Implementing K Nearest Neighbors (KNN)

Maliha Mahjabin Chowdhury
Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204043@aust.edu

***Abstract*—Classification techniques are widely used in modern days. It is a process of predicting the class of an unknown data point, by using a set of correctly classified data points. For instance, determining whether a mail is a spam or not is a great application of classification models. In this report, we thoroughly discuss about one of the simplest classifiers, K Nearest Neighbors Classifier. At first, we design the classifier and following that we use the model to predict classes for some unknown data points.**

***Index Terms*—K Neighbors, Euclidean Distance.**

I. Introduction

K Nearest Neighbors Classifier is one of the easiest, yet a very effective classifier. As the name suggests, the model predicts the class of a data point based on its nearest neighbors. That means, the data point is classified as the majority of its nearest neighbors' class. The distance is measured in different ways, for example, Euclidean distance, Manhattan distance, Minkowski distance, etc. K number of nearest neighbors are taken into consideration, and the according to the majority, the class of test data point is predicted. This is the main principle of K Nearest Neighbors Classifier.

The rest of the report is organised in the following manner: section 2 discusses our experimental design. In section 3, we illustrate the results and present our findings. Finally, section 4 summarizes the report and section 5 demonstrates the algorithm implementation.

II. Experimental Design

In this section, we illustrate the basic algorithm of KNN classifier. Additionally, we demonstrate the implementation of the algorithm.

A. Algorithm

The K Nearest Neighbors algorithm is given below.

B. Implementation Process

We design the classifier in Python. This is accomplished using the most popular libraries: NumPy [1], pandas [2] and Matplotlib [3]. NumPy and pandas help processing the dataset and performing necessary operations and calculations respectively. We visualize the results using Matplotlib. We classify the data points using the equation 1. We use different colors and markers for different classes. Figure 1 presents classified test data points on a plot.

Algorithm 1 K Nearest Neighbors (KNN)

1: Read train file
2: Initialize k, number of neighbors
3: Read test file
4: Calculate distance between test data point and each train data point using the following formula:

$$d = (x_1 - X_1)^2 + (x_2 - X_2)^2 + ... + (x_i - X_i)^2 \quad (1)$$

5: Sort the distances
6: Take the first k distances and classify the test data point based on the majority of top k instances
7: Repeat step 4 to 6 for each data points in test file

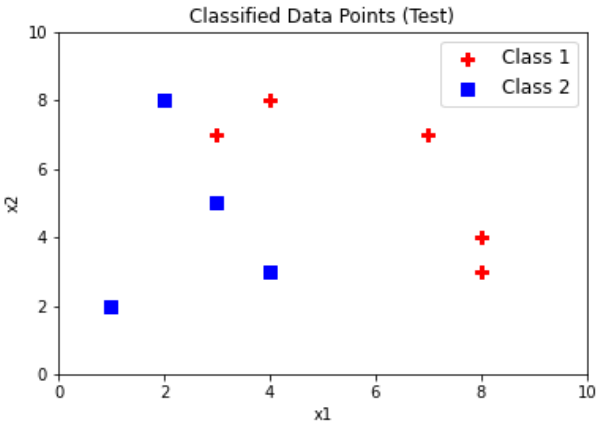

Fig. 1. Classified Test Data Points

III. Result Analysis

We use equation 1 to calculate distance between test data point and train data points. Upon sorting the distances, we take top k distances and based on the majority, we classify the test data point. In figure 1, the classified test data points are showed. The red ones belong to class 1 and the blue ones belong to class 2. Different colors and markers are used to represent different classes. As the test data points are not labelled, we can not calculate accuracy of the model's performance.

IV. Conclusion

K Nearest Neighbors is the simplest Machine Learning algorithm and the easiest to understand and implement. KNN being a non linear classifier, it is useful in many cases. However, as the model calculates distance between test data point and every train data point, for each test sample, the model requires high computational power. Number of neighbors, k and the method for calculating distance often make significant difference.

KNN is not very suitable in case of huge datasets with too many features, as there will be way too many calculations. However, it is an excellent choice for beginners experimenting with small datasets.

V. Code

The classifier is designed in Python. The implementation is given below.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


dataset = pd.read_csv("train_knn.txt",
header = None, sep = ",")


dataset = dataset.iloc[:, :].values


myFig = plt.figure(figsize=[6,4])


plt.scatter([dataset[i][0]
for i in range(len(dataset))
if dataset[i][2] == 1],
[dataset[i][1]
for i in range(len(dataset))
if dataset[i][2] == 1],
color = "red", marker = "P",
label = "Class 1", s = 60)


plt.scatter([dataset[i][0]
for i in range(len(dataset))
if dataset[i][2] == 2],
[dataset[i][1]
for i in range(len(dataset))
if dataset[i][2] == 2],
color = "blue", marker = "s",
label = "Class 2", s = 60)


plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Sample Data Points (Input)")
plt.xlim(0, 8)
plt.ylim(0, 8)
plt.legend(loc = 'upper left',
fontsize = 12)
plt.show()


from google.colab import files
myFig.savefig("Assignment4_1.png")
files.download("Assignment4_1.png")


test = pd.read_csv('test_knn.txt',
header=None, sep=',')
test = np.array(test)


dtype = [('distance', int), ('class', int)]
class_1 = []
class_2 = []


k = int(input())


f = open("Prediction.txt", "w")
for i in range(len(test)):
  values = []
  for j in range(len(dataset)):
    x = dataset[j]
    values.append((((x[0]-test[i][0])**2
    + (x[1]-test[i][1])**2), x[2]))
  knn = np.array(values, dtype)
  knn = np.sort(knn, order = 'distance')
  class1 = 0
  class2 = 0
  f.write("Test Point : " +
  str(test[i]) + "\n")
  for j in range(k):
    f.write("Distance : " + str(knn[j][0]) +
    "    Class : " + str(knn[j][1]) + "\n")
    if knn[j][1] == 1:
      class1 = class1 + 1
    else:
      class2 = class2 + 1
  if class1 > class2:
    class_1.append(test[i])
    f.write("Predicted Class : Class 1\n")
  else:
    class_2.append(test[i])
    f.write("Predicted Class : Class 2\n")


f.close()


myFig = plt.figure(figsize=[6,4])
```

```
plt.scatter([class_1[i][0]
for i in range(len(class_1))],
[class_1[i][1]
for i in range(len(class_1))],
color = "red", marker = "P",
label = "Class 1", s = 60)


plt.scatter([class_2[i][0]
for i in range(len(class_2))],
[class_2[i][1]
for i in range(len(class_2))],
color = "blue", marker = "s",
label = "Class 2", s = 60)


plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Classified Data Points (Test)")
plt.xlim(0, 10)
plt.ylim(0, 10)
plt.legend(loc = 'best', fontsize = 12)
plt.show()


from google.colab import files
myFig.savefig("Assignment4_2.png")
files.download("Assignment4_2.png")
```

REFERENCES

[1] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[2] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[3] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.