F21DL Data Mining & Machine Learning

Coursework 1

ACCAD Dimitri, AUZIMOUR Antoine, DELTEL Clarence, DI MARTINO Thomas

The results of our DMML CW1 takes the form of two Jupyter Notebooks: 1 for questions 1 up to 13 and 1 for research question. Both are provided as .ipynb files + we exported them as .pdf for better readability.

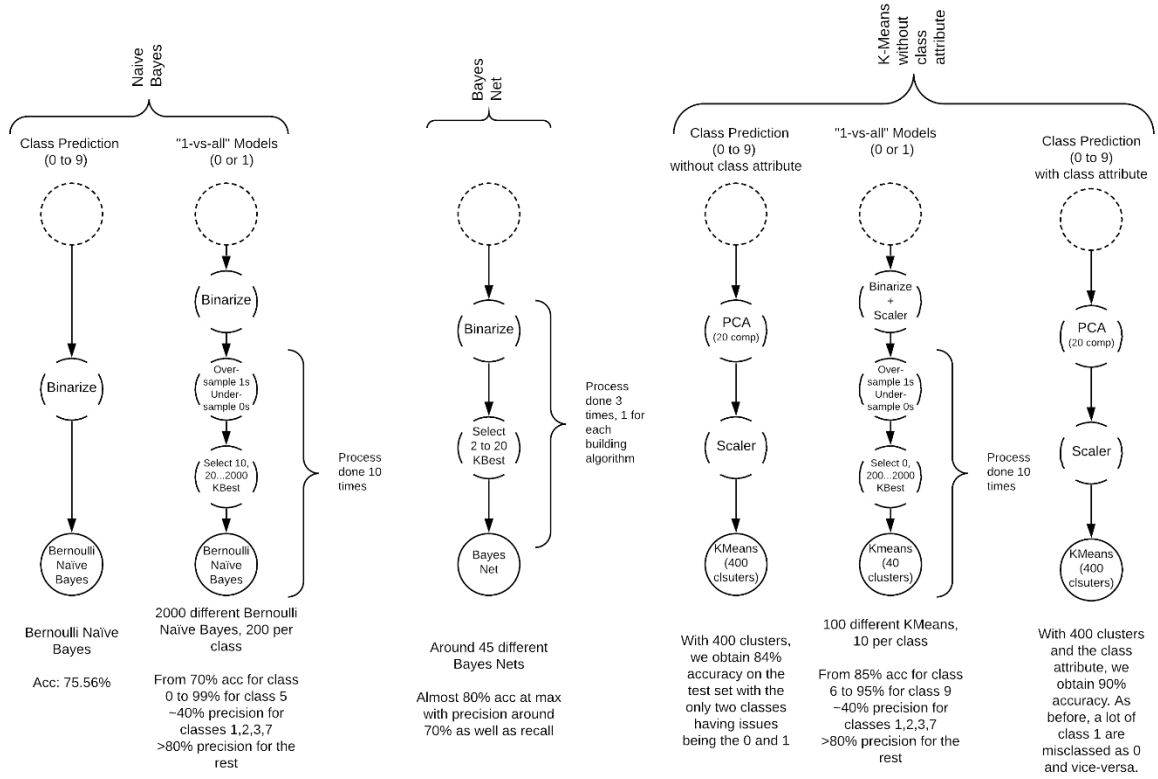
For this coursework we have been using Python and a lots of common Data Science libraries such as: pandas, seaborn, OpenCV, numpy, sklearn, pomegranate, imblearn or even Keras.

I. The Dataset

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

As we can see, our dataset is very unevenly distributed: when we will start training our models, we may have to balance it with under/over sampling strategies. Each of these classes corresponds to one type of German street signs, modelized by a 48x48 image shaped in a row of (2304,) pixels with values between 0 and 255. We preprocessed our Dataset with different filters depending on the algorithm we were using.

II. Our Models



a. Naïve Bayes

The first classifier that we use is Naïve Bayes. We had to work on two problems: a general problem, predicting class from 0 to 9 then on 10 different "*1-vs-all*" problems. For the former, we only used binarization when for the latter, as our dataset suffered from strong imbalances, we used binarization followed by over & under-sampling (using SMOTE and random under sampler) then we created 200 different models for each "*1-vs-all*" problem to find the best number of attributes to keep for each class.

i. General Problem

Our first model has an accuracy of 75.56% on the test against 76.59% on the train set. The results are very good considering the complexity of the task and we are delighted to see homogeneity in the performance between the train & test sets.

On the right side, you will see the confusion matrix of our test sets results:

$$\begin{bmatrix} 202 & 170 & 28 & 2 & 32 & 2 & 0 & 1 & 15 & 2 \\ 137 & 386 & 19 & 4 & 52 & 1 & 1 & 0 & 17 & 2 \\ 6 & 15 & 116 & 0 & 0 & 0 & 0 & 2 & 8 & 0 \\ 6 & 41 & 1 & 376 & 5 & 0 & 0 & 0 & 10 & 6 \\ 17 & 105 & 0 & 1 & 532 & 0 & 0 & 3 & 17 & 13 \\ 16 & 33 & 0 & 0 & 27 & 641 & 4 & 0 & 22 & 4 \\ 0 & 9 & 4 & 0 & 1 & 0 & 232 & 0 & 13 & 0 \\ 10 & 6 & 0 & 0 & 1 & 0 & 0 & 56 & 0 & 0 \\ 19 & 71 & 10 & 1 & 9 & 0 & 6 & 0 & 518 & 0 \\ 2 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 102 \end{bmatrix}$$

As we can see class 0 and 1 (top left corner) have trouble being identified from each other. These two classes which are "60_limit" and "80_limit" street signs are, therefore, very similar to the eye of our classifier and the confusion in the model prediction is, indeed, understandable: the two signs look exactly the same except in the number 6 and 8 that even have the same overall round shape. Also, a surprising fact is how well this algorithm behave on rarer classes such as the class 7: 56 out 62 are classified correctly

ii. "*1-vs-all*" problems

In order to compensate the rarity of some classes and avoid the precision bias (algorithm predicting false every time, while still reaching high accuracy), we will be using oversampling and down sampling.

Here is the distribution of each class on the 1 vs all problem, 0 being false and 1 being true:

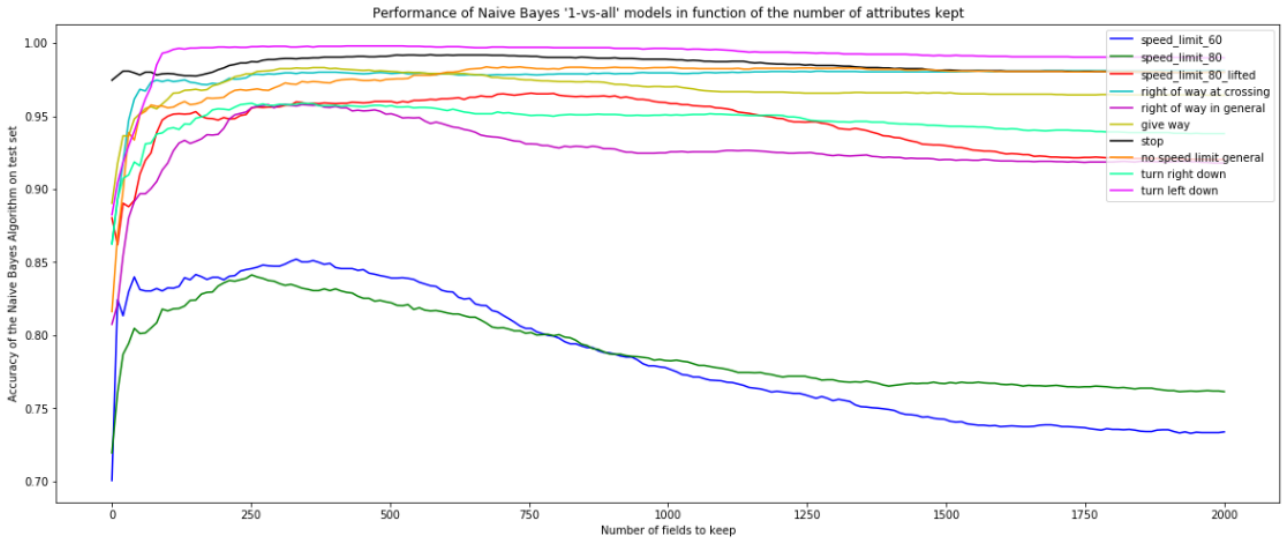
	0	1	2	3	4	5	6	7	8	9
0	7526	7241	8209	7607	7070	7069	7961	8315	7046	8294
1	956	1241	273	875	1412	1413	521	167	1436	188

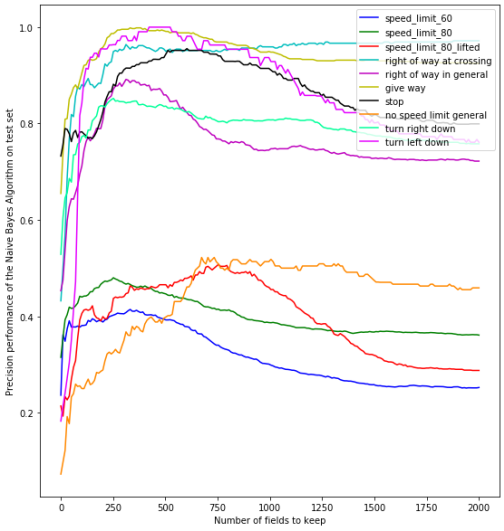
Oversampling will artificially create new data for the class in minority (class 1) from the existing data. Under sampling will delete data in majority (here class 0). We used the SMOTE technique for over-sampling and a random process for under sampling.

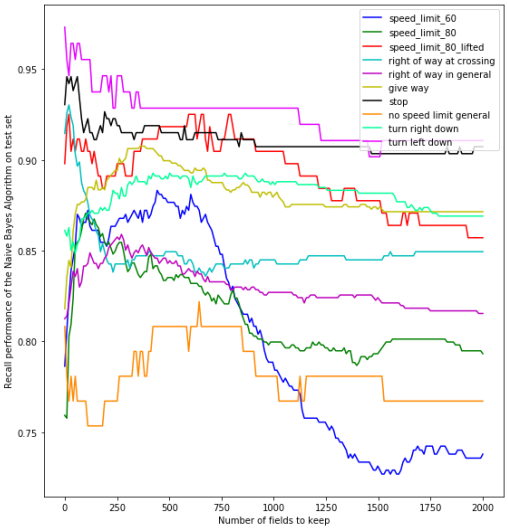
Using these techniques together we have a balance data set for each class with 50% 0s (false) and 50% 1s (true).

	0	1	2	3	4	5	6	7	8	9
0	2508	2413	2736	2535	2356	2356	2653	2771	2348	2764
1	2508	2413	2736	2535	2356	2356	2653	2771	2348	2764

Now that our data is balanced, we will reduce the number of attributes to work with, selecting only the most important ones for each class. Previously we were using the whole 2304 pixels of each picture: we now pick up the n best attributes for each class, with n ranging from 10 to 2000, looping with a step of 10. This is our accuracy performances for each model, ranging from 75% to almost 100%, on our test set.







We now see how different the 10 classes in 10 " *1 vs all* " problems results are:

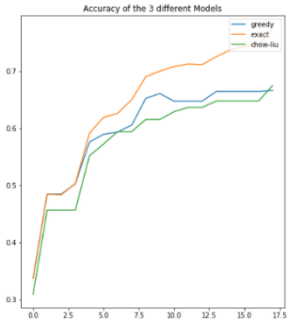
Even with a balanced training dataset, we have **4 classes with a low precision score**: Speed limit 80, 60 no speed limit general and speed limit 80 lifted. As we had expected before, our algorithm still struggles with correct classifications of the speed limit signs: their similarities make them very hard to deal with when just using a Naive Bayes algorithm. Surely a simple Naïve Bayes algorithm does not have enough complexity to make the difference between an "8" and a "6" consistently.

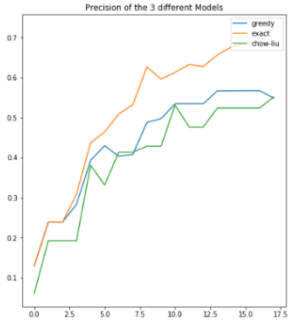
On the other hand, we could really cheer ourselves with the **overall very good results of the 6 remaining classes**, all offering a precision score of above 75% and a top-level accuracy (even if not relevant in this context). This is also the case of some of our initially rare classes like the "*turn_left_down*", representing initially only 3% of the positive cases in its "*1-vs-all*" scenario. This is a proof that our choice of using a mix of over and under sampling was a good call.

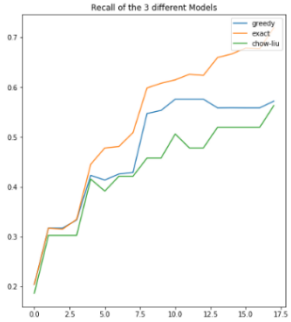
Also, we can clearly see that some attributes have some of the best predictions around 500 to 750 attributes selected but as soon as we get the number of attributes up, the prediction performance down to even reach their worst possible performance. We could interpret this as adding too much noise to the data, making it even harder for Naive Bayes to sort through all kinds of fields for similar signs.

We invite to check our .gif animation to see which attributes is the most important for each class.

b. [Beyond Naïve Bayes: complex Bayesian Network Architecture]



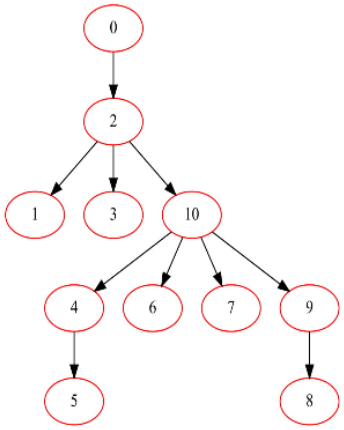




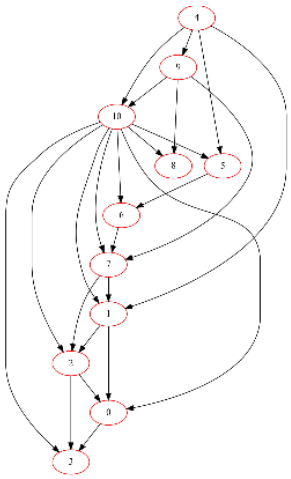
In order to make the development of a Bayes network easier, we will develop it using binarized images, no longer in shades of gray. We had three different algorithms to generate these perf plots.

We see around 75% acc for the 'exact' algorithm when the other two are around the same 65% accuracy. They all hold strong precision/recall scores with 65% each.

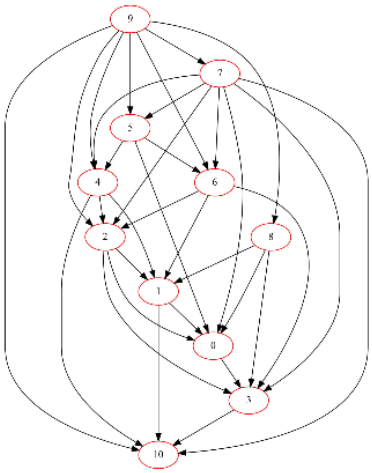
We can see in these curves than every algorithm generates a stable model. For all of them, the more pixels they have, the better they behave. The worse performing is the Chow-Liu algorithm, but its main objective is to come up with the fastest reliable solution possible and it does its job well by being faster than the other algorithms.



Layout for the chow-liu algorithm



Layout for the exact algorithm



Layout for the greedy algorithm

Chow-Liu: this algorithm is supposed to find a fast and somewhat reliable solution. This is by far the simplest and the class attribute has only one direct predecessor: pixel n°2. However, it has a direct influence on the values of the pixels **n°9, n°4**, n°6 and n°7.

Exact: the exact algorithm, using DP/A*, finds the best layout. Here, the class attribute is way up ahead with only two predecessors: **pixels n°4 and n°9.** It however dictates the values of the rest. This graph resembles more what we were used to.

Greedy: the greedy algorithm generates a very complex architecture where the class is at the lowest end of the graph. We can here see that the value of the class (n°10) is directly linked to the values of the **pixels n°9**, n°3, **n°4** and n°1

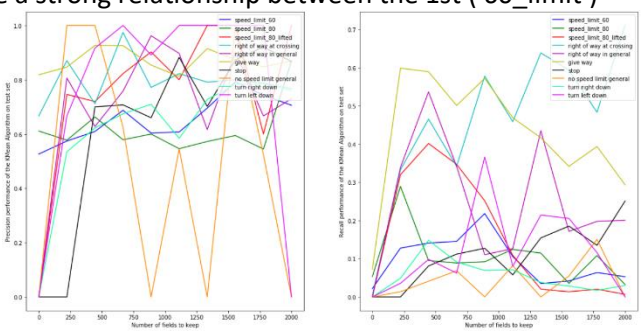
c. [Clustering, k-means]

For K-means, in the general problem, we replaced the binarization preprocessing with a PCA, to reduce pixels dimensions and a scaler, to help calculating distances. When launching our training procedure with 400 (40 * number of classes) clusters, we have the following results:

```
0.8439444710387746
Confusion matrix:
[[257 170   0   5   1   6   1   1  13   0]
 [145 439   2   4   6  11   0   1  11   0]
 [  9  12 112   0   5   1   0   4   3   1]
 [  2   3   0 407   8   6   0   0  14   5]
 [ 10  21   0   2 638  10   1   0   5   1]
 [  1  13   0   3   5 716   1   0   8   0]
 [  3   5   0   0   1   3 240   0   7   0]
 [  5  10  26   0   3   0   1  25   1   2]
 [  6  14   1   0   5   6   6   0 594   2]
 [  3   4   2   2   1   1   0   0   1  98]]
```

```
0.9028243178554333
Confusion matrix:
[[351 102   1   0   0   0   0   0   0   0]
 [163 455   1   0   0   0   0   0   0   0]
 [  3  18 125   1   0   0   0   0   0   0]
 [  0   2   1 409  33   0   0   0   0   0]
 [  0   1   0  16 667   4   0   0   0   0]
 [  0   0   0   0   4 743   0   0   0   0]
 [  0   0   0   0   2   4 248   0   5   0]
 [  0   0   0   0   0   0   1  62  10   0]
 [  0   0   0   0   0   0   2   0 619  13]
 [  0   0   0   0   0   0   0   0  19  93]]
```

First result is without the class attribute, second is with the class attribute. We can see it plays a little role improving overall prediction by 0.05%. We can still note a strong relationship between the 1st ('60_limit') and second class ('80_limit'), which both are speed limit: our algorithm often miss-classifies them both.



When it comes to “1-vs-all”, the results were less interesting: we can see very good behaving class like n°5 or n°0 but the class 7 has very confusing precision results.

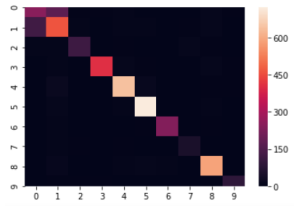
KMeans is better in a general case than “*1-vs-all*”.

d. [Beyond K-Means]

With compared other clustering methods for the general prediction. We have the following results so far:

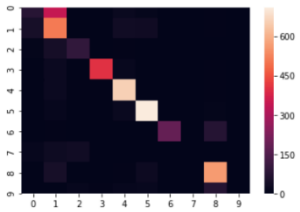
`AffinityPropagation`
Creates clusters by communicating between pairs of samples until convergence.

`Acc: 0.85`



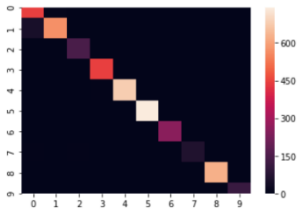
`GaussianMixture`
Generally used for density estimations but is struggling with uneven data distribution

`Acc: 0.76`



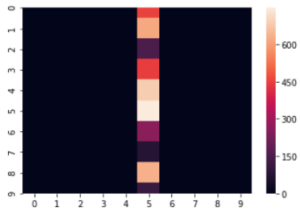
`Birch`
Can be used in large dataset and calculate the Euclidean distance between points + creates Tree of characteristic features of the data.

`Acc: 0.97`



`MeanShift`
Tries to identify blobs in a sample

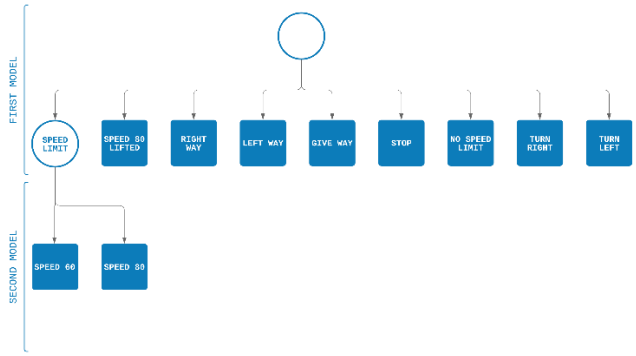
`Acc: 0.17`



As we can see, all our algorithms' performances are nothing compared to the Birch clustering algorithm where a clustering features tree made from the data points is created.

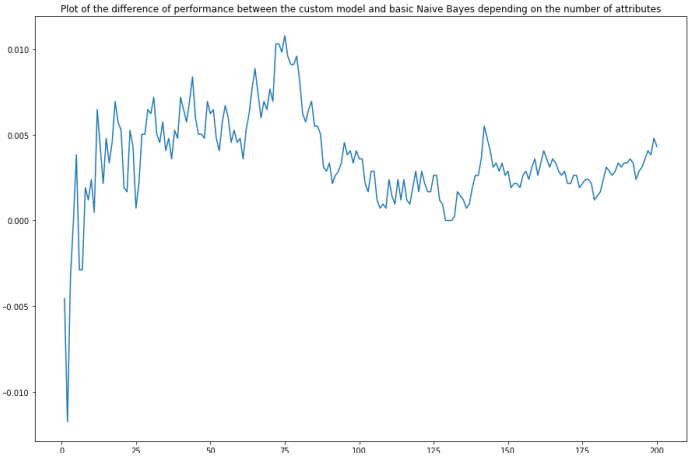
III. Research question: Can we improve Naive Bayes classification by adding a top-level class embedding all 2 speed limit street signs?

We have seen overall good performance by the Naive Bayes algorithm, with accuracy of 75.6% and a confusion matrix depicting very good precision and recall score. However, we could not help but to notice that the most frequent error that occurred with the Naive Bayes algorithm was when he was predicting the speed street-signs.



Therefore, we had an idea: why not create a new class called "speed limit sign" and then, add a second model that would decide what kind of speed limit sign has been predicted. This could be summed up with this architecture: we would have two models, one predicting 9 classes, one predicting two classes. We replaced the two speed limit street signs by a new class called "speed limit". Whenever a speed limit is predicted, we then call the second model to say if it is a 60 or an 80 speed limit sign.

Our first model uses Multinomial Naïve Bayes when the second is a Bernoulli one. When compared with traditional model, we have the following results:



This graph plots the difference of accuracy between our model and the usual model in function of the number of attributes kept: we can see that the more attributes we have, the more consistent our solution accuracy tends to be (around 0.5% improvement). This may be a clear sign that using a tree-like hierarchy of abstract concepts would help to have more precise Naïve Bayes predictions.