Implementing and Analyzing k-Nearest Neighbor Algorithm for Diabetes Prediction on Pima Indians Dataset

Gormery K. Wanjiru

April 30, 2024

Abstract

could not make the full report in time, delivered this to be registered

1 Python Code

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,
    accuracy_score, precision_score, recall_score, f1_score

def load_dataset(filename):
    return np.genfromtxt(filename, delimiter=',')

def euclidean_distance(instance1, instance2):
    return np.sqrt(np.sum((instance1 - instance2) ** 2))

def k_nearest_neighbors(train_data, test_instance, k):
    distances = []
    for index, train_instance in enumerate(train_data[:, :-1]):
        dist = euclidean_distance(test_instance, train_instance)
        distances.append((train_data[index], dist))
    distances.sort(key=lambda x: x[1])
    neighbors = [distances[i][0] for i in range(k)]
    return neighbors

def predict_classification(train_data, test_instance, k):
    neighbors = k_nearest_neighbors(train_data, test_instance, k)
    outputs = [neighbor[-1] for neighbor in neighbors]
    prediction = max(set(outputs), key=outputs.count)
    return prediction
```

```
def kNN(train_data, test_data, k):
    predictions = []
    for test_instance in test_data[:, :-1]:
        prediction = predict_classification(train_data, test_instance, k)
        predictions.append(prediction)
    return predictions

def main():
    # Load the dataset
    data = load_dataset('diabetes.csv')

    # Split dataset into training and testing
    train, test = train_test_split(data, test_size=0.3, random_state=42)

    # Define k
    k_values = [3, 5, 7, 9]
    results = {}

    # Testing different values of k
    for k in k_values:
        predicted_labels = kNN(train, test, k)
        accuracy = accuracy_score(test[:, -1], predicted_labels)
        precision = precision_score(test[:, -1], predicted_labels)
        recall = recall_score(test[:, -1], predicted_labels)
        specificity = (confusion_matrix(test[:, -1], predicted_labels)[0,
                       (confusion_matrix(test[:, -1], predicted_labels)[0
                        confusion_matrix(test[:, -1], predicted_labels)[0
        f1 = f1_score(test[:, -1], predicted_labels)
        results[k] = (accuracy, precision, recall, specificity, f1)

    for k, scores in results.items():
        print(f'k={k}: Accuracy={scores[0]:.2f}, Precision={scores[1]:.2f

if __name__ == "__main__":
    main()
```

References