Experiment 5

Objective:

To implement data classification using KNN.

Theory

The K-nearest neighbors (KNN) algorithm is a simple, yet powerful, supervised machine learning method used for classification and regression tasks. It is based on the principle of similarity between data points, where the classification of a given data point is determined by its closest neighbors in the feature space. KNN is non-parametric, meaning it does not assume a specific form for the underlying data distribution, making it versatile across various types of datasets.

The KNN Algorithm for classification operates as follows:

Choose the Value of K: K represents the number of nearest neighbors to consider when classifying a new data point. A smaller K can lead to a more sensitive model, but may also be more prone to noise, while a larger K may generalize better but could blur class boundaries. Calculate Distance: For each new data point, KNN calculates the distance between this point and all points in the training data. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance. Euclidean distance is often used as it provides a straightforward measure of similarity.

Identify Nearest Neighbors: The algorithm then selects the K data points closest to the new point.Determine the Class: The new data point is assigned the class label that is most frequent among its K nearest neighbors. For instance, if K=5 and three of the neighbors belong to Class A while two belong to Class B, the new point will be classified as Class A.

Choosing the Value of K is crucial to KNN's performance. Common methods include cross-validation, where various K values are tested to determine which yields the best model accuracy. Low values of K may lead to overfitting, where the model is too sensitive to noise. Higher values of K provide smoother decision boundaries but may miss finer patterns.

Advantages and Disadvantages: KNN is easy to understand and implement, making it popular for beginners in machine learning. It performs well on smaller datasets where the data points are easily separable in the feature space. However, it has limitations: it is computationally intensive for large datasets, as it requires calculating distances for all training points for each new prediction. KNN is also sensitive to feature scaling; therefore, normalization or standardization of features is essential to prevent features with large ranges from dominating the distance calculations.

Implementing KNN: In Python, the scikit-learn library provides a simple implementation of KNN through KNeighborsClassifier. Users can set the K value, distance metric, and weighting options. We can evaluate KNN models using performance
Overall, KNN is an effective algorithm for classification tasks where data points exhibit clear clusters or similarities. It is widely used for its simplicity and intuitive approach, providing meaningful insights, especially in tasks where interpretability is key.

Code & OUTPUT

In [2]:
```
print("Experiment No 05 : To implement data classification using KNN.")
```

```
Experiment No 05 : To implement data classification using KNN.
```

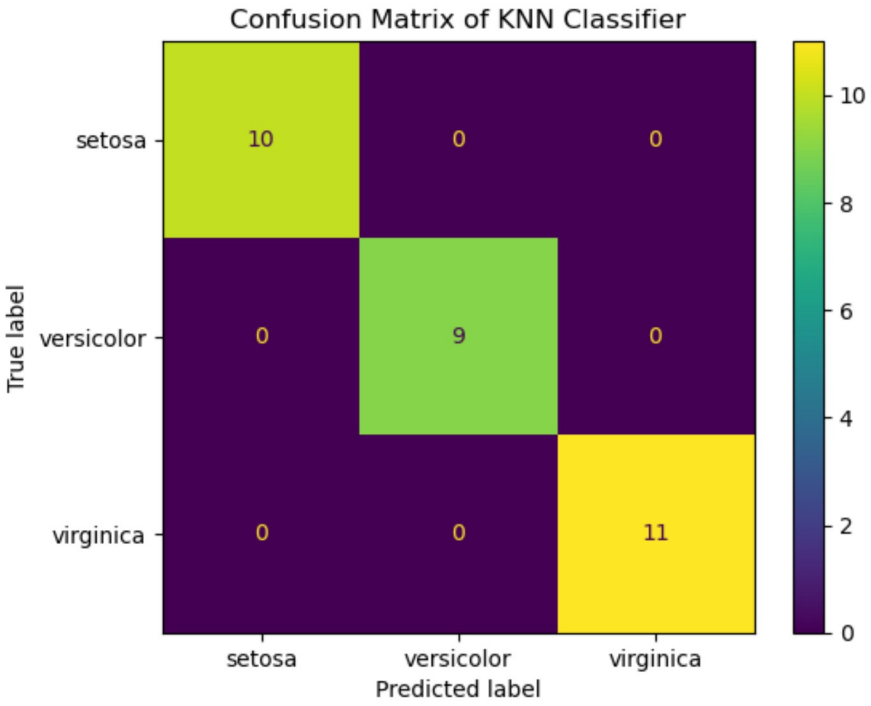
In [4]:
```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

print("OUTPUT:\n\n")

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (sepal length, sepal width, petal length, petal width)
y = iris.target  # Target (species)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the K-Nearest Neighbors classifier with k=3
knn = KNeighborsClassifier(n_neighbors=3)

# Train the model
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display a detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Plotting the Confusion Matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, display_labels=iris.target_names)
plt.title("Confusion Matrix of KNN Classifier")
plt.show()
```

OUTPUT:

Accuracy: 1.00

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30


Confusion Matrix of KNN Classifier
setosa
versicolor
virginica
True label
10
0
0
0
9
0
0
0
11
setosa
versicolor
virginica
Predicted label
10
8
6
4
2
0

In [ ]:

Result

As a result of this Experiment, we successfully wrote and executed the program to implement data classification using KNN.

Learning Outcomes

Understand and apply the K-means clustering algorithm to group data points into clusters, selecting optimal k and interpreting results effectively.