CLASSIFYING IRIS SPECIES

Aim:

To build a machine learning model that can predict different species of IRIS.

Theory:

The Iris dataset is a standard and commonly used dataset in machine learning and statistics. It includes 150 samples of iris blossoms from three distinct species: Setosa, Versicolor, and Virginica. Each sample has measurements(features) for

1. sepal length
2. sepal width
3. petal length
4. petal width

Ronald Fisher generated the dataset in 1936, and it has since become a benchmark for evaluating classification techniques. It is frequently used to demonstrate the usefulness of algorithms in solving classification issues and comparing their performance.

The Iris dataset is regarded as an attractive dataset for machine learning beginners due to its simplicity and well-defined structure. It is relatively small and contains simple numerical features. The dataset is extremely adaptable, as it displays distinct variations across its classes, allowing for the use of various classification algorithms such as logistic regression, decision trees, support vector machines, and more.

Main features of the Iris dataset:

- It includes 150 samples of iris blossoms from three distinct species: Setosa, Versicolor, and Virginica.
- Each sample comprises four characteristics: sepal length, sepal width, petal length, and petal width.
- The dataset was created by Ronald Fisher in 1936 and has since become a benchmark for testing classification algorithms.
- It is regarded as a great dataset for novices in machine learning because of its simplicity and well-defined structure.
- The dataset is rather modest and contains unambiguous, numerical elements that are easily understood.
- One class (Iris Setosa) is linearly separable from the other two classes, which are not linearly separate from each other.
- The dataset is adaptable, as it displays distinct variances across its classes, allowing the use of multiple classification algorithms.

- Users can simply access the dataset because it is provided in both the R base and Python's machine learning module scikit-learn.

Fig.1 IRIS Setosa

Fig.2 IRIS Versicolor

Fig. IRIS Verginica

Libraries Used:

1.Scikit-learn: Scikit-learn is a versatile machine learning library that provides simple and efficient tools for data mining and data analysis. It includes a wide range of algorithms for supervised and unsupervised learning, including classification, regression, clustering, dimensionality reduction, and model selection. Scikit-learn is designed to be easy to use and integrates well with other Python libraries

2.NumPy: NumPy is a fundamental library for scientific computing in Python. It provides support for multidimensional arrays, along with a collection of mathematical functions to operate on these

arrays efficiently. NumPy is the foundation for many other libraries in the Python ecosystem, including Pandas and SciPy.

STEPS INVOLVED:

1. data selection
2. Data splitting
3. Algorithm selection and model fitting
4. Making Predictions
5. Evaluating Model
6. Finding accuracy using score method

PROGRAM:

1.Data selection

```
from sklearn.datasets import load_iris
```

```
import numpy as np
```

The code in the image imports two libraries: `sklearn.datasets` and `numPy`.

- from sklearn.datasets import load_iris :Imports the load_iris function from the sklearn.datasets library. This tool imports the Iris dataset, a classic dataset for machine learning classification tasks. The Iris dataset includes 150 samples of iris blooms, each having four attributes.
- import numpy as np : Imports the numpy library and assigns the alias np. Numpy is a Python library for numerical computing that includes a number of routines for dealing with arrays and matrices.

```
iris=load_iris()
print(iris.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
print(iris['data'][:5])
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

```
print(iris['target'][:10])
```

```
[0 0 0 0 0 0 0 0 0 0]
```

```
print(iris['target_names'][:5])
```

```
['setosa' 'versicolor' 'virginica']
```

```
print(iris['feature_names'][:5])
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

- iris=load_iris() : This line loads the Iris dataset from the sklearn.datasets module and assigns it to the variable 'iris'.
- print(iris.keys()) : This line prints the keys (or attributes) of the iris dataset. The 'Iris' dataset is a structured dataset, and the 'keys()' method is used to retrieve the names of the attributes (or columns) in the dataset.
- print(iris['data'][:5]) : This line prints the first 5 rows of the 'data' attribute of the 'iris, dataset. The 'data' attribute contains the feature (or input) data, which are the measurements of the iris flowers.
- print(iris['target'][:10]) : This line prints the first 10 elements of the 'target' attribute of the 'iris' dataset. The 'target' attribute contains the target (or output) data, which are the class labels (0, 1, or 2) for the different species of iris flowers.
- print(iris['target_names'][:5]) :This line prints the first 5 elements of the 'target_names' attribute of the 'iris' dataset. The 'target_names' attribute contains the names of the three iris flower species (setosa, versicolor, and virginica).
- print(iris['feature_names'][:5]) : This line prints the first 5 elements of the 'feature_names' attribute of the 'iris' dataset. The 'feature_names' attribute contains the names of the features (or input variables) in the dataset, which are the measurements of the iris flowers.

2. Data splitting

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split((iris.data),(iris.target),random_state=0)
print(X_train.shape)
print(y_train.shape)
```

```
(112, 4)
(112,)
```

- from sklearn.model_selection import train_test_split :This line imports the 'train_test_split' function from the sklearn.model_selection module. This function is used to split the dataset into training and testing sets, which is a common step in machine learning workflows.
- X_train, X_test, y_train, y_test = train_test_split((iris.data), (iris.target), random_state=0) : This line splits the Iris dataset into training and testing sets.
- print(X_train.shape) ; print(y_train.shape) : These lines print the shapes of the training feature data (X_train) and the training target data (y_train).

3. **Algorithm selection and model fitting**

```
#Algorithm selection and model fitting
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
KNeighborsClassifier(n_neighbors=1)
```

- from sklearn.neighbors import KNeighborsClassifier :This line imports the 'KNeighborsClassifier' class from the 'sklearn.neighbors' module. This class is used to implement the K-Nearest Neighbors (KNN) algorithm, which is a popular machine learning algorithm for classification tasks.
- knn = KNeighborsClassifier(n_neighbors=1) : This line creates an instance of the 'KNeighborsClassifier' class and assigns it to the variable 'knn'. The 'n_neighbors=1' parameter specifies that the KNN algorithm will use 1 nearest neighbor to make predictions.
- knn.fit(X_train, y_train) : This line trains the KNN classifier on the training data.
- KNeighborsClassifier(n_neighbors=1): The KNN algorithm works by finding the 'k' nearest neighbors of a new data point in the feature space and then assigning the new data point to the class that is most common among its 'k' nearest neighbors. In this case, we've set 'k=1', which means that the algorithm will assign the new data point to the class of its single nearest neighbor.

4.Making Predictions

```
# Making Predictions
X_new=np.array([[5,2.9,2,1],[3,1,3,4]])
print(X_new)
prediction = knn.predict(X_new)
print("X_new.shape:{}".format(X_new.shape))
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(iris['target_names'][prediction]))
```

- X_new = np.array([[5, 2.9, 2, 1], [3, 1, 3, 4]]) ; print(X_new) : This code creates a new 2D NumPy array called 'X_new' with two rows and four columns. Each row represents a new sample of iris flower measurements that we want to classify.
- prediction = knn.predict(X_new) , print("X_new.shape:{}".format(X_new.shape)) : This code uses the trained KNN classifier (knn) to make predictions on the new sample data (X_new). The predict() method returns an array of predicted class labels for each sample in 'X_new'.
- print("Prediction: {}".format(prediction)) ;
  print("Predicted target name: {}".format(iris['target_names'][prediction])):
  These lines print the predicted class labels and the corresponding target names for the new sample data.

5. Evaluating Model

```
# Evaluating Model
# here_test set iscompared against labels.we can measure ..
# how well the model works by computing the accuracy.
y_predict=knn.predict(X_test)
print("test test predictions:{}".format(y_predict))
```

- y_predict = knn.predict(X_test) : This line uses the trained KNN classifier (knn) to make predictions on the testing data (X_test). The 'predict()' method returns an array of predicted class labels for each sample in 'X_test'.
- **print**("test test predictions:{}".format(y_predict)) : This line prints the predicted class labels for the testing data.

7. **Finding accuracy using score method**

```
# finding accuracy using score method
accuracy=knn.score(X_test,y_test)
print("Accuracy of KNN model built for Iris dataset:{:.2f}".format(accuracy))
```

```
Accuracy of KNN model built for Iris dataset:0.97
```

```
print("Accuracy of KNN model built for Iris dataset: {}".format(accuracy))
```

```
Accuracy of KNN model built for Iris dataset: 0.9736842105263158
```

- accuracy = knn.score(X_test, y_test) :This line evaluates the accuracy of the 'KNN classifier' on the testing data. The 'score()' method of the 'KNeighborsClassifier' class calculates the accuracy of the model's predictions on the provided data.The 'X_test' and 'y_test' arguments are the feature data and target data for the testing set, respectively. The 'score()' method compares the

predicted class labels (obtained from the ‘predict()’ method) with the actual class labels (y_test) and calculates the percentage of correct predictions.

- print("Accuracy of KNN model built for Iris dataset: {:.2f}".format(accuracy)) print("Accuracy of KNN model built for Iris dataset: {}".format(accuracy)): These lines print the accuracy of the KNN classifier on the Iris dataset. The first line formats the accuracy to 2 decimal places, while the second line prints the accuracy without any formatting.

Conclusion:

A machine learning model to predict different species of IRIS was developed using KNN classifier .The KNN classifier achieved an accuracy of 97.37% on the Iris dataset, indicating that it is a highly effective model for classifying the different species of iris flowers based on their sepal and petal measurements.