EXPERIMENT 8

Student Name: Samuel **UID: 24MAI10018**
Branch: CSE-AIML **Section/Group: 24MAI-1**
Semester: 2 **Date of Performance:**
Subject Name: Machine Learning Lab **Subject Code: 24CSH–651**

AIM:

Implementing K- nearest Neighbours algorithm using Python.

SOFTWARE REQUIREMENTS:

- Python IDE (e.g., Jupyter Notebook, PyCharm, etc.)
- NumPy Library.
- Pandas Library.
- Scikit-Learn Library.
- Matplotlib & Seaborn Libraries.

THEORY:

K-Nearest Neighbors (K-NN) Algorithm: K-Nearest Neighbors (K-NN) is a supervised learning algorithm used for classification and regression. It classifies a new data point based on the majority vote of its nearest neighbors in the feature space. Here, k is just a number that tells the algorithm how many nearby points (neighbours) to look at when it makes a decision.

The value of k is critical in KNN as it determines the number of neighbors to consider when making predictions. Selecting the optimal value of k depends on the characteristics of the input data. If the dataset has significant outliers or noise, a higher k can help smooth out the predictions and reduce the influence of noisy data. However, choosing a very high value can lead to underfitting where the model becomes too simplistic.
Some applications of K-NN algorithm are recommendation system, spam detection, customer segmentation, speech recognition etc.

Statistical Methods for Selecting k:

1. **Cross-Validation:** A robust method for selecting the best k is to perform k-fold cross-validation. This involves splitting the data into k subsets, training the model on some subsets and testing it on the remaining ones and repeating this for each subset. The value of k that results in the highest average validation accuracy is usually the best choice.
2. **Elbow Method:** In the elbow method, we plot the model's error rate or accuracy for different values of k. As we increase k, the error usually decreases initially. However, after a certain point, the error rate starts to decrease more slowly. This point where the curve forms an "elbow" that point is considered as best k.
3. **Odd Values for k:** It's also recommended to choose an odd value for k especially in classification tasks to avoid ties when deciding the majority class.

Key Features of K-NN:

- **Instance-Based Learning**: It stores all training examples and classifies new data by comparing it to stored examples.
- **Non-Parametric**: Does not assume any specific distribution of the data.
- **Lazy Learning**: No explicit training phase; classification happens during prediction.

Working of K-NN Algorithm:

1. Choose the number of neighbors K.
2. Compute the distance (e.g., Euclidean distance) between the new data point and all training points.
3. Select the K nearest neighbors based on the smallest distances.
4. Perform a majority vote among the selected neighbors.
5. Assign the most common class label to the new data point.

Mathematical Formulation (Euclidean Distance):

For two points $({}_1, {}_1)$ and $({}_2, {}_2)$, the distance is calculated as:

$$(\, , \,) = \sqrt{({}_2 - {}_1)^2 + ({}_2 - {}_1)^2}$$

Choosing the Optimal K:

- Small K: Can lead to overfitting (high variance).
- Large K: Can lead to underfitting (high bias).
- The Elbow Method or Cross-Validation can help select the best K.

Advantages of K-NN:

- Simple and easy to implement.
- No need for explicit training.
- Can handle multi-class classification.

Disadvantages:

- Computationally expensive for large datasets.
- Sensitive to irrelevant or redundant features.
- Requires proper feature scaling for distance-based calculations.

ALGORITHM:

1. Load and preprocess the dataset.
2. Choose the number of neighbors (K).
3. Compute the distance metric (e.g., Euclidean distance).
4. Identify the K nearest neighbors.
5. Perform a majority vote to classify the test instance.
6. Evaluate the model using accuracy, precision, and recall.

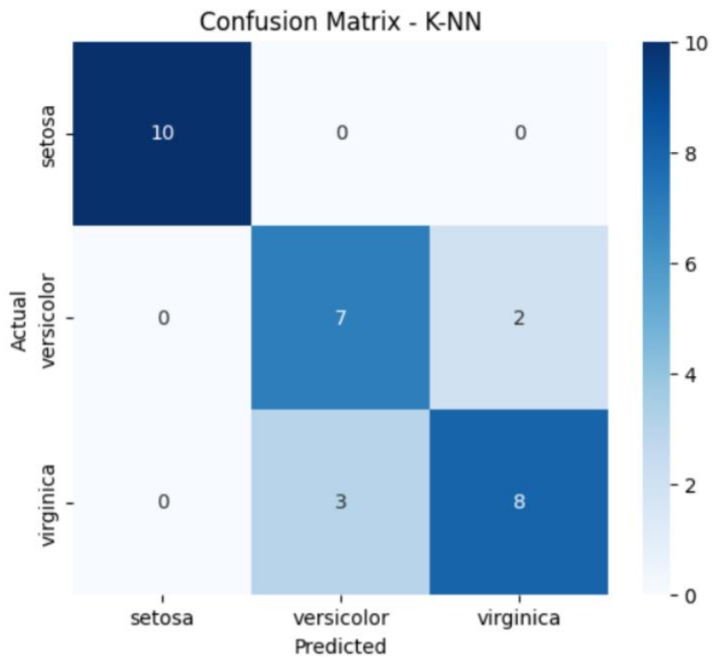
SOURCE CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler from
sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

# Load dataset (Iris dataset)
iris = datasets.load_iris()
X = iris.data[:, :2] # Taking first two features for visualization
y = iris.target
```

```
# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
      random_state=42)

# Standardizing the features (important for K-NN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize K-NN Classifier with K=5
knn_classifier = KNeighborsClassifier(n_neighbors=5, metric='euclidean')

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

# Make predictions
y_pred = knn_classifier.predict(X_test)

# Compute accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")

# Display Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualizing Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d",
xticklabels=iris.target_names,
yticklabels=iris.target_names) plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - K-
NN") plt.show()
```

OUTPUT:

```
Model Accuracy: 83.33%
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.70      0.78      0.74         9
   virginica       0.80      0.73      0.76        11

    accuracy                           0.83        30
   macro avg       0.83      0.84      0.83        30
weighted avg       0.84      0.83      0.83        30
```



LEARNING OUTCOMES:

1. Understood the K-Nearest Neighbors Algorithm and its working principles.
2. Learned how distance metrics (Euclidean, Manhattan) influence classification.
3. Implemented K-NN Classifier using Scikit-Learn.
4. Explored the importance of feature scaling in distance-based algorithms.
5. Evaluated model performance using accuracy, classification report, and confusion matrix.
6. Understood the impact of choosing the right K value using validation techniques.