# Machine Learning Course

## Lab assignment 3: kNN classifier and evaluation

Augustin VIOT

## Contents

## Introduction

This document is a report of the lab assignment done in the course of Machine learning at the University of Genova (UNIGE). It describes the theory and implementations of a KNN classifier (k-nearest neighbors) and the implementation of some quality measurement.

## Theory explanation:

### Framework:

The goal of our exercise is to implement a classifier that will perform digit recognition. Tis classifier will be a KNN classifier. The data we are provided have 10 digits (from 0 to 9). At the end, we want to perform KNN classification for many k values for all binary classes (one class VS all the other). For all these classification, we want to compute the following quality indicators: error Rate (or accuracy), sensitivity, specificity, precision, recall, F1.

### How to determine the appropriate class?

The data is divided in two sets: training set (about 80%) and the test set (about 20%). To determine the computed target for each point from the test set we will follow the following algorithm:

1. Compute distance between the current point to classify and all the points from the training set
2. Select the k closest point based on the computed distances
3. The computed target is the most represented class among the k closest neighbors

### Comments about the KNN classifier:

The KNN classifier is known for being very simple to implement and understand and often giving better results than more complex classifiers.

# Implementation

## Code structure:

Our code contains 3 functions and one file with the main script. Here are the description of each files.

- *knnClassifier* :
  This function perform a KNN classification based on the two parameters it takes (*trainingDataSet* and *testDataSet*). It returns a vector containing the computed target, and the error rate of the classification (if the test set contains the target values). The code is divided in three parts: the check part (where we check the data and arguments given), the classification part (we apply the KNN algorithm), and finally the computation of the error rate (this part is done only if the test set contains the target).
- *knnClassifierWithAnalyze* :
  This function perform the same task as the knnClassifier but also compute and return a confusion matrix. This function only works with binary class!
- *analyzeConfusionMatrix* :
  This function analyze the confusion matrix given as a parameter and returns 6 parameters to evaluate the performances: error rate, sensitivity, specificity, precision, recall, F1.
- Main script :
  The main script (*exercice1.m*) of the exercise just call the different functions. It perform the tasks asked by the exercise. We will make a more precise description of the script below.

The main script (exercice1.m) of the exercise just call the different functions. It contains

- **Build dataset**

  We load the data and preprocess it using the *readdigits* function provided by the exercise. We obtain a matrix containing all the observations. Then we change the data to have only one target column containing the target as a digit (instead of 10 Boolean column). The next step is to sample the data in the two dataset (training: 80%, test: 20%).

- **Test the simple KNN classifier**

  For this task, we test the *knnClassifier* function and display the computed target and the error rate.

- **Test the KNN classifier with binary functions with fixed k, and for one class**

  We use the *knnClassifierWithAnalyze* function to perform this task, and the *analyzeConfusionMatrix* function to compute the quality indicators. (This task is commented in the code)

- **Test the KNN binary classifier for a fixed k and all class**

  Same task as before, but with one more loop to compute all the classes. (This task is commented in the code)

- **Perform KNN binary classifier for all the binary classes and many k values (exercise asked for the lab)**

  Same task as the previous one but with one more loop to compute all the k values. We also added another loop so the values are average, the number of iterations is by default `numberForAverage=1`.

- **Save the results of the KNN classifier (all classes, many k values)**

  Because the previous task may require a long computation time for `numberForAverage` high, we save the results so we can use it later to draw the graph without having to compute again. (Comment: in our files, we saved the matrix for `numberForAverage = 50` ).

- **build some graphic representation of the results**

  We use the saved matrix to draw some useful graphic representation.

For more details about the data structures and the algorithms used, please look the code comments.

# Results

## Analysis of the results:

All the graphs are available in the annexes:

- Annex 1: Error rate and F1 graphs
- Annex 2: Precision and Recall graphs
- Annex 3: Sensitivity and Specificity graphs
- Annex 4: Quality indicators averages

Comments: as mentioned earlier, all the graphs were built by computing the average of each indicators for 50 samples.

Firstly by observing the quality indicators we got from the classifications, we can at first say that the **classifier used (KNN) is a good classifier** for this data. The error rate is under 8% for every class for all the computed k values, and all the other quality indicators show good overall results.

Secondly, we can see that **the classifier does not perform equally for all the classes**. As we the graph show, the classes corresponding to digits “8” and “9” often have worst results than the other class. This shows that the KNN classifier has more trouble to classify these two digits correctly. This may be explain by how close the graphemes of these digits are. We can also notice that the digits “3” and “1” have a bad precision (fraction of positive outputs that was actually positive), which means that the observations are classified as “3” or “1” more often than they should.

Then the graphs show the influence of k for the quality of the classification. The overall interpretation tells us **that the higher k is, the worst is the quality of the classification**. We can see that because we see that when k increase most of the quality indicators show a decrease of the classification quality. This can be explain by the fact that many classes are too close to each other, so an increase of the k value cause an “overlap” of the target computed by the classifier.

Finally, one last observation can tell us that the quality indicators behave differently. Actually, all of them show a decrease of the classification quality and k increase, but the indicator **precision is the only one that increases** when the k value increase. This can be explain by the fact that we use binary classification. In fact, the higher k is, the more the classifier will classify an observation as negative (which means that the number of false negative will increase and the number of positive will increase, so also the percentage of correct positive).

## Possible improvements

### Other possible distances:

In our program we used the Euclidian distance to implement the KNN classification, but other distance computation could be used. As an example the Mahalanobis distance could be a better measurement since it tend to decrease the importance of the noise. An interesting experience would be to compare these two distance and see which one provide the best results. Other possible distances could be the Manhattan and the Minkowski distances.

## References

Useful references:

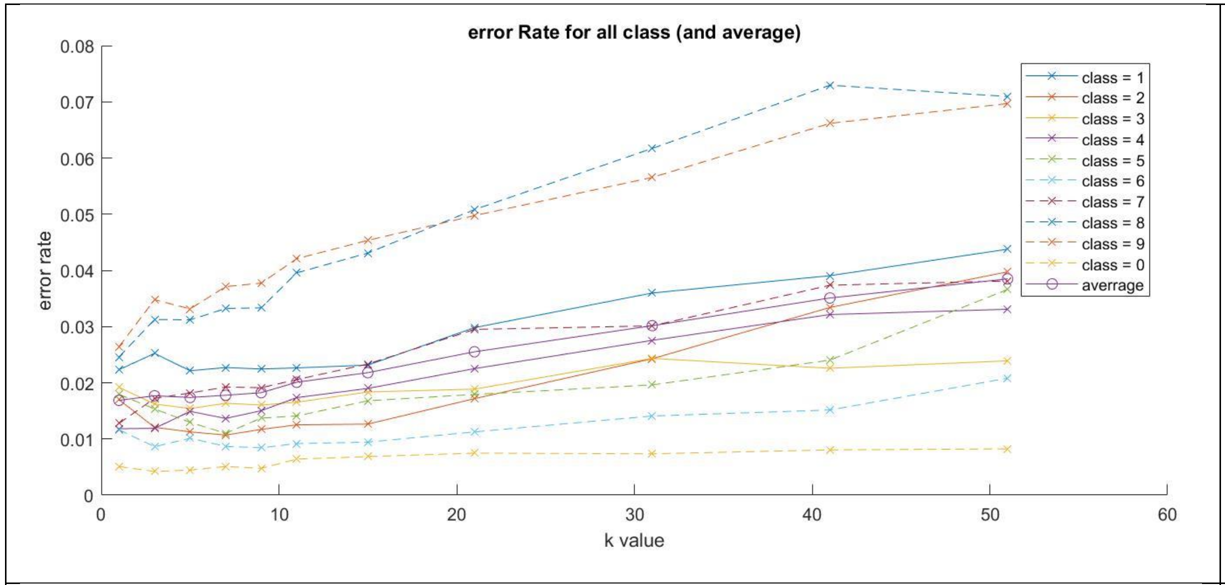
- Machine learning course
- “Pattern classification”, second edition, Richard O. Duda, Peter E. Hart, David G. Stork

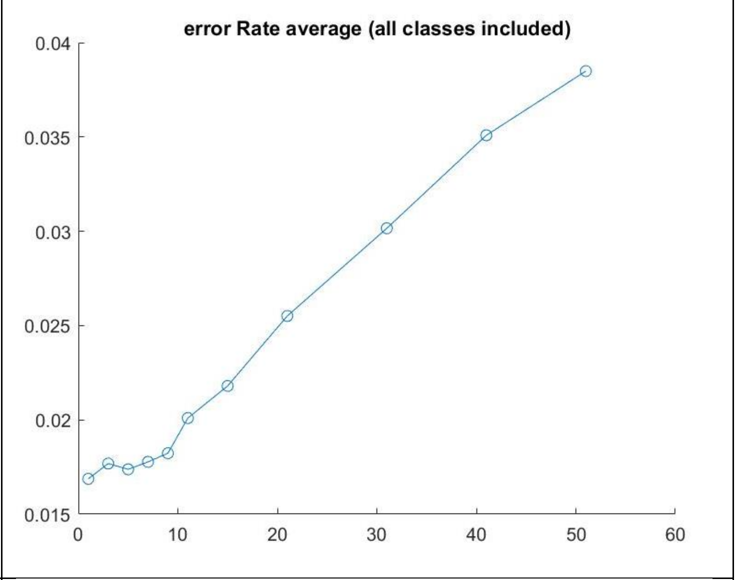
Other useful references:

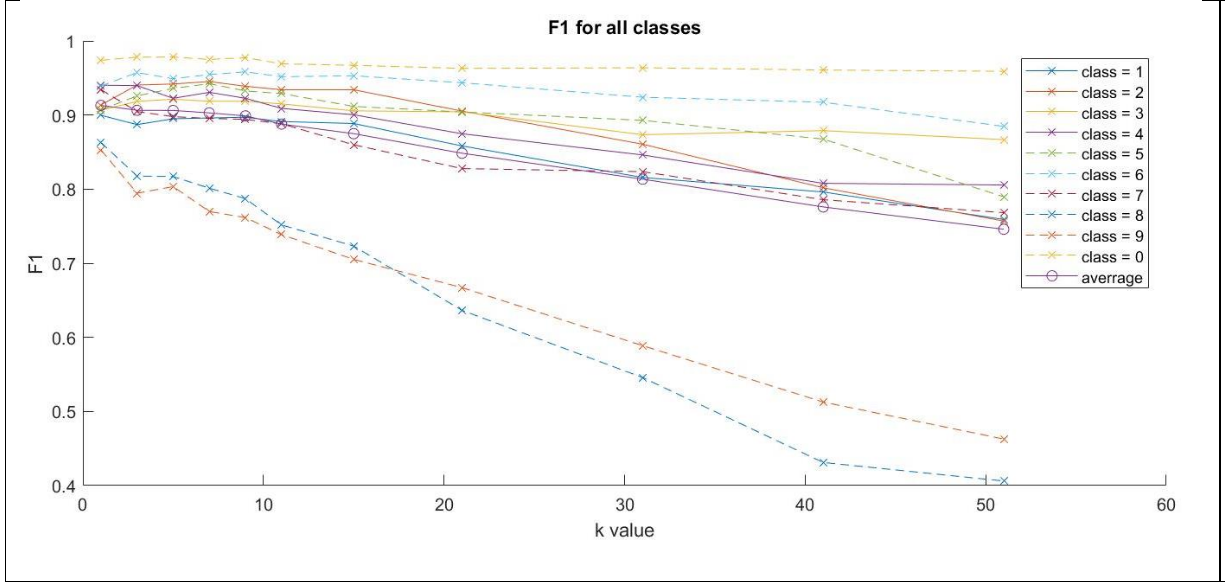
- https://www.youtube.com/watch?v=v6278Cjf_qA
- https://www.youtube.com/watch?v=4ObVzTuFivY&t=234s

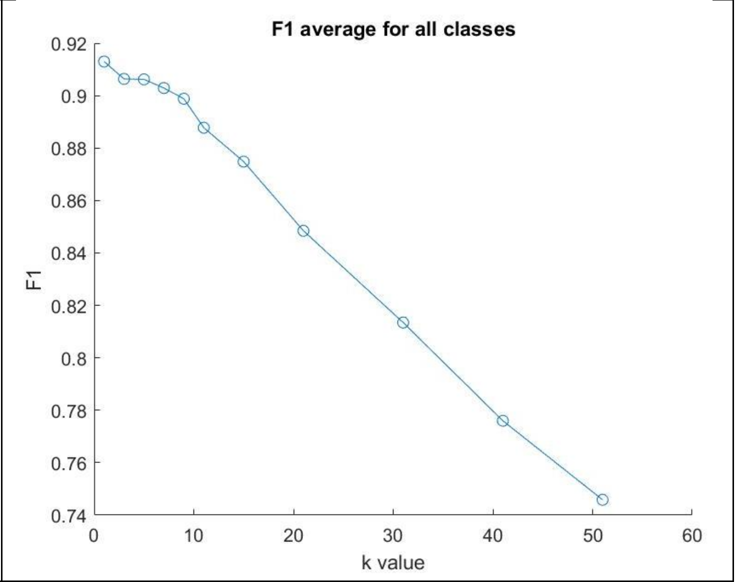
## Annexes:

## Annex 1: Error rate and F1 graphs

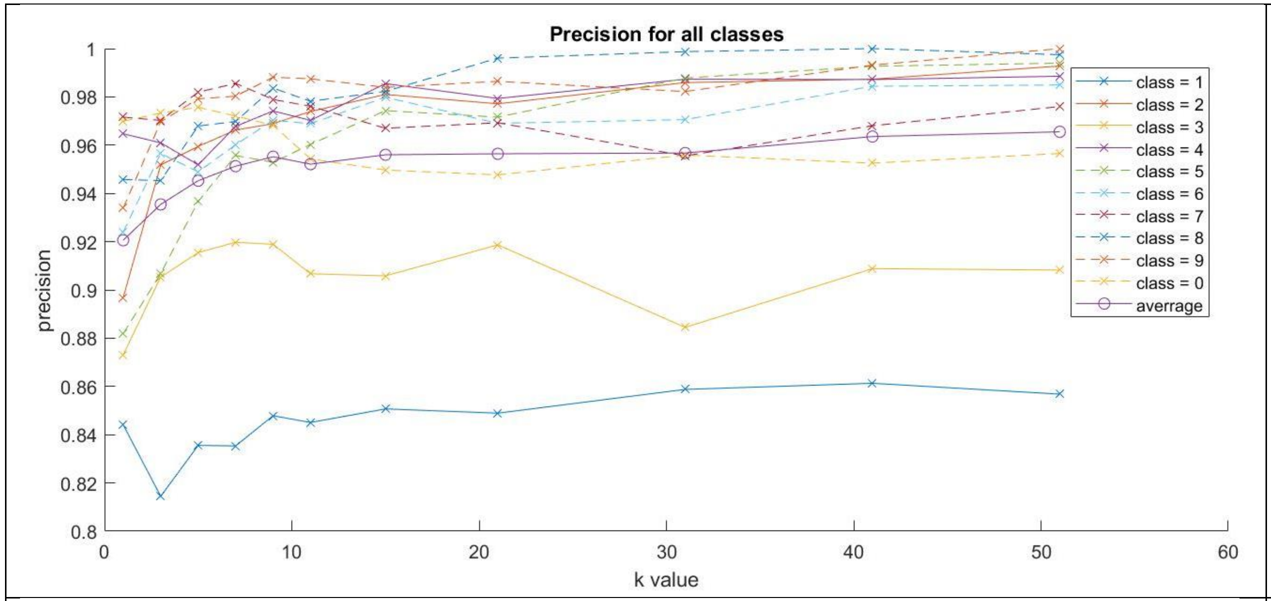


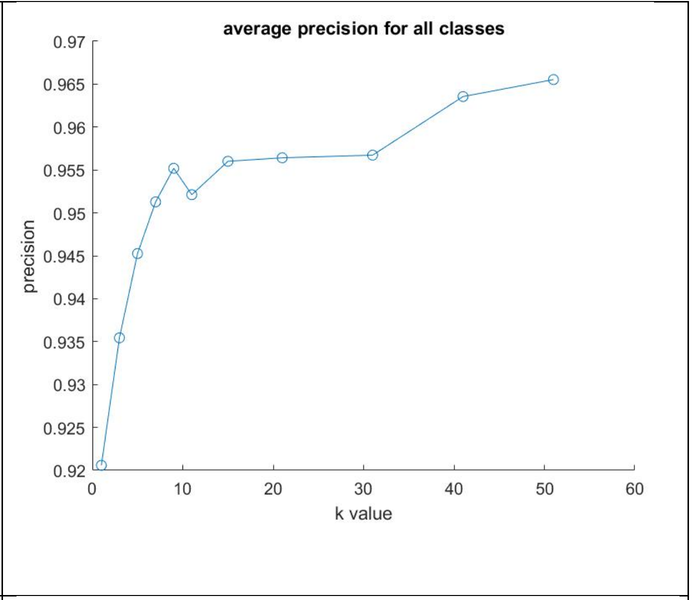


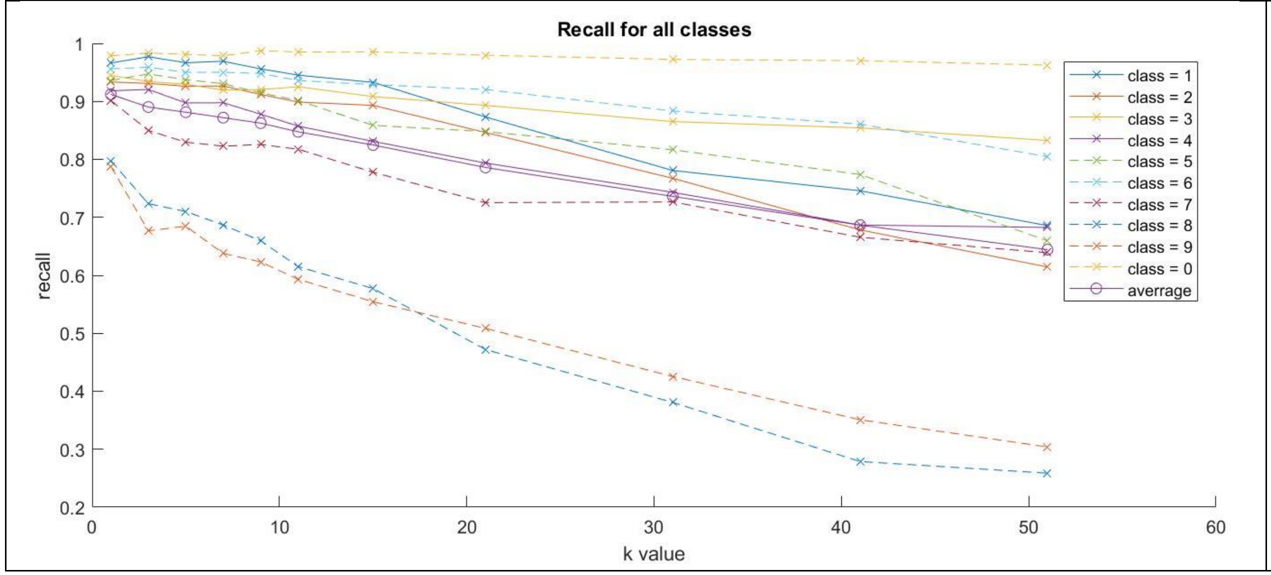


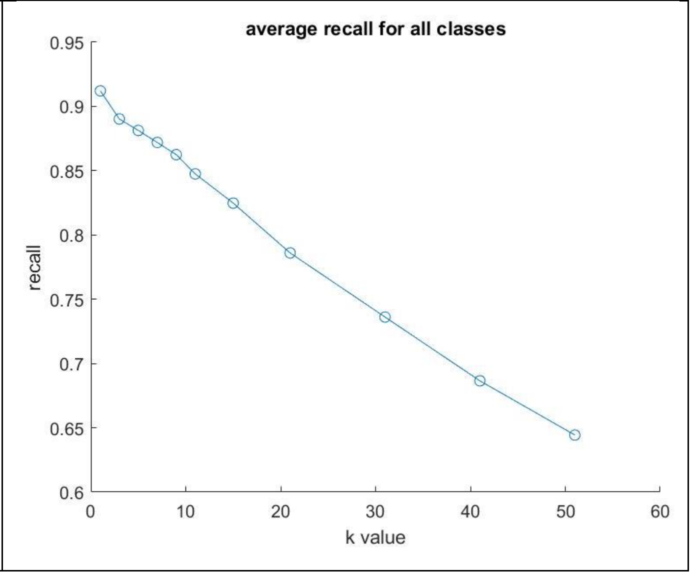


## Annex 2: Precision and Recall graphs

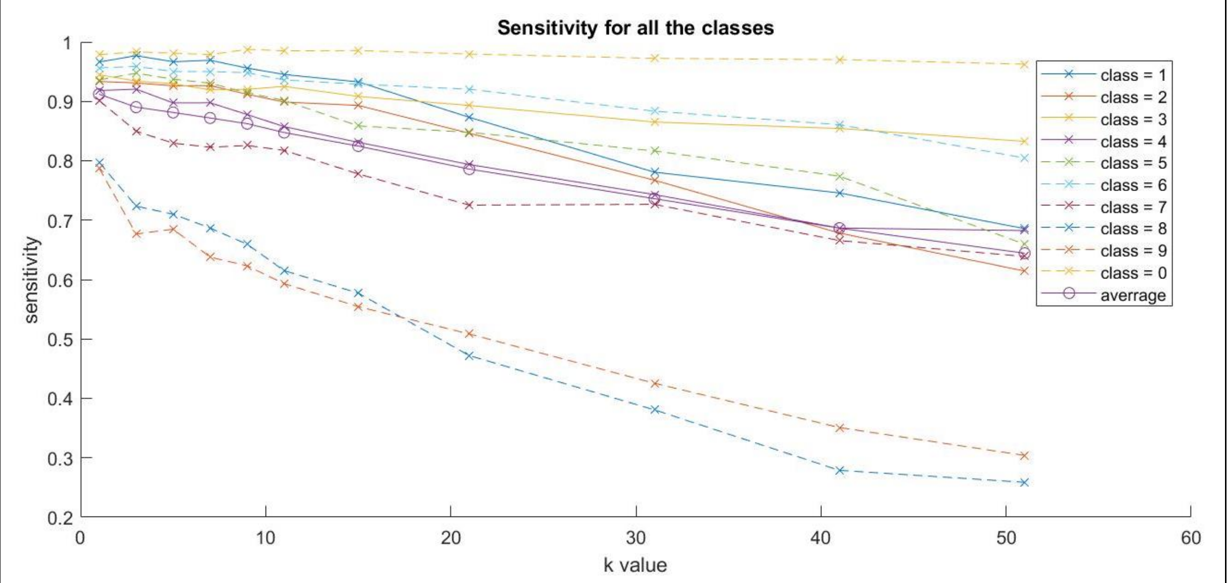


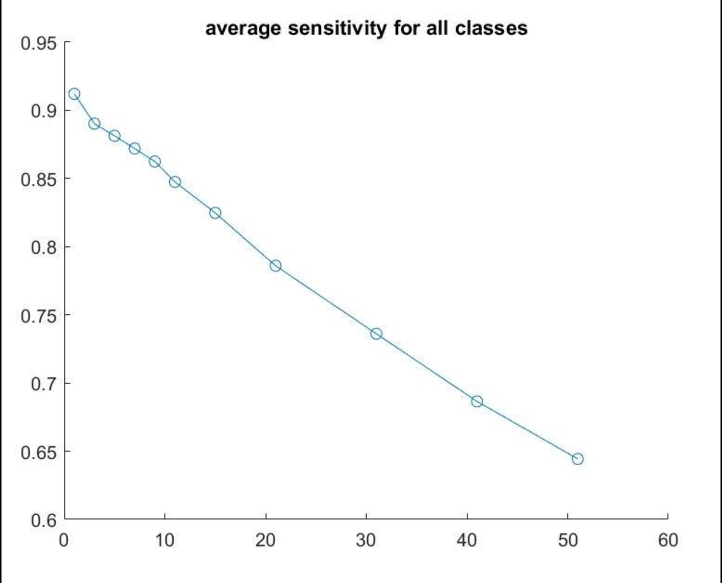


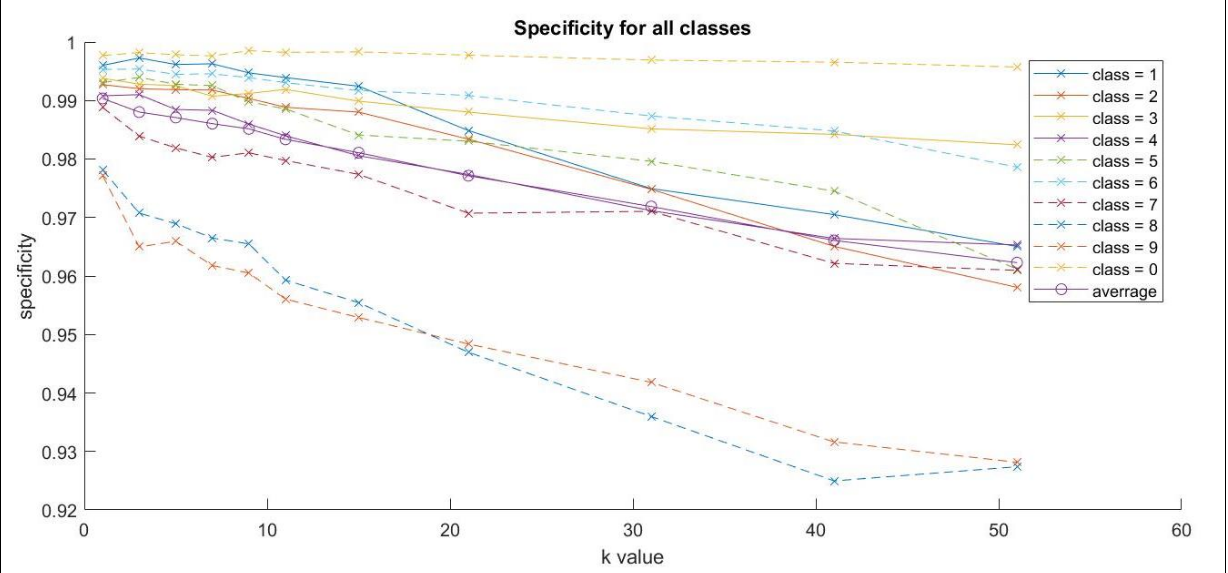


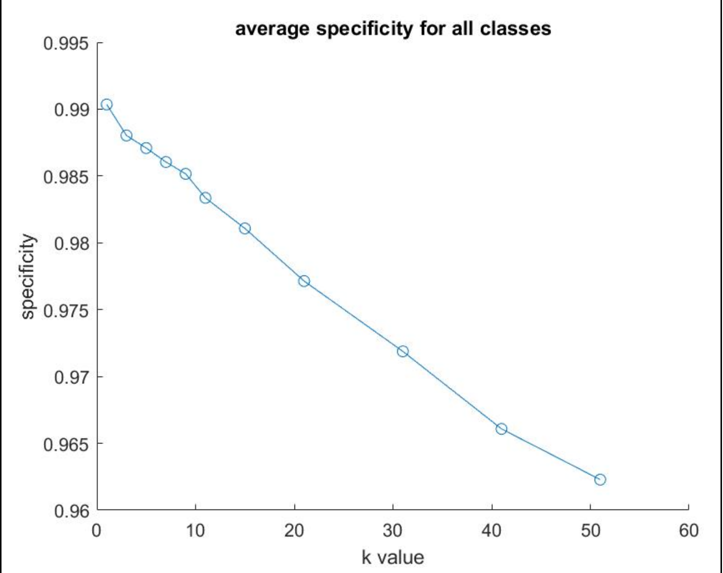


## Annex 3: Sensitivity and Specificity graphs

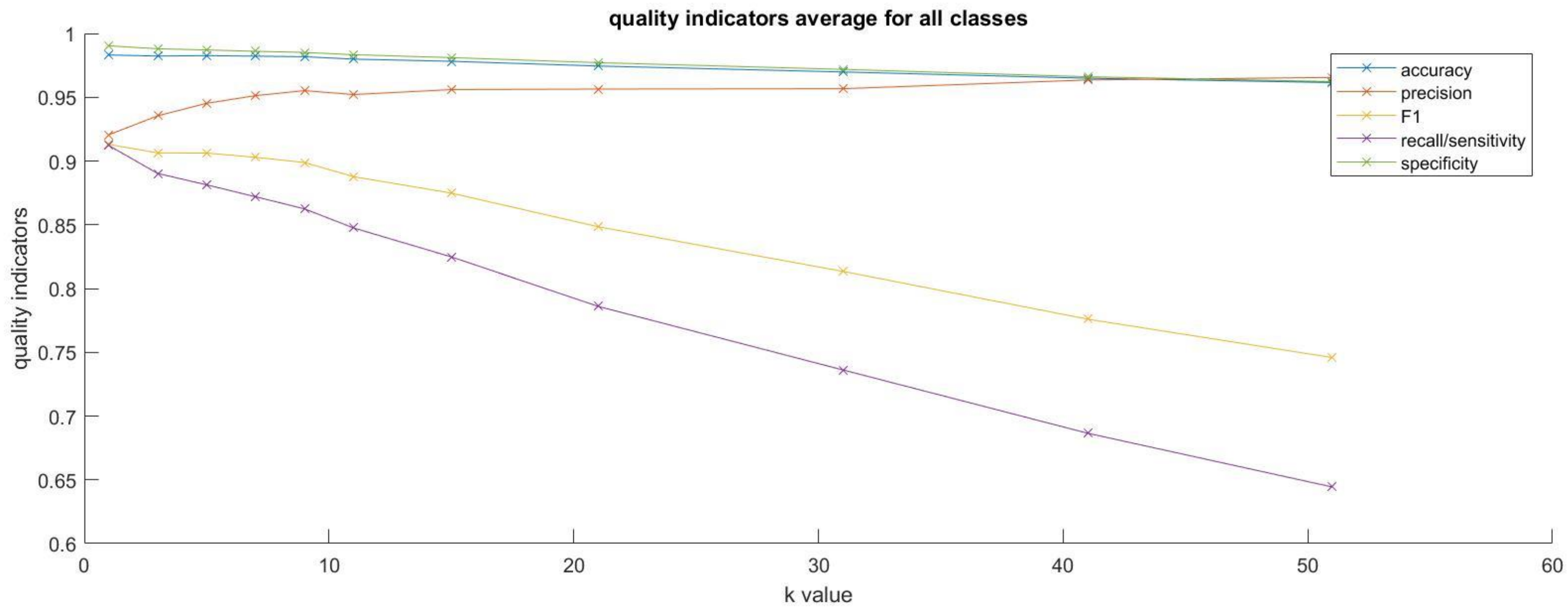








## Annex 4: Quality indicators averages


quality indicators average for all classes
quality indicators
k value
accuracy
precision
F1
recall/sensitivity
specificity