ROB311 - TD1 - KNN

BRAMBILLA Davide Luigi - GOMES DA SILVA Rafael

16 September 2019

In order to execute the program the command to run is *pyhton TD1.py*.
The code is available on the git lab repository:
https://gitlab.data-ensta.fr/gomesdasilva/rob311.git

Introduction

The *k-Nearest-Neighbours*(kNN) algorithm in *Machine Learning* is used for the classification of samples. This algorithm takes as input a set of *already classified* samples and a ensemble of samples that we want to classify. The output is the ensemble of the classifications given to the samples. Since the classification (*label*) for each sample is already known, it means that the problem to be treated is a problem belonging to the *Supervised Learning*.
The algorithm developed was implemented with different parts:

- The part of reading and analysing the data from the data sets in order to split the characteristics from the labels
- The part of splitting the data between the ones that will be used to test the algorithm and the ones that will be used to test it
- The classification with the method of *k-Nearest-Neighbours*
- The study of the *performances* and the *plot* of the test data

The Data Sets

The two data sets used to verify the performance of the *k-NN* algorithm were the Breast Cancer Wisconsin (Diagnostic) data set and the Haberman's Survival data set.

Breast Cancer Wisconsin (Diagnostic)

This data set is related to different factors that permit to classify a cancer between *benign* or *malignant* and reports 11 characteristics for each sample. The

first one represents the ID code associated to the person. The last one represents the cancer classification: benign or malignant. The rest of the data gives information on the status of the nuclei of person analysed. These characteristics take a value between 1 and 10, where 1 represents a value that brings to a benign cancer detection and 10 brings to a malignant cancer detection.

Index	Attribute	Domain
0	Sample code number	Id number
1	Clump thickness	1 - 10
2	Uniformity of cell size	1 - 10
3	Uniformity of cell shape	1 - 10
4	Marginal Adhesion	1 - 10
5	Single epithelial cell size	1 - 10
6	Bare Nuclei	1 - 10
7	Bland Chromatin	1 - 10
8	Normal Nucleoli	1 - 10
9	Mitoses	1 - 10
10	**Class**	2 or 4[a]

[a] 2 = the person has a benign cancer - 4 = the person has a malignant cancer

Haberman's Survival

The data set contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for *breast cancer*. The first three parameters represent the characteristics of the patient at the moment of the operation. The last characteristic is the class where the patient is classified 5 years after the operation.

Index	Attribute	Domain
0	Age of patient at time of operation	Numerical
1	Patient's year of operation	Numerical
2	Number of positive axillary nodes detected	Numerical
3	**Survival status**	1 or 2 [a]

[a] 1 = the patient survived 5 years or longer - 2 = the patient died within 5 year

The Algorithm

The Read and Analysis of the data

The first part of the algorithm is dedicated to read the data set and to split them into different vectors related to their nature.
Two different functions were created in order to adapt them to the two data sets: *load_data_winsconsin()* and *load_data_haberman()*.

The objective of *load_data_winsconsin()* is to split the data in order to separate the 11 attributes in three different groups: one *vector* composed by all the *Id_numbers*, one *vector of vector* composed by all the characteristics, named *x_data*, and the last one that will be a *vector* containing the class attributed to each sample, named *y_data*.

Knowing that in the data set there are 16 characteristics that are missing for the characteristic *Bare Nuclei*, it was chosen to eliminate the sample where the information is missing: this operation doesn't affect the data set because the samples having elements missing represent only the 2 % of the entire data set.

The objective of *load_data_haberman()* is the same of the previous code and it allows to obtain two different vectors: *x_data* that contains all the characteristics of the samples and *y_data* that contains the classifications of the samples.

The Split between *Train data* and *Test data*

The second part of the code aims to divide the data set in two different parts: the data that will be used to *train* the *kNN* algorithm and the data that will be used at the end to *test* it. In order to do so, a percentage of values that will compose the *test* part is chosen, and we eliminated them from the vector *x_data* and *y_data* and they are added them to the vectors *x_test* and *y_test*. This way, it is possible to get, instead, *x_train* and *y_train* directly from the remaining *x_data* and *y_data*.

At the end of this part there will be the four vectors *x_test*, *y_test*, *x_train* and *y_train* ready to be treated by the *kNN* algorithm.

The *k-Nearest-Neighbours* Implementation

In this part of the code the *kNN* algorithm is implemented with the goal to classify each element of the test data.

For each element of the *test* data it is calculated the distance between the element and all the samples that belong to the *train* part. Then the values are

reorganized in an array in order to have the nearest neighbours in the first positions. Finally only the first k elements of the array are selected and the current sample is classified: The classes of the k-nearest neighbours is analysed and the class that appears the most is selected. This way, the class of each sample is chosen using the *majority vote*.

The function *calculate_distance* was implemented, and this function calculates the distance between two given points: at first the method used to calculate the distance was the *Euclidean* distance. Later, other forms to calculate the distance were added, using the *Manhattan* distance and the *Chebyshev* distance.

In order to implement the *majority vote*, the function *count_most_common* was implemented, that gives as output the predicted class for the sample that is currently under treatment.

The *Accuracy* and the *Confusion Matrix*

For the algorithm, the *accuracy* value and the *confusion matrix* were also calculated.

For the *accuracy* value, a comparison between the classes assigned to *x_test* contained in the vector *y_pred* and the real classes contained in *y_test* is made. For each right prediction a counter is incremented and, at the end, the value obtained is divided by the total number of *test* data. In this way a *percentage* of the right predictions is obtained.

For the *confusion matrix* an algorithm was implemented, that works for a generic data set: in particular in this case the output is a 2D matrix containing the number of samples that has been assigned correctly to the class and the number of samples that are not correctly classed. By analysing the confusion matrix calculated by the function *conf_matrix*, it was possible to verify that its result is the same as the one given by the function *confusion_matrix* imported from the *sklearn* library.

The results showed an high accuracy for the *Breast Cancer* database since the data are well separated: the class to which they belong strongly influence their characteristics and the classification from the features results easier and very precise (around 95%). On the other hand, the results showed a lower accuracy for the other database since the distribution of the data is more various(around 65%).

We have reported below the different values that have been obtained in the algorithm changing the parameters:

Changing the k value

The k value permits to choose how many neighbours will be taken into account in the final vote to assign the class: in particular it is expected that the accuracy and the performances of the algorithm will increase when the k is increasing since the classification will be based on more samples.

For the value of k, only impair values of k were used in order to eliminate the ambiguity problem when having the same number of neighbours belonging from different classes: in this case, since the two data sets are composed only by two classes, by setting an impair value of k it is possible to totally eliminate this ambiguity on the result.

The results below show the results of the algorithm for three values of k and the tables below show the three value of accuracy obtained and the three *Confusion matrices*(**BC** representing the *Breast Cancer* data set and **H** representing the *Haberman's* data set):

k	Accuracy - BC (%)	Accuracy - H (%)
3	94,28	62,27
5	95,58	65,93
7	95,75	65,20
Mean Value	95,09	64,46

k	Confusion Matrix - BC	Confusion Matrix - H
3	118 5 4 77	47 24 14 6
5	129 6 4 65	52 11 16 12
7	121 7 6 70	47 16 19 9

Changing the percentage of the test data

The percentage of the test data permits to change the quantity of data that will be used to train the algorithm and to test it: in particular it is expected that the performance of the algorithm will increase when the test data percentage is decreased. The tables below will present the results for a $k = 5$.

Test Data %	Accuracy - BC (%)	Accuracy - H (%)
30%	94,44	63,01
15%	94,77	64,44
5%	96,07	77,77
Mean Value	95,09	68,41

Test Data %	Confusion Matrix - BC	Confusion Matrix - H
30%	126 5 5 68	46 25 9 11
15%	63 5 0 34	21 11 5 8
5%	21 1 0 12	8 3 1 3

Changing the method to calculate the distance

Since the *kNN* algorithm uses the concept of distance in order to classify their samples, an additional analysis was made, by change the function of the distance in order to see if the results will be influenced by the different methods tested.
The three different methods to calculate the distances were: the *Euclidean* distance, the *Manhattan* distance and the *Chebyshev* distance.

Euclidean distance It is the most common distance and, for a two given points x_1 x_2 with different characteristics, its formula is:

$$d(x_1, x_2) = \sqrt{\sum (x_1(i) - x_2(i))^2} \tag{1}$$

Manhattan distance This distance is similar to the *Euclidean* one. It is also known as *rectilinear distance*and it is defined as the sum of the lengths of the projections of the line segment between the points onto the coordinate axes. Its formula is:

$$d(x_1, x_2) = \left|\sum (x_1(i) - x_2(i))\right| \tag{2}$$

Chebyshev distance This distance is defined in vector spaces and tells that the distance between two vectors is the maximum value of their difference along the axes. Its formula is:

$$d(x_1, x_2) = max(|x_1(i) - x_2(i)|) \tag{3}$$

Test Data %	Accuracy - BC	Accuracy - H
Euclidean	97,05	68,13
Manhattan	98,52	62,63
Chebyshev	96,07	64,83
Mean Value	97,21	65,19

Test Data %	Confusion Matrix - BC	Confusion Matrix - H
Euclidean	123 1 5 75	51 18 11 11
Manhattan	127 3 0 74	46 20 14 11
Chebyshev	133 3 5 63	48 17 15 11

The data representation

In order to visualize concretely the behaviour of *kNN* with the data, the plots in 2D and 3D were analysed: the goal was to put in evidence the right and the wrong classification of the data of test.

In particular, for the first database (*Breast Cancer*), the samples have 9 characteristics so it was chosen to represents 2 characteristics per time obtaining in this way some 2D plots: in each plot the test data and classifying was reported using a different color. The samples that have been assigned to the wrong class were also differentiated.

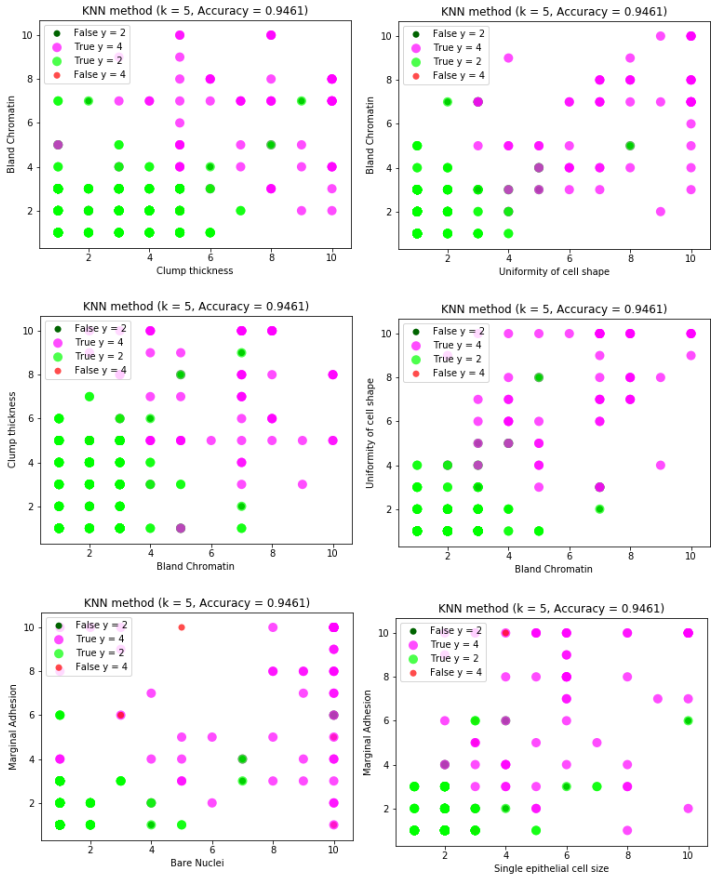

Figure 1: Data representation for database *Breast Cancer*

Analysing the results, it is possible to verify that the classification works well, as the accuracy told us.

For the other data set, since the samples are characterised by only three features both 2D and 3D plots were plotted reporting, like in the previous case, the classification of the data and the data which classification was not corrected.

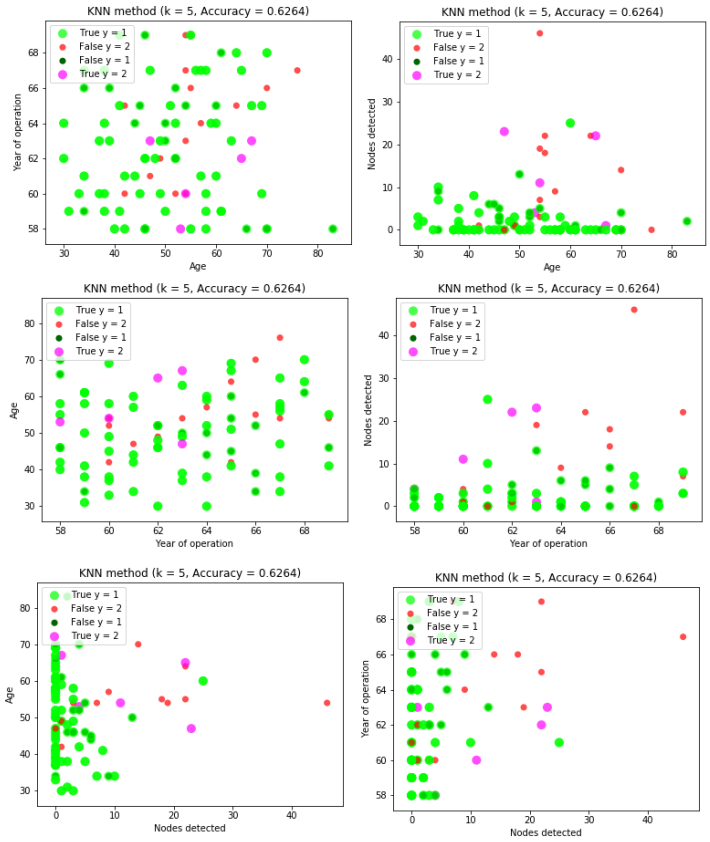

Figure 2: 2D data representation for database *Haberman*

It is possible to see that the classification works worse than in the previous case since a *non-negligible* number of samples is classified in the wrong class.

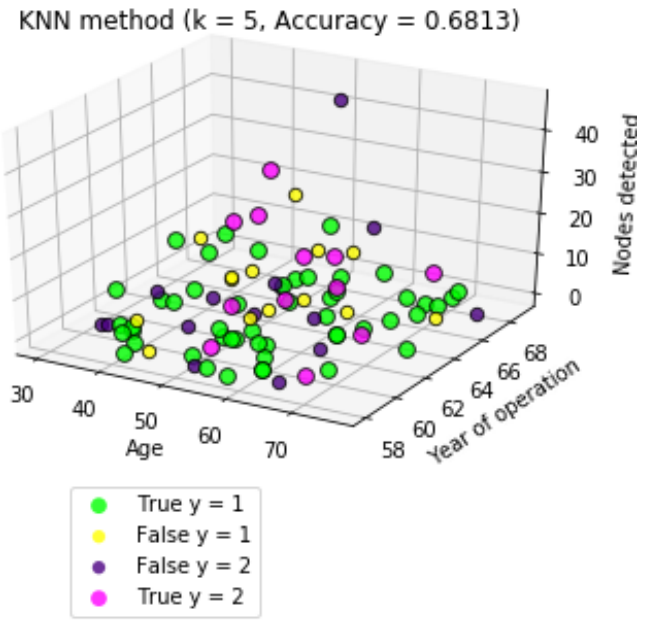

Figure 3: 3D data representation for database *Haberman*