kNN Classifier

Machine Learning 2021/22: 3^{rd}Assignment

Matteo Maragliano
EMARO-European Master on Advanced Robotics
DIBRIS - Dip. Informatica, Bioingegneria, Robotica, Ing. dei Sistemi
Università degli Studi di Genova

***Abstract*—The goal of the assignment is to implement a *kNN Classifier* (k-Nearest Neighbors Classifier) among a certain data given.**
In kNN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (k is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

In particular we were provided different data, from which we took the train and the test set. In particular we pointed out two variables for each data set:

- ***X*: representing the different classes (there are number as we know, for example 1,2 ... and so on).**
- ***T*: giving the representation of each class as a matrix.**

I. Introduction

THE kNN algorithm is mainly used for classification problems, even if it can also be used in case of regression problems.
The algorithm uses the entire training instances to predict output or unseen data; the model is not learned using training data prior and the learning process is postponed to a time when prediction is requested on the new instance.

II. Obtain a data set

As we explained before, the first step to compute is getting data. The function used is *loadMNIST* which takes different parameters as inputs:

- 0 : used to take the *X* and *T* for the training set;
- 1 : used to take the *X* and *T* for the test set.

There is also the possibility to give a second input: a parameter that describe the type of label to take; in our case digits from 0 to 9 where the 0 represents the 10 while the other are the same digit, so the loading data steps is:

$$[X_{train}, T_{train}] = loadMNIST(0, 0:9);$$
$$[X_{test}, T_{test}] = loadMNIST(1, 0:9);$$

Each item represents a different matrix, in particular:

- *X*s are matrices with dimensions:
  - Train: 60000 x 784;
  - Test: 10000 x 784;
- *T*s are matrices with dimensions;
  - Train: 60000 x 1;
  - Test: 10000 x 1.

Another parameter used is the *k* which value represents the class to compute the output according to the algorithm.

III. Build a kNN classifier

This task concerned the building the classifier.
First of all there is the definition of the parameters the function has to take:

- a set of data, as a n x d matrix, to be used as the training set;
- a corresponding column (a n x 1 matrix) of targets;
- another set of data, as a m x d matrix, to be used as the test set;
- an integer k.

First of all the program should verify the correctness of the dimension of the data sets given.
Before starting yo implement the total classifier for different value of *k*, the easiest way is to implement for *k* = 1.
There are various steps to be passed to have the classifier ready. Once all the dimensions are correctly checked, we computed the Euclidean distances between each row of the train with each row of the test, storing all the results into a matrix with dimensions *m* x *n*.

Once filled the matrix, we searched, for each row, the minimum value for the *k* searched (1 in the first case). the software MATLAB does not give the value itself but the index, so with a simple operation we can take the row with the minimal value of the distance.
Once arrived here we computed the mode of the row selected, to determine the most frequent value: this will be our target.
At the end was computed the error rate for our result, to determine how far the classifier went from the test: to solve this we compared the target matrix with the test matrix and summed an error for each mistaken value.
The error was computed like:

$$err = err + \frac{err}{length(target)}$$

With this formula the error is proportioned to the length of the data set taken.

IV. Test the kNN Classifier

This section provided the testing of the classifier built before.
First of all, since the data sets given are too wide to compute a result for all, we took a subsets:

- 5% for the training set: we trained the classifier with 3000 rows;
- 15% for the test set: we tested the accuracy of our classifier with 1500 rows.

The classifier has been tested for multiple value of *k*,represented the number of neighbours required, as we were suggested for example *k* = 1,2,3,4,5,10,15,20,30,40,50.
To better visualise the result we provided a graph in which we plotted on the *x-axes* the different value of *k* while on the *y-axes* the *accuracy*. The accuracy was computed as:

$$accuracy = 1 - error_{rate}$$

where the *error rate* is a matrix, with properly dimensions, containing all the error committed computed for every row of the test with respect to the target obtained.
The graph result is the following:

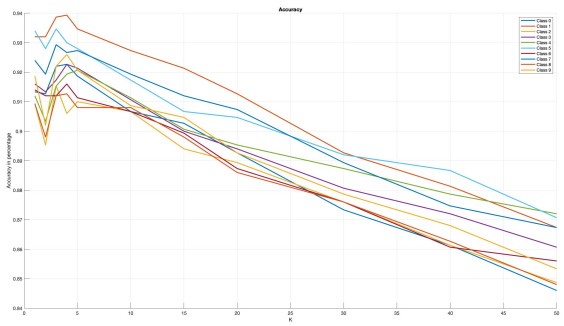

Fig. 1. Representation of the accuracy of the kNN classifier built.

It can be clearly seen that the accuracy decreases while the number *k* of neighbor searched increases: this is because, with the increasing of the number of neighbours required, more labels satisfied the condition, even if while testing they represent an error.
From the graph it is clear that the accuracy decrease with the increasing of the number of neighbours and the decreasing is, with a good approximation, a linear decrease. In particular there are two situations in which there is an abnormal situation:

- k = 2: we can see the presence of a huge peak down. this behavior is due to an ambiguity in the choice of values, since there is the possibility to have the same number of values for each neighbour.
- k = 4: as before also in this case there is the possibility of an ambiguity, even if it is less probably since there are more values to analyze.

In particular we can say that, already the 35^{th} neighbours is not in the same class as the first one, so with *k* too high we consider also values not corrected fot the experiment.
In the following graph it is also represented the error computed:

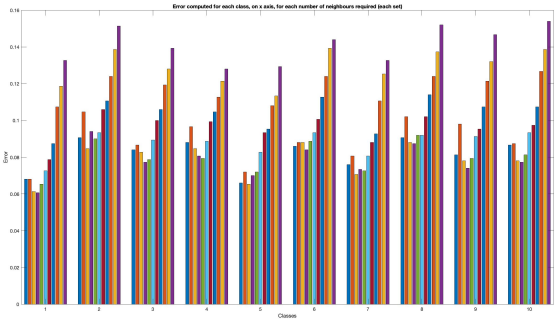

Fig. 2. Representation of the error computed by the classifier

As it is reasonable, for big *k*s there is more or less a bigger error. The bar graph is the dual to the first one since they are obtained by $1 - error_{rate}$ for the first and $1 - accuracy$ for the second.