Assignment 1 : Selected Topics in Distributed Multimedia Systems

Baptiste CALLARD

August 12, 2022

Table des matières

I Introduction

The aim of this assignment will be to apply the theoretical knowledge learned during the course to concrete examples. This first assignment will use machine and deep learning methods to perform supervised multi-class classification. We will use the very popular MNIST dataset which corresponds to a set of handwritten numbers.

II Presentation of results

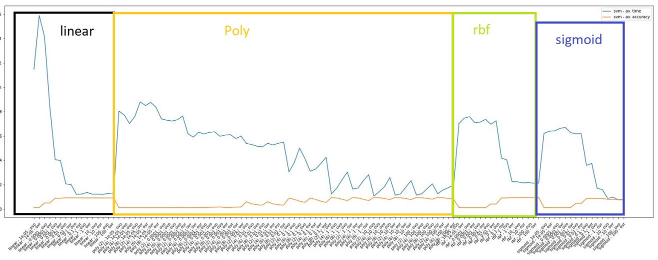
I made the choice to do my assignment on Python. As suggested in the assignment, I used a part of the total game to reduce the overall execution time, 5000 training and 500 testing [1]. I made the choice because to test the different combinations of hyper-parameters one has to train the model many times (the notebook run more thant 10hours). Furthermore, to have more robust and stable estimates I used a cross validation with 5 splits and took the average accuracy and computation time. [2]. I will use accuracy as statistic for seeing if the model as well learned as it is the default output of **cross_val_score**. We can also use f1 or recall but if we want to optimize by looking at all these parameters at the same time it can be really hard. At the end I will compare all these values for top models.

II.1 (a) Analyze different parameter choices for SVM

The SVM will consider an image as a vector with only one dimension. Thus, a first step is to reshape the vector. Then it is important to rescale the data. For example, for a black and white image each pixel can take values between 0 and 255. So, to rescale the data an obvious method is to divide each pixel by 255. I have chosen to study the following hyper-parameters :

parameters	value
kernel	linear, poly (deg = 2, 4, 6, 8), rbf, sigmoid
C	$10^{-5}, 10^{-4}, ..., 10^{2}$
list decision function shape	ovo, ovr

(a) comparison of all parameters



(b) accuracy and time vs parameters (the legend corresponds to kernel - C - decision function shape)

FIGURE 1

We can see on the picture below that for each of the kernels, the model manages to learn correctly and in a reasonable time the data for a certain combination of hyperparameters. An interesting remark is that when the accuracy increases the average time decreases. This is quite understandable because SVM is an optimisation problem. So if the model has trouble finding a solution it may mean that it is complex. This would be a case of overfitting or simply that the model has not found a viable solution because SVM is very sensitive to outliers.

II.2 Influence decision function shape

We will now compare the influence of the different parameters. We will stat by looking at decision function shape

1. I check in the notebook and the classes are balanced see appendix 1
2. Also to find the hyperparameters, there are gridseach methods but I chose not to use them because I also wanted to know the execution time. And to see the process by coding myself for learning purpose

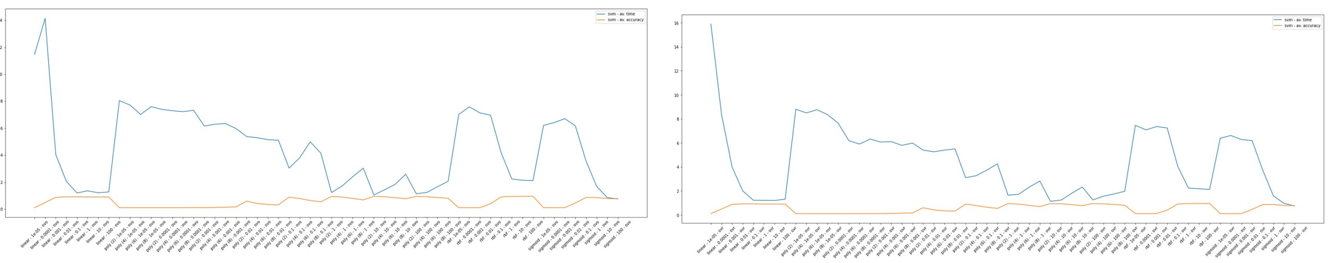


Figure 2 – comparison ovr and ovo (the legend corresponds to kernel - C - decision function shape)

We can see that the choice between ovo and ovr is not crucial. Indeed, there is no special improvement for accuracy or time. So we will continue with ovr which is the default setting of sklearn in the version I am using.

II.3 Influence of the degree of the polynomial

We will filter to study only polynomial kernel.

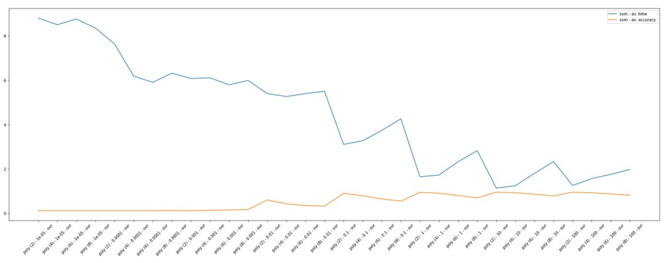


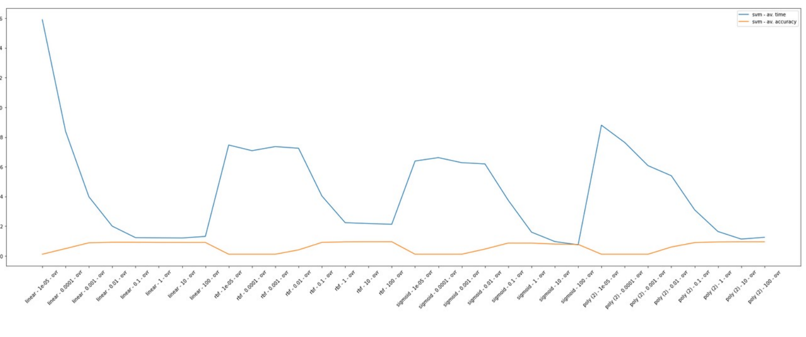
Figure 3 – comparison polynomial degrees (the legend corresponds to kernel - C - decision function shape)

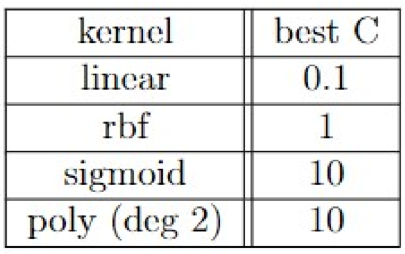
It can be noticed that when the degree of the polynomial increases (with the other parameters fixed) then the time increases and the performance decreases. Thus, for the following we will use the degree 2 for the study which presents the best results in time and accuracy.

II.4 Influence of the parameter C

Here, we selecting only ovr and polynomial degree 2. The parameter C is the most important parameter in the SVM. It is used to define whether we want to have hard or soft margin. It is a regularisation parameter. A too large C can lead to relaxation and over fitting while a too small C constrains the solution and can lead to inconsistent results or underfitting (if a data is not separable then it can't work). Thus, one should go for the smallest C for equivalent performance.
Here we can see that all kernels except the linear one need a C getter than 0.01 to get interesting results in time and accuracy. The linear model manages to have good results from C = 0.0001 but the time remains high.

Thus, considering both time and accuracy with the same importance. The best models per kernel are :





kernel	best C
linear	0.1
rbf	1
sigmoid	10
poly (deg 2)	10

Figure 4 – Comparison performances with respect to C and table with best C

Finally, if a choice has to be made. I think I would go for the linear model with 0.01 or 0.1 depending on the extent to which the execution time is limiting. Here, we only work with a part of the data. One could imagine that with the 60,000 images there is more noise and therefore more difficulty in finding a linear separation. In this case, I would opt for poly, sigmoid or rbf. I think the advantage of SVM is the rapidity because for sure if we want high accuracy it is better to use CNN.

II.5 (b) Run a PCA-prepossessing step and analyze the performance

Here, rather than using a vector of dimension 28*28 = 784. We can choose to decrease this dimension by projecting the data into a smaller dimensional space and keeping most of the information. To do this, we can use PCA which allows us to represent the data in the space of eigenvectors. Then, by keeping only the values with the largest eigenvalues, it is possible to summarise the data set by overwriting the noise. Of course, we risk losing information if we keep 90% of the variance but the predictions will be faster and not necessarily less good. We will study this in this section. We can know the number of axis for an amount of variance by looking at the figue 5.1.

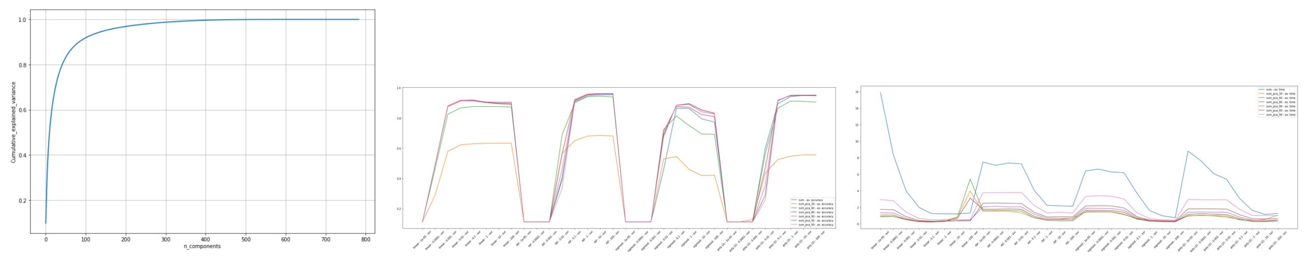


FIGURE 5 – Cumulative variance explaination and accuracy and time comparison for different amount of variance

This graph is very interesting. As you would expect, the variance and the results are in some way proportional. On the other hand, we can see that from 80% of variance onwards we obtain better results than without PCA despite the fact that we lose information. The data are in a space where only the relevant information has been kept and the noise has been removed. Thus, they are more easily separable. This is very interesting. If we keep little variance for example 30% then we don't have enough information and the predictions are again less good. Then, for the time, except for linear - 100 - ovr, the execution time is longer if we keep more variance. Thus, the longest will always be the model without PCA. In view of the results, we should choose, whatever our parameters, to do a pre-processing with a PCA 80% because it is the best compromise between accuracy and time.

II.6 (c) Replace SVM with Random Forests or with a simple Neural Network and analyze different parameter choices

We will now look at the Random Forests algorithm. This is an algorithm that is very powerful and relatively simple to implement. Indeed, it can be used with all types of data, missing values, all kind of inputs (discret, continious) and is not sensitive to outliers. We will still try to find the best hyper-parameters.

II.7 Random Forest

For Random forest, I tried to optimize the following hyper-parameters :

parameters	value
number estimators	10, 50, 100, 200, 500, 1000
max depth	10, 5, 100, 200, 500, 1000, None
max features	sqrt, log

For the maximum number of features I wanted to compare if there was a difference between sqrt and log.

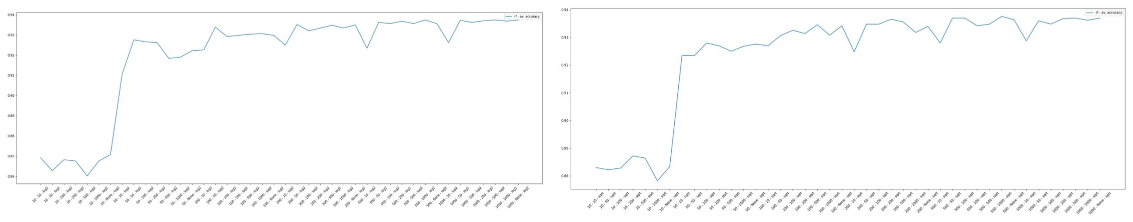


FIGURE 6 – comparison between sqrt and log for maximum features (the legend corresponds to n_esti - max_depth - max_features)

There is no big difference. So, I would continue the study only with sqrt (which is the default on sklearn). Then, we can also compare for example the number of estimators. On the same figure it is obvious from the figures that the more estimators you have, the better the predictions are.

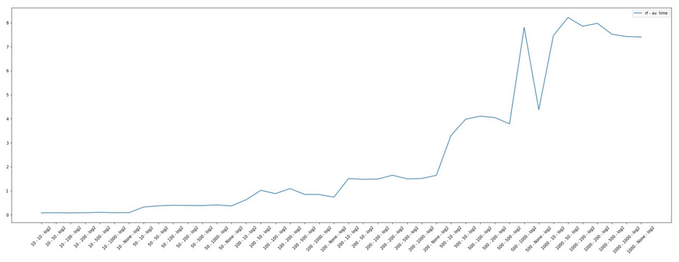
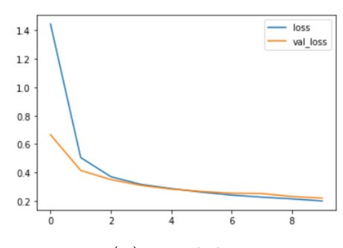


FIGURE 7 – time comparaison (the lgend corresponds to n_esti - max_depth - max_features)

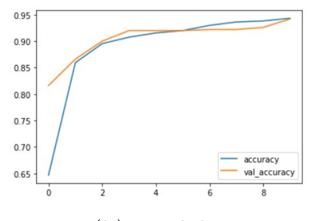
On the other hand, even if the performance increases with the number of estimators the time increases accordingly. Finally, we can compare max depth. Limiting the depth gives better results. The worst results are when we do not limit the depth and the best when we limit to 10. This highlights the problem of trees with over fitting. Indeed, on the training set, if we don't set a limit then the model will put one instance per leaf and will have 100% accuracy but we will be overfitting. Taking accuracy into account the best models are max_depth = 10 and max_features $\in$ {200, 500, 1000}. In terms of time, my choice would be max_features $\in$ {200, 500} because 1000 is really longer. To compre with SVM, random forests is clearly longer than SVM.

II.8 CNN

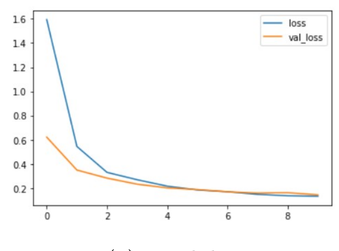
I have implemented 2 relatively simple CNNs. They consist of 54,410 and 34,826 weights. Using the same data, we get very good results for the accuracy. The first model has only 1 layer with 32 convolution filters and the next one has a second layer with 64 filters. I put in appendix the summary of the models.



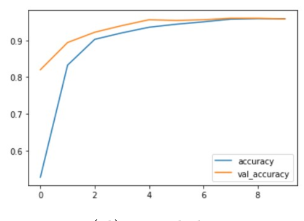
(a) model 1



(b) model 1



(c) model 2



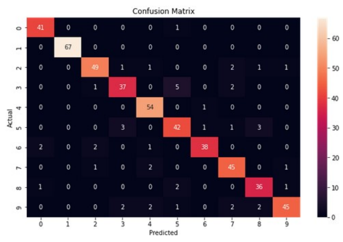
(d) model 2

FIGURE 8 – CNN and performances

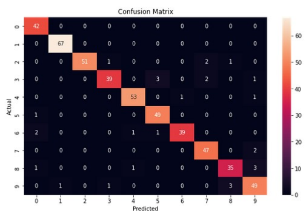
The models are slower than the SVM for training. Afterwards, if we consider the time for cross validation and not the average time, we have closer execution times. The CNN is very simple to implement and has better performance. The second because the performances are better in accuracy and close in time. One can imagine increasing the number of weights by adding layers for example if one increases the training set.

III Comparison

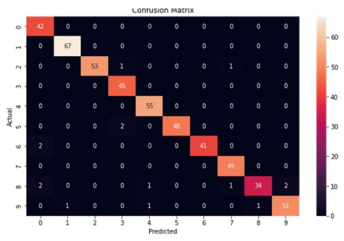
Now we will compare best models for each part :



(a) svm with pca 0.8 and kernel = linear, ovr and C = 0.1

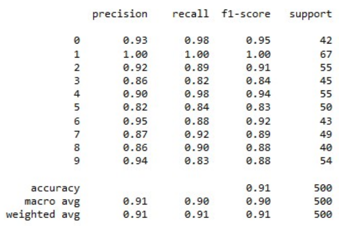


(b) rf with max_dept = 10, max_esti = 200 and max feature sqrt



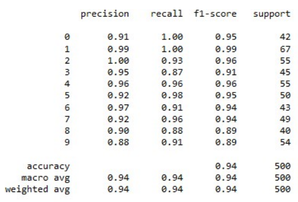
(c) CNN 2

Figure 9 – CNN and performances



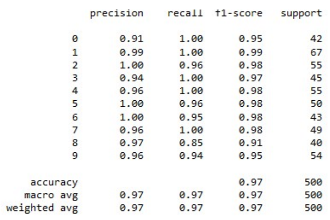
```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95        42
           1       1.00      1.00      1.00        67
           2       0.92      0.89      0.91        55
           3       0.86      0.82      0.84        45
           4       0.90      0.98      0.94        55
           5       0.82      0.84      0.83        50
           6       0.95      0.88      0.92        43
           7       0.87      0.92      0.89        49
           8       0.86      0.90      0.88        40
           9       0.94      0.83      0.88        54

    accuracy                           0.91       500
   macro avg       0.91      0.90      0.90       500
weighted avg       0.91      0.91      0.91       500
```

(a) svm with pca 0.8 and kernel = linear, ovr and C = 0.1



```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        42
           1       0.99      1.00      0.99        67
           2       1.00      0.93      0.96        55
           3       0.95      0.87      0.91        45
           4       0.96      0.96      0.96        55
           5       0.92      0.98      0.95        50
           6       0.97      0.91      0.94        43
           7       0.92      0.96      0.94        49
           8       0.90      0.88      0.89        40
           9       0.88      0.91      0.89        54

    accuracy                           0.94       500
   macro avg       0.94      0.94      0.94       500
weighted avg       0.94      0.94      0.94       500
```

(b) rf with max_dept = 10, max_esti = 200 and max feature sqrt



```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        42
           1       0.99      1.00      0.99        67
           2       1.00      0.96      0.98        55
           3       0.94      1.00      0.97        45
           4       0.96      1.00      0.98        55
           5       1.00      0.96      0.98        50
           6       1.00      0.95      0.98        43
           7       0.96      1.00      0.98        49
           8       0.97      0.85      0.91        40
           9       0.96      0.94      0.95        54

    accuracy                           0.97       500
   macro avg       0.97      0.97      0.97       500
weighted avg       0.97      0.97      0.97       500
```

(c) CNN 2

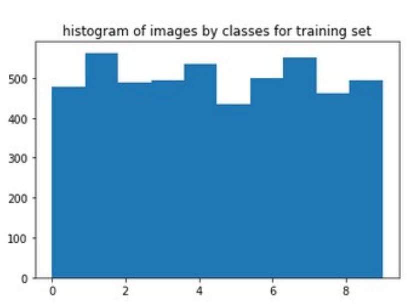
Among these models, the CNN model clearly gives better results if we do not care about time. Precision, recall and f1 are more often better than the other models, followed by random forests and finally SVM. However, if we want a fast model, we have to go for SVM with PCA as it is much faster than RF and CNN. Moreover, the results are also good.

IV Conclusion

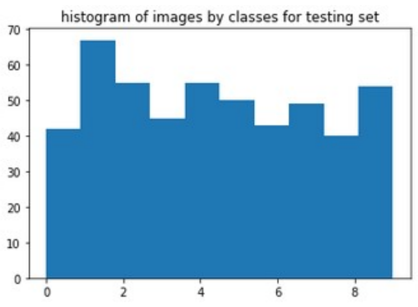
In conclusion, the SVM is a model that allows to obtain very good performances by combining it with the PCA without long execution time. On the other hand, it takes more time to optimise the SVM because it is a very sensitive model to outliers, missing values etc. The most important parameters are the kernel and the value of the relaxation parameter C. In comparison, the random forest has the advantage of being very simple to train and easily adaptable. It supports raw data. One just has to be careful about overfitting by limiting the depth of the tree. Accuracy performance can easily be improved by increasing the number of trees if there are no time constraints. It is necessary to find the trade of. Finally, the CNN is the most popular method today for image classification because of its performance. In our case, for such a "simple" dataset, it is very easy to implement and has accuracy performances that surpass machine learning methods, the only drawback may be time. If we don't care about time the good choice is surely CNN.

V appendix

V.1 histogramme instances in classes

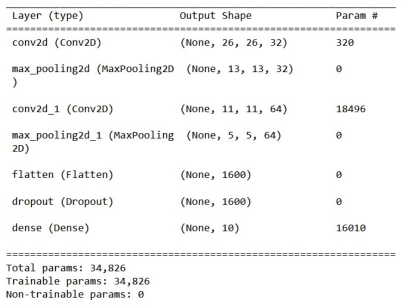


(a) train



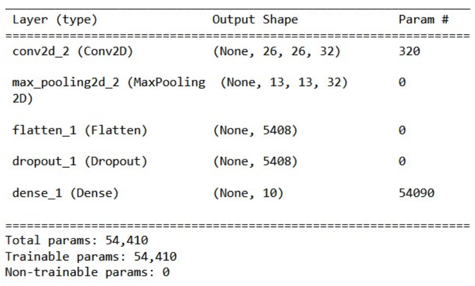
(b) test

V.2 Model used as CNN



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dropout (Dropout)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
```

(a) CNN 1



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 flatten_1 (Flatten)         (None, 5408)              0

 dropout_1 (Dropout)         (None, 5408)              0

 dense_1 (Dense)             (None, 10)                54090

=================================================================
Total params: 54,410
Trainable params: 54,410
Non-trainable params: 0
```

(b) CNN 2