continuous features.

Model training:

- Here we imported Gaussian Naive Bayes from the scikit-learn naïve bayes.
- Then, we created an object and named it classifier.
- After that, we feed the training dataset into the algorithm by using the classifier.fit() method.
- calculate model building time.

Naive Bayes classifier.

```
In [521]: from sklearn.naive_bayes import GaussianNB
```

```
In [522]: classifier = GaussianNB()
          classifier.fit(X_train, y_train)
          y_pred = classifier.predict(X_test)
```

```
In [523]: #calculate model building time
          from time import time
          t0 = time()
          classifier.fit(X_train, y_train)
          print ("training time:", round(time()-t0, 3), "s")

          training time: 0.003 s
```

Training time Output: 0.003 s.

Model Evaluation:

Now we predict the classes from the test dataset using our trained model. Then we check the accuracy score of the predicted classes. Accuracy_score() takes true values and predicted values and returns the percentage of accuracy.

```
In [524]: from sklearn.metrics import accuracy_score
          print(accuracy_score(y_pred,y_test))

          0.9333333333333333
```

Accuracy Output: 93.33 %

The classification report gives a detailed report of the prediction:
- Precision defines the ratio of true positives to the sum of true positive and false positives.
- Recall defines the ratio of true positive to the sum of true positive and false negative.
- F1-score is the mean of precision and recall value.
- Support is the number of actual occurrences of the class in the specified dataset.

```
In [525]: # A detailed classification report
          from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        17
Iris-versicolor       0.87      0.95      0.91        21
 Iris-virginica       0.95      0.86      0.90        22

       accuracy                           0.93        60
      macro avg       0.94      0.94      0.94        60
   weighted avg       0.94      0.93      0.93        60
```

Testing the model:

Here we take some random values based on the average plot to see if the model can predict accurately.

```
In [520]: X_new = np.array([[3, 2, 1, 0.2], [  4.9, 2.2, 3.8, 1.1 ], [  6.4, 2, 4.6, 4]])
          #Prediction of the species from the input vector
          predictions = classifier.predict(X_new)
          print("Prediction of Species: {}".format(predictions))
```

```
Prediction of Species: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

6. Logistic Regression:

Logistic Regression is a supervised machine learning model used mainly for categorical data, and it is a classification algorithm. Seeing the name logistic regression, you may think it will be a regression algorithm. But the fact is that it is a classification algorithm, and it is a generalization of the linear regression model.

Logistic Regression is used to find the relationship between dependent and independent variables. This is done by using a logistic regression equation. This is a very easy to implement, understand, and also easy method to train the model.

Model training:

- Here we imported Gaussian Naive Bayes from the scikit-learn naïve bayes.
- Then, we created an object and named it logisticReg.
- After that, we feed the training dataset into the algorithm by using the logisticReg.fit() method.
- calculate model building time.

Logistic Regression

```
In [528]: from sklearn.linear_model import LogisticRegression

In [534]: logisticRegr = LogisticRegression(max_iter=1000)
          logisticRegr.fit(X_train, y_train)

Out[534]: LogisticRegression
          LogisticRegression(max_iter=1000)

In [530]: #calculate model building time
          from time import time
          t0 = time()
          logisticRegr.fit(X_train, y_train)
          print ("training time:", round(time()-t0, 3), "s")

          training time: 0.014 s
```

Model Evaluation:

Now we predict the classes from the test dataset using our trained model. Then we check the accuracy score of the predicted classes. Accuracy_score() takes true values and predicted values and returns the percentage of accuracy.

```
In [524]: from sklearn.metrics import accuracy_score
          print(accuracy_score(y_pred,y_test))

          0.9333333333333333
```

Accuracy Output: 93.33 %

The classification report gives a detailed report of the prediction:

- Precision defines the ratio of true positives to the sum of true positive and false positives.
- Recall defines the ratio of true positive to the sum of true positive and false negative.
- F1-score is the mean of precision and recall value.
- Support is the number of actual occurrences of the class in the specified dataset.

In [532]:
```
# A detailed classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        17
Iris-versicolor       0.87      0.95      0.91        21
 Iris-virginica       0.95      0.86      0.90        22

       accuracy                           0.93        60
      macro avg       0.94      0.94      0.94        60
   weighted avg       0.94      0.93      0.93        60
```

Testing the model:

Here we take some random values based on the average plot to see if the model can predict accurately.

In [533]:
```
X_new = np.array([[3, 2, 1, 0.2], [  4.9, 2.2, 3.8, 1.1 ], [  6.4, 2, 4.6, 4]])
#Prediction of the species from the input vector
y_pred = logisticRegr.predict(X_new)
print("Prediction of Species: {}".format(y_pred))
```

```
Prediction of Species: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

Comparison Graph between Different Models