Name : Susmita Rani Saha , ID : B180305047

Name : Tanvir Ahammed Hridoy , ID : B180305020

Classification Implementation :

Step 1 : importing all necessary libraries

First of all we have to import all necessary libraries

In [25]:
```
import pandas as pd
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

Step 2 : loading the CSV file here.

Then load the csv file,so that we get a look at how to load and preprocess data. Now we just put the data file in the same directory as our Python file.The Pandas library has an easy way to load in data, read_csv():

In [26]:
```
data = pd.read_csv('iris.csv')

# It is a good idea to check and make sure the data is loaded as expected.
print(data.head(5))
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

Step 3 : Preprocessing data

Because the dataset has been prepared so well, we don't need to do a lot of preprocessing. Only we drop the "ID" column. As this isn't helpful we could drop it from the dataset using the drop() function:

In [27]:
```
data.drop('Id', axis=1, inplace=True)
```

Step 4 : Define features and labels

We now need to define the features and labels. We can do this easily with Pandas by slicing the data table and choosing certain rows/columns with iloc(). The slicing notation selects every row and every column except the last column (which is our label, the species).

In [28]:
```
# Pandas ".iloc" expects row_indexer, column_indexer
X = data.iloc[:,:-1].values

# Now let's tell the dataframe which column we want for the target/labels.
y = data['Species']
```

Step 5 : Split into train and test sets

Now that we have the features and labels we want, we can split the data into training and testing sets using sklearn's handy feature train_test_split():

In [18]:
```
# Test size specifies how much of the data you want to set aside for the testi
# Random_state parameter is just a random seed we can use.
# You can use it if you'd like to reproduce these specific results.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, rand
```

Step 6 : Print train sets

We can print the results to be sure our data is being parsed as we expect:

In [19]:
```
print(X_train)
print(y_train)
```

```
[[6.3 2.7 4.9 1.8]
 [5.  3.5 1.3 0.3]
 [6.4 2.7 5.3 1.9]
 [6.2 2.9 4.3 1.3]
 [6.7 3.1 4.7 1.5]
 [4.4 3.  1.3 0.2]
 [6.8 2.8 4.8 1.4]
 [6.3 2.5 5.  1.9]
 [5.8 2.7 3.9 1.2]
 [4.8 3.1 1.6 0.2]
 [4.6 3.4 1.4 0.3]
 [6.3 2.9 5.6 1.8]
 [5.9 3.2 4.8 1.8]
 [7.2 3.2 6.  1.8]
 [4.6 3.6 1.  0.2]
 [6.  2.9 4.5 1.5]
 [6.8 3.  5.5 2.1]
 [7.4 2.8 6.1 1.9]
 [6.1 2.9 4.7 1.4]
```

Step 7 : instantiate the models

Now we can instantiate the models. Follow this steps : First fit the classifiers,this train the model. Then predict and store the prediction in a variable, And print accuracy score, classification report and confusion matrix.

i. Support Vector Machine (SVM)

Support Vector Machines work by drawing a line between the different clusters of data points to group them into classes. Points on one side of the line will be one class and points on the other side belong to another class.

The classifier will try to maximize the distance between the line it draws and the points on either side of it, to increase its confidence in which points belong to which class. When the testing points are plotted, the side of the line they fall on is the class they are put in.

In [24]:
```
# Support Vector Machine (SVM)
SVC_model = SVC()
SVC_model.fit(X_train, y_train)
SVC_prediction = SVC_model.predict(X_test)
print(accuracy_score(SVC_prediction, y_test))
print(classification_report(y_test, y_test))
print(confusion_matrix(SVC_prediction, y_test))
```

```
0.9333333333333333
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      1.00      1.00        11
 Iris-virginica       1.00      1.00      1.00        12

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30

[[ 7  0  0]
 [ 0 10  1]
 [ 0  1 11]]
```

ii. Logistic Regression

Logistic Regression outputs predictions about test data points on a binary scale, zero or one. If the value of something is 0.5 or above, it is classified as belonging to class 1, while below 0.5 if is classified as belonging to 0.

Each of the features also has a label of only 0 or 1. Logistic regression is a linear classifier and therefore used when there is some sort of linear relationship between the data.

In [29]:
```
# Logistic Regression
logreg_clf = LogisticRegression()
logreg_clf.fit(X_train, y_train)
logreg_prediction = logreg_clf.predict(X_test)
print(accuracy_score(logreg_prediction, y_test))
print(classification_report(y_test, y_test))
print(confusion_matrix(logreg_prediction, y_test))
```

```
0.9333333333333333
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      1.00      1.00        11
 Iris-virginica       1.00      1.00      1.00        12

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30

[[ 7  0  0]
 [ 0 10  1]
 [ 0  1 11]]

E:\ana\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWa
rning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

iii. K-Nearest Neighbors

K-Nearest Neighbors operates by checking the distance from some test example to the known values of some training example. The group of data points/class that would give the smallest distance between the training points and the testing point is the class that is selected.

In [30]:
```
# K-Nearest Neighbors
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
knn_y_pred = knn_model.predict(X_test)
print(accuracy_score(knn_y_pred, y_test))
print(classification_report(y_test, y_test))
print(confusion_matrix(knn_y_pred, y_test))
```

```
0.9666666666666667
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      1.00      1.00        11
 Iris-virginica       1.00      1.00      1.00        12

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30

[[ 7  0  0]
 [ 0 11  1]
 [ 0  0 11]]
```

```
E:\ana\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarn
ing: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default
behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.
0, this behavior will change: the default value of `keepdims` will become Fal
se, the `axis` over which the statistic is taken will be eliminated, and the
value None will no longer be accepted. Set `keepdims` to True or False to avo
id this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

iv. Decision Tree

A Decision Tree Classifier functions by breaking down a dataset into smaller and smaller subsets based on different criteria. Different sorting criteria will be used to divide the dataset, with the number of examples getting smaller with every division.

Once the network has divided the data down to one example, the example will be put into a class that corresponds to a key.

In [31]:
```
# Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
dt_y_pred = dt_model.predict(X_test)
print(accuracy_score(dt_y_pred, y_test))
print(classification_report(y_test, y_test))
print(confusion_matrix(dt_y_pred, y_test))
```

```
0.9
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      1.00      1.00        11
 Iris-virginica       1.00      1.00      1.00        12

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30

[[ 7  0  0]
 [ 0 10  2]
 [ 0  1 10]]
```