Multi-Layer-Perceptron/Support-Vector-Machine Python Implementations to solve the MNIST Classification Problem

Bernard Gütermann, Vincent Robert - Proceedings of the Pattern Classification and Machine Learning Course, EPFL - 18.12.2013.

In this report we discuss the results of the Machine Learning Project on Pattern Classification, more precisely using both a Multi-Layer Perceptron and Sequential Minimal Optimization to solve a sample of two binary classification problems on the MNIST dataset. We implement, evaluate, discuss and compare both methods results in their respective sections of this report

1 Introduction

Pattern classification and machine learning provide a large set of powerful tools and techniques. The sheer number and the diversity of these tools can make selecting the ideal technique for a given problem quite difficult. We could simply decide we want to use the best techniques available to make sure our system is optimal, but with complexity comes time consumption. Therefore, when using an elaborate tool to solve a classification problem, it is generally useful to test it and make sure it actually performs the way we expect it to, thus leveraging on the impending costs. This report compares the performance of Multi-Layer Perceptron technique, with stochastic gradient descent over a logistic error criterion, against Sequential Minimal Optimization support vector machine algorithm. The evaluation is made on subsets of the widely available MNIST dataset.

2 Methods

2.1 MLP : Binary Multi-Layer Perceptron

Model and Parameters

There are two binary classification problems that we want to solve : Separation of patterns representing '3"s from the ones depicting '4', and separation of '4's from '9's. Both datasets are subsets of the MNIST dataset. We perform stochastic gradient descent on a Multi-Layer Perceptron with two layers (1 hidden layer), for a variable number of hidden units. The datasets are normalized between [0,1] in a preprocessing phase before the actual processing can take place. We implement the algorithm in Python 2.7.

Forwardpropagation The vectorized forward propagation equation corresponding to the given 2-layers binary MLP is as follows :

$$a^{(2)}(\mathbf{x}_i) = \left(diag\left(\mathbf{w}^{(2)}\right)\mathbf{a}_o^{(1)}\right)^T \sigma\left(\mathbf{a}_e^{(1)}\right) + \mathbf{b}^{(2)}$$

where $\mathbf{a}_o^{(1)} = \mathbf{w}_o^{(1)}\phi(\mathbf{x}_i) + \mathbf{b}_o^{(1)}$ and $\mathbf{a}_e^{(1)} = \mathbf{w}_e^{(1)}\phi(\mathbf{x}_i) + \mathbf{b}_e^{(1)}$ are the odd, respectively even components of the activators of the first layer for input pattern $\mathbf{x}_i$, with $\mathbf{a}_o^{(1)}, \mathbf{a}_e^{(1)} \in \mathbb{R}^{h_1 \times 1}$ and $\mathbf{w}^{(2)}, \mathbf{w}_o^{(1)}, \mathbf{w}_e^{(1)} \in \mathbb{R}^{h_1 \times 1}$. $\sigma(.) = tanh(.)$ is the transfer function. Note that we deliberately chose to split the matrices $\mathbf{w}^{(1)}$ and $\mathbf{a}^{(1)}$ into even and odd sub-matrices upstream from the computations, as it shows to be a more elegant solution for the backpropagation phase.

As a side note, we point out that $diag\left(\mathbf{w}^{(2)}\right)\mathbf{a}_o^{(1)}$ corresponds to the point-wise multiplication of $\mathbf{a}_o^{(1)}$ with $\mathbf{w}^{(2)}$, which is computationally very fast. From now on, W

we slightly abuse notation and write $\mathbf{w}^{(2)}.*\mathbf{a}_o^{(1)}$ to denote this vector-space closed operation.

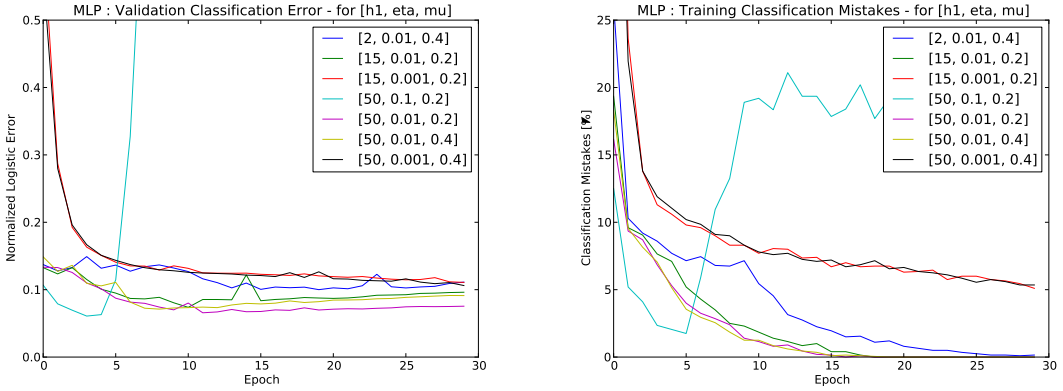

Figure 1: MLP validation classification error (left) and classification mistakes (right) samples, for the '3'-vs-'5' classification problem

Backpropagation We start from the Logistic Error equation: $E_{log} = \frac{1}{n}\sum_{i=1}^{n} E_i(\mathbf{x}_i)$, where $E_i(\mathbf{x}_i) = log\left(1 + e^{-t_i a^{(2)}(\mathbf{x}_i)}\right)$. Furthermore, we denote $\tilde{t_i} = \frac{t_i+1}{2}$ as the normalized label for input pattern $\mathbf{x}_i$. The vectorized backpropagation equations that we implemented for the binary MLP are as follows :

$$
\begin{aligned}
r^{(2)} = \sigma\left(a^{(2)} - \tilde{t_i}\right) \quad &\Rightarrow \quad \nabla_{\mathbf{w}^{(2)}} E_i = r^{(2)}\mathbf{z}_i \ , \in \mathbb{R}^{h_1 \times 1} \\
&\Rightarrow \quad \nabla_{b^{(2)}} E_i = r^{(2)} \ , \in \mathbb{R} \\
\mathbf{r}_o^{(1)} = r^{(2)}\mathbf{w}^{(2)} .* \sigma\left(\mathbf{a}_e^{(1)}\right) \quad &\Rightarrow \quad \nabla_{\mathbf{w}_o^{(1)}} E_i = \mathbf{r}_o^{(1)} (\mathbf{x}_i)^T \ , \in \mathbb{R}^{h_1 \times d} \\
&\Rightarrow \quad \nabla_{\mathbf{b}_o^{(1)}} E_i = \mathbf{r}_o^{(1)} \ , \in \mathbb{R}^{h_1 \times 1} \\
\mathbf{r}_e^{(1)} = r^{(2)}\mathbf{w}^{(2)} .* \mathbf{a}_o^{(1)} .* \sigma\left(\mathbf{a}_e^{(1)}\right) .* \sigma\left(-\mathbf{a}_e^{(1)}\right) \quad &\Rightarrow \quad \nabla_{\mathbf{w}_e^{(1)}} E_i = \mathbf{r}_e^{(1)} (\mathbf{x}_i)^T \ , \in \mathbb{R}^{h_1 \times d} \\
&\Rightarrow \quad \nabla_{\mathbf{b}_e^{(1)}} E_i = \mathbf{r}_e^{(1)} \ , \in \mathbb{R}^{h_1 \times 1}
\end{aligned}
$$

where the residuals are computed using the derivative of the transfer function given in terms of itself, i.e : $\sigma'(x) = \sigma(x) * \sigma(-x)$.

Model selection

Our MLP has one native parameter, h_1, and relies on stochastic gradient descent to converge to the optimal solution. This adds two more parameters, *learning rate* η and *momentum term* μ, to the set of parameters from which we will draw our sample of optimal solutions.

Unfortunately, the impact of each parameter on the overall performance of the MLP is very difficult to predict. Indeed, the standard model selection method of computing logistic errors on the Cartesian product of a reasonable range of parameters values was overall lacking in consistency because of the random nature of the computations (we reshuffle the training and validation sets at each epoch).

We circumvent this issue by performing the experiments on a wide range of parameters values in the dataset, and subsequently perform an analysis on *both* the Logistic Error during validation at each epoch, and the *mistake rate*. The latter one, which is the proportion of misclassified patterns during training, provides us with a quantitative aspect in the analysis of the classification results. This criterion allows to narrow down the iterative research, as it provides an *efficiency* factor in the decision factor. Indeed, we observe that the training phase is usually "over" when all training patterns have been classified correctly. Our experiments have shown us that further iterating and reducing error *may not* necessarily give better

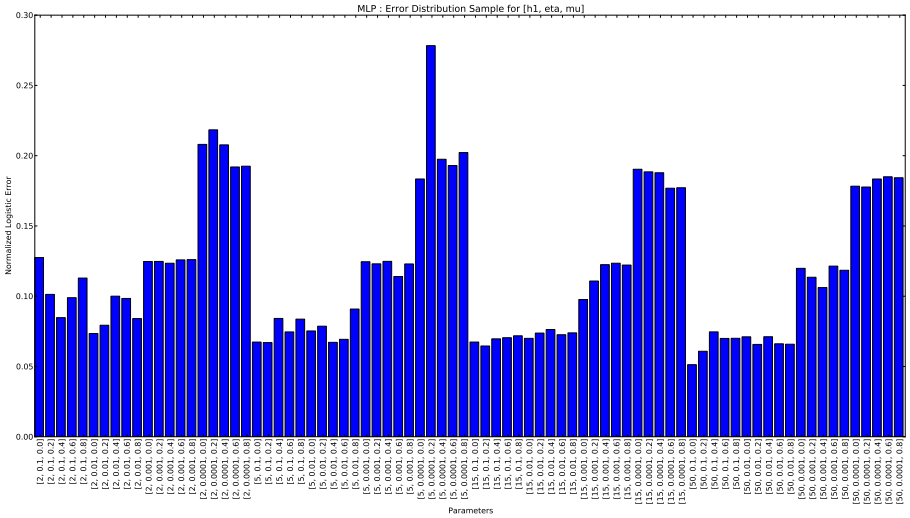

Table 1: MLP Testing phase result samples. Shown are the minimum logistic errors on the validation set, on a run of 30 epochs, '3'-vs-'5' classification problem

results on the validation dataset, and are therefore usually not worth the additional cost. Figure 1 illustrates this phenomenon, during the preliminary testing phase. Table 1 shows a wide range of results that were gathered during this early training phase, for a wide subset of parameter space. Note that we implemented early stopping but did not use it in this phase in order to gather uniform results.

We perform model selection on both sets, but only show the intermediary results for the '3'-vs-'5' classification problem.

2.2 SMO : Sequential Minimal Optimization

Model and Parameters

We perform soft margin *support vector machine* (SVM) binary classification on the "4vs9" subset of MNIST dataset. We use the *sequential minimal optimization* (SMO) algorithm to solve the dual problem of the soft margin:

$$\min_{\boldsymbol{\alpha}}\{\boldsymbol{\Phi}(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j t_i t_j K_{ij} - \sum_{i=1}^{n}\alpha_i\}$$

Subject to: $\alpha_i \in [0, C]$, $i = 1, ..., n$, $\sum_{i=1}^{n} t_i\alpha_i = 0$, and where $\boldsymbol{K} = [K_{ij}] = [K(\boldsymbol{x}_i, \boldsymbol{x}_j)]$ is the kernel matrix. We solve this optimization problem over the training dataset $D = \{(\boldsymbol{x}_i, t_i)|i = 1, ..., n\}$, where $t_i \in \{-1, 1\}$ are the labels associated with each pattern. We use a Gaussian kernel which is defined as $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{\tau}{2}||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}$. The SMO algorithm minimize iteratively $\boldsymbol{\Phi}$ with respect to α_i and α_j for some $j \neq i$. Once the convergence criterion is met, the discriminant is $sgn(y(\boldsymbol{x}))$, where $y(\boldsymbol{x}) = \sum_{i=1}^{n}\alpha_i t_i K(\boldsymbol{x}, \boldsymbol{x}_i) - b$. Furthermore, we have $b = \frac{1}{|S|}\sum_{i\in S}(t_i - \tilde{y}_i)$, with $\tilde{y}_i = \sum_j \alpha_j t_j K(\boldsymbol{x}_j, \boldsymbol{x}_i)$ and $S = \{i|0 < \alpha_i < C\}$.

There are two free parameters: C and τ. In order to determine the best pair, we use 10-fold cross-validation (see next section).

Before starting the training, the dataset is preprocessed in the same fashion as in the MLP. Furthermore, in order to save computation time, the training set is split into $k = 10$ folds up-front, and saved to disk.

C/tau	0.001	0.01	0.1	1	10
0.1	52.1	27.2	7.0	24.1	31.2
1	52.1	300	10.6	293.8	31.1
10	300.0	295.1	7.7	292.5	294.4
100	300.9	275.6	7.7	292.5	294.4

Table 2: Exponential step size exhaustive search across parameter space. The results in the table is the average number of misclassified element over the 10-fold validation $\hat{R}(D^{(i)}$, where $|D^{(i)}| = 600$.

C/tau	0.05	0.07	0.09	0.1	0.11	0.12	0.15
1	250.9	47.2	7.6	10.6	18.2	31.7	157.2
3	60.9	13.1	6.5	7.7	10.6	19.9	83.9
5	43.7	14.4	6.6	7.7	10.6	19.9	83.9
7	53.0	14.4	6.6	7.7	10.6	19.9	83.9
9	53.0	14.4	6.6	7.7	10.6	19.9	83.9
10	53.0	14.4	6.6	7.7	10.6	19.9	83.9

Table 3: Linear step size exhaustive search across parameter space. The results in the table is the average number of misclassified element over the 10-fold validation $\hat{R}(D^{(i)}$, where $|D^{(i)}| = 600$.

Model selection: 10-fold cross validation

This method is very used in the area of machine learning. It provides a good way to evaluate the parameters of a given model. The idea is to split the training set D into k subsets of equal size where $k = 10$. Formally, we have $D = \bigcup_{i=1}^{10} D^{(i)}$, thus each subset $D^{(i)} = D \backslash \bigcup_{j \neq i} D^{(j)}$. This also implies that $|D^{(i)}| = \frac{1}{10}|D|$. In order to evaluate how good the 2 parameters are, we compute an estimation of the risk $\hat{R}(D)$. In order to do so, we perform the following: Use the first $D^{(1)}..D^{(k-1)}$ subsets to make the training. The last subset $D^{(k)}$ is used to validate the results, i.e. compute the risk $\hat{R}_i(D^{(k)})$. We iteratively perform this task 10 times, while choosing each time a different validation set $D^{(k)}$ in a round robin fashion over all $D^{(i)}$. At the end of the 10th iteration, we compute the overall risk $\hat{R}(D) = \frac{1}{10}\sum_{i=1}^{10} \hat{R}_i(D^{(i)})$. This estimator is the used to compare each different parameter setup.

To find the best parameters, we proceeded in two phases, similarly to the MLP. The first is to perform an exhaustive search across the parameter space, this time using orders of magnitude steps. In a second phase, we narrow down the area of research to the neighborhood of the optimal values found in the first phase to seek with more precision the optimal parameters. sample tables of the search results are shown in Table 2 and Table 3. The best pairs for both phases batches are highlighted in green.

3 Results and Discussion

3.1 Binary MLP

We narrowed down our parameter space by recursively running the test described in Table 1 on an increasingly narrower parameter space. Unfortunately, due to processing time constraints, we stopped when we could deem the precision as "sufficient" enough with respect to the problem. This was achieved after running the testing phase on two magnitude orders on each parameters. Figure 2 shows the logistic error computed with the optimal parameters for the binary MLP for the 3vs5 classification problem, while Figure 3 shows the results for the 4vs9 classification problem.

The both the classification Errors, respectively mistakes, quickly converge to 0. It is interesting to notice that the classification error, respectively mistakes, never reach 0, even in our optimal case. This further confirms the intuition expressed in the previous section that whenever the classifier reach the 0 mistake mark on the training pattern, the classification error for the validation set become unpredictable. We used this observation in the implementation of our early stopping algorithm, where we consider the mistakes as an important criterion for convergence evaluation.

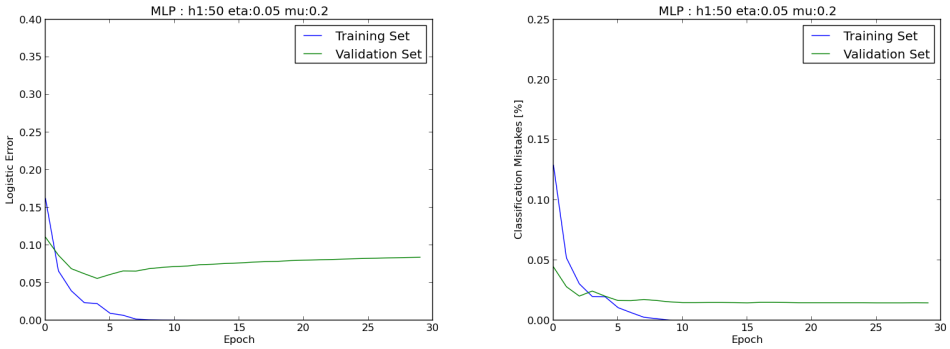

Figure 2: MLP Optimal parameters classification error (left) and classification mistakes (right), for the '3-vs-5' classification problem

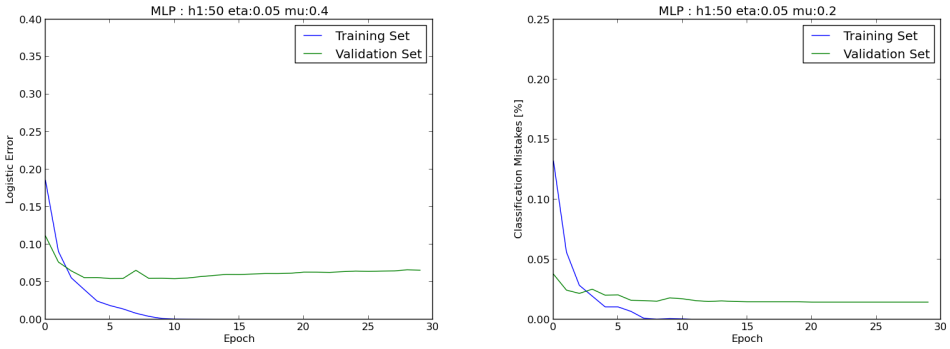

Figure 3: 'MLP Optimal parameters classification error (left) and classification mistakes (right), for the '4-vs-9' classification problem

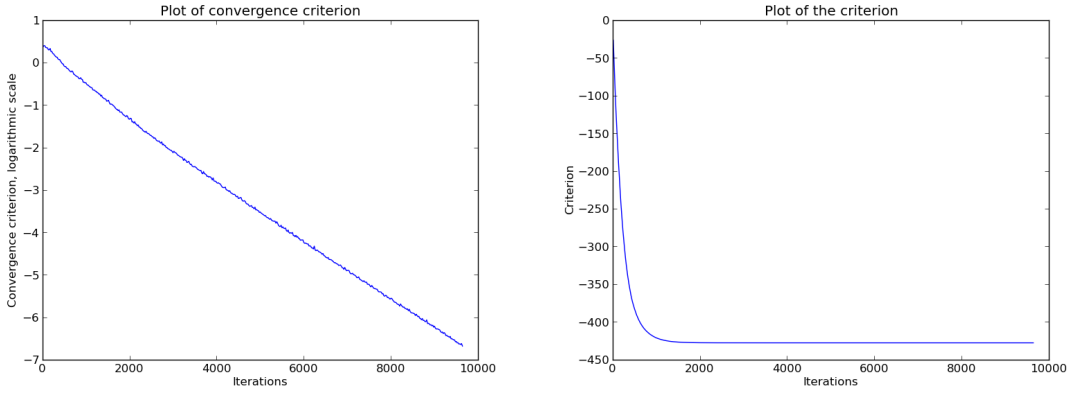

Figure 4: Plots of SVM criterion (right) and convergence criterion (left)

3.2 SMO

Figure 4 plots the algorithm running trace of the two main criteria, over the optimal parameters. These results are compiled over cross-validation on the dataset, as explained above. The figure on the left represents the convergence criterion based on KKT conditions, computed as $log_{10}(b_{up} - b_{low})$. The figure on the right illustrate the SVM error criterion $\mathbf{\Phi}$.

At convergence, we have on average (over all cross-validations) 6.5 misclassified patterns, giving us a mistake rate of **1.0833%**

3.3 Classification Results

The results for the classification rate on the test set are shown in Figure 5. We can observe that on average the SMO algorithm provides better classification results over the '4vs9' test dataset. On the binary classification dataset '3vs5' the MLP misclassified on average $\mu = 1.31\%$ of input patterns, with a standard deviation of $\sigma = 0.234\%$. On the '4vs9' classification dataset, the MLP misclassified on average $\mu = 1.36\%$ of input patterns, with a standard deviation of $\sigma = 0.166\%$; whereas the SMO misclassified exactly **0.9041%** of input patterns. The results on the MLP were computed on runs of 20 full iterations with the early stopping implemented as described above. We demonstrate some example of patterns that were misclassified by our implementations in Figure 6

From these observations we can argue that the SMO overall performs both qualitatively and quantitatively better than the MLP for this binary classification problems. The fact that the SMO is deterministic whereas the MLP is not, renders this observation even more pertinent, as in this case we were asked to solve a very precise binary classification problem "4vs9", over a predetermined dataset. In such a setup, it is thus very easy to settle for the SMO for classification. However, it is our belief that the SMO is harder to adapt to more general setups, setups where the problem is wider and more general, without necessarily predetermined datasets. Furthermore, it is our belief that, for the current setup, a MLP with more layers could have given way better classification results, possibly better than the SMO; But at a way greater cost.

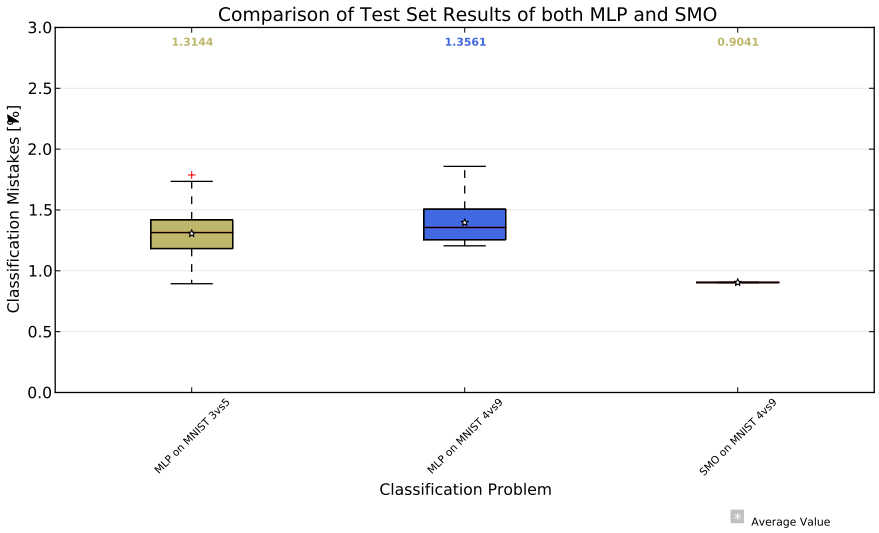

Figure 5: Comparative plot of the test results of each classifier over their respective test set

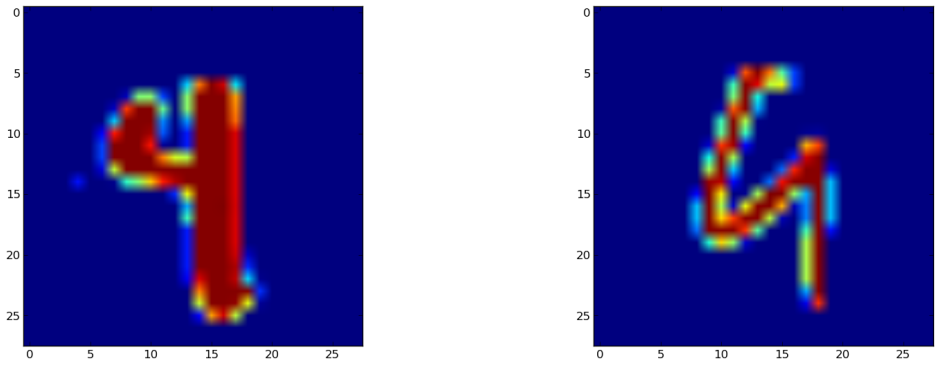

Figure 6: Example of misclassified patterns: A misclassified 4 (left), and a misclassified 9 (right)