K-Nearest Neighbours Classifier
Assignment 3 of Machine Learning I, 2022/2023

Mura Alessio

DIBRIS - Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
Università Degli Studi di Genova

1 Introduction

Classification in machine learning and statistics is a supervised learning approach in which the computer program learns from the data given to it and makes new observations or classifications.

One of the most common classifier is the **k-Nearest Neighbour classifier [KNN]**. KNN is a supervised learning algorithm that estimates how likely a data point (instance) belongs to one class or the other depending upon which class its 'k' nearest instances belong to.

It is a non-parametric model that means it does not make assumptions about the data pre-hand like they do in case of linear regression that the data must linear. KNN is referred to as a *Lazy Learner*, because it does not really do anything during the training phase, actually, there is no as such a training phase for KNN, means it does use the training data points to do any kind of generalization

2 Theory of KNN Classifier

Given an input point $\bar{x}$ called *query*, the goal is to classify this point using all the other points that surrounds it. The classifier then gives the predicted class ω as output, such that ω = *y($\bar{x}$)*, where *y()* is the rule of the kNN.

The decision rule is based on the concept of the **majority voting**, meaning that the $\bar{x}$ will be classified with the class $\omega 1$ only if the majority of the k-closest objects around him belongs to the class $\omega 1$.

Let's define X as the training set given as input to the kNN:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{bmatrix} \tag{1}$$

where x_i is a single observation of the training set.

The k-Nearest Neighbour classifier computes the *distance* (norm) between each observation x_i and the query point $\bar{x}$, and stores it in a row vector **N**:

$$N = (\|x_1 - \bar{x}\|, \|x_2 - \bar{x}\|, ..., \|x_n - \bar{x}\|) \tag{2}$$

where each value stored inside **N** tells how much the query point is similar to the considered observation.

The kNN classifier than takes into account the first k points that are closest to x_i (i.e. the k points that are most similar to the query one)

$$\{n_1, ..., n_n\} = top_k \|x_i - \bar{x}\| \tag{3}$$

The class given to x_i is then the class that appears the most:

$$\omega = mode\{t_{n1}, ..., t_{nk}\} \tag{4}$$

where t_{n1} is the class of the class 1.

Moreover, it is possible to visualize the performance of our prediction model by a **Confusion Matrix**.

A Confusion matrix is an NxN matrix used for evaluating the performance of a classification model, where N is the number of target classes. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall, and F1-score.

In our case, we used a Confusion Matrix for Multi-Class Classification in order to check the performance for each digits.

3 THE ASSIGNMENT

The given assignment consist of three tasks:

- **Task 1**: Obtain a data set
- **Task 2**: Build a kNN classifier
- **Task 3**: Test the kNN classifier

3.1 Task 1: Obtain a data set

This assignment takes into account the MNIST set, a standard benchmark for machine learning tasks.
The data represent 70000 handwritten digits in 28x28 greyscale images, already split into a 60000-image training set and a 10000-image test set, 784 attributes and 10 classes.

Since the two sets are very big, two random subsets are selected inside the main function [600 for training set and 100 for the test set]. By using Python solution, the machine learning library *keras*, part of *tensorflow*, contains ready-made functions. Just do the following:

from tensorflow.keras.datasets import mnist
$$(train_X, train_Y), (test_X, test_Y) = mnist.loadData()$$

3.2 Task 2: Build a kNN classifier

In the second tasks, we have to actually implement a k-Nearest Neighbour classifier. In order to check the correctness of the input variables, we created some functions: *CheckDataset()*, *Check-K-value()*.

Subsequently, we classified the dataset according to the kNN theory, described with formulas 2, 3 and 4, and returned the classification obtained.

We computed the classification by a function we created: *KNNclassifier()*. After applying the reshape function, in order to work in 2-Dimensions, and initilizing the lists, we computed a prediction list and stored it into a dictionary. Then we sorted in ascending order by *Euclidean distance* and got the count of the maximum class in result list.

Finally, we computed the accuracy and error rate by checking if the values inside pred-list were equal to test-y. The function returns: accuracy, error rate and pred-list.

3.3 Task 3: Test the kNN classifier

In task 3 it is required to run the kNN classifier using different values of k. The values used are:

$$K_1 = [1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 50]$$

By a *for loop*, we passed each of the previous k-value inside *KNNclassifier()* and plotted the respective results in a subplot graph where we reported the accuracy and error rate.

Then, we created a **Confusion Matrix** by some functions: *ConfusionMatrix()* and *ClassificationQualityIndexes()*. First of all, we created a matrix $10x10$ (10 is the number of digits) and computed all the quality indexes by the following four terminologies:

- **True Positives (TP)**: when the actual value is Positive and predicted is also Positive.
- **True negatives (TN)**: when the actual value is Negative and prediction is also Negative.
- **False positives (FP)**: When the actual is negative but prediction is Positive.
- **False negatives (FN)**: When the actual is Positive but the prediction is Negative.

In order to have a clear representation of the performance, we reduced the set of k value:

$$K_2 = [1, 3, 5, 10, 25, 50]$$

Finally, we compared the accuracy of the classifier on the different classes (i.e. digits), checking each digit vs the remaining nine, by repeating the previous steps.

4 RESULTS

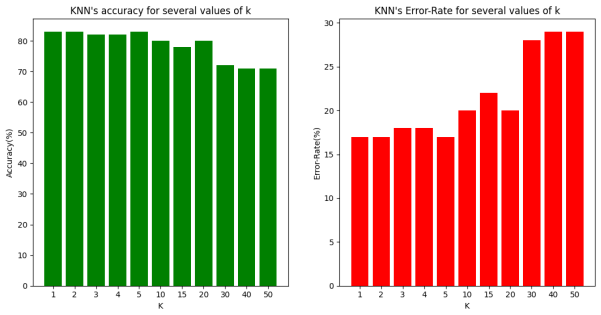

Figure 1. Accuracy and Error rate for several values of k.

As shown in Figure 1, the error rate increases as the value of k gets higher. The best value of k should then be lower then 10, in order to avoid taking considering some wrong neighbours.

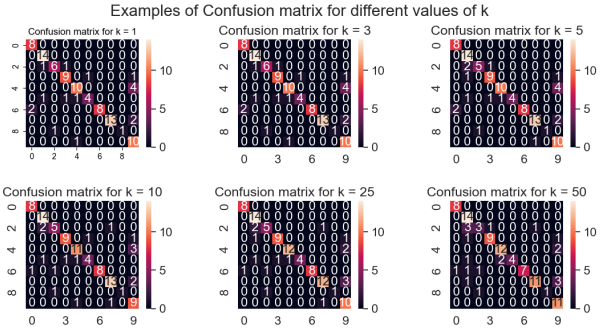

Figure 2. Confusion matrix for different values of k.

As shown in Figure 2, we can clearly see that the higher values are along the diagonal. This agrees with the theory, because the cells along the diagonal represent the **True Positives (TP)** values, that is when the actual value is well predicted.

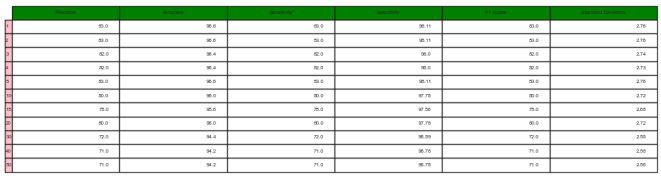

Figure 3. Quality indexes for different values of k.

First of all, we have to define what each Index means:

- **Precision**: It is a measure defined as the ratio of the total number of correctly classified positive classes divided by the total number of predicted positive classes.
- **Accuracy**: It is a measure defined as the ratio between the number of correct predictions and the total number of predictions.
- **Sensitivity**: It is a measure defined as the ratio of the total number of correctly classified positive classes divide by the total number of positive classes.
- **Specificity**: It is a measure that tells us what fraction of all negative samples are correctly predicted as negative by the classifier.
- **F1-score**: It is a number between 0 and 1 (in our case we use percentage) and is the harmonic mean of precision and recall.
- **Standard deviation**: It is a statistic that measures the dispersion of a dataset relative to its mean and is calculated as the square root of the variance.

By Figure 3, we can infer that our prediction was quite successful. Now, in a perfect world, we'd want a model that has a precision of 1 and a recall of 1. That means a F1-score of 1.

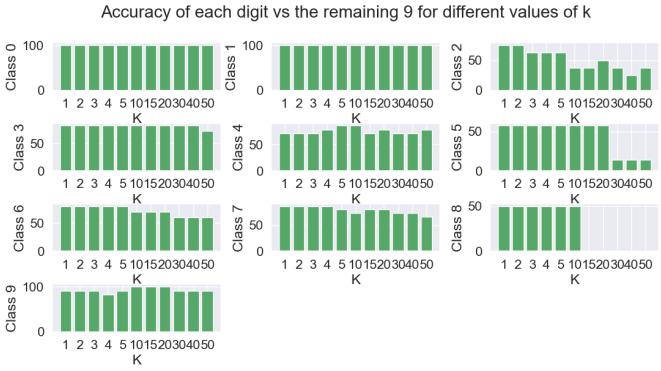

Figure 4. Accuracy for each digit vs remaining nine.

By Figure 4, It is clearly visible that the easiest classes to recognize in general are 0 and 1 (with an accuracy of 100%), while the hardest class to recognize in general is 8 (with an accuracy of 50%, meaning that one classification every two is wrong). Moreover, we can see that, as the value of k gets higher, the accuracy decreases.

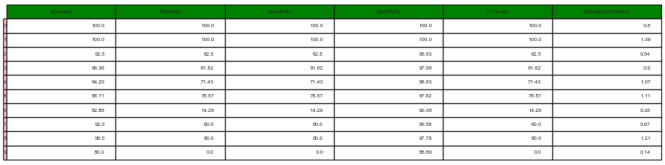

Figure 5. Quality Indexes for each digit vs remaining nine.

Finally, by Figure 5, we can infer what we got in the previous case, that is the correctness of the prediction for each digits vs the remaining nine, in the case that K was equal to 5.

In fact, the easiest classes to recognize are 0 and 1. Whereas, with this k value, the hardest one is class 9.

Obviously, for each value of K, the hardest class to recognize will change.