Digit classification with the Kernel Perceptron

Marco Del Treppo

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Abstract

In this experimental project it's implemented from scratch Kernel Perceptron for multiclass classification using the Polynomial Kernel. The Algorithm is then used for predicting the label of handwritten digits of the MNIST dataset. The performance are evaluated using the zero-one loss and the multiclass prediction are made as $\arg\max_i g_i(x)$ where g_i correspond to the binary classifier for the class i. The implementation and the analysis are performed using Python.

Introduction

The following Introduction is mainly copied from the notes of Professor Nicolò Cesa-Bianchi

The Perceptron algorthm is one of the earliest machine learnings algorithms. It's an on-line linear classification algorithms. It learns a decision function based on a hyperplane by processing training point one at the time. As a linear classifier it learns a predictor of the form $h : \mathbb{R}^d \to \{-1, 1\}$ defined by

$$h(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w}^\top \boldsymbol{x})$$

Knowing that the update rule of the Perceptron is $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_t \boldsymbol{x}_t$ where $\boldsymbol{w}_0 = \boldsymbol{0}$ then the linear classifier learned through the Percpetron are of the form

$$h(\boldsymbol{x}) = \text{sgn}\left(\sum_{s \in S} y_s \boldsymbol{x}_s^\top \boldsymbol{x}\right)$$

where S is the set of indices s of the training examples $(\boldsymbol{x}_s, y_s)$ on which the Perceptron made an update. At this point is possible to write the pseudo code of the Perceptron Algorithm based only on the set S of points where was made an update. The reason of this variation form the standard code was necessary in order to implement the Kernel Trick.

Algorithm 1 Perceptron

Let S be the empty set.
For all $t = 1, 2...$

1. Get next example $(\boldsymbol{x}_t, y_t)$
2. Compute $\hat{y}_t = \text{sgn}\left(\sum_{s \in S} y_s \boldsymbol{x}_s^\top \boldsymbol{x}_t\right)$
3. If $\hat{y}_t \neq y_t$ add t to S

Linear predictors may potentially suffer from a large approximation error because they are described by a number of coefficients which can not be larger than the number of features. A technique to reduce this bias is feature expansion which adds new parameters by constructing features through nonlinear combinations of the base features. Fixing a feature expansion maps $\phi : \mathbb{R}^d \to \mathbb{R}^N$ and consider the classifier $h : \mathbb{R}^d \to \{-1, 1\}$ defined by

$$h(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w}^\top \phi(\boldsymbol{x}))$$

If we run the Perceptron in the space $\mathbb{R}^N$, the linear classifier h becomes

$$h(\boldsymbol{x}) = \text{sgn}\left(\sum_{s \in S} y_s \phi(\boldsymbol{x}_s)^\top \phi(\boldsymbol{x})\right)$$

Where S is again the set of indices s of the training examples $(\boldsymbol{x}_s, y_s)$ on which the Perceptron made an update

The Polynomial Kernel, corresponding to the polynomial feature-expansion map is

$$K(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{z}) = (1 + x^\top z)^d$$

So given a Kernel K we can then write the linear classifier generated by the Perceptron as

$$h_K(\boldsymbol{x}) = \text{sgn}\left(\sum_{s \in S} y_s K(\boldsymbol{x}_s, \boldsymbol{x})\right)$$

Below is given the pseudo code of the Kernel Perceptron as it was implemented in Python. Specifically only the set S of training examples was stored with the time index t.

Algorithm 2 Kernel Perceptron

Let S be the empty set.
For all $t = 1, 2...$

1. Get next example $(\boldsymbol{x}_t, y_t)$
2. Compute $\hat{y}_t = \text{sgn}\left(\sum_{s \in S} y_s K(\boldsymbol{x}_s, \boldsymbol{x}_t)\right)$
3. If $\hat{y}_t \neq y_t$ add t to S

Data Preparation

The MNIST Dataset downloaded from Kaggle comes in two different files. The train set is composed by 60000 rows and 785 columns. The first columns is the label of the picture that goes from 0 to 9 corresponding to 10 different handwritten digits. The other 784 columns are the features of the row and corresponds to the value, in gray scale, of a pixel ranging from 0 to 255. Each pixel value is divided by 255 in order to have a scale ranging from 0-1. This pre-computation normalization is done in order to speed up the latter phases of the analysis. The train data are also randomly permuted before the analysis.

Binary Classification

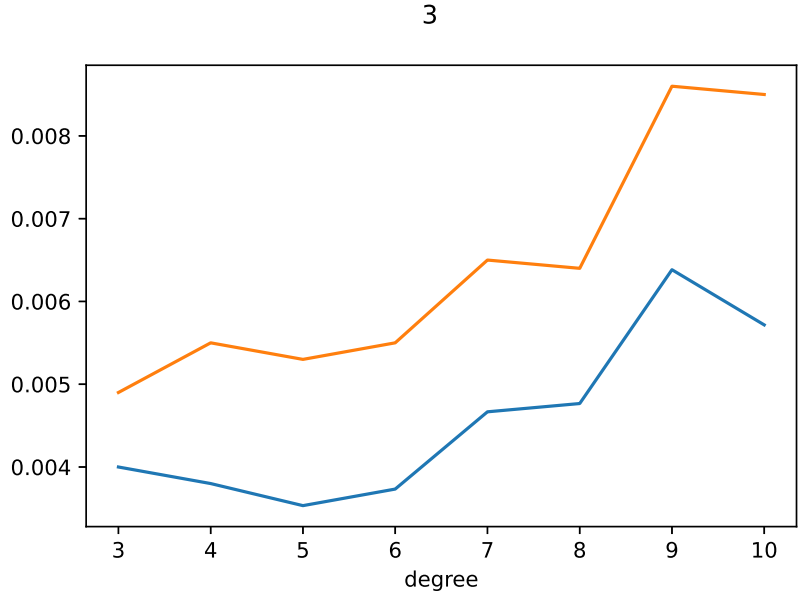
Average Predictor

Defining S_t as the active set of indices of examples where was made a mistake at time t, S_0 by definition is the empty set. Also by definition of the Kernel Perceptron Alghoritm $S_1 = \{1\}$. At time $t = 2$ is possible that $S_2 = \{1, 2\}$ or $S_2 = \{1\}$ if the second training example was correctly classified. At time $t = n$ the active set is S_n and defined as $w_s = \frac{n+1-s}{n}$ the average importance of the misclassified training example $(\boldsymbol{x}_s, y_s)$ the classifier becomes

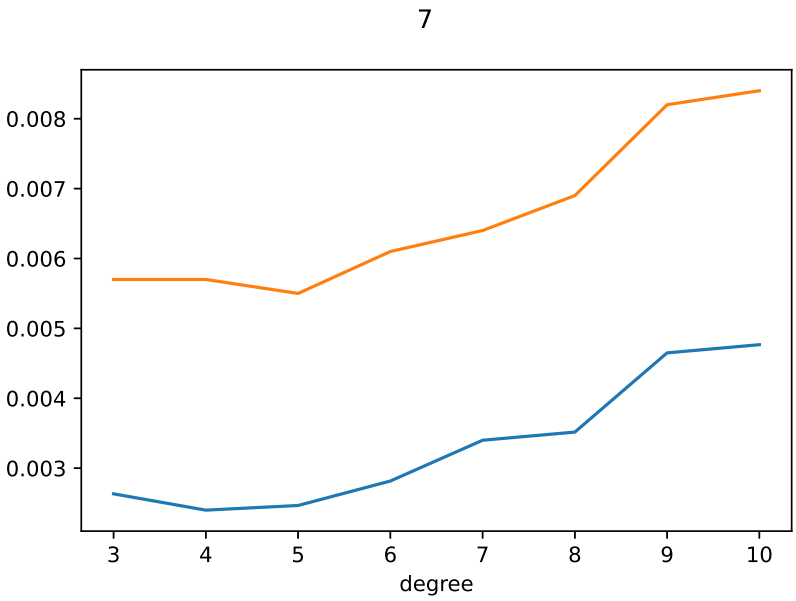
$$h_w(\boldsymbol{x}) = \text{sgn}\left(\sum_{s \in S} y_s w_s K(\boldsymbol{x}_s w_s, \boldsymbol{x})\right)$$

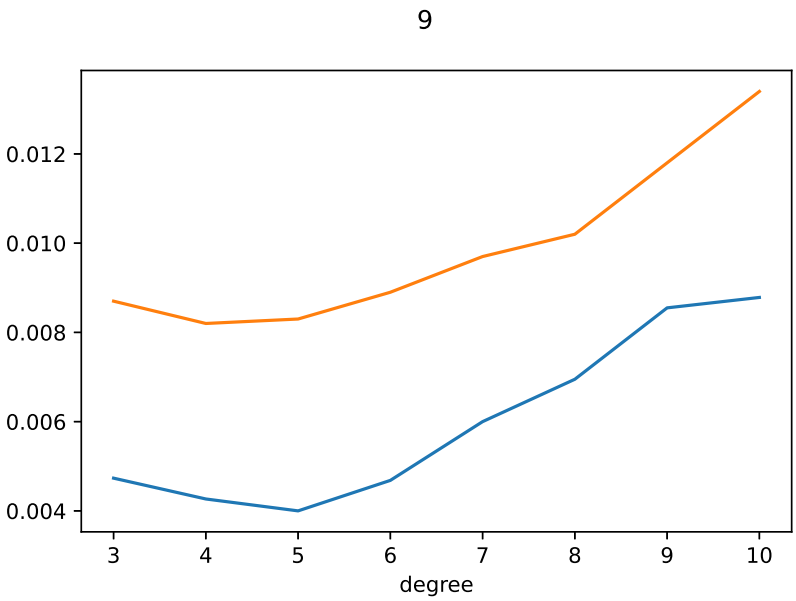
By definition $w_1 = 1$ and $w_t < 1$ for all $t \neq 1$. This give a bigger importance to the set of early misclassified examples. Heuristically the advantages of this predictor is backed by the fact that the major reduction in the training error in the binary classification was observable in the early time stamp. Also an advantages could be a reduction in the probability of overfitting for a large enough number of epochs.

Down below are shown the result with a fixed number of epochs T = 4 for different binary classification. In orange is the test error and in blue the training error. The evaluation is performed for different value of the degree of the polynomial kernel ranging from 3 to 10. For the binary classification task on number 3 the best performances are obtained with low degree. The best accuracy on the test set is near 99.5%.



The same binary classification task is performed for all the class. Down below are plotted the results for number 7 and number 9. Are the accuracy are over 99% for the binary classification. And the best region for the degree of the polynomial kernel is between 3 and 5.





Best Training Predictor

At time $t = T$ we have $S_1, S_2, ...S_T$ active sets of indices and we know that $S_1 \subseteq S_2 ... S_{T-1} \subseteq S_T$. Defining the set of classifier as

$$h_1(\boldsymbol{x}) = \text{sgn}\left(\sum_{s \in S_1} y_s K(\boldsymbol{x}_s, \boldsymbol{x})\right)$$

$$h_2(\boldsymbol{x}) = \text{sgn}\left(\sum_{s \in S_2} y_s K(\boldsymbol{x}_s, \boldsymbol{x})\right)$$

$$\vdots$$

$$h_T(\boldsymbol{x}) = \text{sgn}\left(\sum_{s \in S_T} y_s K(\boldsymbol{x}_s, \boldsymbol{x})\right)$$

Given $\Omega = \{h_1, h_2, ...h_T\}$ the best predictor is the one minimizing the training error formally

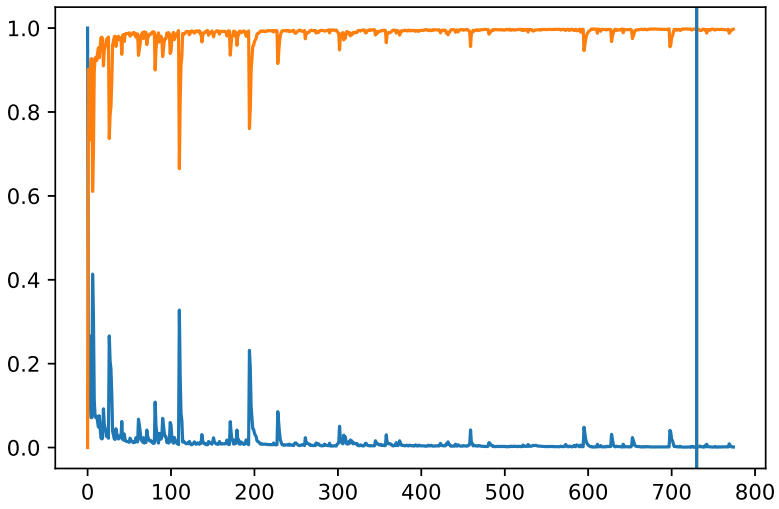
$$h_b(\boldsymbol{x}) = \arg\min_{h \in \Omega} \frac{1}{T} \sum_{t=1}^{T} L(h(x_t), y_t)$$

The plot below shows the results on the training and test set using all the possible different predictor (aka number of misclassified point). In this case the number of epochs is fixed at T = 3 and the degree of the polynomial kernel at D = 3. In orange the test accuracy and in blue the training error.

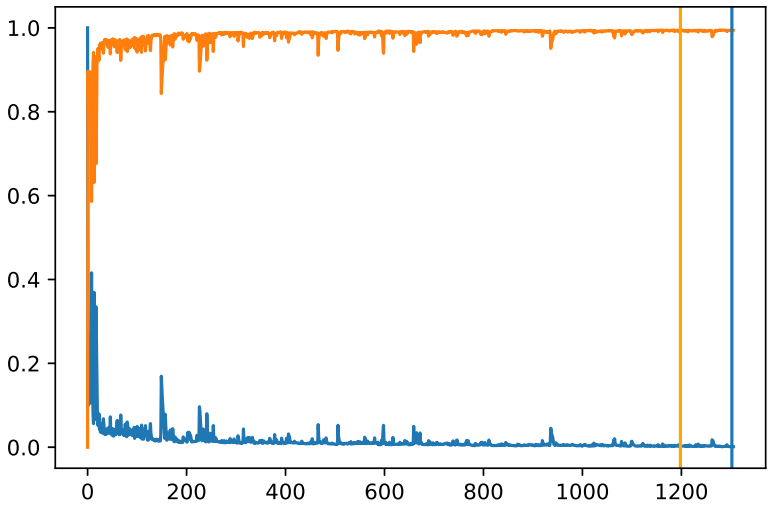
Is also plotted **vertically** in orange the predictor with maximal test accuracy and in blue the predictor with minimal training error. Is interesting to notice that only for number 0 the max test accuracy predictor and the minimal training error predictor are the same. The y-axis shows the cardinality of the active set S.

Also using the Best Training Predictor the accuracy on the test set is generally over 99%. This approach has the disadvantage of being more computational and time consuming given the needs of trying all the different predictor on the training set.
Interestingly enough the two plot seems to be identically. The graph reported below are just for the binary classification for number 0 and 2 but the analysis was performed for all the classes not highlighting any particular differences.

0


1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
600
700
800

2

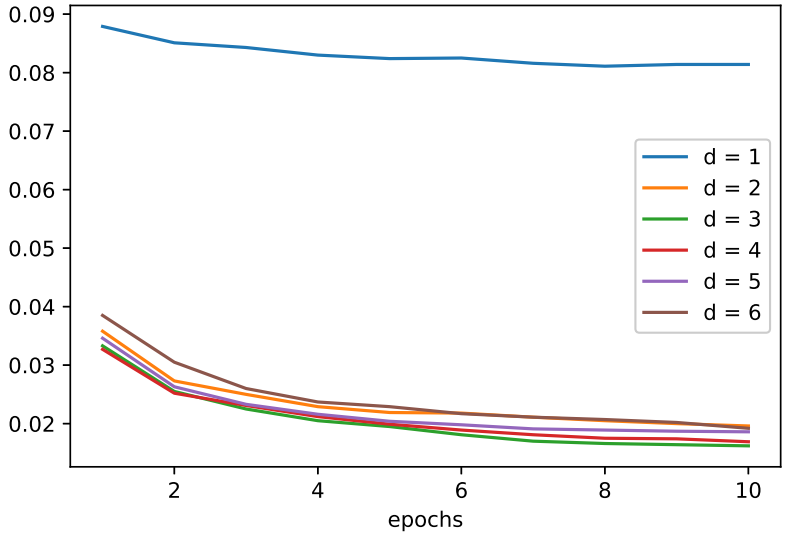

1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
1200

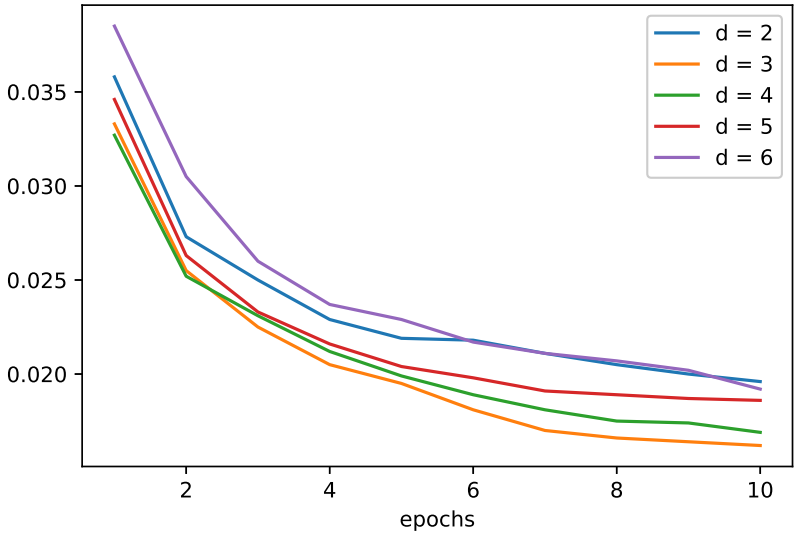
Multiclass Prediction

Average Predictor

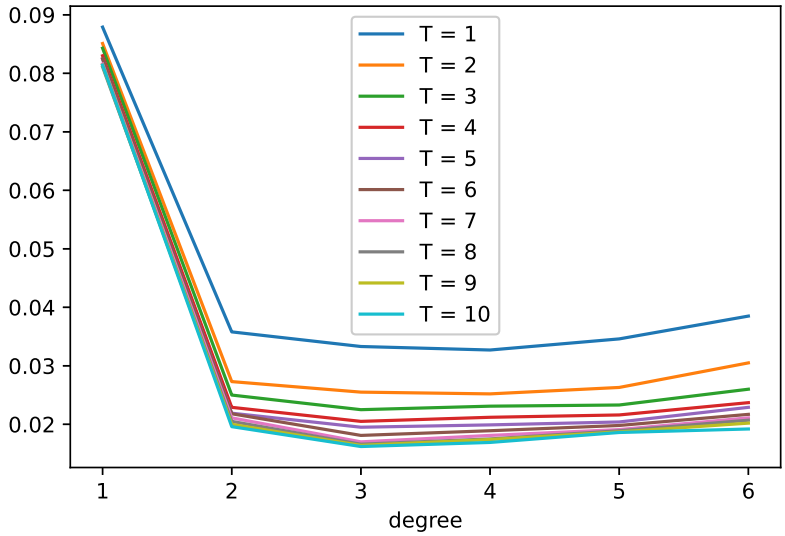
Given the absence of any particular advantages in using the "Best Training Predictor" the analysis in the multiclass scenario was performed using the Average Predictor. The performance are evaluated using the zero-one loss and the multiclass prediction are made as $\arg\max_i g_i(x)$ where g_i correspond to the binary classifier for the class i.

The analysis was first performed for a number of epochs ranging from 1 to 10 and the degree of the polynomial kernel ranging from 1 to 6. The plot below shows the Test Error for different number of epochs and for different value of the degree of the polynomial kernel. Only d = 1 shows way worst performance on the Test set. Is evident a decreasing trend over the number of epochs but from T = 6 to T = 10 there is not a huge improvement.

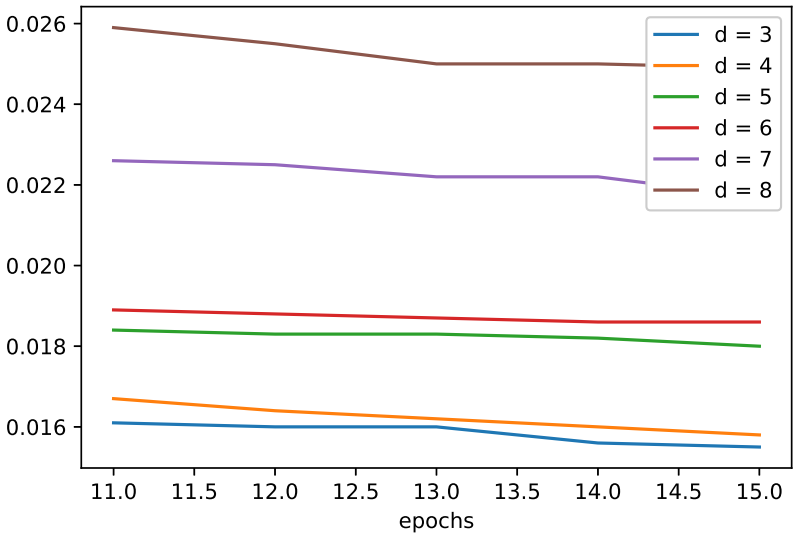


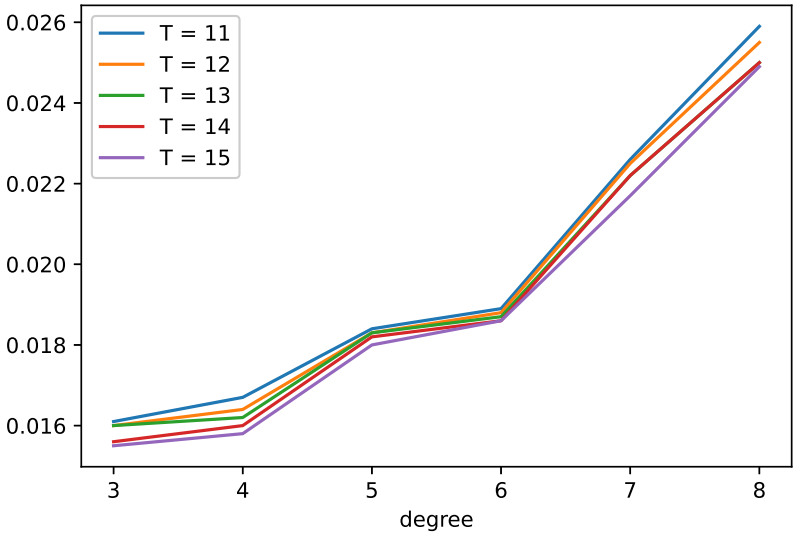


Excluding D = 1 the results plotted below highlights D = 3 as the winner in terms of accuracy on the test set for almost all the different number of epochs. In the multiclass scenario the best performance are obtained with D = 3 and T = 10 and the test error is 1.5%



Given the decreasing trend with respect to the number of epochs, despite being almost stable from epoch 6 to 10, I wanted to perform a second analysis using as a range of degree of the polynomial kernel from 3 to 8 and as number of epochs from 11 to 15. As shown in the graph below this second experiment did not bring improvements. Also in this case D = 3 allows better performance for each T.





Conclusion

The results for the binary classification and for the multiclass classification were in my opinion astonishing. Even with a single run over the training set given D = 3 the accuracy was high. Certainly the dataset contributed to these types of results. I would like to point out in the conclusions that the biggest difficulties arose in the computation time. The analysis was done from my laptop and the implementation in Python was optimized, to the best of my current ability, to be as fast as possible.

The choice of not using a mistake counter (choice proposed in the Wikipedia page of Kernel Perceptron) was effective in this case but certainly problematic if the active set should contain too many times the same point. Faster variants of Kernel Perceptron would have been more efficient. In particular, the possibility of limiting the maximum size of the active set to contain the maximum computation time for a very large number of epochs caught my curiosity.

I also want to make explicit my doubts about my implementation of the Average Predictor that perhaps it would be better to define it as Average Active Set in the form that I have implemented. The better performances in comparison to other variations that I have tried have pushed me to continue with this approach.