F21DL:
Data Mining & Machine Learning
Coursework 1

Students:

Mohammad ALKHALDI

Anis BENAMER

Quentin DUCASSE

Sylvain TOUANEN

Lecturers:

Diana BENTAL

Ekaterina KOMENDANTSKAYA

HERIOT WATT UNIVERSITY

Table of Contents

Introduction

In the scope of the F21DL-Data Science and Machine Learning course, the first coursework uses a bank of images of street signs and wants to predict the type of a given image. In order to respond to this objective, several tools are used.

The first part of the coursework, and therefore the first one of this report is centered around data *pre-processing*, how we managed files, how we processed, transformed and selected the data available. The second part is focused on *Naïve Bayes* and both its accuracy and issues. Along with this part comes a reflection on the information we learned about the dataset. The third part revolves around *Complex Bayesian Architectures*, how to build them and how they perform compared to the previous model. Along with this part come reflections on the new properties found about the data as well as the help *Bayesian Nets* provide over *Naïve Bayes*. The next part is centered on *K-Means and Clustering* and how it can be applied to our problem. This part comes with a discussion on the results obtained along with the clustering part. Finally, a research question is asked and a solution to it is provided in the final part.

This coursework has been the occasion to train the machine learning skills we obtained throughout the course, either during lectures or labs. As the subject was completely different from the ones in the labs, because it is related to computer vision, we were taking the coursework as a new challenge to prove our understanding of the course. Our initial objective was to use Python in order to complete the coursework as it is looking like the most used language in industry. However, parts of our reflection are based on Weka as well due to its accessibility.

The project is hosted on GitHub under *https://github.com/QDucasse/dm_cw1* along with the installation instructions and milestones of the project. In order for it to work, the actual datasets need to be downloaded at https://www.dropbox.com/s/n36lewuw0alaja9/data.zip?dl=0 and unzipped inside the root project. The project was run under Weka 3.8.3 and Python 3.7.4.

Data Pre-Processing

The first part of the coursework revolves around *Data Pre-Processing*, and we will here explain the objectives we set for ourselves. The four main points we took in consideration are *File Management, Pre-Processing, Transformation* and *Selection.*

File Management

In order to load the dataset, it is needed to link the "base" dataset (12660 instances of images composed of 2304 pixels each) and the labels those images are given. Images are stored under "`./data/x_train_gr_smpl.csv`" while the labels are in "`./data/y_train_smpl.csv`". Moreover, Boolean masks are provided under "`./data/x_train_smpl_<NB>.csv`" where NB is the wanted label. Based on those files were generated the files in the `arff` folder allowing the dataset to be loaded in Weka. The conversion is made possible thanks to the `arff_converter.py` file and can be done again by executing the file with the following command while in the root project:

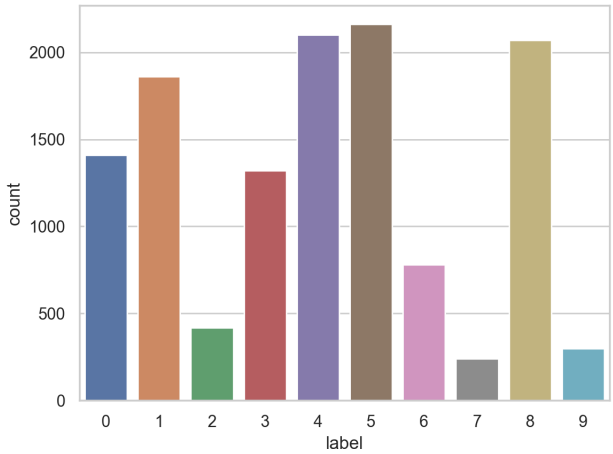
bash command

```
$ python dm_cw1/arff_converter.py
```

The converter in `arff_converter.py` performs similar actions to the `loader.py`. Both of those scripts fully load the dataset, either the *Weka's* way through *arff* files or the *Python* way using *pandas*. The labels are linked to the corresponding images and it is possible to load a Boolean mask of the labels rather than the labels themselves.

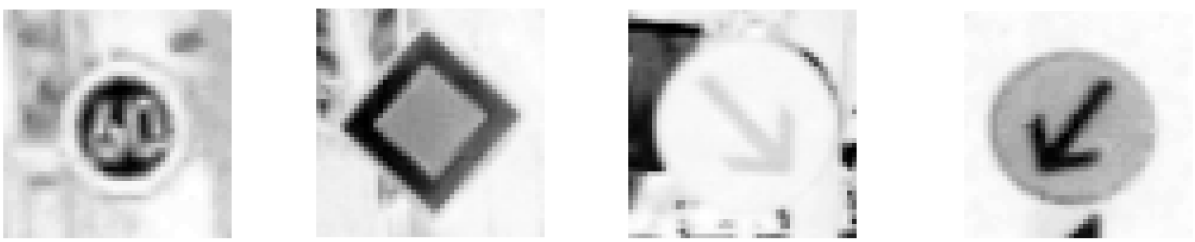
Visualisation

Python could handle the whole dataset, so we tried to load it in order to visualize it and have a better understanding of what it is made of. Using *seaborn,* we can plot the number of instances by label as well as their size using `groupby('label').size()`:



label	
0	1410
1	1860
2	420
3	1320
4	2100
5	2160
6	780
7	240
8	2070
9	300

The first thing we can notice is that the numbers of instances of the different type of signs is extremely unbalanced, while label 5 is represented by 2160 instances, label 7 only holds 240 instances. This imbalance may cause the filter to perform badly and will need to be fixed. We can then inspect the images by using the `display_nth_sign()` function (indexes 1300, 5600, 10400 and 12550 here).



By seeing the image, we can suppose that not every pixel (feature) will be equivalent. The center of the image seems extremely important and the exterior of the sign and therefore the border of the image should less significant.

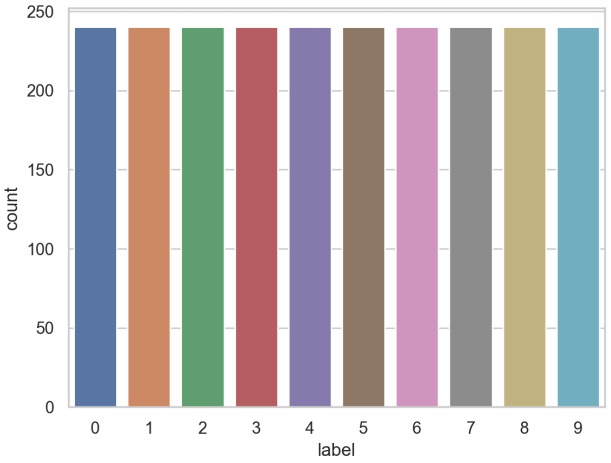
Selection

The first objective of the selection is to reduce the number of instances [2]. To do so, we created the function `select_instances()` that will compute the minimal number of instances a class attribute is represented by and select this same number out of all the other class attributes representants. The selection is either made by taking the first instances representative of the different class attributes or by picking them randomly. This option can be triggered by specifying the function parameter `rd=True` or `False`.

Moreover, the instances are put in random order each time a new computation is done [1]. This is made possible by using the function `sample(frac=1)` applied to the correct *pandas* data frame. The final datasets are the following (obtained by using `print_head_tail()`).

```
Head:
   label        0         1         2  ...      2300      2301      2302      2303
0      5  0.066667  0.082353  0.054902  ...  0.031373  0.031373  0.031373  0.031373
1      1  0.070588  0.066667  0.074510  ...  0.227451  0.231373  0.235294  0.235294
2      0  0.941176  0.941176  0.945098  ...  0.247059  0.250980  0.247059  0.247059
3      0  0.133333  0.137255  0.141176  ...  0.058824  0.058824  0.054902  0.054902
4      8  0.509804  0.623529  0.560784  ...  0.074510  0.074510  0.074510  0.078431

[5 rows x 2305 columns]
Tail:
       label        0         1  ...      2301      2302      2303
12655      5  0.894118  0.894118  ...  0.862745  0.854902  0.858824
12656      5  0.125490  0.105882  ...  0.090196  0.086275  0.098039
12657      8  0.133333  0.125490  ...  0.129412  0.137255  0.145098
12658      8  0.090196  0.094118  ...  0.262745  0.266667  0.266667
12659      4  0.952941  0.960784  ...  0.192157  0.313725  0.360784
```

```
Head:
   label        0         1         2  ...      2300      2301      2302      2303
0      2  0.050980  0.054902  0.054902  ...  0.101961  0.101961  0.105882  0.105882
1      2  0.156863  0.223529  0.215686  ...  0.211765  0.258824  0.254902  0.207843
2      0  0.329412  0.305882  0.298039  ...  0.156863  0.156863  0.156863  0.152941
3      4  0.294118  0.298039  0.298039  ...  0.200000  0.188235  0.176471  0.164706
4      3  0.074510  0.074510  0.078431  ...  0.066667  0.062745  0.062745  0.066667

[5 rows x 2305 columns]
Tail:
      label        0         1  ...      2301      2302      2303
2395      9  0.074510  0.078431  ...  0.101961  0.109804  0.125490
2396      6  0.129412  0.141176  ...  0.180392  0.188235  0.156863
2397      4  0.098039  0.098039  ...  0.137255  0.149020  0.164706
2398      9  0.141176  0.137255  ...  0.098039  0.094118  0.094118
2399      1  0.482353  0.505882  ...  0.078431  0.070588  0.070588
```




```
label
0    240
1    240
2    240
3    240
4    240
5    240
6    240
7    240
8    240
9    240
```

Transformation

The main transformation that we did on the images was to normalize everything. To do so, we used to different techniques. The first one, as the images are composed of greyscale values (between 0 and 255) was to divide everything by 255 to get a value between 0 and 1. The second method uses the function `scale()` from the *preprocessing sklearn* subpackage and will produce a Gaussian distribution with zero mean and variance. You can see in the logs above the result of using the `divise_by_255()` filter on the dataset.

Arff Conversion

The conversion of the dataset to .arff files in order to load the dataset in Weka is made through the `arff_converter.py` file. As Weka cannot handle large files, the selection is done at generation here following the same guidelines the *pandas* data frame would handle. Executing the file will generate the base dataset, two sampled datasets (random selection or not) and the datasets with the Boolean masks.

Naïve Bayes Nets

First use

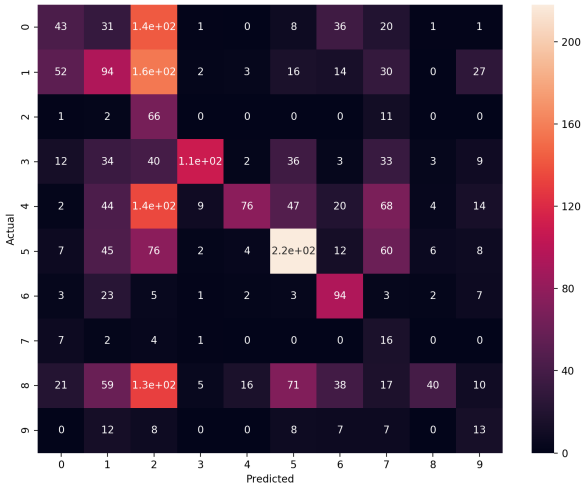
Once the dataset is reduced, randomised and normalised, we can apply the first model and algorithm in order to try to identify the different signs: Naïve Bayes [3]. We will run the algorithm on both Weka and Python in order to compare the results and see if the whole dataset that Python can compute holds better results. For all the runs, a confusion matrix will be provided. We will run the *Naïve Bayes classifier* on the following datasets and under the languages:

- Full dataset (randomised version) in Python (1)
- Small dataset (randomised version) in Python (2)
- Small dataset (randomised version) in Weka (3)

We obtain the following:

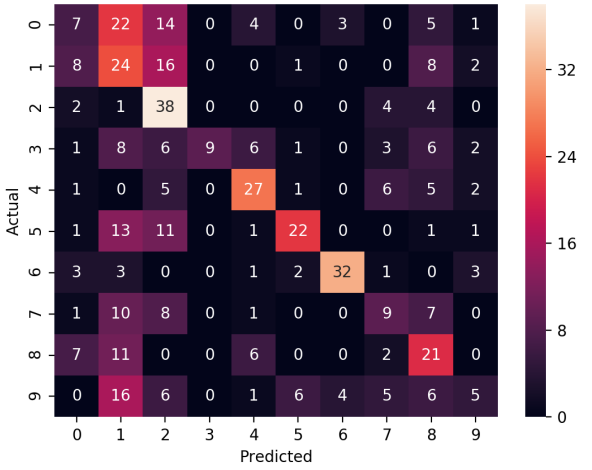
(1)

```
Running Naive Bayes on dataframe 'Signs randomised'
with class feature 'label'
Naive-Bayes accuracy:  0.2843601895734597
Predicted   0   1    2   3   4    5   6   7   8   9
Actual
0          51  29  144   0   0    8  52  14   1   2
1          38  63  158   5   3   18  23  27   0  17
2           0   0   71   0   0    0   0   8   0   0
3           8  43   45  95   5   30   1  29   3  10
4           8  39  118  15  87   65  21  48   6  13
5           7  44   84   1   2  186  14  63   4  15
6           2  25   10   1   7    6  85   5   1   8
7           3   3    9   3   1    0   6  20   0   0
8          27  50  136   8  13   90  36  14  47  12
9           0  14   10   0   0   13   4   7   0  15
```



(2)

```
Running Naive Bayes on dataframe 'reduced Dataframe 'Signs' randomised'
with class feature 'label'
Naive-Bayes accuracy:  0.4375
Predicted   0   1   2   3   4   5   6   7   8  9
Actual
0           1   9  16   0   3   0   2   0   5  0
1           2  22  12   0   0   0   0   0   8  0
2           1   1  50   0   0   0   0   1   0  0
3           0   8   5  12   7   3   0   6   3  0
4           2   2   6   0  32   0   1   2   4  4
5           2  11   8   0   0  20   0   0   2  3
6           2   5   0   0   2   1  31   0   6  0
7           0  12   9   1   1   3   0  16  11  0
8          10  16   0   0   5   0   0   2  18  0
9           0  19   7   0   1   5   2   4   7  8
```



From Python point of view, the first thing to notice is that the classifier performs better on the reduced dataset (28.4% against 43.8%) even though it struggles to output correct answers as its accuracy stays under 50%. Another thing when looking at the confusion matrix is that the classifier tends to predict signs labelled 1 a lot and often miss the point, especially when looking at signs labelled 0 or 9 for example. The same issues are noticeable on the Weka logs shown below.

(3)

```
=== Summary ===

Correctly Classified Instances         921               38.375  %
Incorrectly Classified Instances      1479               61.625  %
Kappa statistic                          0.3153
Mean absolute error                      0.1233
Root mean squared error                  0.3509
Relative absolute error                 68.5006 %
Root relative squared error            116.9692 %
Total Number of Instances             2400

=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   j   <-- classified as
  36  28 119   3   0  10  27  11   4   2 |   a = label0
  23  44 126   3   2   7  12  11   0  12 |   b = label1
   6   2 211   0   0   0   0  21   0   0 |   c = label2
   8  32  40  84   5  32   2  24   7   6 |   d = label3
   5  22  65   6  51  31   9  33  10   8 |   e = label4
   4  34  43   0   5 102   6  35   5   6 |   f = label5
   9  39  11   1   3   7 144   4   4  18 |   g = label6
  27  11  39   2   6   2  18 135   0   0 |   h = label7
   8  32  72   1   7  41  27   7  40   5 |   i = label8
   1  51  47   2   0  16  23  26   0  74 |   j = label9
```

Determining Best Attributes

Out of the 2304 attributes corresponding to the pixels composing every image, we want to narrow them down to the two, five and ten most representative of each label representant. In order to do this, we will use the absolute top correlation value with the class feature "label" [4]. The correlation for a label is computed from the dataset with the corresponding Boolean mask. This is done in Python by loading this dataset and then using:

naive_bayes.py > best_cor_attributes()

```
cor_target = abs(cor[class_feature])
relevant_features = cor_target[cor_target>0.01].sort_values()
```

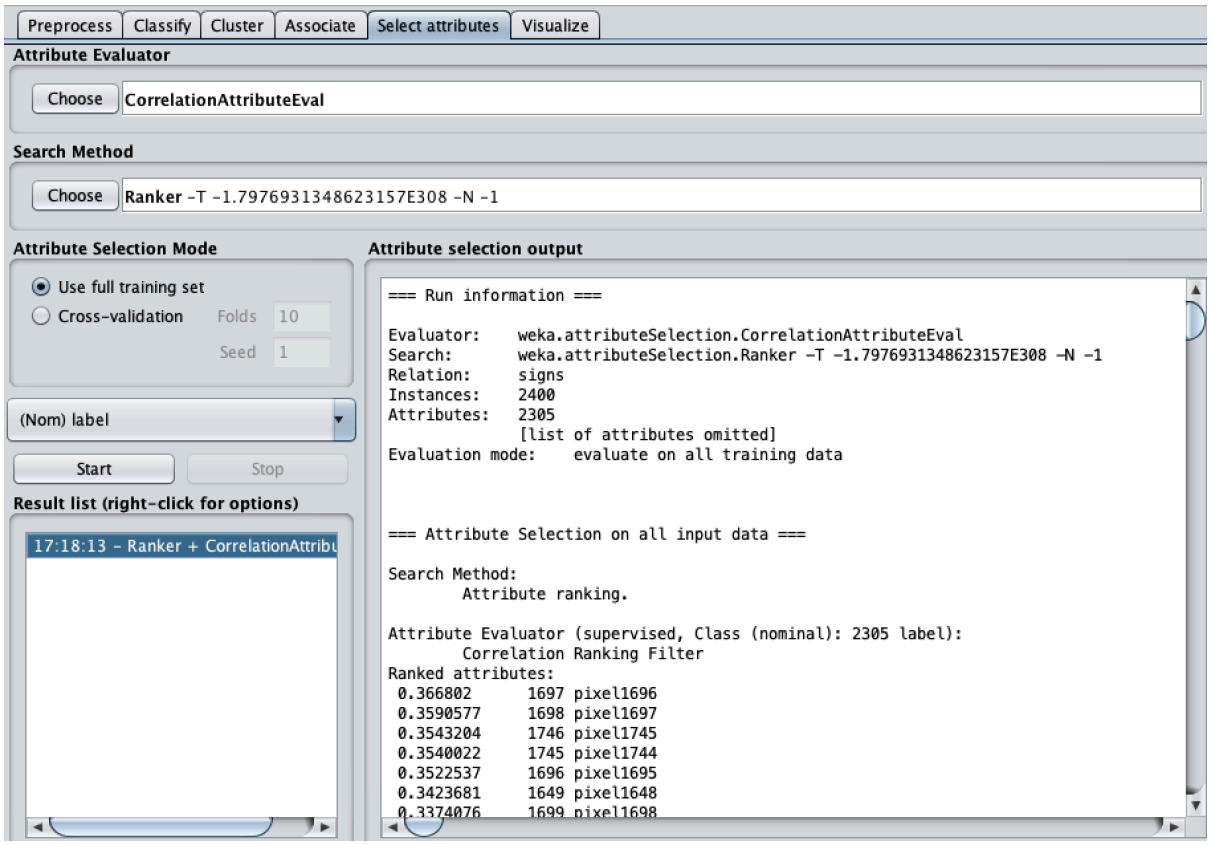
Those features are sorted from the top to the least correlating feature while staying above a score of 0.01 in absolute correlation. The features are then stored to the corresponding files and can be re-generated by using the function `store_best_attributes()`. Out of those files are extracted the following top correlating attributes for each label:

naive_bayes.py

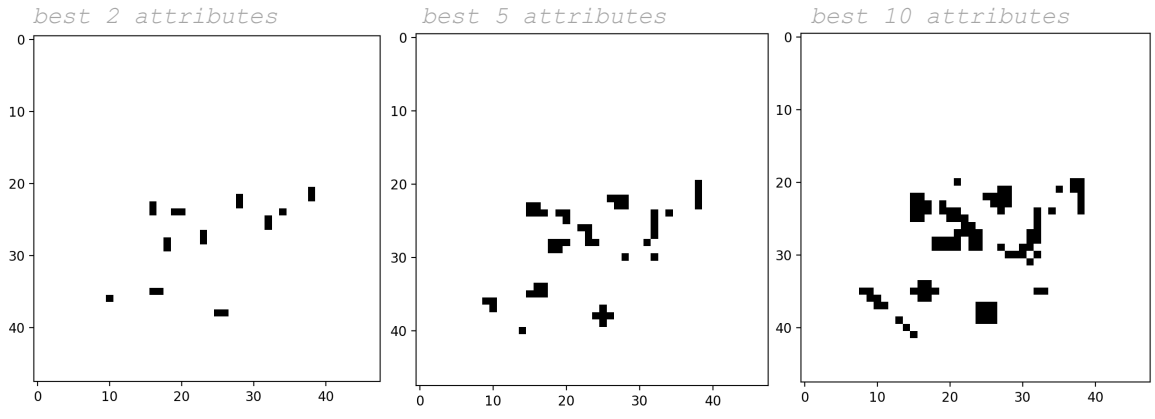
```
ba0 = [1172, 1171, 1468, 1220,
1472, 1221, 1123, 1469, 1419,
1519, 1124, 1471]

ba1 = [1094, 1046, 1172, 1142,
998, 1190, 1173,  997,  981, 1045,
950, 1143]

ba2 = [1084, 1132, 1083, 1131,
1082, 1130, 1036, 1081, 1179,
1035, 1129, 1178]

ba3 = [1696, 1697, 1648, 1649,
1695, 1698, 1745, 1744, 1713,
1712, 1647, 1650]

ba4 = [1849, 1850, 1848, 1897,
1801, 1802, 1898, 1800, 1896,
1043, 1847, 1309]
```

```
ba5 = [1319, 1367, 1271, 1368,
1270, 1318, 1320, 1415, 1416,
1222, 1366, 1223]

ba6 = [1186, 1738, 1934, 1737,
1786, 1787, 1885, 1689, 1983,
1688,  633, 1836]

ba7 = [1168, 1120, 1169, 1119,
1167, 1121, 1216, 1072, 1071,
1215, 1217, 1118]

ba8 = [1280, 1232, 1328, 1184,
1375, 1376, 1327, 1470, 1423,
1422, 1471, 1469]

ba9 = [1362, 1410, 1363, 1364,
1411, 1365, 1412, 1317, 1318,
1413, 1361, 1314]
```

On Weka, the same thing can be done by loading the dataset with the Boolean mask and running the `CorrelationAttributeEval` paired with the Ranker in the *"Select Attributes"* tab.



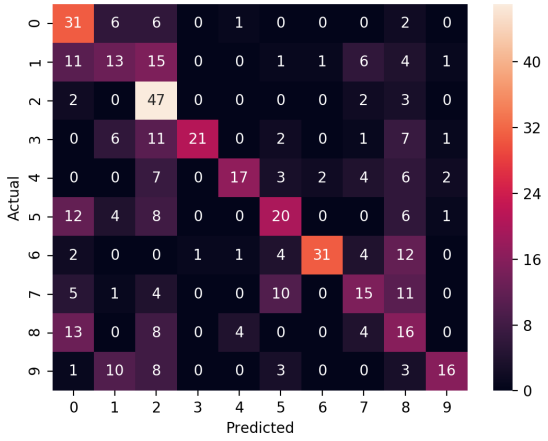
Now that we have the top correlating attributes, it is possible to visualise them in order to look at the most important pixels:



The centre of the image looks important to the recognition and this confirms the idea we had when looking at the images provided in the dataset. The environment surrounding the sign is changing and must not be as important as the sign itself. We can now rerun the classifier on the sampled dataset for the 2, 5 and 10 best attributes for each type of sign.

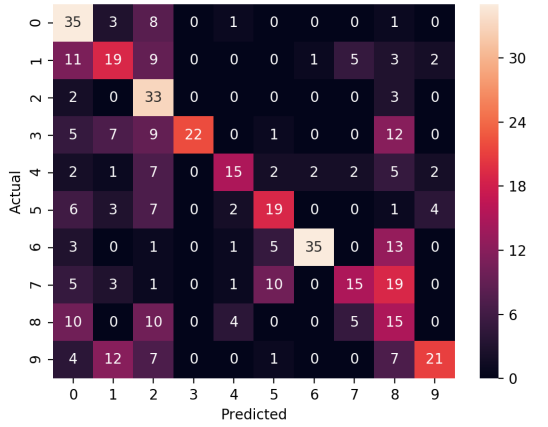
best 2 attributes

```
============================================================
Running Naive Bayes on dataframe 'reduced Dataframe 'Signs' with
 best 2 attributes randomised'
with class feature 'label'
Naive-Bayes accuracy:  0.47291666666666665
Predicted   0   1   2   3   4   5   6   7   8   9
Actual
0          31   6   6   0   1   0   0   0   2   0
1          11  13  15   0   0   1   1   6   4   1
2           2   0  47   0   0   0   0   2   3   0
3           0   6  11  21   0   2   0   1   7   1
4           0   0   7   0  17   3   2   4   6   2
5          12   4   8   0   0  20   0   0   6   1
6           2   0   0   1   1   4  31   4  12   0
7           5   1   4   0   0  10   0  15  11   0
8          13   0   8   0   4   0   0   4  16   0
9           1  10   8   0   0   3   0   0   3  16
```



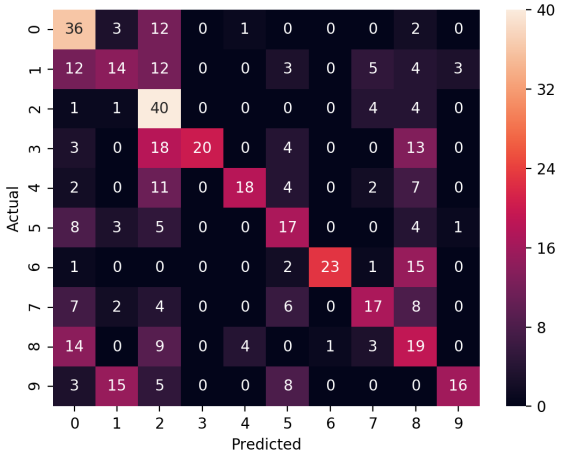
best 5 attributes

```
============================================================
Running Naive Bayes on dataframe 'reduced Dataframe 'Signs' with
 best 5 attributes randomised'
with class feature 'label'
Naive-Bayes accuracy:  0.47708333333333336
Predicted   0   1   2   3   4   5   6   7   8   9
Actual
0          35   3   8   0   1   0   0   0   1   0
1          11  19   9   0   0   0   1   5   3   2
2           2   0  33   0   0   0   0   0   3   0
3           5   7   9  22   0   1   0   0  12   0
4           2   1   7   0  15   2   2   2   5   2
5           6   3   7   0   2  19   0   0   1   4
6           3   0   1   0   1   5  35   0  13   0
7           5   3   1   0   1  10   0  15  19   0
8          10   0  10   0   4   0   0   5  15   0
9           4  12   7   0   0   1   0   0   7  21
```



best 10 attributes

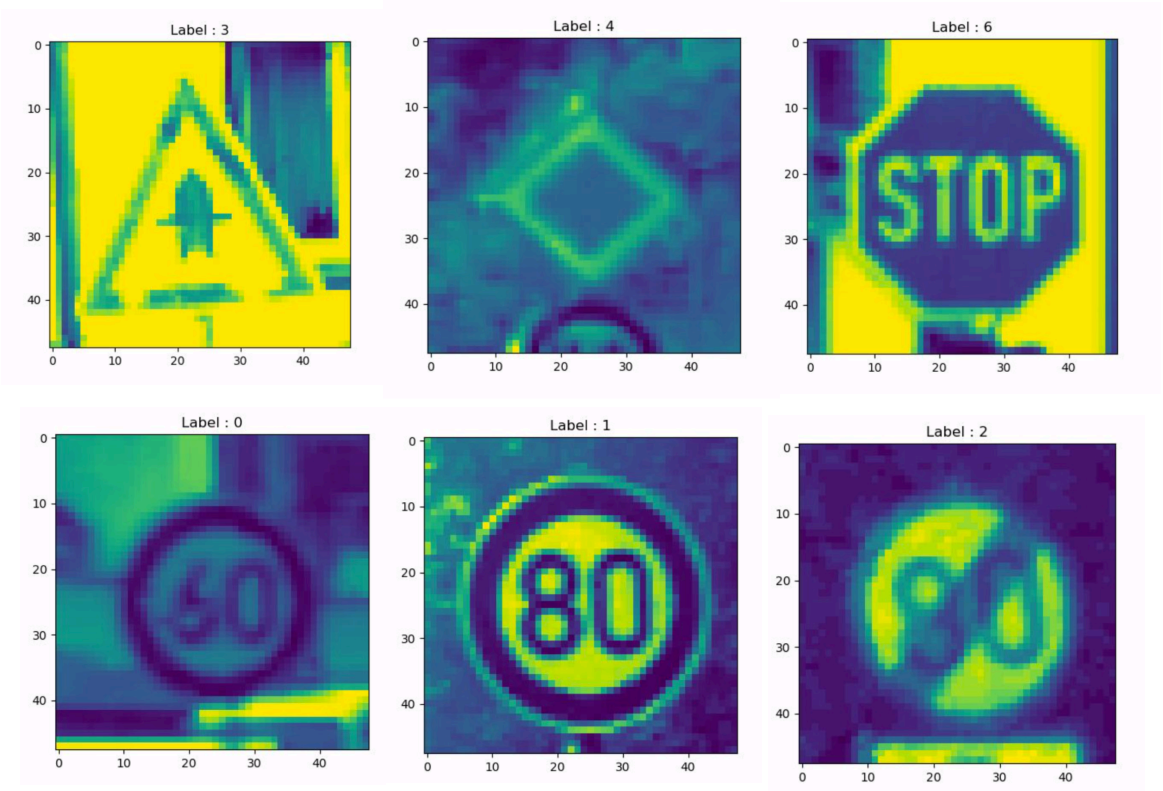
```
============================================================
Running Naive Bayes on dataframe 'reduced Dataframe 'Signs' with
 best 10 attributes randomised'
with class feature 'label'
Naive-Bayes accuracy:  0.4583333333333333
Predicted   0   1   2   3   4   5   6   7   8   9
Actual
0          36   3  12   0   1   0   0   0   2   0
1          12  14  12   0   0   3   0   5   4   3
2           1   1  40   0   0   0   0   4   4   0
3           3   0  18  20   0   4   0   0  13   0
4           2   0  11   0  18   4   0   2   7   0
5           8   3   5   0   0  17   0   0   4   1
6           1   0   0   0   0   2  23   1  15   0
7           7   2   4   0   0   6   0  17   8   0
8          14   0   9   0   4   0   1   3  19   0
9           3  15   5   0   0   8   0   0   0  16
```



The accuracy of the classifier improved by taking only some of the attributes and we can see in the example that the best accuracy is obtained with 5 best attributes for each label and therefore the 50 attributes dataset. However, the results vary from a run to another, but it is safe to say that this dataset is the most consistent. We obtained ~51% with the 100 attributes dataset but it often stays around ~46%.

Reflections on Naïve Bayes:

Thanks to the above experiments, we learned a lot about our data. First, each label is not represented equally within the initial dataset. Each image contains both a sign and its surrounding environment. This environment can be considered as noise and we can see that when selecting the top-correlating attributes, most of those are located in the centre of the image, in the place the sign is placed. Secondly, by looking at the confusion matrix we obtained, the signs 3, 4 and 6 are easily recognised. Those corresponds to the signs the most recognisable because they have a specific shape and does not look like any other. On the other hand, the round-shaped signs are often confused. We can look for example at the 2, 0 and 1 that are easily confused.



The randomisation plays a role in the selection of the test and training sets as if a test set is selected from an ordered dataset, the classifier will only have to perform on the same labels and be biased due to the lack of those particular labels in the training set. Other ways can be used to improve our results such as the ones presented in [6].

Complex Bayes Nets

This task was made entirely on Weka because the construction of complex Bayesian nets is complicated with Python [5]. We looked at *pomegranate* which seems to be the go-to package for this type of classifier, but the more straightforward usage and implementation and Weka was more attractive. For the next experiments, we will use the small dataset version with the best 5 features from previous experiment. The three models are the following:

- K2 algorithm with number of parents varying from 1 to 3
- TAN algorithm
- Hill-Climbing algorithm with number of parents varying from 1 to 3

K2

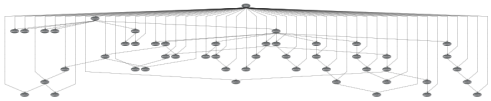
The algorithm outputs the same results after 3 parents and tops at 76.8% accuracy.

K2 maxparents:1

```
=== Summary ===

Correctly Classified Instances        1217               50.7083 %
Incorrectly Classified Instances      1183               49.2917 %
Kappa statistic                          0.4523
Mean absolute error                      0.0986
Root mean squared error                  0.2972
Relative absolute error                 54.7959 %
Root relative squared error             99.0699 %
Total Number of Instances             2400
```

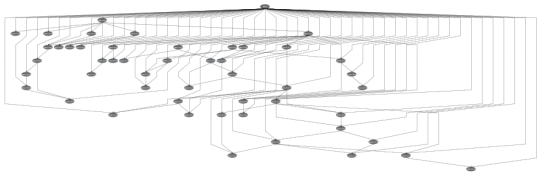
K2 maxparents:2

```
=== Summary ===

Correctly Classified Instances        1771               73.7917 %
Incorrectly Classified Instances       629               26.2083 %
Kappa statistic                          0.7088
Mean absolute error                      0.057
Root mean squared error                  0.2009
Relative absolute error                 31.6927 %
Root relative squared error             66.9619 %
Total Number of Instances             2400
```



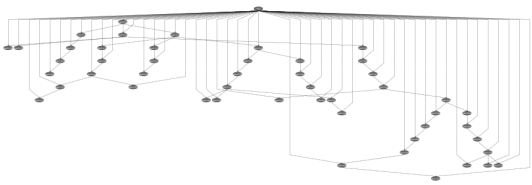
K2 maxparents:3

```
=== Summary ===

Correctly Classified Instances        1844               76.8333 %
Incorrectly Classified Instances       556               23.1667 %
Kappa statistic                          0.7426
Mean absolute error                      0.0502
Root mean squared error                  0.1894
Relative absolute error                 27.8798 %
Root relative squared error             63.1461 %
Total Number of Instances             2400
```



TAN

```
=== Summary ===

Correctly Classified Instances        1813               75.5417 %
Incorrectly Classified Instances       587               24.4583 %
Kappa statistic                          0.7282
Mean absolute error                      0.0532
Root mean squared error                  0.1921
Relative absolute error                 29.5499 %
Root relative squared error             64.0304 %
Total Number of Instances             2400
```



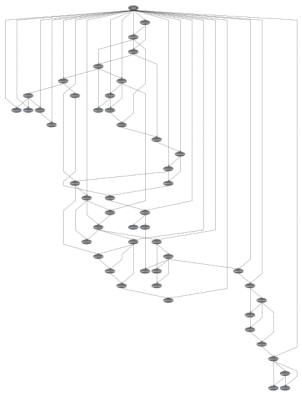
HILL-CLIMBING

HillClimbing maxparents:1

```
=== Summary ===

Correctly Classified Instances        1217               50.7083 %
Incorrectly Classified Instances      1183               49.2917 %
Kappa statistic                          0.4523
Mean absolute error                      0.0986
Root mean squared error                  0.2972
Relative absolute error                 54.7959 %
Root relative squared error             99.0699 %
Total Number of Instances             2400
```

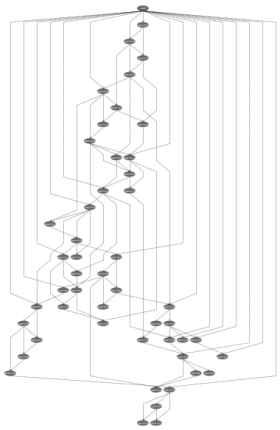
HillClimbing maxparents:2

```
=== Summary ===

Correctly Classified Instances        1721               71.7083 %
Incorrectly Classified Instances       679               28.2917 %
Kappa statistic                          0.6856
Mean absolute error                      0.0657
Root mean squared error                  0.1993
Relative absolute error                 36.489  %
Root relative squared error             66.4311 %
Total Number of Instances             2400
```



HillClimbing maxparents:3

```
=== Summary ===

Correctly Classified Instances        1734               72.25   %
Incorrectly Classified Instances       666               27.75   %
Kappa statistic                          0.6917
Mean absolute error                      0.0652
Root mean squared error                  0.1964
Relative absolute error                 36.2101 %
Root relative squared error             65.4587 %
Total Number of Instances             2400
```



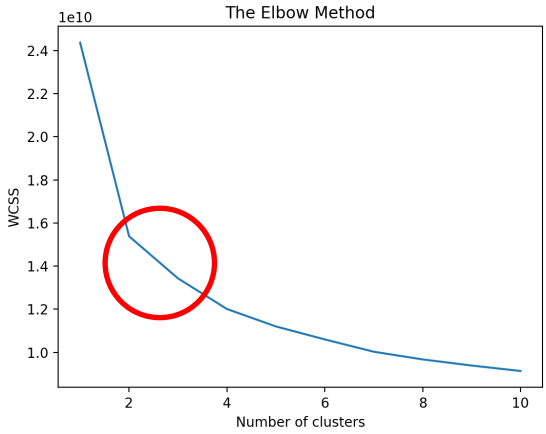
The *Bayesian Nets* help consider the weight of each of the attributes by looking at the number of their parents and children. Additionally, it helps showing dependencies in the data as well as what pixels are most representative. On our example, the best accuracy that can be obtained is the one we got when using the K2 algorithm with a maximal number of parents of three.

Clustering

In order to perform a *KMeans clustering* [7] on the dataset, we will use the small dataset generated after instance selection and we will try it as well on the dataset with selected attributes. To simulate the unsupervised idea of the classifier, we will drop the 'label' column in Python and use "Classes to clusters" in Weka.

Python

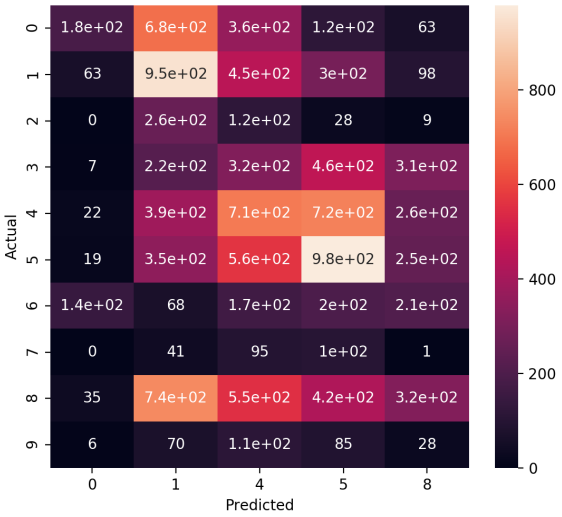
In Python, if we do not know the number of clusters we want or know our data to be separated in, we can use the elbow method [] that plots the WCSS (Within Clusters Sum of Squares) given the number of clusters. In our case, the result is the following:



In an unsupervised manner, the *KMeans clustering* algorithm aims at 2/3 clusters. Now, if we input the number of clusters we want to be 10, we can obtain the following on the full dataset and reduced one:

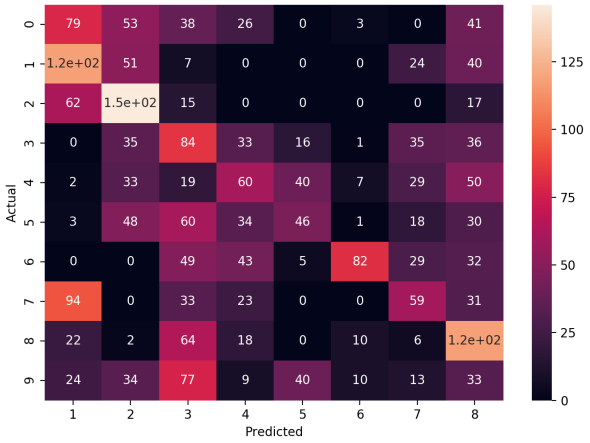
full dataset

```
Running KMeans unsupervised clustering with 10 clusters on data
frame 'Signs randomised'
Shape of the clusters centers: (10, 2304)
KMeans accuracy score: 0.24842022116903634
Predicted    0    1    4    5    8
Actual
0          184  683  364  116   63
1           63  953  449  297   98
2            0  262  121   28    9
3            7  221  318  462  312
4           22  390  707  719  262
5           19  349  564  979  249
6          142   68  166  197  207
7            0   41   95  103    1
8           35  740  552  421  322
9            6   70  111   85   28
```



reduced dataset

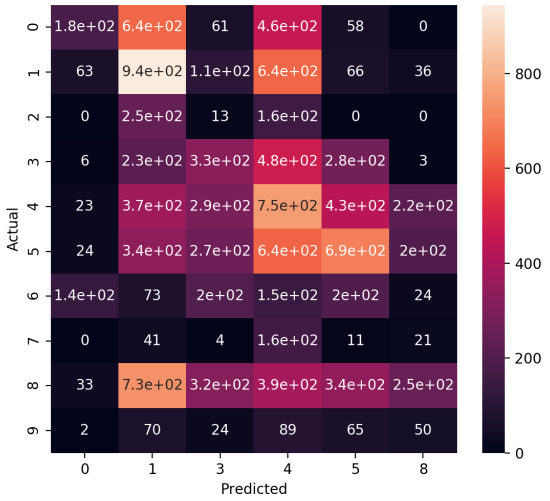
```
Running KMeans unsupervised clustering with 10 clusters on data
frame 'reduced Dataframe 'Signs' randomised'
Shape of the clusters centers: (10, 2304)
KMeans accuracy score: 0.2970833333333333
Predicted    1    2  3   4   5   6   7    8
Actual
0           79   53  38  26   0   3   0   41
1          118   51   7   0   0   0  24   40
2           62  146  15   0   0   0   0   17
3            0   35  84  33  16   1  35   36
4            2   33  19  60  40   7  29   50
5            3   48  60  34  46   1  18   30
6            0    0  49  43   5  82  29   32
7           94    0  33  23   0   0  59   31
8           22    2  64  18   0  10   6  118
9           24   34  77   9  40  10  13   33
```



If we now include the class feature and therefore make it a supervised classifier:

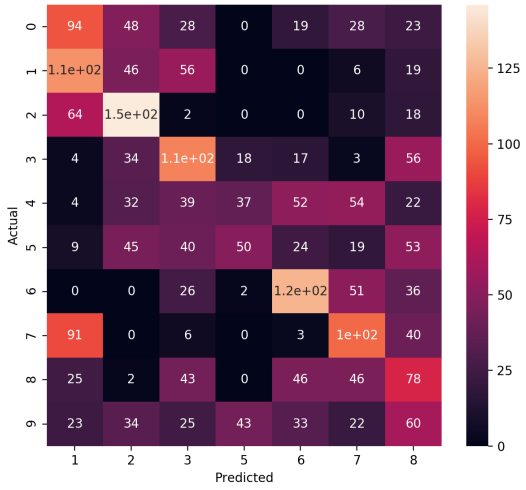
full dataset

```
Running KMeans supervised clustering (on feature label) with 10
 clusters on dataframe 'Signs randomised'
Shape of the clusters centers: (10, 2305)
0.2678515007898894
Predicted    0     1    3    4    5    8
Actual
0          181  1086   62   15   66    0
1           64  1542  135   38   66   15
2            0   408   12    0    0    0
3            6   613  404    9  288    0
4           24   892  305  339  443   97
5           22   820  288  270  700   60
6          139   194  199   19  214   15
7            0   147   13   59   11   10
8           32  1069  310   60  374  225
9            1   155   28    1   42   73
```



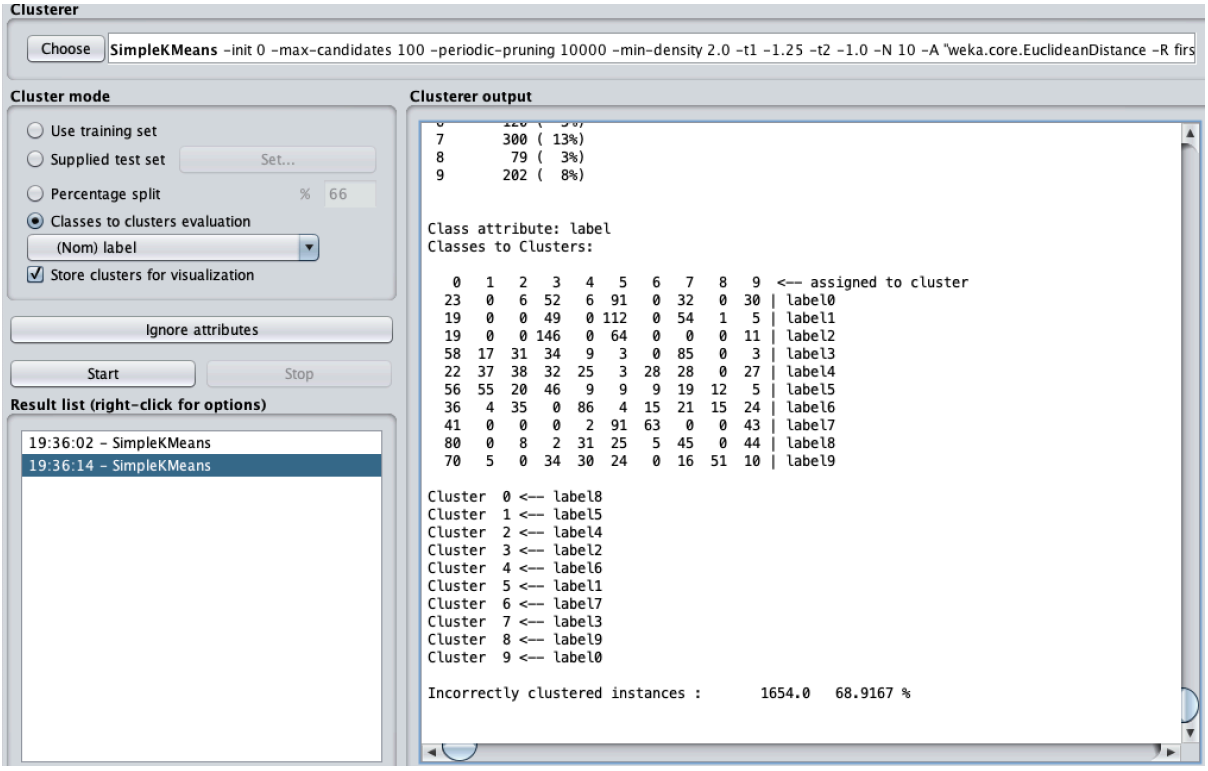
reduced dataset

```
Running KMeans supervised clustering (on feature label) with 10
clusters on dataframe 'reduced Dataframe 'Signs' randomised'
Shape of the clusters centers: (10, 2305)
0.3
Predicted    1    2    3   5    6    7   8
Actual
0           94   48   28   0   19   28  23
1          113   46   56   0    0    6  19
2           64  146    2   0    0   10  18
3            4   34  108  18   17    3  56
4            4   32   39  37   52   54  22
5            9   45   40  50   24   19  53
6            0    0   26   2  125   51  36
7           91    0    6   0    3  100  40
8           25    2   43   0   46   46  78
9           23   34   25  43   33   22  60
```



Weka

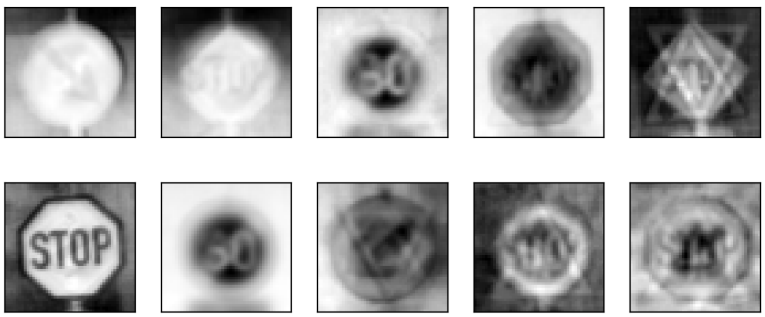
On Weka, using the KMeans classifier in an unsupervised way consists in specifying "*Use classes to clusters*". Moreover, we set the number of clusters to be 10 and we obtain the following:



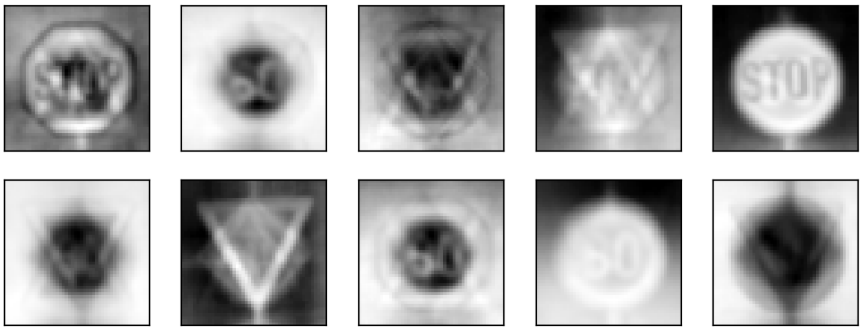
The instances are incorrectly clustered to up to ~73% depending on the seed. However, inputting the datasets with Boolean masks and reducing the number of clusters to 2 brings the accuracy up. We obtain an accuracy of 60% to 70% for the different labels, using seed 1,5 and 8:

Label0: 64.5%
Label1: 59.9%
Label2: 59.5%
Label3: 65.1%
Label4: 70.1%
Label5: 68.7%
Label6: 72.3%
Label7: 62.3%
Label8: 66.6%
Label9: 65.6%

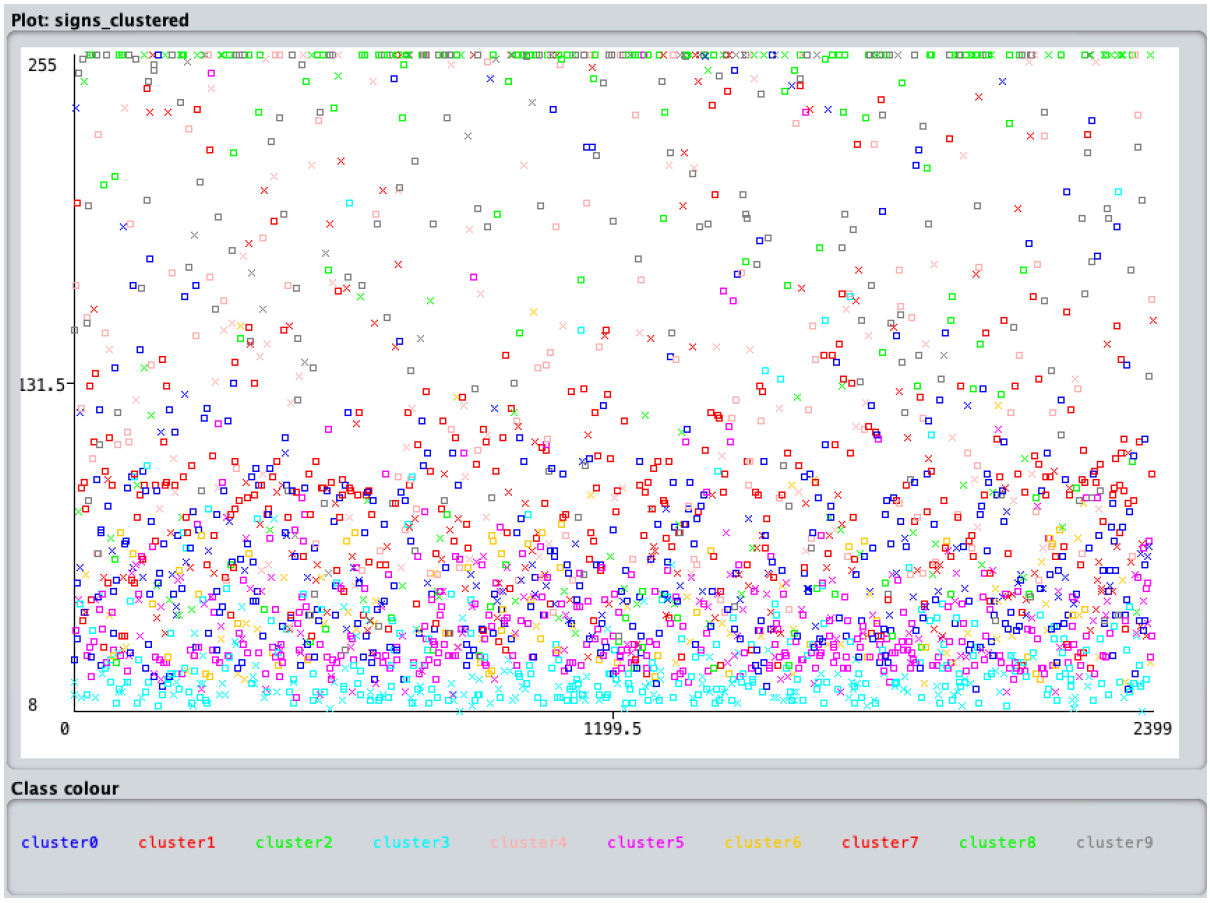
The results on the base dataset while asking to match the whole dataset or part of it to 10 clusters result in poor accuracy. The following image depicts the result of the centres of the clusters when shown for the small dataset:



Most of the signs are poorly recognised and it gets even worse when using *KMeans* algorithm on the full dataset:



Therefore, the *KMeans* algorithm is performing well on the dataset when it is Boolean masked and it has to split it into two clusters. The elbow method shows us that the algorithm should perform well with 2/3 clusters. The performance drops when we try to use it on the full or reduced dataset and ask it to split the data into 10 clusters. The centres of the clusters seem to mismatch the actual labels as they look like combination of different signs.



KMeans improvements

We can try to refine this clustering method by using the TSNE (T-distributed Stochastic Neighbours Embedding algorithm) in order to classify better. This algorithm is a non-linear embedding algorithm that is particularly adept at preserving points within clusters [11]. This is done by using the TSNE object from *sklearn.manifold* to project the instances of the dataset. This step does not bring any particular improvement to the classification unfortunately. Using the Manhattan distance helps the accuracy as well.

We tried the following clustering algorithms using the classes to cluster evaluation:

- **Canopy**: Acc: 24.3%
- **FarthestFirst**: Acc: 12.5%
- **FilteredClusterer**: Acc: 16.2%
- **MakeDensityBasedClusterer**: Acc: 17.4%

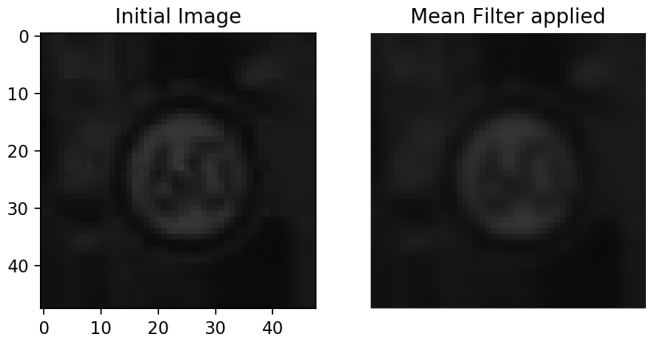
Unfortunately, we did not have enough time to implement those classifiers in *Python* but we were interested by the article [9] that presented interesting types of clustering algorithms.

Research Question

The research question we chose and tried to answer to is: *"What type of image specific filters exist?"* and how to apply them to our dataset. The normalisation method we used in the coursework is very basic and straightforward, we simply divide the greyscale image by 255 in order to have final values between 0 and 1. The other method was to output a gaussian distribution with zero mean and variance. Here are other filters [10] that we will try to apply on a single image in order to see the output:

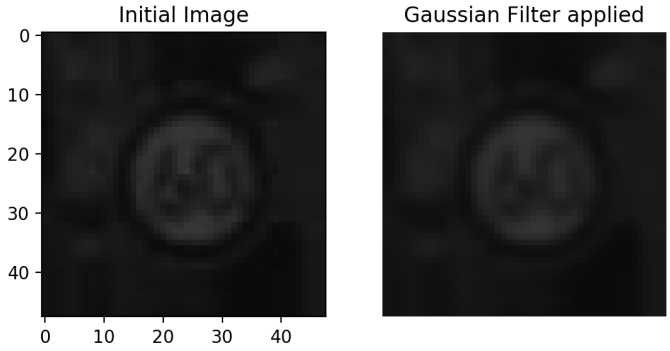
Mean Filter

The mean filter consists of a basic blur placed on the image. This is done by taking the mean of the surrounding pixels' intensities for each pixel in the image. This filter mostly aims at reducing speckle noise but comes at the cost of losing some details of the image.



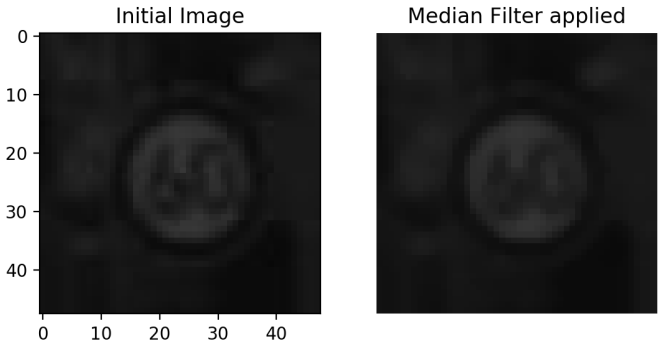
Gaussian Filter

The Gaussian filter has the same objective as the mean filter but manages to preserve the edges of the image better than the corresponding mean filter. It creates the mean for each filter by using a weighted average of the surrounding pixels. It can be supplied with a sigma value in order to specify the standard deviation in both the x and y directions.



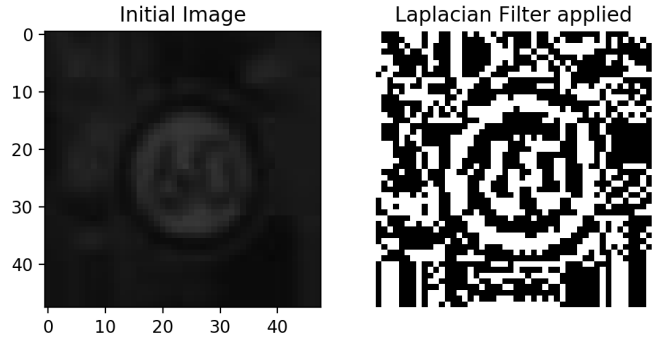
Median Filter

The Median filter calculates the median intensity values of the surrounding pixels. This filter gets better result when removing salt and pepper noise but struggle with speckle noise.



Laplacian Filter

The Laplacian of an image shows better the zones with quick changes in intensity. It is mostly used for edge detection and performs well in this field. It outputs a heavily contrasted image corresponding to the edge of the provided image. It can then be coupled to another filter in order to produce a blurred image with accentuated edges.



Unfortunately, we did not have enough time to perform the experiments again with the new versions of our image in order to test changes in accuracy induced by those new filters. This could have been a way to produce a benchmark on the best filters to add to your images.

Conclusion and Contribution

This coursework felt extremely interesting as it both challenged us to the use of a new language used industry-wise, Python and on the other side let us explore Weka. It was the occasion for us to challenge ourselves to a state-to-the-art issue that can and will be extremely important regarding driverless cars.

Using both Python and Weka was interesting in order to spot the eases each of the languages have. Python through pandas, sklearn and cv2 is extremely vast and permitting to a point we can lose ourselves quickly. On the other hand, Weka simplifies some aspects while allowing fine tuning in the classifiers. The outputs, both visual and textual are well produced as we have to create them on our own with Python.

The project is hosted on GitHub under *https://github.com/QDucasse/dm_cw1* along with the installation instructions and milestones of the project. In order for it to work, the actual datasets are gathered under a 'data' folder but most of it can be generated from the base datasets.
The generated datasets can be found online as they are stored on Dropbox and need to be downloaded at https://www.dropbox.com/s/n36lewuw0alaja9/data.zip?dl=0 then unzipped inside the root project. The project was run under Weka 3.8.3 and Python 3.7.4.

Contribution of the different members:
Python code: Quentin DUCASSE **80%**, Sylvain TOUANEN **20%**
Weka code: Quentin DUCASSE **90%**, Mohammad ALKHALDI **5%**, Anis BENAMER **5%**
Report: Quentin DUCASSE **50%**, Sylvain TOUANEN **50%**

References

[1] Jason Brownlee,
"*The Role of Randomization to Address Confounding Variables in Machine Learning*",
July 2018
https://machinelearningmastery.com/confounding-variables-in-machine-learning/?fbclid=IwAR0yKGk4CGV1e7AmxW6E-q2HmmgW4ad7gZRQC2q_vsOOSzUyZnUVngSwXDM
[visited 11/11/2019]

[2] Håkon Hapnes Strand,
"How do machine learning algorithms handle such amounts of data?"
April 2018
https://www.forbes.com/sites/quora/2018/04/10/how-do-machine-learning-algorithms-handle-such-large-amounts-of-data/#4cce8a8b730d
[visited 11/11/2019]

[3] Gaurav Chauhan,
"All about Naïve Bayes"
October 2018
https://towardsdatascience.com/all-about-naive-bayes-8e13cef044cf [visited 11/11/2019]

[4] Jason Brownlee,
"How to Perform Feature Selection With Machine Learning Data in Weka"
July 2016
https://machinelearningmastery.com/perform-feature-selection-machine-learning-data-weka/?fbclid=IwAR36sWL3X340ZRuYdQJ7zrwJVidqpFCBVFnInIbQuIPMXDY7iQcjfaVLEqY
[visited 11/11/2019]

[5] Ana M.Martínez-Rodrígueza, Jerrold H.Mayb, Luis G.Vargas,
"An optimization-based approach for the design of Bayesian networks"
October 2008

[6] George Seif
"3 ways to improve your Machine Learning results without more data"
December 2018
https://towardsdatascience.com/3-ways-to-improve-your-machine-learning-results-without-more-data-f2f0fe78976e
[visited 11/11/2019]

[7] Michael J. Garbade,
"Understanding K-means Clustering in Machine Learning",
September 2018
https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1
[visited 11/11/2019]

[8] Trupti M. Kodinariya, Prashant R. Makwana
"Review on determining number of Cluster in K-Means Clustering",
November 2013

[9] George Seif,
"The 5 Clustering Algorithms Data Scientists Need to Know"
February 2018
https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68
[visited 11/11/2019]

[10] Manvir Sekhon, "*Image Filters in Python*",
https://towardsdatascience.com/image-filters-in-python-26ee938e57d2 [visited 11/11/2019]

[11] Laurens van der Maaten, *"TSNE"*
https://lvdmaaten.github.io/tsne/ [visited 11/11/2019]