Experiment 9 - kNN Classifier

May 1, 2023

1 Experiment Details

1.1 Submitted By

Desh Iyer, 500081889, Year III, AI/ML(H), B5

1.2 Problem Statement

Write a program to implement k-Nearest Neighbour (k-NN) algorithm to classify the iris dataset. Print both correct and wrong predictions. Java/Python ML library/classes can be used for this problem.

1.3 Theory

The k-NN algorithm is a type of supervised machine learning algorithm used for classification and regression. In the k-NN algorithm, a new data point is classified based on the k-nearest neighbours to that data point in the training set. The algorithm determines the k-nearest neighbours by calculating the distance between the new data point and all the data points in the training set.

The iris dataset is a commonly used dataset in machine learning for classification problems. It consists of 150 samples of iris flowers, with 50 samples from each of three different species of iris flowers. Each sample contains four features: sepal length, sepal width, petal length, and petal width.

1.4 Steps

Here are the steps to implement the k-NN algorithm to classify the iris dataset:

1. Load the iris dataset.
2. Split the dataset into training and testing sets.
3. Define the value of k.
4. For each data point in the testing set:
    1. Calculate the distance between the data point and all the data points in the training set.
    2. Select the k-nearest neighbours.
    3. Classify the data point based on the majority class of the k-nearest neighbours.
    4. Print whether the prediction is correct or wrong.
5. Calculate the accuracy of the classifier.

1.5 Pseudocode

Here's the pseudocode for the k-NN algorithm:

```
function knn_algorithm(train_set, test_set, k):
    predictions = []
    for i in range(len(test_set)):
        distances = []
        for j in range(len(train_set)):
            dist = euclidean_distance(test_set[i], train_set[j])
            distances.append((train_set[j], dist))
        distances.sort(key=lambda x: x[1])
        neighbors = [distances[m][0] for m in range(k)]
        prediction = majority_vote(neighbors)
        if prediction == test_set[i][-1]:
            print("Correct prediction:", prediction)
        else:
            print("Wrong prediction:", prediction)
        predictions.append(prediction)
    accuracy = calculate_accuracy(test_set, predictions)
    print("Accuracy:", accuracy)
```

2 Import libraries and data

[ ]:
```python
# Import 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
import matplotlib.pyplot as plt

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target
```

```
/home/volt/.local/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version
of SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

3 Train-test Split

[ ]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

4 Fitting data to a KNN Model

5 Determining the optimal K value through the elbow method

```
[ ]: # Determine the best value for K using the elbow method
     k_range = range(1, 21)
     scores = []
     for k in k_range:
         knn = KNeighborsClassifier(n_neighbors=k)
         knn.fit(X_train, y_train)
         scores.append(knn.score(X_test, y_test))

     best_k = k_range[scores.index(max(scores))]
```

```
[ ]: print(f'Scores: {scores}\nOptimal K value: {best_k}')
```

```
Scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Optimal K value: 1
```

6 Creating the model

```
[ ]: knn = KNeighborsClassifier(n_neighbors=best_k)
```

7 Fitting the data

```
[ ]: knn.fit(X_train, y_train)
```

```
[ ]: KNeighborsClassifier(n_neighbors=1)
```

8 Make predictions on the testing data

```
[ ]: y_pred = knn.predict(X_test)
```

9 Printing wrong and right predictions

```
[ ]: # Print correct and wrong predictions
     for i in range(len(y_pred)):
         if y_pred[i] == y_test[i]:
             print(f"Correct prediction: actual={y_test[i]}, predicted={y_pred[i]}")
         else:
             print(f"Wrong prediction: actual={y_test[i]}, predicted={y_pred[i]}")
```

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

```
Correct prediction: actual=1, predicted=1
Correct prediction: actual=0, predicted=0
Correct prediction: actual=2, predicted=2
Correct prediction: actual=1, predicted=1
Correct prediction: actual=1, predicted=1
Correct prediction: actual=0, predicted=0
Correct prediction: actual=1, predicted=1
Correct prediction: actual=2, predicted=2
Correct prediction: actual=1, predicted=1
Correct prediction: actual=1, predicted=1
Correct prediction: actual=2, predicted=2
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Correct prediction: actual=1, predicted=1
Correct prediction: actual=2, predicted=2
Correct prediction: actual=1, predicted=1
Correct prediction: actual=1, predicted=1
Correct prediction: actual=2, predicted=2
Correct prediction: actual=0, predicted=0
Correct prediction: actual=2, predicted=2
Correct prediction: actual=0, predicted=0
Correct prediction: actual=2, predicted=2
Correct prediction: actual=2, predicted=2
Correct prediction: actual=2, predicted=2
Correct prediction: actual=2, predicted=2
Correct prediction: actual=2, predicted=2
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Correct prediction: actual=1, predicted=1
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Correct prediction: actual=2, predicted=2
Correct prediction: actual=1, predicted=1
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Correct prediction: actual=2, predicted=2
Correct prediction: actual=1, predicted=1
Correct prediction: actual=1, predicted=1
Correct prediction: actual=0, predicted=0
Correct prediction: actual=0, predicted=0
Accuracy: 1.0
```

10 Conclusion

The iris dataset has relatively few features and classes, with a clear separation between them. This means that the decision boundaries for a KNN classifier will be relatively simple, leading to high accuracy. A more complex dataset with overlapping classes and more ambiguous feature relationships would pose a greater challenge for a KNN classifier.