Report Lab3 for the Machine Learning course

Nicholas Attolino

November 2023

Contents

Introduction

This work is a report for the third laboratory of the Machine Learning course. The subject of this laboratory is a **k-Nearest Neighbors Classifier** using MATLAB, a versatile platform that serves as a numerical computation and statistical analysis environment, as well as a programming language[2].
The chapters are divided in:

1. Goal;
2. Methods used;
3. Results.

1 Goal

The goals of this project are:

- Obtain a dataset;
- Build a kNN classifier;
- Test the kNN classifier.

The k-nearest neighbors algorithm, abbreviated as k-NN, functions as a non-parametric method for classification. Instead of creating a model with defined parameters, it constructs a discriminatory rule directly from the available data. The dataset employed for both training and testing is known as the *mnist* dataset. This dataset contains images of handwritten digits depicted in grayscale, each measuring 28 by 28 pixels. These images are divided into 10 distinct classes corresponding to the numbers ranging from 0 to 9.
In the training set, each class comprises a minimum of 5800 instances, while in the test set, there are at least 890 instances for each class.

1.1 Data Files overview

```
- Attolino-Lab3.zip
    - Attolino-Lab3
        - Lab3.m
        - mnist
            - (the dataset files)..
```

The *Lab3* is the main program while *mnist* is the dataset used.

2 Methods used

2.1 Dataset elaboration

Due to the abundance of instances within each class in the dataset, there's a need to downsize the training set. This involves selecting a defined percentage from the entire dataset.
The data set contains classes arranged in sequence. To ensure equal representation of all classes within the subset, it's imperative to randomly select observations.
An alternate approach involves generating individual subsets for each class and subsequently merging these 10 subsets to compose the training set.

2.2 kNN Classifier

The Nearest Neighbor rule is based on the assumption that observations which are close together (in some appropriate metric) will have the same classification[1].
So, given a training set:

$$X = \{x_1, ..., x_l, ..., x_n\}$$

and a query point $\bar{x}$, the estimated class is the same as the point:

$$q = arg\ min||x_l - \bar{x}||$$

The k-Nearest-Neighbour is a variant of the previous algorithm: it takes from the training set the firsts k observations ordered by crescent euclidean distance

$$\{n_1, ..., n_k\} = top\ k||x_l - \bar{x}||$$

and chose as class the most frequent between them

$$y = mode\{t_{n_1}, ..., t_{n_k}\}$$

2.3 The Matlab Function

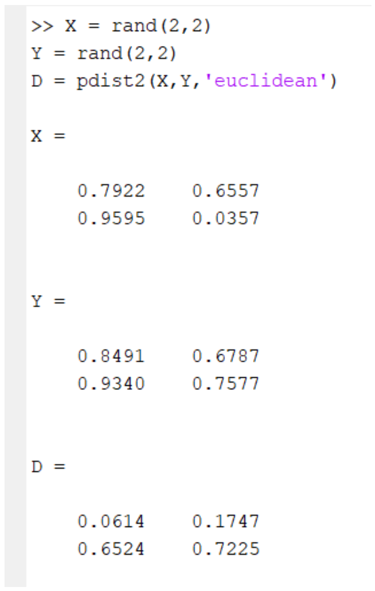
The classifier function has at least 4 input arguments:

- The **training set**, a *n x d* matrix where *n* is the number of observations and *d* is the number of attributes. In the studied case each observation has 784 (28x28) attributes: each image, sampled as grey scale, is represented as a row vector of 784 numbers;
- The **training set labels**, a *n x 1* matrix with each value equal to the number the corresponding observation represents. Note that the "0" class is represented by the number 10, in order to use it as index;
- The **test set**, a *m x d* matrix where *m* is the number of observations to classify and *d* remain the number of attribute. One of the firsts operations the function does, is to check if the *d*-s are the same;
- The **k** number of neighbors. This could be either a scalar or a vector of *k*-s. The function will return as many estimated classes as the number of *k*-s in the vector.

Eventually, there is an additional input that might be the **test set classes**. When the latter is present, it is used to compute the accuracy once the function classify all the observations.
The program executes a number of checks before it starts the classification:

1. Initially it checks that the number of arguments are at least 4;
2. After that, it checks that the number of attributes of training and test sets matches;
3. Finally it checks that all the values in the *k* vector are greater than 0 but smaller than the total number of observations (The function cannot find 10 nearest neighbors if there exist only 5 observations).

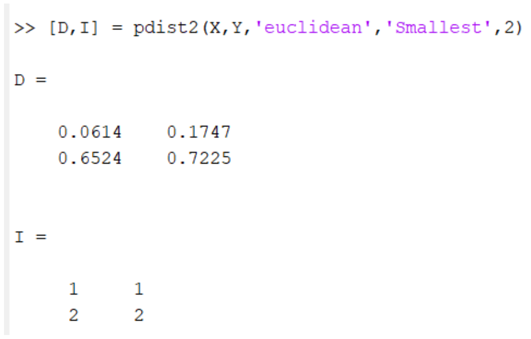
The main operation the function does is to call another function, the *pdist2* function [3]: this returns the euclidean distance between the two given observations (if we don't specificate it, by default it returns always the euclidean distance).
When two matrices (training and test sets) are given in input, it returns a matrix with the distances of each observation in the first matrix from each observation in the second one.



```
>> X = rand(2,2)
Y = rand(2,2)
D = pdist2(X,Y,'euclidean')

X =

    0.7922    0.6557
    0.9595    0.0357


Y =

    0.8491    0.6787
    0.9340    0.7577


D =

    0.0614    0.1747
    0.6524    0.7225
```

Figure 1: The *pdist2* function

An optional input argument it has is the '*Smallest*' string followed by an integer n: for each observation in the test matrix, *pdist2* finds the n smallest distances by computing and comparing the distance values to all the observations in the training matrix.
The function then sorts the distances of every test observation in ascending order; in this case, the second output contains the indexes of the observations in the training set corresponding to the distances just computed.



```
>> [D,I] = pdist2(X,Y,'euclidean','Smallest',2)

D =

    0.0614    0.1747
    0.6524    0.7225


I =

     1     1
     2     2
```

Figure 2: The *pdist2* function with a second input/output

The function set up provides an 'n' that matches the largest 'k' value in the corresponding vector, given that additional neighbors aren't necessary.

With every 'k' value and for every test observation, the classifier calculates the class label by identifying the mode among the initial 'k' class labels obtained using the earlier indices.
The result will be a matrix having rows equivalent to the number of test set observations and columns equivalent to the quantity of values in the 'k' vector. For each observation in the test set, the function delivers the most likely class for each 'k'.

3 Results

The test involved utilizing the entire training set, along with 50% of the test set, while employing specific k values:

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

Using a *for* loop, the function was invoked to categorize each test set observation into either the i_{th} class or any other.
As the test set labels were provided as input, the function became capable of determining the accuracy in correctly identifying each digit against the remaining digits, considering various k values within the specified vector.

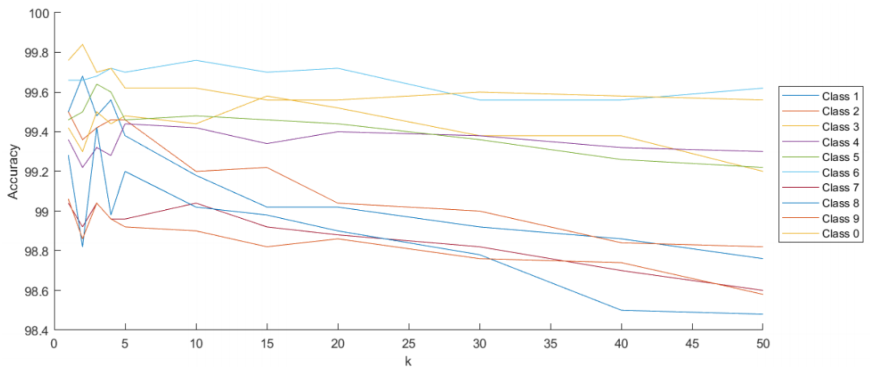


Figure 3: Class Accuracy variation with k

With $k = 3$, the accuracy spanned between 99.04 and 99.7, averaging at approximately 99.42, making it the highest average accuracy. The better accuracy was achieved when identifying the digit 0 using $k = 2$, reaching around 99.84.

4 Conclusions

The classifier's performance varies based on the specific class it aims to differentiate from the rest.
Overall, the classifier demonstrates high effectiveness, achieving an accuracy surpassing 98.8% when using the appropriate k value.
Directly providing a K vector to the classifier, rather than a single scalar each time, significantly enhances the execution speed by circumventing unnecessary repetitions in the computation process.

References

[1] T. Cover and P. Hart, *Nearest neighbor pattern classification.* 1967, vol. 13, pp. 21–27. DOI: `10.1109/TIT.1967.1053964`.

[2] T. M. Inc., *MATLAB Version: 23.2.0.2365128 (R2023b).* The MathWorks Inc., 2023.

[3] *pdist2 function.* [Online]. Available: `https://it.mathworks.com/help/stats/pdist2.html`.