Experiment 3

Aim : Study and Implement K Nearest Neighbours (KNN).

Theory : K-Nearest Neighbors (KNN) is a simple, intuitive, and widely used machine learning algorithm, especially in classification and regression tasks. It is a type of instance-based learning or lazy learning, where no explicit model is trained during the training phase, and predictions are made based on stored training data**.**

Working Principle of KNN :

KNN works by storing the entire training dataset and making predictions for new data points by finding the most similar data points (neighbors) in the training set. The steps involved in KNN are:

- **Step 1: Choose the number of nearest neighbors (k).**
- **Step 2: Compute the distance** between the new data point and all the points in the training dataset using a distance metric.
- **Step 3: Sort the distances** in increasing order and select the `k` closest data points (neighbors).
- **Step 4: Classification (for KNN Classification)**: Take a **majority vote** among the `k` nearest neighbors. The most frequent class among the neighbors is the predicted class for the new data point.
- **Step 5: Regression (for KNN Regression)**: Take the **average** (or weighted average) of the values of the `k` nearest neighbors to predict the continuous output value.

Choosing the Value of k :

- **Small k**: If `k` is too small (e.g., `k = 1`), the algorithm becomes highly sensitive to noise and outliers, as the prediction is based only on the closest neighbor.

- **Large k**: If `k` is too large, KNN may misclassify points, as it will consider neighbors that are far away, thus diluting the influence of closer, more relevant neighbors.

The value of `k` is a hyperparameter and is often chosen through techniques like **cross-validation**.

Distance Metrics in KNN :

Euclidean Distance: The most commonly used distance metric, suitable for continuous variables. It calculates the straight-line distance between two points in Euclidean space:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Code with Output :

```
[61] # Import necessary libraries
     import numpy as np
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[62] data = pd.read_csv('/IRIS.csv')
     data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
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

```
X = data.drop('species', axis=1)
y = data['species']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier()
```

```
y_pred=knn.predict(X_test)
```

```
accuracy=accuracy_score(y_test,y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 100.00%
```

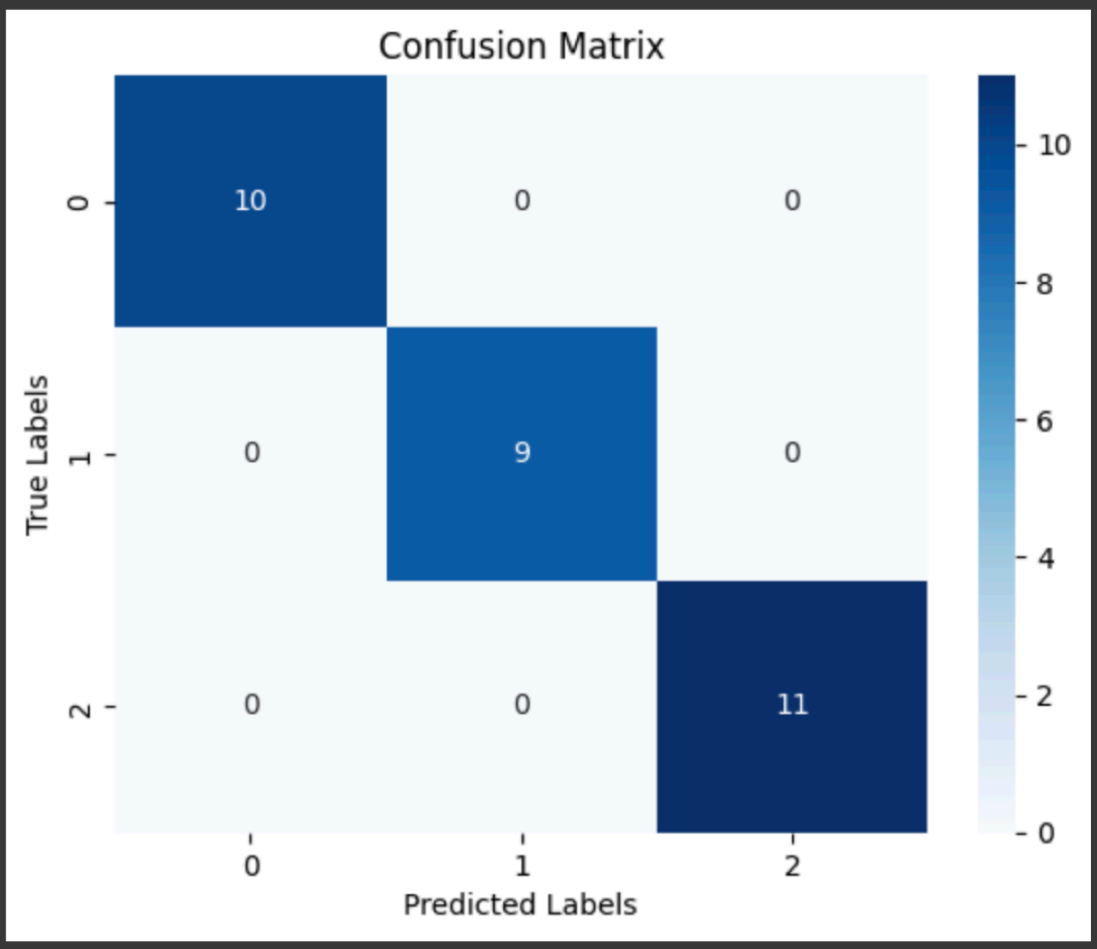
```
[77] cm=confusion_matrix(y_test,y_pred)
     print(cm)
```

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
[80] print(classification_report(y_test, y_pred))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-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
```

```
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



Learning Outcome :

Experiment 4

Aim : Study and Implement classification using SVM.

Theory : Support Vector Machine (SVM) is a powerful supervised learning algorithm used for both classification and regressiontasks, although it is more commonly used for classification. SVM works by finding the optimal hyperplane that best separates data points of different classes in the feature space.

Key Concepts of SVM :

1. **Hyperplane**: In SVM, a hyperplane is a decision boundary that separates different classes in the feature space. The optimal hyperplane is the one that maximizes the margin between the classes.

2. **Margin**: The margin is the distance between the hyperplane and the closest data points from both classes, known as support vectors. The goal of SVM is to maximize this margin.

3. **Support Vectors**: These are the data points that lie closest to the hyperplane and are most difficult to classify. They are critical in defining the position and orientation of the hyperplane.

4. **Kernels**: Kernels allow SVM to handle non-linear classification problems by transforming the original input space into a higher-dimensional space where the data becomes linearly separable.

5. **Linear and Non-Linear SVM:**

- **Linear SVM**: Used when the data is linearly separable, meaning it can be separated by a straight line (or hyperplane in higher dimensions).
- **Non-Linear SVM**: Used when the data is not linearly separable. SVM uses the **kernel trick** to project the data into a higher-dimensional space where it can be linearly separated.

Code with Output :

```
[1] import numpy as np
    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.svm import SVC
    from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
    import matplotlib.pyplot as plt
    import seaborn as sns
```

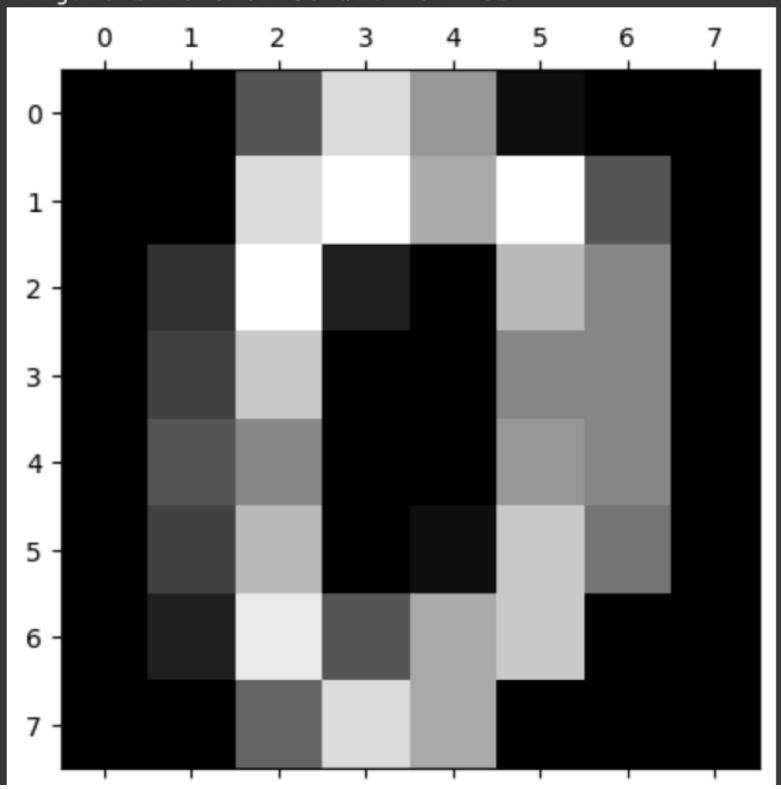
```
[5] from sklearn.datasets import load_digits
    digits = load_digits()
```

```
[3] dir(digits)
```

```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

```
[45] plt.gray()
     for i in range(2):
         plt.matshow(digits.images[i])
```

```
<Figure size 640x480 with 0 Axes>
```



```
[46] df=pd.DataFrame(digits.data)
     df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0

5 rows × 64 columns

```
[47] X=digits.data
     y=digits.target
     print(X)
     print(y)
```

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
[0 1 2 ... 8 9 8]
```

```
[48] X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```

```
[49] # Train the SVM model with RBF kernel
     svm_classifier = SVC(kernel='rbf', random_state=30)
     svm_classifier.fit(X_train, y_train)
```

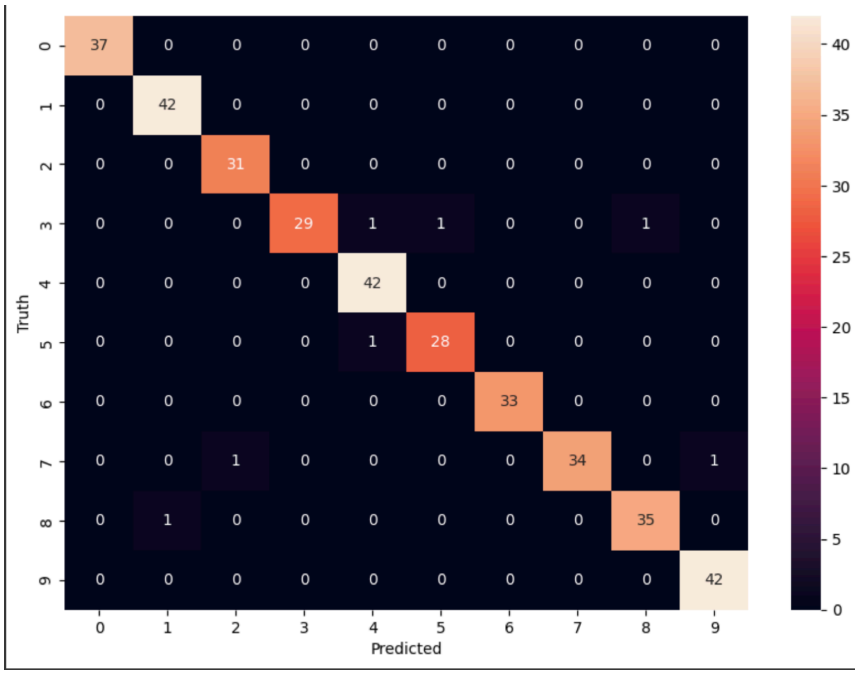
```
SVC
SVC(random_state=30)
```

```
[50] y_pred = svm_classifier.predict(X_test)
     accuracy = accuracy_score(y_test, y_pred)
     print("Accuracy:", accuracy)

     Accuracy: 0.9805555555555555
```

```
[38] Suggested code may be subject to a license | Ishpreet21698/H
     cm = confusion_matrix(y_test, y_pred)
     print("Confusion Matrix:")
     print(cm)

     Confusion Matrix:
     [[37  0  0  0  0  0  0  0  0  0]
      [ 0 42  0  0  0  0  0  0  0  0]
      [ 0  0 31  0  0  0  0  0  0  0]
      [ 0  0  0 29  1  1  0  0  1  0]
      [ 0  0  0  0 42  0  0  0  0  0]
      [ 0  0  0  0  1 28  0  0  0  0]
      [ 0  0  0  0  0  0 33  0  0  0]
      [ 0  0  1  0  0  0  0 34  0  1]
      [ 0  1  0  0  0  0  0  0 35  0]
      [ 0  0  0  0  0  0  0  0  0 42]]
```

```
[40] model.score(X_test,y_test)

     0.9638888888888889

[ ]  y_predicted=model.predict(X_test)
```

```
[41] plt.figure(figsize=(10,7))
     sns.heatmap(cm,annot=True)
     plt.xlabel('Predicted')
     plt.ylabel('Truth')
     plt.show()
```



Learning Outcome :