COMP 472 – Mini project 2

Machine Learning - Analytical report

November 17th, 2018

Victor Debray (ID: 40102554)
Theo Penavaire (ID: 40102474)

This report provides a description of the research made for the COMP 472 course's second project. For this project we had to use a machine learning framework to experiment with different algorithms and datasets.

We chose to use scikit-learn, which is an open-source machine learning library for Python. We chose it over WEKA because we are feeling more familiar with Python than with Java.

In this report, we will first explain our experimental setup, and give a description of the algorithms and hyper parameters we used. We will then study the resulted output, before drawing final conclusions.

I. Experimental setup

A. Requirements

For this project, we were given two datasets, representing black and white images of latin and greek characters. An instance (row) of the data is represented by 32x32 pixels, either true if the pixel is black, or false if it is white.

For each dataset, we had to generate six output files, as follow:

- Validation set:
    - Decision Tree classifier output
    - Naïve bayes classifier output
    - Another ML algorithm classifier output
- Test set:
    - Same files as for the validation set.

We used the scikit-plot library to generate charts from the results. The results are based on the validation sets, as they are the only ones (apart from the test sets) with both features and correct labels.

B. Naïve Bayes Algorithm

While working on the Naïve Bayes (NB) algorithm, we noticed that several implementations were possible:

Bernoulli NB

- Multinomial NB
- Gaussian NB
- Complement NB

We chose to study two of them, because of their contrasted output.

Bernoulli Naïve Bayes

This algorithm is suitable for discrete data, that is, data that can only take particular values. The Bernoulli NB algorithm is designed around booleans expressing the occurrence or absence of features, rather than around their occurrence frequencies. This is therefore suitable for vectors of binary features (0s and 1s).

Gaussian Naïve Bayes

On the other hand this algoritm is better when used for features that follow a normal distribution.

Intuitively, it would seem that the Bernoulli Naïve Bayes is more appropriate for our type of data.

C. Decision Tree

The Decision Tree (DT) algorithm can be used for classification and regression problems. We will be using classification, as requested in handout, to solve our problem.

The decision tree is the easiest classification algorithm to understand, as it tries to solve the problem with a tree representation. Each node of the tree corresponds to a feature and each leaf node to a class label.

The scikit-learn framework enables us to play with many hyper-parameters. We will choose 2:

- **Criterion**: measures the information gain, by entropy or Gini impurity
- **Splitter**: choice of split, either best split or random of best split

Intuitevely, it would seem that the best split method is more appropriate than random split. Best split seems to be more precise.

D. Multi-layer Perceptron

We decided to challenge our knowledge of perceptrons, acquired in class. That is why we chose to play with multi-layer perceptrons for our third set of experimentations.

To study multi-layer peceptron classifiers, we focused on the influence of the following hyper-parameters:

- **Activation**: This is the activation function to use for hidden layers
- **Solver**: This refers to the weight optimization when backpropagating
- **Number of hidden layers**

To simplify our process, we used a utility provided by the machine learning framework itsef. This is an object that allows the user to test a combination of hyper parameters and to pick the best one. It also provides a detailled report of the tests. This object is called “GridSearchCV” and is available in the “sklearn.model_selection” namespace. Using this object, we are sure to use the best possible hyper-parameters for the datasets.

The classifier we will use implements an algorithm that trains using backpropagation.

II. Analysis of the results

A. Naïve Bayes algorithm

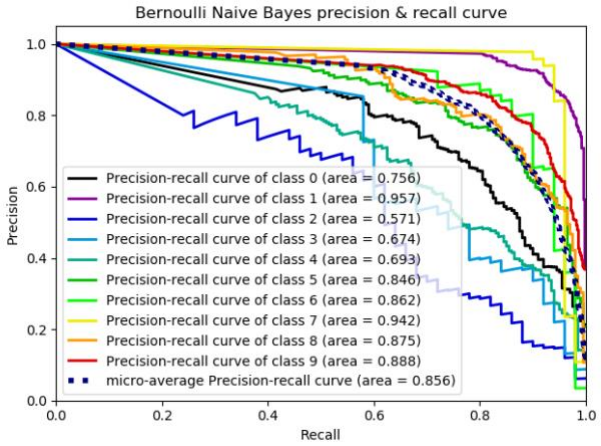
For these experimentations, we tested the Naïve Bayes algorithm with the second dataset, which produces a more readable output for charts (less features and output classes). It is also the biggest dataset, with 6400 instances in the training set and 2000 in the validation one.

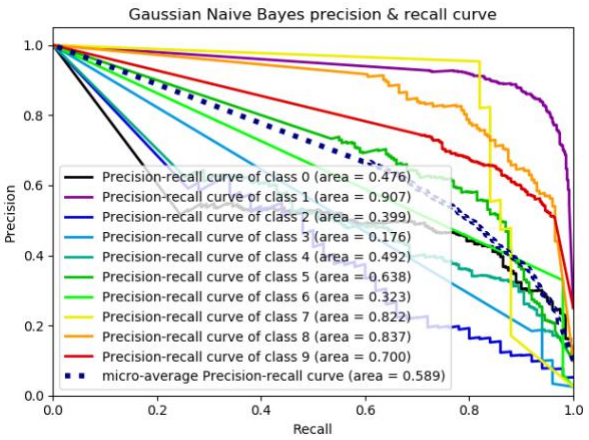
The simplest comparable output is the accuracy.

	Accuracy
Bernoulli Naïve Bayes	80.05%
Gaussian Naïve Bayes	64.2%

Obviously, the Bernoulli NB algorithm seems to be more accurate. But accuracy alone is not enough to measure the quality of our models, particularly if we don't know if the set is correctly balanced.

This is why we need the following graph to examine the precision and recall of these two models:





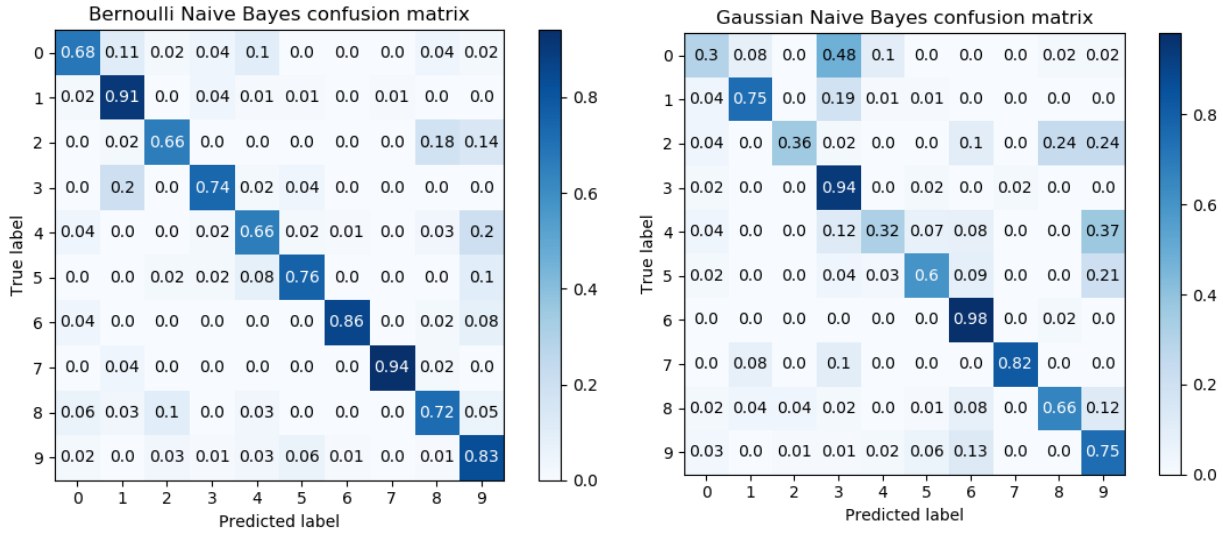
Precision and recall are smaller with the Gaussian NB algorithm. Indeed, if we compare the average precision-recall curve (dashed line), for a precision of 60% we have a recall of 90% with Bernoulli, and a recall of about 70% with the Gaussian algorithm.

Here is a comparison of F-Measures for these two models:

	F-Measure
Bernoulli Naïve Bayes	≈ 75.66%
Gaussian Naïve Bayes	≈ 58.22%

Again, the Bernoulli NB model has a better F-Measure than the Gaussian NB model

Finally, here are the error matrices for these algorithms. Again, they corroborate our conclusion.



Even if most of the predictions are correct for both, the output of the Gaussian Naïve Bayes model contains more approximations (wrong predictions for *pi:* only 30% recognition, *beta:* 36% and *gamma:* 32%).

B. Decision Tree

To experiment, we used the combination of best and random split with the Gini Impurity and Information Gain Entropy criterion.

The values of Accuracy and F-measure are all average of 10 launches of the algorithm on the same data set, with the same parameters. As we can see, neither the Gini nor the Entropy parameters has a significant impact on Accuracy or on F-Measure, wether on a small or large dataset.

Entropy is however computationally intensive as it uses logarithms.

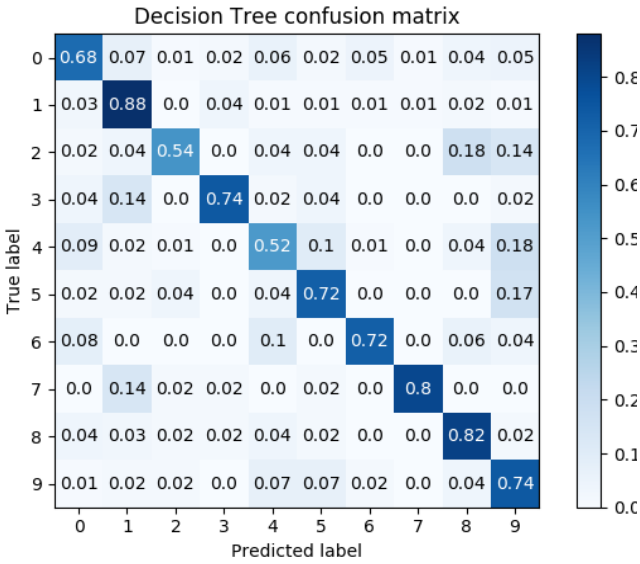
Accuracy in DS1	**Best**	**Random**
Entropy	32.5%	30.3%
Gini	30.6%	27.4%

Accuracy in DS2	**Best**	**Random**
Entropy	75.9%	74.8%
Gini	76.1%	75.4%

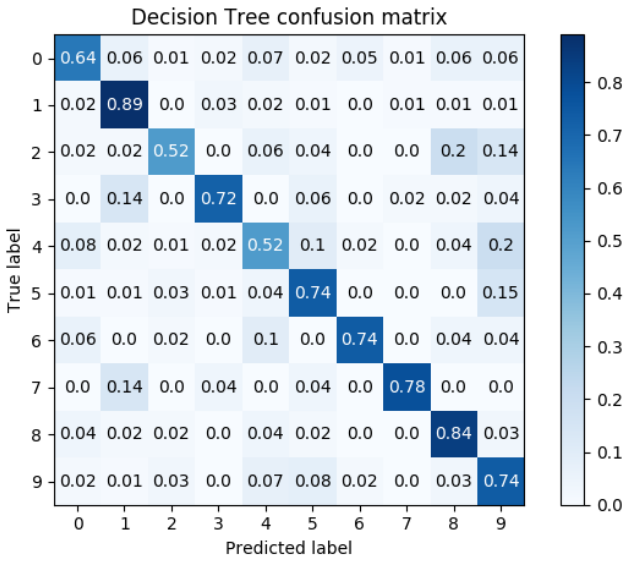
F-measure in DS1	**Best**	**Random**
Entropy	31.3%	29.2%
Gini	29.8%	26.4%

F-Measure in DS2	**Best**	**Random**
Entropy	72%	70.2%
Gini	71.1%	70.4%

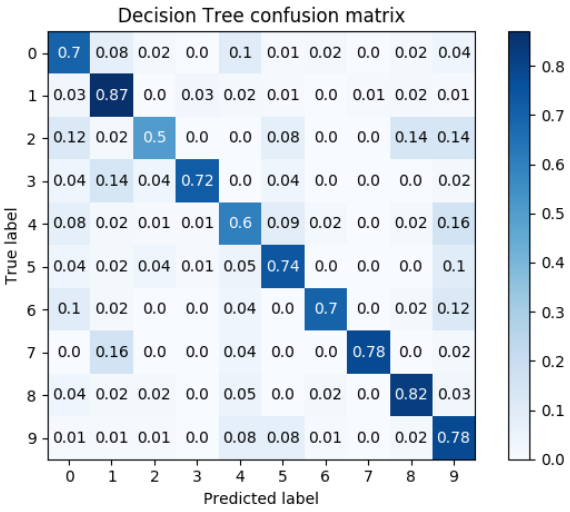
Again with the confusion matrices, we can see very light difference, the difference being the same between small or large dataset.



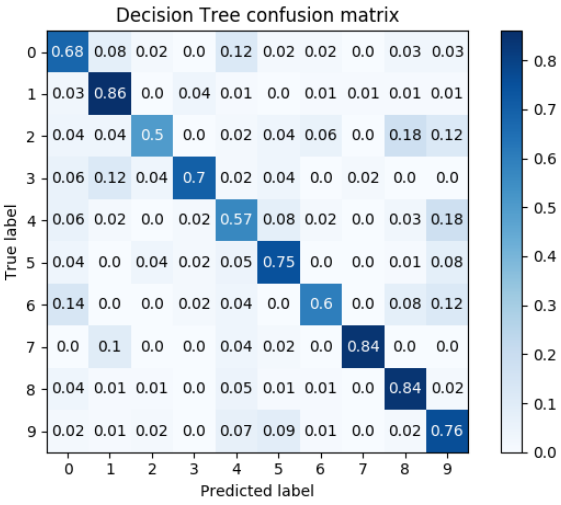
Entropy-Best Confusion Matrix



Entropy-Random Confusion Matrix



Gini-Best Confusion Matrix



Gini-Random Confusion Matrix

C. Multi-layer Perceptron

For these experimentations, we also tested the classifier with the second dataset. It is the biggest dataset, with 6400 instances in the training set and 2000 in the validation one.

The purpose of the following tests is to find the best possible hyper-parameters. A little reminder, these results are obtained with a utility provided by scikit-learn (GridSearchCV), that tests each parameter and chooses the best one. The tests are performed individually, so that other hyer parameters do not influence the results.

Activation

In scikit-learn, the activation function can be one of the following:

- **Identity**: f(x) = x
- **Logistic**: Logistic sigmoid function: f(x) = 1 / (1 + exp(-x))
- **Tanh**: Hyperbolic tan function : f(x) = tanh(x)
- **Relu**: Rectified linear unit function: f(x) = max(0, x)

The scores for the different activation functions based on accuracy, precision and recall were the following:

	Accuracy	**Precision**	**Recall**
Identity	0.825	0.807	0.775
Logistic	0.852	0.849	0.813
Tanh	0.849	0.846	0.808
Relu	0.834	0.827	0.796

The colored row represents the highest values for this test. The logistic sigmoid function is therefore the best one for our model. Another thing to notice though, is that changing this hyper parameter does not seem to have a huge impact on the output. Indeed, all the values above are relatively close to each other (scores between 0.775 and 0.850 on average for each measure).

Solver

This is the parameter responsible for the implementation of the weight balancing during backpropagations. It can be one of the following:

- **lbfgs**: Optimizer in the family of quasi-Newton methods
- **sgd**: Stochastic gradient descent
- **adam**: Stochastic gradient based optimized

The scores for the different weight balancc systems based on accuracy, precision and recall were the following:

	Accuracy	**Precision**	**Recall**
lbfgs	0.837	0.804	0.779
sgd	0.722	0.433	0.459
adam	0.851	0.847	0.809

Here again, the colored row represents the best parameter (the one with highest values). This hyper-parameter seems to have a greater influence on the output than the activation function. In particular, the optimized stochastic gradient descent system seems to be much better than the “classic” one. It is almost 2 times better in terms of precision (43% for sgd, 85% for adam), and recall (46% for sgd, 81% for adam).

Number of hidden layers

We increased the number of hidden layers by 10 and then 25, from 0 to 100.

	Accuracy	**Precision**	**Recall**
1	0.371	0.590	0.100
10	0.795	0.755	0.680
20	0.839	0.826	0.787
30	0.849	0.842	0.800
40	0.849	0.853	0.786
50	0.856	0.851	0.807
75	0.853	0.853	0.807
100	0.855	0.850	0.811

The higher a score is, the darker its cell. This is to point out that the optimal number of hidden layers should be around 75. An important thing to notice is that around this point, all of the scores seem to have reached a plateau. The improvements when adding layers, if present, are very small.

With about 75 hidden layers, we can say that backpropagation is optimized, and errors are minimized.

III. Conclusion

A. Naïve Bayes

Our hypothesis was that the Bernoulli implementation of the Naïve Bayes algorithm better suited our situation.

The results corroborate this hypothesis. Indeed, our dataset contains only boolean values, which is why it doesn't work well with the Gaussian algorithm. The latter focuses on features that follow a normal distribution.

On average with the two datasets, we achieved to reach an accuracy of 70% and an F-Score of 67.5% with the Bernoulli implementation.

If we were to continue working on this project, we would have liked to perform tests with other implementations of the Naïve bayes classifier. Each implementation should probably perform better than the others with different datasets and configurations.

B. Decision Tree

As we intuitevely predicted, using the best split criterion gives a better accuracy and f-measure. This comes without surprise as best split makes consistent decisions along the tree building.

According to the paper 'Theoretical comparison between the Gini Index and Information Gain criteria': the most important remark is that it only matters in 2% of cases wether you use the one or the other. However it does not mean that for some specific datasets there might not be significant differences Plus Entropy Information Gain using logarithms, it is computationnaly more intensive.

With our datasets and according to the experiments, we see that we will prefer to use Entropy Information Gain, the f-measure being higher on average.

C. Multi-layer Perceptron

By adjusting the activation function, the solver and the number of hidden layers, we were able to improve our results.

Before thinking about the parameters, and with the default values the accuracy for the first dataset was 62%. The F-Measure score was 60%. For the second dataset the accuracy was 85% and the F-Score was 81%.

With the new configuration, we reached an accuracy and F-Score of both 67% for the first dataset. For the second dataset, respectively 90% and 87%.

In average, the scores were increased by 7 to 8 percent.

If we were to continue working on this project, we would have liked to experiment with other hyper parameters. The utility we used to improve our search of the best combination unfortunately used a lot of time and processor resources. Indeed, it has to test the best possible configuration for each of the combinations.

D. Comparison of the classifiers

We wanted to finish these experimentations by giving a comparison of all of the classifiers that we studied. Here is a table to summarize the results.

Accuracy:

	Naïve bayes	**Decision Tree**	**Multi-layer perceptron**
Dataset 1	60%	29%	65%
Dataset 2	80%	71%	90%

F-Score:

	Naïve bayes	**Decision Tree**	**Multi-layer perceptron**
Dataset 1	60%	29%	64%
Dataset 2	75%	71%	87%

It is safe to say that the multi-layer perceptron classifier, for these datasets, is more efficient. Then comes the Naïve Bayes classifier, and the Decision tree classifier.

IV. Acknowledgements

Scikit-Learn documentation (machine learning framework for Python): https://scikit-learn.org/stable/documentation.html

Scikit-plot documentation (a tool to generate charts directly from Python's code): https://scikit-plot.readthedocs.io/en/stable/index.html

A list of classifiers: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

Tuning the hyper-parameters of an estimator: https://scikit-learn.org/stable/modules/grid_search.html

Theoretical comparison between the Gini Index and Information Gain criteria: https://www.unine.ch/files/live/sites/imi/files/shared/documents/papers/Gini_index_fulltext.pdf