EXP 2:- KNN Classifier Using Open-Source Dataset

Aim:- To implement the KNN classifier using an open source dataset (iris) and observe the prediction results.

Algorithm:-

1. Load the dataset (Iris from sklearn)
2. Split the dataset into training and test sets.
3. Normalize the feature data using standardscaler
4. Train a KNN classifier with K = 3
5. predict outcomes on test data
6. Display actual vs predicted labels.

Code:-

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standardscaler
from sklearn.neighbors import kNeighborsclassifier

# step 1: Load Iris dataset
    iris = load_iris()
    x = iris.data
    y = iris.target

# step 2: Train_test split
    x_train, x_test, y_train, y_test = train_test
                    -split(x, y, test_size=0.3,
```

```
random_state=1)

# step 3: Feature scaling

scaler = standardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# step 4: Initialize and train KNN

knn = kNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)

# step 5: prediction

y_pred = knn.predict(x_test)

# step 6: Display output

print("Actual labels :", y_test)
print("predicted labels:", y_pred)

# step 7: Evaluate

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred,
                            average='macro')
recall = recall_score(y_test, y_pred, average=
                      'macro')
f1 = f1_score(y_test, y_pred, average='macro')
cm = Confusion_matrix(y_test, y_pred)
```

```
# step 8: Display results

print("Accuracy :", round(accuracy, 2))
print("precision :", round(precision, 2))
print("Recall :", round(recall, 2))
print("F1 score :", round(f1, 2))
print("\n classification Report :\n", classification
_report(y_test, y_pred, target_names = iris.target
                                        _names))

# step 9 : Confusion matrix
sns.heatmap(cm, annot = True, cmap = "Blues",
    fmt = "d", xticklabels = iris.target_names,
    yticklabels = iris.target_names)
plt.title("Confusion matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Conclusion:-

KNN classifier performed exceptionally well on the iris dataset. This high accuracy can be attributed to: The simplicity and clean separation between classes in the iris dataset. The use of feature scaling, which is important for distance-based models like KNN.

output:

Actual labels: [0 1 1 0 2 1 2 0 0 2
1 0 2 1 1 0 1 1 0 0 1 1 1 0 2 1 0 0 1
2 1 2 1 2 2 0 1 0 1 2 2 0 2 2 1]

predicted labels: [0 1 1 0 2 1 2 0 0 2
1 0 2 1 1 0 1 1 0 0 1 1 1 0 2 1 0 0
1 2 1 2 1 2 2 0 1 0 1 2 2 0 1 2 1]

Accuracy : 1.0 (0.9333)

precision : 1.0 (0.9444)

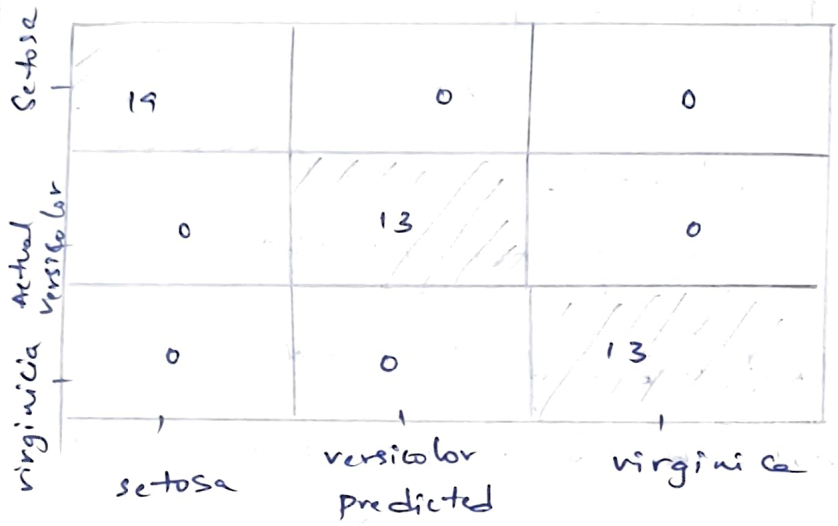
Recall : 1.0 (0.9333)

F1 score : 1.0 (0.9327)

classification Report:

	precision	recall	F1 score	support
setosa	1.00	1.00	1.00	19
versicolor	1.00	1.00	1.00	13
virginica	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Confusion matrix



[5]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report
)
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Load data
iris = load_iris()
X = iris.data
y = iris.target

# Step 2: Split and scale
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 3: Train model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

```
# Step 4: Evaluate
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
cm = confusion_matrix(y_test, y_pred)

# Step 5: Display results
print("Accuracy :", round(accuracy, 4))
print("Precision:", round(precision, 4))
print("Recall   :", round(recall, 4))
print("F1 Score :", round(f1, 4))
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=iris.target_names))

# Step 6: Confusion Matrix
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d",
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Accuracy : 0.9333
Precision: 0.9444
Recall   : 0.9333
F1 Score : 0.9327
```

```
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.83      1.00      0.91        10
   virginica       1.00      0.80      0.89        10

    accuracy                           0.93        30
   macro avg       0.94      0.93      0.93        30
weighted avg       0.94      0.93      0.93        30
```