k-NEAREST NEIGHBOUR CLASSIFIER

Machine Learning 2018/19: Assignment 3

Torielli Davide Fusaro Fabio
EMARO - European Master on Advanced Robotics
DIBRIS - Dip. Informatica, Bioingegneria, Robotica, Ing. Dei Sistemi
Università Degli Studi Di Genova, November 16, 2018

I. INTRODUCTION

CLASSIFICATION is a model for recognition. With a given input, the role of the *classifier* is to categorize this input in one of the suitable class. For example, we have a patient record as input and we want to put the patient in one of the categories healthy/at risk/ill.
The k-Nearest Neighbour classifier is a solution for this problem. Differently from others (for example the Naive Bayes Classifier) this is a non-parametric model. This means that we do not explicitly implement a model with parameters, but we directly build a discrimination rule from data. So, there is not a *training* phase.
In this work we implement in Matlab the *k-Nearest Neighbour Classifier*, and test it on different data with different values of k.

II. THE THEORY

A. The kNN Classifier

Given a *query* point $\bar{\boldsymbol{x}}$ we want to classify it. So we build a rule (the classifier) $y()$ that receives the *query* point $\bar{\boldsymbol{x}}$ and outputs a *class* (the category) $\omega = y(\bar{\boldsymbol{x}})$.
The idea is to do a mechanism of *voting* for the choice of the class; the majority will decide the right class. We have the *training set* X:

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_n^T \end{bmatrix}$$

where each $\boldsymbol{x}_l^T$ is a row with $d+1$ columns which is a training point with a known class $x_{l,d+1}$.
The algorithm performs the norm between each training point $\boldsymbol{x}_l^T$ and the *query* point $\bar{\boldsymbol{x}}$:

$$\boldsymbol{N} = \{\|\boldsymbol{x}_1 - \bar{\boldsymbol{x}}\|, \ldots, \|\boldsymbol{x}_n - \bar{\boldsymbol{x}}\|\}$$

If the columns of $\bar{\boldsymbol{x}}$ are greater than the columns of $\boldsymbol{x}_l^T$ (minus the $d+1$ that is the class) the additional columns are discarded at the beginning and so not considered, in order to perform correctly the norm.
The values of $\boldsymbol{N}$ tell us the *distance* of the query point from each training points, so the smallest k values refer to the k nearest neighbours.
Thus, we extract the smallest k norms from $\boldsymbol{N}$:

$$\{N_1, \ldots, N_k\} = min - k\{\boldsymbol{N}\}$$

and let the corresponding training points $\boldsymbol{x}_{N_1}, \ldots, \boldsymbol{x}_{N_k}$ *vote* for the output class ω:

$$\omega = mode\{x_{N_1,d+1} \ldots x_{N_k,d+1}\}$$

where the mode is an operator that extracts the most frequent value.
For the choice of the value of k, please refer to the section III.
Many variants exist (for example using a approximated, but faster, formula to calculate the distances) but they are not covered in this work.

B. Pros and Cons of kNN Classifier

Having no *training* phase, the cost for it is O(1): just for storing the training set. But, what we gain in learning phase, we pay in classification complexity: O(nd). In particular, the most critical phase is the computation of the min-k: we have to scroll the vector of the norms k times. For not small values of k it is more convenient to sort the vector before, but this require time, too (O(nlog(n))).
This kind of classifier is *theoretical guarantee*: for $n \to \infty$, the error rate is $\leq 2 \times BayesError$, where Bayes Error is the best theoretically achievable error.
However, in practice, errors can be consistent if the decision region have "jagged" borders. This is the case when the training points are near (so, very far to be *linearly separable*).

III. Our Work

Our work consists in the development of a k-Nearest-Neighbour classifier on a *MNIST data set*. So the goal is to give a classification of the handwritten digits.
This was implemented with three tasks:

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

A. Task 1

The data that we analyzed is a MNIST data set; the data is composed by 60000 instances, 784 attributes and 10 classes. Each instance represents a handwritten digit in 28x28 greyscale image.
The task 1 loads the data specifying if we want load the training set and the respective labels, or the test set and its labels.

B. Task 2

In this task we implemented the k-Nearest-Neighbour classifier. This was done applying the theory (**??**) the classifier is based on. In particular the task calculates the norms between all the training points and a query point. After, we must extract from this norms the k smallest ones: this was done ordering the norms vector and taking the first k values. With this method, we improve the computational time with respect to research k times the minimum as explained in the subsection II-B.
At the end, if the test set has number of columns equal or higher than the training set, the $d+1$ column (where d is the column number of the training set) is used to calculate the error rate between the obtained classifications and the real results.

C. Task 3

In this task, we computed the accuracy on the test set using the MNIST character recognition data. We checked the accuracy on 10 tasks: each digit *vs* the remaining 9, i.e. identify whether the observation has a class value or not. Also we controlled the correctness of the classifier for several values of k using the rule that k should not be divisible by the number of classes to avoid the ties in the voting mechanism.

IV. Experimental Results

The data, as described in III-A, has 60000 instances for the training set and 10000 observations for test set. Being the dimension of the sets very large, for this experiment we choose 1000 instances and 100 observations randomly, so we actually use a *condensed kNN*. The last columns of both data set are the classifications. We use the last column of the test set to compute the error rate (number of wrong classification / total number of observations).
We choose 13 different values of k (1 to 9, 11, 16, 21, 31) (number of neighbours) to test the classification. Please note that we have chosen them such that they are never multipliers of the number of classes (10) to avoid ties in the *voting* phase.

The chosen k are more ”condensed” for little values and more sparse for big values because we expect that differences decrease when we use bigger values.
Having the classifications for all test set, we build a One*vs*All matrix where each column x represent the problem *”Is this observation (the row) classified as class x or not?”*.
From this information, we compute also the *Accuracy*, the *Sensitivity*, the *Sensibility* and the *Precision*, to better judge the classification.

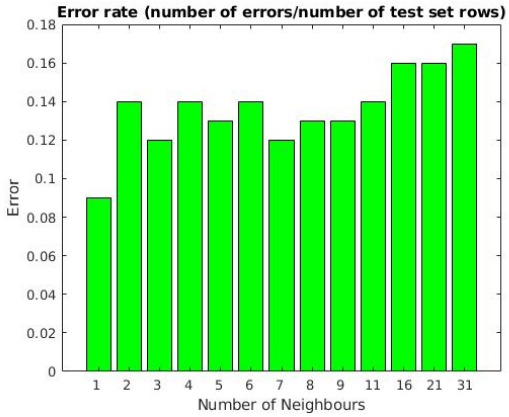


Fig. 1: *Error rate between prediction (classification) and real target*

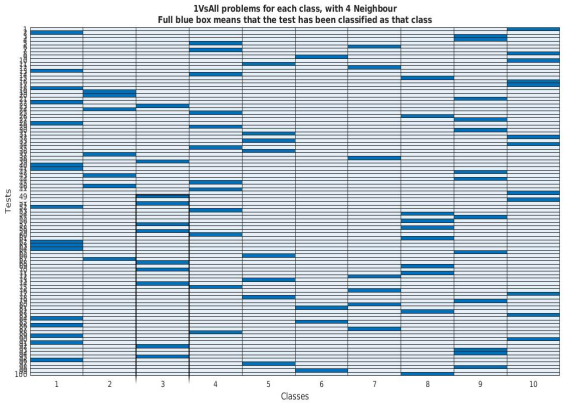
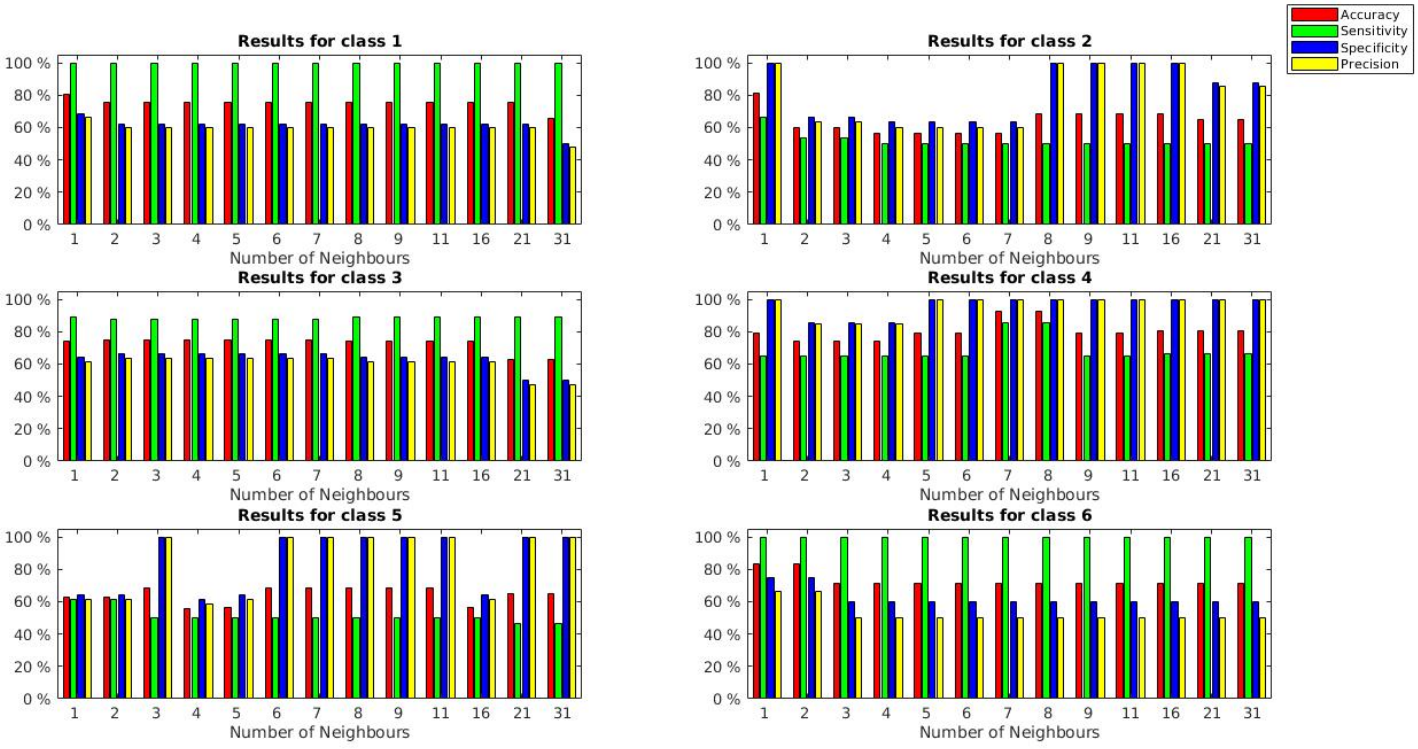


Fig. 2: *Heat-map describing One vs All problems*



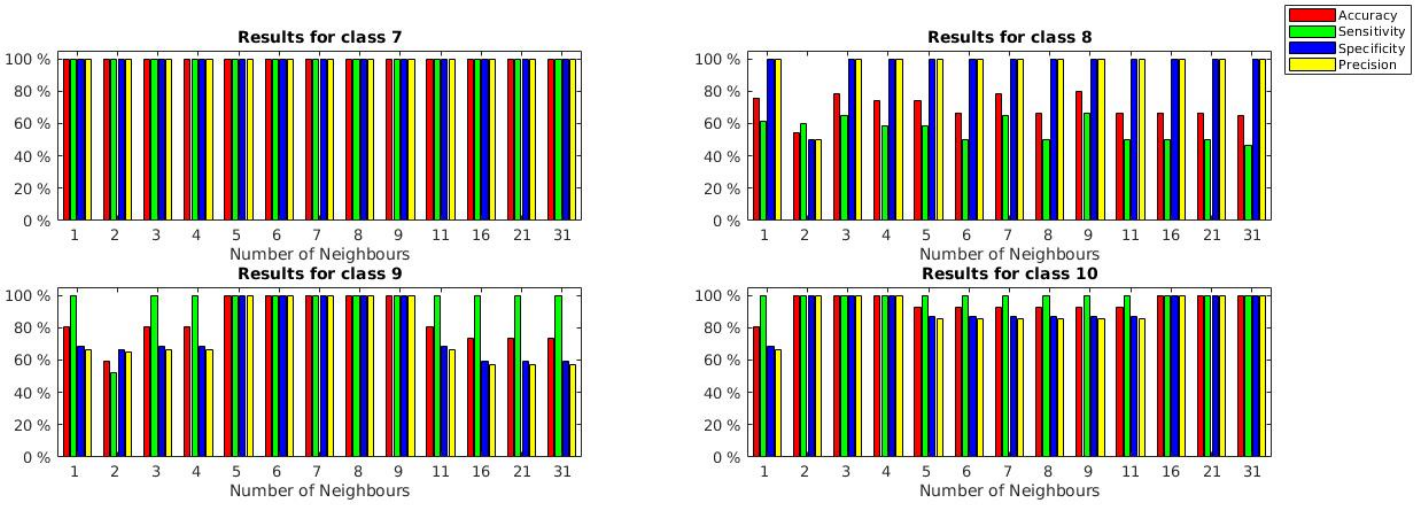


Fig. 3: *Accuracy, sensitivity, specificity and precision for each of the 10 classes*

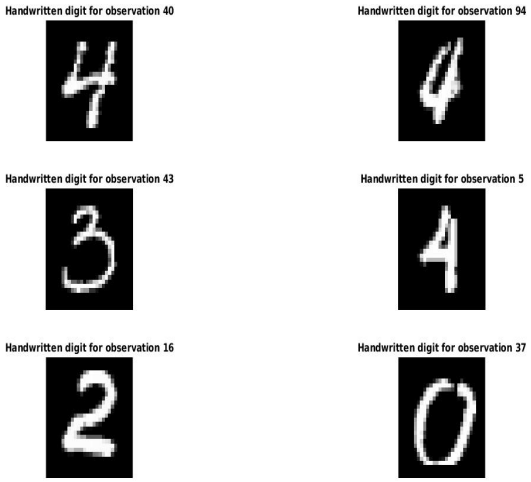


Fig. 4: *Handwritten digits (observations)*

	40	94	43	5	16	37
k=1	4	4	3	4	2	10
k=2	4	4	3	4	2	10
k=3	4	4	3	4	2	10
k=4	4	4	3	4	2	10
k=5	4	4	3	4	2	10
k=6	4	4	3	4	2	10
k=7	4	4	3	4	2	10
k=8	4	4	3	4	2	10
k=9	4	4	3	4	2	10
k=11	4	4	3	4	2	10
k=16	4	4	3	4	2	10
k=21	4	4	3	4	2	10
k=31	4	4	3	9	2	10

TABLE I: Classification for different k

As we can see in the plots, nice values of k (number of neighbours) are around 4, 5, 6. The error (Fig. 1) increases for values of k too big, because too many neighbours are taken and so they vote even if they should not because they are not so "near" values. Instead, little values of k bring to wrong classifications because "unfortunates" neighbours are taken, which vote for the wrong class.
The One*vs*All problems is described by a heat-map that shows for each observation (rows) if it is classified (dark blue) or not (light blue) as the specific class (columns) (Fig. 2).
The Accuracy, Sensitivity, Sensibility and Precision are values that we want high. As we can see in Fig. 3 in each class the better scores are in the middle values of k (4,5,6), in concordance to the plot of errors. It is important to notice that choosing the size of the testing set and training set can modify the results a lot. The greater the size of the training set is, the better the classification works. This because the procedure has more informations. But if it has too much size, it will take too much time to do all the distances.
The test set should be big enough to test well the classification. We found that choosing 1000 instances for the training set, and 100 for the test set is a good compromise, considering also that we chose 13 values of neighbours k.
In Fig. 4 six examples of the test set are shown. In the Tab. I there are the corresponding results (classifications) for different values of k.

It is interesting to notice the nature of the dataset: they are handwritten digits. Some of them are very easy to recognize and others can be confused for their shape.
For example, if we see the result (Fig. 3) for class one (digit 1), the sensitivity is, generally, higher that other classes and sensibility lower. This because it is easy to classify a true 1 as 1 (which means high sensitivity). But it is also easy to confuse other numbers as 1 because 1 has a simple shape, so it has more false positive than other classes (which means low sensibility).
Other classes, like 8, have a precise (different from others) shape, so if something is classified as 8, it is probably a true 8 (many true positive, high sensibility).