Data Mining Project

Classification with KNN and KNN+

Author: Leandro Pereira Almeida, K-5600

INDEX

Introduction

The aim of this project is classification with KNN (k-nearest neighbour) and KNN+. This classification can be done in numerical, categorical or mixed (numerical and categorical) datasets. To these type of datasets, different measures to compute distances are used. In this case was used Euclidean distance to compute distances (in numerical datasets) and *dGower* distance (to mixed dataset).

The difference between KNN and KNN+ is that KNN must consider only K neighbours, however in KNN+ the number of chosen neighbours can be greater than K (if the distance from the sample classified and the K neighbour is the same as the K+1, K+2..). In KNN+, the result to the sample *p* is the set of all points that are distant from *p* by no more than any k-neighbour of *p*. After finding the K (or more in the case of KNN+) neighbours, to classify the sample is needed to find the class in majority in the neighbours found.

In order to understand better the difference between KNN and KNN+ is illustrated an image (figure 1). To classify the 3 neighbours of the sample *p*, with KNN it will return the samples *z,v,q* or *z,v,w.* Nevertheless, with KNN+ it will return the samples *z,v,q,w* (once the distance *p - w = p - q).*

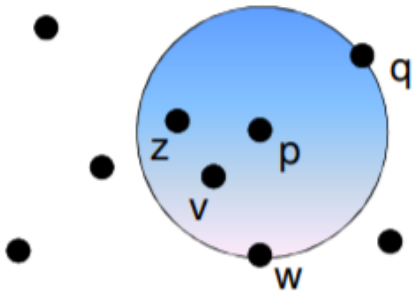

Figure 1: Example of classification with KNN+. Taken from [1]

In all the decisions made in the program, I have tried to optimize the temporal and memory complexity, that is going to be analyse later in this report.

There are a lot of different useful functionalities that user can use in the program.

The implementation of the program was accomplished in **C language.**

Input data and user guide

The program is prepared to all types of numerical datasets (independently of the number of features and samples). The values must be separated by spaces (not commas), and in the first row must appear the name of the features (each one in a single word). The user can specify the dataset that he wants to evaluate when runs the program. The program works also to a mixed (categorical + numeric) dataset. In this case the number of features has to be the same as the dataset used to test the program.

To create the executable is only needed to put $*make* in the command line.

The user has got a lot of options to perform the program. The way that the program runs depend on these decisions. User has to run the program in the command line and has to run with some flags. Each flag specifies a different option.

- **Dataset file (flag -f) -** User must specify the dataset to be used. It is a mandatory input. **-f car.txt** indicates that the dataset is the file *car.txt*.
- **Type of algorithm (flag -a) –** User can specify the algorithm to be used (KNN – > 0 or KNN+ -> 1). This is a mandatory parameter. **-a 1** indicates that KNN+ will be carried out.
- **Number of neighbours K (flag -k) –** User can define the number of neighbours to be used. If he does not define, it is used the number log(N), where N is the number of samples. If he wants k = 5, he must specify **-k 5.**
- **Type of dataset (flag -t) –** User must inform what is the type of dataset. If is numerical is 0 (he must specify **-t 0**). If is mixed dataset is 1. The default value is the numerical one.
- **Classification samples (flag -c) –** User can choose to classify only one sample that is part of the dataset or can classify samples that are not part of the dataset. He can also let the program decide. In this case, the program will choose randomly 1/3 of the number of samples of the dataset.
    1. **-c 0 indexOf_the_sample** will classify the sample indexOf_the_sample that is inside the dataset (**-c 0 0** will classify the sample with index 0).
    2. **-c 1** will classify 1/3 of the dataset. The samples to be classified will be choose randomly.
    3. **-c 2** will perform Leave-One-Out method. All samples will be classified (test set just with one sample) and all the remaining will be training set.
    4. **-c 3 textfile.txt** will classify the samples that are in *textfile.txt*
- **Normalization data (-n) –** If user uses this flag, the data will be normalized before the classification. The default value is without normalization.

In the figure 2 is depicted an example of how to run the program.

```
/mnt/c/Users/leand/Desktop/dataMiningProject/KNN-KNN-and-VPTree$ ./main -a 1 -f DataSets/categNumTeste.txt -t 1 -c 1 -k 2 -n
```

Figure 2: Running the program – example 1

In this case the algorithm to be used is KNN+ with the file categNumTeste.txt (in folder DataSets), the type of dataset is categoric+numeric, the samples to be classified will be randomly chose (1/3 of the dataset). The value of K is 2, and the data will be normalized.

Another example is shown in figure 3.

```
leandroljpa@LAPTOP-JUNB6O12:/mnt/c/Users/leand/Desktop/dataMiningProject/KNN-KNN-and-VPTree$ ./main -a 1 -f DataSets/categNumTeste.txt
```

Figure 3: Running the program – example 2

In this case has the values were not specified, the program will be performed with the default values.

It was used small datasets to confirm that everything is working (computation of the distances, as well as the choice of neighbours) and to be easier to analyse, and it was carried out. Then there were performed experiments to different datasets (with a lot of samples and different number of features). In both datasets the spaces between the attribute values were removed.

```
:/mnt/c/Users/leand/Desktop/dataMiningProject/KNNandKNN+$ ./main -a 0 -f DataSets/teste.txt -t 0  -c 3 testset/test.txt -k 3
```

In this case the program will classify the samples that are in *testset/test.txt* (test set), with the training set that is inside the file *DataSets/teste.txt.*

Output Data

The results are written in an output file that is inside the folder "Results". If the algorithm used is KNN, the results are written in Results/KNN_'datasetName'_K=3.txt, to the case of K=3. Otherwise is written in Results/KNN_PLUS_'datasetName'_K=5.txt, to the case of K=5.

Next figure is an example of the output file.

KNN_PLUS.txt - Bloco de notas

Ficheiro Editar Formatar Ver Ajuda

```
4-nearest Neighboors of the sample number: 0, with feature values:    vhigh       vhigh       2       2       small       low       unacc
    1. Sample nr: 1, with distance: 0.166667
feature values:
   vhigh       vhigh       2       2       small       med       unacc


  2. Sample nr: 2, with distance: 0.166667
feature values:
   vhigh       vhigh       2       2       small       high       unacc


  3. Sample nr: 3, with distance: 0.166667
feature values:
   vhigh       vhigh       2       2       med       low       unacc


  4. Sample nr: 4, with distance: 0.166667
feature values:
   vhigh       vhigh       2       2       small       med       unacc


  THe sample if classified as class =  unacc
```

Figure 4: Output file - example

In this example the 4 neighbours of the sample number 0 are: sample number 1,2,3 and 4. As all of them belong to the class *unacc* the sample is classified as class *unacc*.

Program Architecture

The flowchart depicted in the figure below shows the architecture of the program and has information on it works.

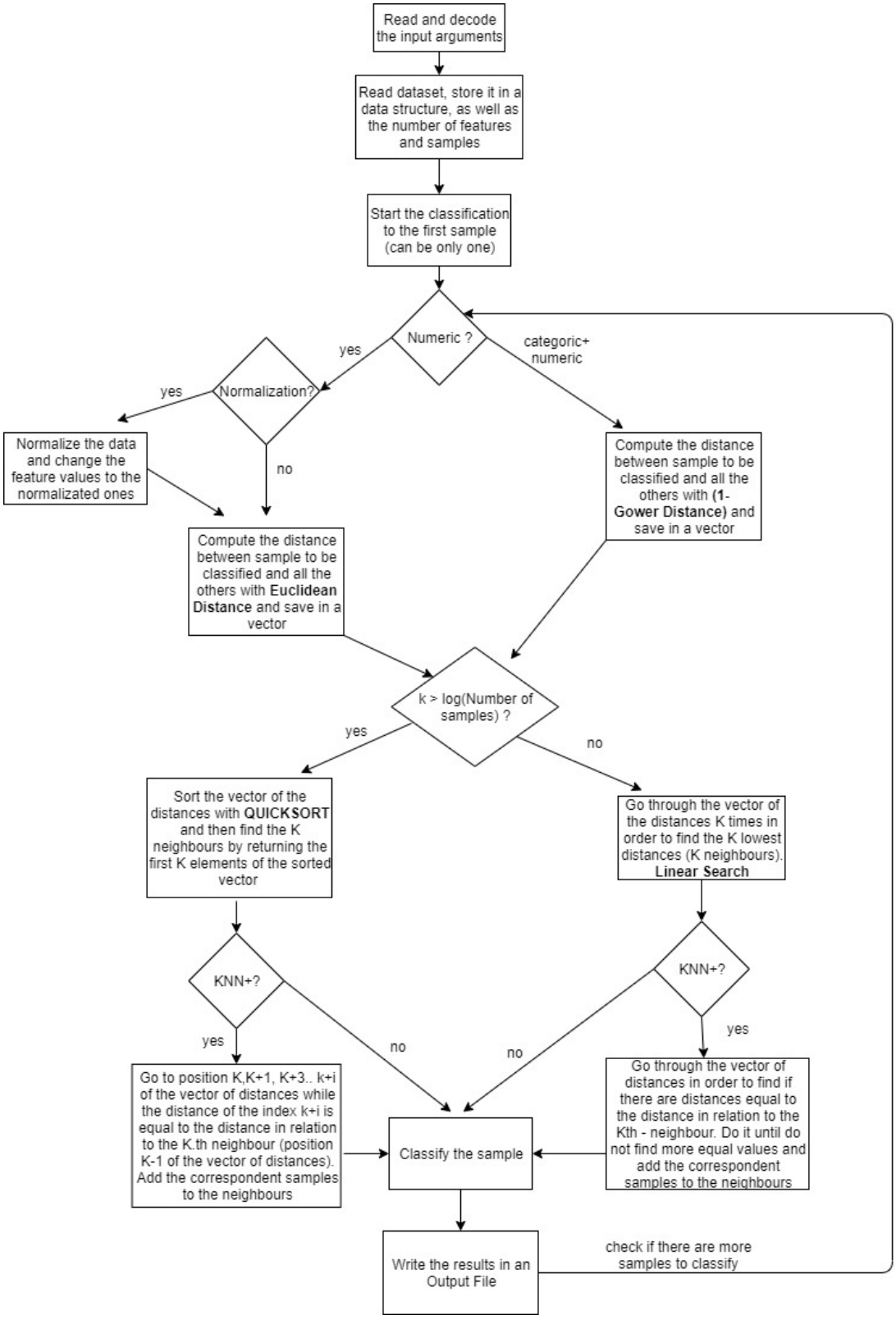

Figure 5: General architecture

Description of the data structures

In order to describe the data structures used during the program, is important to define:

1. N – number of samples
2. F – number of features
3. K – number of neighbours

The main data structures used are:

- **dataset structure with the fields:**
    1. **matrix** with dimension NxF (memory complexity is O(NxF))that stores all the dataset in memory;
    2. **number of samples** – integer that stores the number of samples of the dataset;
    3. **number of features -** integer that stores the number of features of the dataset;
- **distances vector –** vector of type float with size N (memory complexity is O(N)). This vector stores the distance between the sample classified and all the other samples. For example, if the sample to be classified is the sample 0, the position 5 of the vector represents the distance between the sample 0 and 5;
- **neighbour vector –** vector of type int with size K (in the case of K++ the size is bigger). Memory complexity is O(K). This vector stores the index of the nearest neighbours. For example, if the nearest neighbours of the sample 0 are 2,5 and 8 the position 0,1,2 of the vector will be 2,5 and 8, respectively.
- **Variables of type int** – that stores the options of the program (for example type of algorithm, number of K, type of dataset).

Implementation and complexity analysis

- **Read and decode the input arguments**
  The first step of the program is to decode the information given by the user in command line. This way the program knows which algorithm he intends, as well as which samples is supposed to classify, the dataset to analyse.
- **Read dataset and store it in data structures**
  Since file reading is very fast, the dataset is read twice. The goal of the first one is to know the number of samples and the number of features, then the data structures are allocated with the correct size. This way, user does not have to care about the number of features and records, once the program compute it automatically. In the second time all the dataset is stored in the data structures allocated. The temporal complexity to read and store the dataset is *O(NxF).*
- **Normalization**
  In order to normalize the data, it has to go through all the samples and for each feature check what is the maximum and minimum value. And then go through all the samples again and change the value to the normalized one. The normalized value is given by:

$$\text{Normalized} = \frac{initial\_value - \min\_value}{\max\limits_{value} - \min\limits_{value}}$$

  Where *max_value and min_value* are computed to each feature (each one has its own).

The temporal complexity of this process is *O(NxF),* once it has to iterate over all dataset. The results are changed in the matrix that stores all the dataset to the normalized values.

- **Compute distances**

As this program is prepared to two different type of datasets (numerical and numerical+categoric) there are two different method to compute distances. The first one is to numerical datasets – Euclidean distance. The Euclidean distance between *x* and *y* is given by:

$$d_E(x,y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

The second one is to mixed types -dGower distance. This distance can be used to compare objects with attributes of mixed types. In the standard dGower measure, "weights" of all attributes are the same and equal to 1(dGower works on somehow normalized data). In this project was assumed always the same weight.

As can be found in [2], dGower distance is given by:

$$G(p,q) = \frac{\sum_{d=1}^{m} w(p_d,q_d) \times s(p_d,q_d)}{\sum_{d=1}^{m} w(p_d,q_d)}$$

where *m* is the number of attributes, *s* is the similarity of the objects *p* and *q* on the attribute *d.* For numerical attributes:

$$s(p_d,q_d) = 1 - \frac{|p_d - q_d|}{range_d}$$

For nominal attributes *s=1* if $p_d = q_d$, else is equal to zero. As there are no unknown values and all the attributes have the same weight, the denominator is equal to the number of features. The value stored in the vector of distances is *1- dGower* in order to select the lowest values.

For both cases the temporal complexity of the measure of the distance between 2 samples is *O(F)* once it iterates over all the features. The complexity to compute the distance between the sample to be classified and all the others is *O(N*F).*

- **Find the nearest neighbours**

After having the distances between the sample to be classified and all the other samples computed, to find the K (or K++) nearest neighbours there are two possibilities. The first one (linear search) is go through the vector K times and, in each time, find the smallest distance that was not found before. The other case is to sort the vector of distances and return the first K elements of the sorted vector. For example, imagine if we want to classify the sample number 0. If the vector of distances is [-1, 5.9, 4.8, 3, 1, 2, 5, 0.5] (-1 to sample 0 because it is the sample to be classified), and if we want to find the 2-nearest neighbours, the first approach (linear search) in the first time will go through the vector and choose the neighbour with distance 0.5 (neighbour with index 7). After this, with this

algorithm, it will go through the vector again and it will choose the neighbour with distance 1 (neighbour with index 4). The temporal complexity to this process is *O(K*N).*

As far as the second approach is concerning, the algorithm Quicksort in its recursive representation creates data partitions in subtables and each available part is sorted. The complexity associated to this algorithm is *O(Nlog(N)).* In the example above, after sorting the vector of distances will be [-1, 0.5, 1, 2, 3, 4.8, 5, 5.9], and it would be created another auxiliary vector with the correspondent index [-1, 7, 4, 5, 3, 2, 6, 1]. After sorting, it is only need to return the first elements of the vector (the complexity is *O(1)*). Therefore, the final complexity of this process, is the complexity of sort the data: *O(N*log(N)).*

This way, which approach is more worth it to use? As the first one has complexity *O(N*K)* and the second has complexity *O(N*log(N)),* in the case of K < log(N) the first approach (linear search – without sorting) is more worth it. However, if log(N) < K, N*log(N) < N*K, and in this case the second approach (with Quicksort) is more worth it.

- **Find KNN+**

To find KNN+ nearest neighbours, there are also two different ways, depending how approach was taken to choose the K nearest neighbours. With linear search, as the vector of distances is not sorted, the program will iterate over all the vector and check if there are distances equal to the distance of the K^{th} neighbour. If the distance is the same, it is added this sample to the vector of neighbours. This process costs *O(N).*

If the second approach was taken, the vector of distances is sorted, therefore is enough to check the positions k, k+1, k+2… until the distance value is different from the value stored in the position k-1. If the values are equal, the correspondent samples are added to the vector of neighbours. This process costs *O(1).*

- **Classify a sample**

To classify a sample what is need is to iterate the vector of neighbours and check which class is more frequent (save the values to each class in an auxiliary vector). The temporal complexity is *O(K).*

Examples

1) Example number 1 - KNN vs KNN+ in numerical dataset

In the first example, the dataset shown below was used.

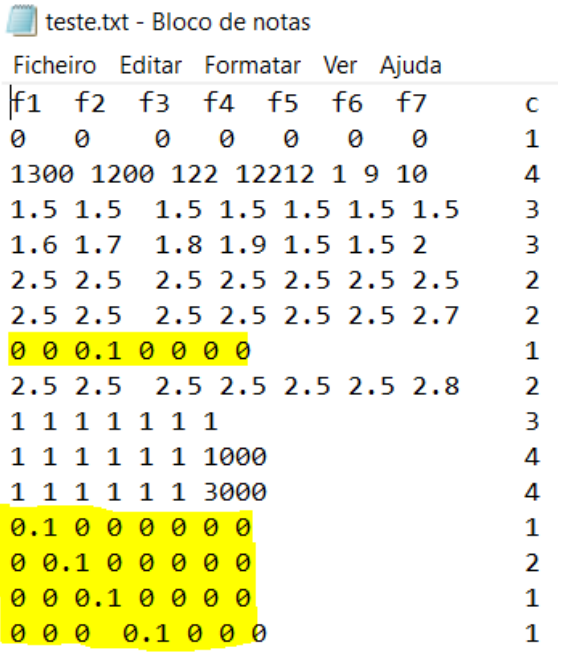

Figure 6: Dataset used

In the first time the program was ran this way:

```
/mnt/c/Users/leand/Desktop/dataMiningProject/KNN-KNN-and-VPTree$ ./main -a 0 -f DataSets/teste.txt -t 0 -c 0 0 -k 2
```

As we saw before, with these flags the algorithm used is KNN, the type of dataset is numeric, there will be classified only one sample (it is the sample number 0 of the dataset), and K is equal to 2.

As we can observe in the dataset, the nearest samples of the sample number 0 are the shaded samples of yellow (with Euclidean distance equal to 0.1) – samples 6, 11, 12, 13 and 14.

The output is:

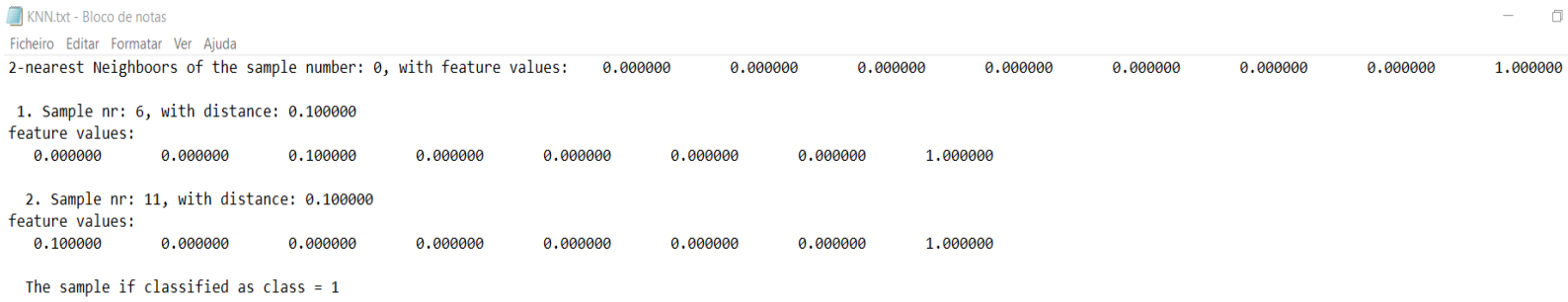

Figure 7: output example 1 with KNN

Despite having 5 samples with distance 0.1 as we can observe only the samples number 6 and 11 were selected. As both samples belong to class number 1, the sample is classified as class number 1. If instead of using KNN, it is used KNN+, to run (in the same conditions) the program is with:

```
/mnt/c/Users/leand/Desktop/dataMiningProject/KNN-KNN-and-VPTree$ ./main -a 1 -f DataSets/teste.txt -t 0 -c 0 0 -k 2
```

In these conditions, the output is:

KNN_PLUS.txt - Bloco de notas

Ficheiro Editar Formatar Ver Ajuda

```
2-nearest Neighboors of the sample number: 0, with feature values:    0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      1.000000

 1. Sample nr: 6, with distance: 0.100000
feature values:
   0.000000      0.000000      0.100000      0.000000      0.000000      0.000000      0.000000      1.000000

  2. Sample nr: 11, with distance: 0.100000
feature values:
   0.100000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      1.000000

  3. Sample nr: 12, with distance: 0.100000
feature values:
   0.000000      0.100000      0.000000      0.000000      0.000000      0.000000      0.000000      2.000000

  4. Sample nr: 13, with distance: 0.100000
feature values:
   0.000000      0.000000      0.100000      0.000000      0.000000      0.000000      0.000000      1.000000

  5. Sample nr: 14, with distance: 0.100000
feature values:
   0.000000      0.000000      0.000000      0.100000      0.000000      0.000000      0.000000      1.000000

  The sample if classified as class = 1
```

Figure 8: Output example number 1 with KNN+

Since we are using KNN+, with K = 2, in this case the number of nearest neighbours found was equal to 5, as expected, once the samples number 12, 13 and 14 has the same distance (0.1) as the sample number 11 (2nd neighbour). As 4 out of 5 samples belong to class 1, the sample is classified as class 1. All the sample in yellow were selected.

2) Example number 2 - KNN vs KNN+ in categoric dataset

The dataset used was:

categNumTeste.txt - Bloco de notas

Ficheiro Editar Formatar Ver Ajuda

```
buying maint doors persons lug_boot safety class
vhigh vhigh 2 2 small low unacc
vhigh vhigh 2 2 small med unacc
vhigh vhigh 2 2 small high unacc
vhigh vhigh 2 2 med low unacc
vhigh vhigh 2 2 small med unacc
vhigh vhigh 2 2 med med unacc
vhigh vhigh 2 2 med high unacc
high low 4  5 small high acc
high low 4  5 med low unacc
high low 4  5 med med acc
high low 4  5 med high acc
high low 4  5 big med acc
high low 4  5 big high acc
med low 2  5 med high good
med low 2  5 big low unacc
med low 2  5 big med good
med low 2  5 big high vgood
```

Figure 9: Dataset example 2

In order to classify the sample number 0 with KNN (K = 2) was ran the line

```
/mnt/c/Users/leand/Desktop/dataMiningProject/KNN-KNN-and-VPTree$ ./main -a 1 -f DataSets/categNumTeste.txt -t 1 -c 0 0 -k 2
```

As can be shown in yellow, the first four samples are at the same distance in relation to the sample 0 (once only one categoric parameter change in relation to 0). As it was carrying out KNN, the output achieved was:

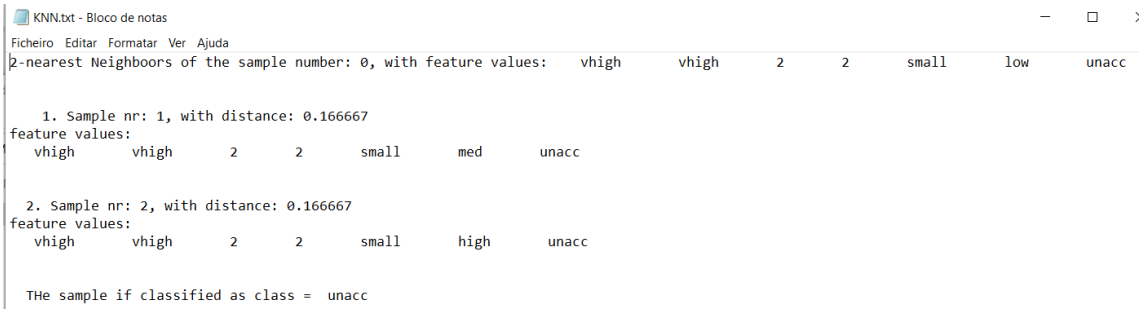

Figure 10: Output example 2 with KNN

Only 2 samples were chosen. However, if the experiment is carried out with KNN+, the output is:

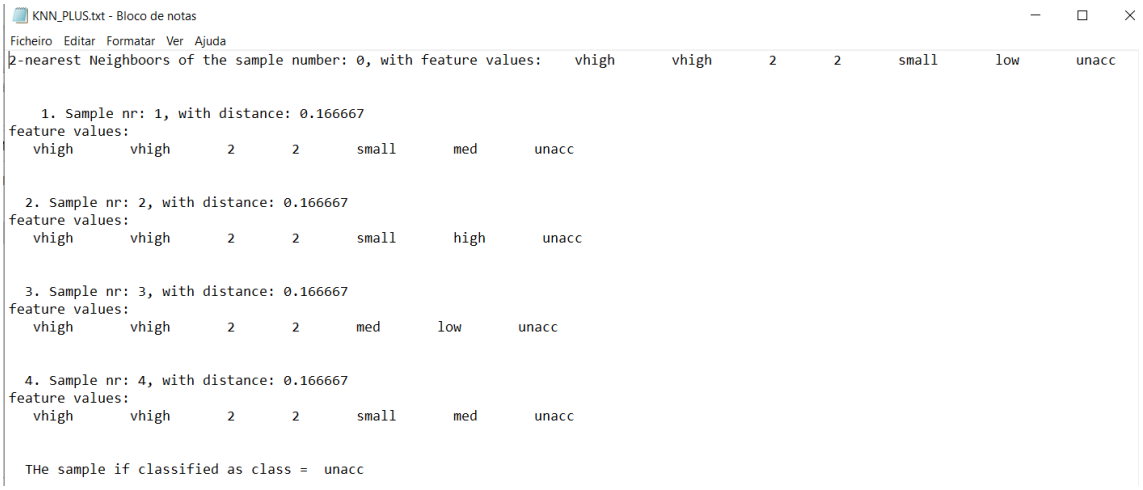

Figure 11: Output example 2 with KNN+

By applying the *dGower* distance we can confirm that the values obtained to the distance are correct.

3) Quicksort vs Linear Search

In the dataset shown above the number of samples is equal to 17. As it was mentioned before, the search of the neighbours can be done with linear search or with quicksort. To k < log2(N) linear search is perform and to k>= log2(N) quicksort is carried out. In this case, log2(N) = 4.087, therefore to k>= 5 quicksort is performed, once there will be less computations this way. The next figures illustrate what happens with the vector that contain the distances between sample number 0 and all the others when the above experiment is performed to k=2 (left) and k = 5 (right). If with k=5, the sorted algorithm was not performed, there would be much more computations once it would have to go through the vector of 17 elements 5 times (85 executions). This way, after sorting (around 21 executions), it just has to return the first 5 elements of the vector.

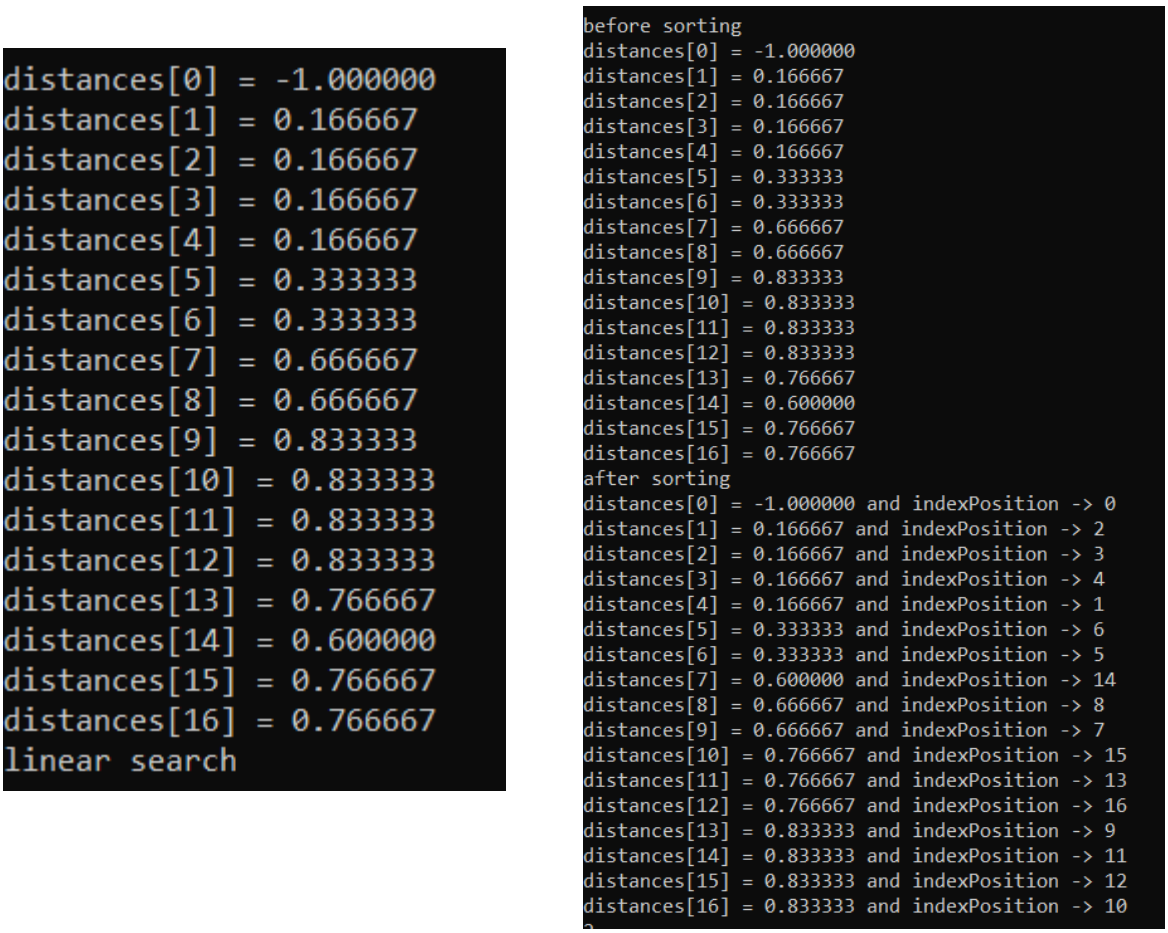

Figure 12: Vector of distances in case of linear search (left) and quicksort (right)

Results analysis

As KNN is a problem classification, it was decided to analyse the accuracy of the model to different datasets. Just to exemplify how to get the accuracy, there were performed tests with the small datasets created by hand. Accuracy is given by number samples well classified divided by all samples classified. In order to show how to get the accuracy, an example is provided.

Figure 13: Example of accuracy computation to this output

In this example, there were 5 samples classified, and as we can see (in yellow) 3 out of 5 are well classified, and the other 2 (the 3rd one and the 4th one) were not well classified. So, the accuracy is given by 3/5 = 0.6.

The same is suitable to compute to mixed dataset.

Figure 14: Example of accuracy computation to this output

To analyse the program with large datasets, there were used 3 numeric datasets and 1 mixed dataset.

The datasets used to analysis were:

Dataset 1 - Numeric:

Name file: iris.txt

Dimensions: 150 samples, 4 features

https://archive.ics.uci.edu/ml/datasets/iris

Dataset 2 - Categoric + numeric:

Name file: car.txt

Dimensions: 1728 samples, 6 features

http://archive.ics.uci.edu/ml/datasets/Car+Evaluation

Dataset 3 - Numeric:

Name file: wine.txt

Dimensions: 178 samples, 13 features

https://archive.ics.uci.edu/ml/datasets/Wine

Dataset 4 - Numeric:

Name file: redWine.txt

Dimensions: 1600 samples, 11 features

https://archive.ics.uci.edu/ml/datasets/Wine+Quality

Table 1 and table 2 resume the accuracy to the different datasets to both algorithms. Table 2 can be shown the accuracy after performing data normalization. These values were obtained with Leave-One-Out method (there were classified all of the samples of the training set with all the other samples working as training set).

Table 1: Accuracy to each dataset to different values of K and to KNN and KNN+

Dataset	K =	1	2	3	4	5	6	7	10	13	15	19	23	25
1	KNN	96,053	94,737	96,05	96,05	96,71	96,05	96,72	96,70	96,70	97,37	98,03	96,71	96,71
	KNN+	96,053	94,737	96,05	96,05	96,71	96,71	97,37	97,37	96,71	97,37	98,03	96,71	96,71
2	KNN	69,98	69,98	70	70	62,34	59,9	58,99	59,1	66,9	68,3	67,72	66,10	66,16
	KNN+	70	70	70,2	71,88	61,88	59,41	56,91	67,78	67,78	67,78	65,12	65,12	65,12
3	KNN	76,96	67,41	72,47	66,29	69,6	68,5	66,2921	66,8	69,1	70,02	71,34	71,91	71,91
	KNN+	76,97	67,41	72,47	66,29	69,6	68,5	66,2	66,8	69,1	70,02	71,34	71,91	71,91

Table 2: Accuracy to different datasets, after **normalization**, to different values of K and KNN and KNN+

Dataset	K =	1	2	3	4	5	6	7	10	13	15	19	23	25
1	KNN	94,737	96,05	94,73	95,41	95,42	95,39	96,05	95,39	96,05	95,39	95,4	95,42	95,39
	KNN+	94,737	96,05	94,74	95,39	95,39	95,39	96,05	95,39	96,06	95,40	95,39	95,39	95,39
3	KNN	94,94	95,5	96,62	95,5	94,94	95,5	96,62	96,62	97,75	97,75	97,19	97,75	97,75
	KNN+	94,94	95,5	96,62	95,5	94,94	95,5	96,62	96,62	97,75	97,75	97,19	97,75	97,75

The table 3 represents the runtime obtained also with Leave-One-Out method. There are 2 values: one without writing the results in the text file (first one), and the total runtime.

Table 3: Runtime values to different datasets. The first value of each entrance is without writing the results in the text file, and the second value is the total runtime. Time is in [ms].

Dataset	K =	1	2	3	4	5	6	7	10	13	15	19	23	25
1	KNN	1,89 and 4,68	1,6 and 4,79	1,57 and 5,09	1,08 and 5,59	1,22 and 5,64	1,49 and 7,53	1,97 and 9,38	2,36 and 11,7	2,5 and 12,4	2,69 and 15,13	2,52 and 16,68	2,72 and 22,61	2,75 and 23,15
	KNN+	1,85 and 4,3	1,65 and 4,86	2,42 and 6,34	1,78 and 6,93	1,9 and 8,69	1,68 and 7,5	1,91 and 8,41	2,69 and 12,33	2,06 and 11,21	2,56 and 14,77	2,59 and 16,9	2,25 and 22,6	2,86 and 23,85
2	KNN	730,6 and 771,89	710,6 and 763,7	720,9 and 788,1	752,6 and 825,5	711,7 and 886,78	793,15 and 881,89	830,9 and 936,54	812,9 and 934,46	1100 and 1209	1208 and 1385	1201 and 1379	1194 and 1451	1196 and 1362
	KNN+	782,3 and 798,79	789 and 803,1	770,2 and 818,2	791,3 and 898,6	811,24 and 916,72	833 and 931,9	860,92 and 946,64	900,2 and 1000,47	1111 and 1278	1278 and 1401	1248 and 1385	1201 and 1499	1196 and 1362
3	KNN	2,26 and 7,24	2,35 and 10,07	2,18 and 10,18	2,96 and 13,68	2,56 and 14,84	2,76 and 20,31	2,55 and 17,87	4,48 and 30,8	5,55 and 38,1	4,78 and 43,04	4,3 and 44,8	4,5 and 46,06	4,025 and 51,85
	KNN+	2,26 and 7,5	2,4 and 10,2	2,59 and 11,92	3,01 and 14,1	2,91 and 16,59	3,12 and 22,1	2,99 and 18,99	3,9 and 23,99	4,22 and 30,85	4,44 and 40,14	4,13 and 43,9	4,5 and 45,16	4,017 and 52,35
4	KNN	879 and 950	816 and 911,7	815 and 926	816 and 946	894 and 1059	839 and 1011	919 and 1124	883 and 1137	890 and 1236	913 and 1334	967 and 1449	1016 and 1569	1303 and 1602
	KNN+	780 and 846	819 and 942,1	854 and 996	898 and 1084	954 and 1109	967 and 1044	1008 and 1151	937 and 1207	919 and 1268	1033 and 1361	1078 and 1509	1117 and 1584	1298 and 1611

As we can observe in table 1 and 2, in generally there are better results to odd values of K, because with even values the model is most likely to produce draws. Draw can happen also to odd values and depend on the number of classes and the value of K. In case of draw, there is no rejection. The program will choose one of the classes. Because of that, it will produce less accuracy. KNN+ can settle down this problem. Because of that in some cases it can be seen the improvements that KNN+ produces.

In nearest neighbour classifier it is evident that all features should have the same order of magnitude. With normalization there were improvements in the accuracy of dataset 3. These improvements were around 25%. It was because in the original dataset the features had not the same order of magnitude. Regarding to dataset1, normalization did not produce improvements, since all features have the same order of magnitude in the original dataset.

The best value to K depends on the dataset. For example, to dataset1 the best result was achieved to K = 19, while to dataset3 was to k = 1.

With KNN+ the results achieved are a little bit better in relation to KNN, but it depend on the dataset, once if one sample is close to the decision boundary can be wrong classified with KNN+ and well classified with KNN (the same can happen with the increase of K). Same can

happen in another situations, like with outliers. To k=10 in dataset2, the improvements of KNN+ were around 10% (maybe there are a lot of samples in the border between the two classes or mixed samples of the 2 classes, that is solved with KNN+). To dataset 2 the best results are achieved to low values of K.

As expected, as larger is the dataset (or with more features), more time is consumed.

In general, KNN+ takes more time than KNN. It was expecting, once the KNN+ must verify if exists more neighbours with the same distance as the last one.

When the accuracy changes more from KNN to KNN+, the runtime also changes more (because there were performed more executions to find the 'extra' neighbours). For example, to dataset 2, to K= 10, where there was a great improvement and a great difference of the runtime.

When the K increases, the runtime in general increases also, since there are more computations in the search of neighbours

A good observation is that the difference between the runtime in KNN and KNN- is less to greater K. It is because for k > log2(N) quicksort is performed. This way, to find the 'extra' neighbours in KNN+ is very fast once is to check only the next entrances of the distances. It can be noticed in the large datasets used. Sometimes, if we compare K=7 with K=10 we can see that the increase of the runtime is not so long (because of the use of quicksort to k=10 most of the times).

Even for the largest dataset (dataset4) the runtime is not so long (it took around 1.6 seconds in the worst case). So KNN and KNN+ are a fast method. Dataset 3 and 4 take much more time than the other ones, once the number of samples is much bigger. Besides that, the computation of the *dGower* distance takes more time then Euclidean distance (more comparisons).

Conclusions

All in all, KNN simply assumes that class of a test sample according to the majority class of the K closest training samples. There are several distance metrics to compute the distance between two samples. KNN and KNN+ can be used in different type of datasets, where to each one it is need to adequate the distance metric used to measure distances.

KNN is generally used when there are enough samples (especially near between class boundaries). In this case, KNN is quite good classifier. The problem is that requires memory to store samples and good implementation. When there is a well-defined metric and enough samples in the training set it can be used KNN for such a set.

As far as the value of K is concerning, we have two contradictory requirements for k: it has to be large, in order to minimize probability of misclassification, but at the same time small - with relation to training set size - to achieve true class boundaries. For a fixed size training set we should experiment with different k values and select the value giving best classification results on test set.

In binary classification problems, KNN classifier tries to avoid using an even number as the value of K. It is because this it Is possible to produce the same number of the two main classes. The same problem also exists also with K>2. With KNN+ this problem can be settled down.

Normalization can be used to improve the accuracy of the model, once in nearest neighbour classifier it is evident that all features should have the same order of magnitude. Detection of outliers could be a good idea to apply in this project that could increase the accuracy.

References

[1] - mkr_Clustering53_notes.pdf

[2] - EDAMI19Z_mkr_kNN%20&%20Gower_11_notes.pdf

[3] https://www.geeksforgeeks.org/quick-sort/