p2

December 7, 2023

[2]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import warnings

# Suppress SettingWithCopyWarning
warnings.filterwarnings("ignore", category=pd.errors.SettingWithCopyWarning)
```

[3]:
```python
# Load the "diabetes.csv" dataset.
data = pd.read_csv('diabetes.csv')

# The features and targets are separated
x = data.drop(columns=['Outcome'])
y = data[['Outcome']]
```

[4]:
```python
# The data is shuffled and split into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
↪random_state=100)
```

[5]:
```python
# Features are Normalized using Min-Max Scaling.
x_train_max = x_train.max()
x_train_min = x_train.min()
range_x_train = x_train_max - x_train_min
x_test_scaled = (x_test - x_train_min) / range_x_train
x_train_scaled = (x_train - x_train_min) / range_x_train
```

[6]:
```python
# Convert data to Numpy array
x_train_np = x_train_scaled.to_numpy().reshape((-1, 8))
x_test_np = x_test_scaled.to_numpy().reshape((-1, 8))
y_train_np = y_train.to_numpy().reshape((-1, 1))
y_test_np = y_test.to_numpy().reshape((-1, 1))
```

[7]:
```python
# Function to calculate Euclidean Distance
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

# Function for Distance-Weighted Voting
```

```python
def distance_weighted_vote(distances):
    weights = 1 / (distances + 1e-10)  # Adding a small constant to avoid␣
↪division by zero
    return weights / np.sum(weights)

# Function to predict the class using KNN
def knn_predict(train_data, train_labels, test_instance, k):
    distances = np.array([euclidean_distance(test_instance, train_instance) for␣
↪train_instance in train_data])
    sorted_indices = np.argsort(distances)

    # Break ties using Distance-Weighted Voting
    vote_weights = distance_weighted_vote(distances[sorted_indices[:k]])
    class_votes = np.zeros(np.max(train_labels) + 1)

    for i in range(k):
        class_votes[train_labels[sorted_indices[i]]] += vote_weights[i]

    predicted_class = np.argmax(class_votes)
    return predicted_class

# Function to evaluate KNN for a given k value
def knn_evaluate(train_data, train_labels, test_data, test_labels, k):
    correct_count = 0

    for i in range(len(test_data)):
        predicted_class = knn_predict(train_data, train_labels, test_data[i], k)
        if predicted_class == test_labels[i]:
            correct_count += 1

    accuracy = correct_count / len(test_data) * 100
    return correct_count, len(test_data), accuracy
```

[ ]:
```python
# Set the range of k values for iterations
k_values = [2, 3, 4 ,7 ,23]
accuracies=[]
# Perform iterations and print results
for k in k_values:
    correct, total, accuracy = knn_evaluate(x_train_np, y_train_np, x_test_np,␣
↪y_test_np, k)
    print(f"k value: {k}")
    print(f"Number of correctly classified instances: {correct}")
    print(f"Total number of instances: {total}")
    print(f"Accuracy: {accuracy:.2f}%\n")
    accuracies.append(accuracy)
```

```
k value: 2
```

```
Number of correctly classified instances: 163
Total number of instances: 231
Accuracy: 70.56%

k value: 3
Number of correctly classified instances: 167
Total number of instances: 231
Accuracy: 72.29%

k value: 4
Number of correctly classified instances: 164
Total number of instances: 231
Accuracy: 71.00%

k value: 7
Number of correctly classified instances: 170
Total number of instances: 231
Accuracy: 73.59%
```

[ ]:

```

```