AI Project 2 : Character recognition with supervised Machine Learning

Yann Kerichard

17th November 2018

1 Introduction

During this second project, I was asked to use Scikit-learn, a popular machine learning framework based on Python, to create and train models able to recognize handwritten characters. These characters take the form of 1024 binary features, each corresponding to one pixel. We had to use different techniques such as Naive Bayes and Decision Tree classifiers to train the model. The goal was to be able to have an overview of different supervised machine learning techniques, to be able to compare them, and to see the impact of hyper-parameters on the results.

We were provided .csv files containing the whole data-set (training, validation and testing sets), representing handwritten characters. The first data-set provided was about our classical alphabet (with upper cases and lower cases), whereas the second data-set was about a Greek alphabet composed of 10 letters.

This report tries to bring an answer to the following questions: How to learn to a program to recognize characters? What are some of the different solutions that can be applicable to a such problem? And how does each of the techniques I used compete between each other (Which one is the best)?

To bring an answer to those questions, I will detail exactly how I generated and trained the asked models, then, you will also learn what I experimented above the requirements, so you can get an overview of my whole work. Then, you will find a second section focusing on the analysis between each technique used and discussing of each hyper-parameter's impact. Each technique will be described in detail, so you will be able to understand how they work to recognize characters, and how they compete with each others. In a third part, I will let you know the difficulties I encountered and how I addressed them. You will also be able to read what would be, in my opinion, interesting to investigate by continuing this project.

2 Realization

2.1 Required techniques

My solution is composed of one python script which read .csv files, load them in 3 maching learning models, train the models, realize some experimentation with different configurations and generate the output files. Only 2 techniques were asked : Naive Bayes and Decision Tree classifiers.

Concerning the first classifier, it is based on probability calculations, using Bayes formulas. It assigns a weight to each feature, depending on their probability to occur regarding other features of a particular item. In our case, each feature is a binary number (e.g. 1 is colored and 0 is white). The, using the provided data-set, the model will assign weights to each pixel, corresponding to their probability to appear for a particular letter. Then, when the model has used all the data in the training set, we also give him the validation set, so our model can compute the accuracy, and we can then adjust hyper parameters in order to increase this accuracy. Finally, we provide to the model the test set, for which we do not have the results, and the Naive Bayes classifier predicts the corresponding letter based on what he learned from the previous examples. For the Naive Bayes classifier, there was different algorithms such as Gaussian, Bernoulli, etc... but I stuck to the multinomial one because it was the one asked in the requirements. I also found that it was a bit more efficient to modify the alpha parameters. This parameter corresponds to the smoothing. I will discuss about the impact of each parameter in another section, but I set the smoothing one to 0.5.

Concerning the Decision Tree classifier, it is based on information gain. I will not re-explain how it works, but I could choose between two different methods of calculation. The most classical is the one seen in class which is based on entropy. However, there is also an other method based on Giny impurity and which is pretty complex. To generate the output files, I chose to work with entropy and an alpha parameter set to 1.

However, the project description was mentioning about choosing a third classifier or my choice. Then, I decided to work with neural networks, because I really found interesting to compare the techniques that we studied theoretically in class. In scikit-learn framework this model is named as multi-layer perceptrons, because there are many different kinds of neural networks. I really like this model because you can choose the number of perceptrons that you want, each corresponding to a small computation unit. So, for this model, I put the smoothing parameter to 0.5 and I set the number of perceptrons to 200 per layer, with 80 layers.

2.2 Experimentation

During the project, I also decided to make some experimentation by changing hyper-parameters or also by using other classification algorithms.
First, about the decision tree classifier, I decided to compare the entropy to the giny impurity. I did not know about giny impurity [1], but it allows to indicate the frequency of a random element to be misclassified. I found that it was interesting to compare with entropy. I also changed the splitter parameter. When you work with decision tree classifier, the different possible classes are set into a decision tree with many possible configurations. Generally, a good approach is too split the leaves of the tree regarding the most probable frequency of

apparition for a specific class. However, two options were available : best and random. It allows us to see the consequences on the final accuracy if the tree has different configurations.

Then, I also modified the hyper parameters for Naive Bayes classifier. I decided to change the way to calculate the probabilities. So, I experimented it with Gaussian and Bernoulli approaches and I compared it to the multinomial one. In my opinion, it particularly relevant because, it demonstrates that using the same classifier, but associated with different calculation for the weights, has consequences on the final result. I also modified the smoothing parameter (alpha) to see how important the smoothing function can be for the final accuracy, and how much it can fake the results if we take a too large number. Concerning this smoothing parameter, I put it to a very low number and to a huge number to show the difference.

Finally, I found relevant to see the impact of the number of perceptrons and hidden layers, both on the results and on the computation time, for the neural network classifier. More specifically, I tried the following dimensions : 5x2, 50x20, 100x40, 200x80.

And, obviously, I made those experimentation for both data-sets.

3 Analysis of the results

I will divide the analysis of the results in 4 sections. 3 sections will be used to compare the impact of the hyper parameters I experimented for each of the 3 first classifiers. Then, the last section will be dedicated to the comparison of the classifiers. All the accuracy results in the tables below are converted into percentage.

3.1 Naive Bayes classifier

In the following table, you will be able to see the results of my experimentation with Naive Bayes classifier, for data-sets 1 and 2.

Dataset	Algorithm	Smoothing	Accuracy
1	Bernoulli	1	59,92
2	Bernoulli	1	80,05
1	Gaussian	1	43,77
2	Gaussian	1	64,2
1	Multinomial	1	49,81
2	Multinomial	1	78,6
1	Multinomial	1000	34,44
2	Multinomial	1000	70,65
1	Multinomial	No	49,61
2	Multinomial	No	78,6

Figure 1: Experimentation with Naive Bayes classifier

In the table above, we can easily see that the 3 algorithms used to calculate the probabilities are not equals. As you can quickly notice, there is a big difference between the

results provided for data-set 1 and those provided for data-set 2. This difference will have an explanation in the last subsection of this section.

Concerning the hyperparameters, Bernoulli algorithm outclasses Gaussian by far, and is a bit above the multinomial one, for both data-sets. However, we can see that this parameter does not create the same difference for both data-sets. As an example, whereas the difference for the first data-set between Bernoulli and Multinomial is around 10%, for the second data set (which contains less classes), this difference drops to 1.4%. We can then wonder if using data-set much bigger or much smaller, would change the efficiency of those algorithms.

I also wanted to compare the smoothing value. I think that this parameter has not a very big impact in this project, for 2 main reasons. Firstly, I would say that the number of 0-values is not so high comparing to the number of 1 for each letter.So, that would explain why we have almost no difference with or without any smoothing. Secondly, as you can see, when I set this parameter to a big number such as 1.000, the accuracy drops. It was expected because, in proportion, the more the smoothing value increases, the more the difference of weights of the features is reduced. Considering that we have 1.024 features, putting a smoothing parameter of 10.000, for example, would completely false the result: the information (true) value is lost in the proportion of smoothing (fake) value.

3.2 Decision tree classifier

In the following table, you will be able to see the results of my experimentation with Decision Tree classifier, for data-sets 1 and 2.

Dataset	Information	Smoothing	Splitter	Accuracy
1	Gini impurity	1	Best	28,4
2	Gini impurity	1	Best	76,95
1	Entropy	1	Random	29,18
2	Entropy	1	Random	76,5
1	Entropy	1	Best	30,73
2	Entropy	1	Best	75,3

Figure 2: Experimentation with Decision Tree classifier

The first hyperparameter I used is the information value that we use in the decision tree classifier. In class, we saw the entropy which corresponds to a quantity of information, but I was not aware of the second one : the Giny impurity [2]. The Gini impurity is simply the probability of obtaining two different outputs, which is an "impurity measure", during the random classification of a class. We can notice that there are very few differences between both techniques.

The second hyperparameter is the splitter. The strategy used to choose the split at each node. Supported strategies are best to choose the best split and random to choose the best random split [3]. The impact of this parameter is, once again, not very important. We notice almost the same accuracy for configurations using Best and Random strategies.

3.3 Neural network : Multilayer perceptron classifier

In the following table, you will be able to see the results of my experimentation with Multilayer perceptron classifier, for data-sets 1 and 2.

Dataset	Perceptrons/Lyr	Hidden layers	Perceptrons	Accuracy
1	5	2	10	2,33
2	5	2	10	25
1	50	20	1000	55,45
2	50	20	1000	86,55
1	100	40	4000	60,31
2	100	40	4000	88,1
1	200	80	16000	64,55
2	200	80	16000	89,1

Figure 3: Experimentation with Multilayer perceptron classifier

If you observe the table with attention, you will notice that with few perceptrons in the network, the result is not accurate at all : accuracy of data-set 1 is only 2.33% with 10 perceptrons whereas data-set 2 has 25%. I suppose that this huge difference with the same configuration for both data-sets is due to the fact that data-set 1 has very few training instances comparing to data-set 2 1960 against 6400. Moreover, the information brought by theses instances is kind of split between the number of possible classes which is 51 for data-set 1 and only 10 for data-set 2. Considering the fact that we only have a few number of perceptrons, the quantity and the quality of information provided for each class in both data-set has really a strong impact. However, in the opposite case, if we have a big number of perceptrons in the network, the quantity and the quality of information provided for each class in both data-sets is not as much important.

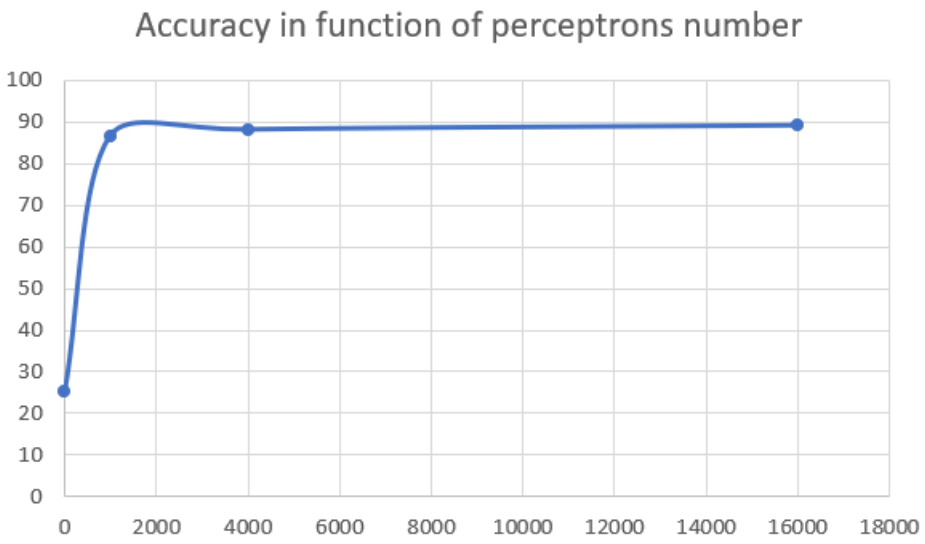

Figure 4: accuracy in function of the number of perceptrons in the network

Here is a curve of the accuracy in function of the number of perceptrons in the network. I also that with a few number of perceptrons, the model is not well trained because each perceptron in the network did not practice enough, resulting in giving bad results. By increasing a bit this number, the accuracy rockets, and the computation time is a bit longer, but still good. However, inserting a very huge number of perceptrons becomes inefficient at a certain point because the results tend to stabilize and the computation time rockets.

3.4 Comparison between data-sets and classifiers

In order to facilitate the understanding of what I am going to explain, I decided to compute the average accuracy for each data-set and classifier.

	Dataset 1	Dataset 2	**Classifier average**
Naive Baye	47,51	74,42	**60,97**
Decison Tree	29,44	76,25	**52,84**
Neural Network	60,10333333	87,92	**74,01**
Dataset average	**45,68**	**79,53**	-

Figure 5: Average accuracy for each data-set and classifier

In this table, all the data used to compute the average come from the configurations used in the experimentation part.

We can notice that the accuracy average for data-set 1 is drastically different from the one computed for data-set 2. This is easily explainable, and I see 3 main reasons to this observation. The two first reasons are about the quantity, whereas the third one is about quality.

First, the amount of training data for data-set 1 is much lower comparing to the amount for data-set 2. This is logical : if two systems having the same configuration, and one of the system has less examples to learn and progress than the second system, it is obvious that the result of the second system will be better than the first one. Here, our data-set 1 has only 1.960 examples to learn, whereas data-set 2 has 6.400 examples.

Secondly, the number of classes also plays a direct role regarding the results. It is logical that it is harder to learn to distinguish 51 letters (data-set 1) than 10 (data-set 2). There are more chances to find similarities (and so it is harder to distinguish) among the 51 letters rather than among the 10 letters. To compensate this difference, I think that we should provide at least 5 times more training instances to compensate this weakness.

Finally, I also found something that could have an impact, but it is way less explicit. In data-set 1 we are using upper-cases and lower-cases whereas in data-set 2 we do not have that. Moreover, I think that both are not always easy to distinguish because the symbol is pretty much the same and has similarities. In my opinion, there are some cases where the models have the good letter but have difficulty to know if a letter is the upper-case or the lower-case.

In my opinion, those 3 factors explain well the differences that we obtain regarding the accuracy average for both data-sets. Now, considering the accuracy average for the classi-

fiers, we can see that the best results come from the neural network. Also, in data-set 2, the results between Naive Bayes and Decision Tree are very close, contrary to what we could have thought after looking at data-set1.

I have an explanation to why we have those differences. First, I think that neural network classifier outperforms the 2 other because we have a lot of features. If you look more precisely at how a neural network work, you will see that most of the time they come with a back-propagation mechanism which allows the network to correct the errors made on the weight for each perceptron. The errors are then fixed for each feature and recomputed, and back and forth. So, I would say that this model is very scalable and particularly adapted to huge data-sets with tons of possible classes.

Secondly, Naive Bayes methodology assumes the conditional independence. This means that in situations where we have completely different letters without any dependence with each other, it works well, such as in data-set 2. But when it comes to situations with similar and linked letters, like in data-set 1 with upper-cases and lower-cases, it struggles. Also, each binary feature for a letter has some kind of dependence with each other. For example, for the letter H, we know that the binary features on the left-side will be linked in a certain form (a single line from bottom to top). Naive Bayes does not assume this link and may lead to a lack of information.

And finally, the least efficient classifier overall is the decision tree classifier. I think this kind of algorithm work very well in situation where we do not have a lot of classes to distinguish. The amount of information for each class is then very high, and a decision tree can be a good and efficient solution. However, in data-sets with a lot of classes, it becomes harder to make distinction between 2 classes, and the quantity of information brought by each class is not very good. So, when the decision tree find a class that could be related to a particular symbol, it takes it without even checking at all other possible classes. This classifier is not very good when data-sets come along with tons of classes to distinguish.

4 Conclusion and future work

Finally, here we are! After this project, I think that even though neural network outperforms the 2 other classifiers, it ask more computation time. Naive Bayes seems to be a good alternative to neural networks when the features are not related with each other. However, decision tree classifier is only a possible solution when we have few classes to distinguish. Overall, those results confirm what I thought in the first place.

If I had to pursue this project, I would like to further investigate the scalability of those techniques, using huge data-sets, in order to see which algorithms are particularly efficient in term of performances. Also, changing the characters, such as taking Chinese letters would be something truly interesting.

A solution that would be efficient to improve the accuracy of the algorithms, would be to also take into account the context. By context, I mean that generally, characters are not written alone, so a good point to investigate further would be to analyze the letters before and after. With Natural Language Processing (which is the next project!) we should be able to increase our chances of guessing the correct letter.

References

[1] Wikipedia. Arbre de decision (apprentissage), 2018.

[2] StackExchange. Giny impurity, 2017.

[3] Scikit-learn. Decision tree splitter parameter, 2018.