Mini project report

Fangyu YE
Master of Communication System
N^O Sciper: 232225

Yilin HU
Master of Communication System
N^O Sciper: 232522

Introduction

The aim of this miniproject is to implement some widely used machine learning methods and apply them to a time-honored machine learning problem: hand-written digits pattern recognition. First we will implement Multi-Layer-Perceptron algorithm to classify two classes on two different binary classification problems, then implement the SMO algorithm for SVM training.

1. Learning Multi-Layer Perceptron

1.1 Split the Data Set & Preprocess the Data

Before we do all other operation related to training ,we first randomize all the order of training part with randperm in Matlab and split the Xtrain part into a training set (2/3 of cases, i.e. 4000) and a validation set (1/3 of cases, i.e. 2000).

Then in MLP we need to preprocess our Data in order to avoid the non-linear sigmoid being easily saturate. The method is working as following:

1) Compute the minimum α_{min} and maximum α_{max} over all the coefficients of all training patterns (*Do not touch the test set).
2) Replace each input pattern x$\in R^{784}$ by $\frac{1}{\alpha_{max}-\alpha_{min}}$(X-$\alpha_{min}$1)

Apply this normalization to all patterns of the dataset, training and test.

1.2 Final setup of the MLP

We use a two-layer MLP, with one hidden and one output layer. Use the gating transfer function $g(a_1, a_2) = a_1\sigma(a_2)$, $\sigma(x) = \frac{1}{1+e^{-x}}$ .The binary classifier is f(x)=sgn($a^{(2)}(x)$) and our code is aimed to minimize the logistic error function. More deeply we choose an optimization algorithm: stochastic gradient descent with a momentum term.

1.2 Model Evaluation

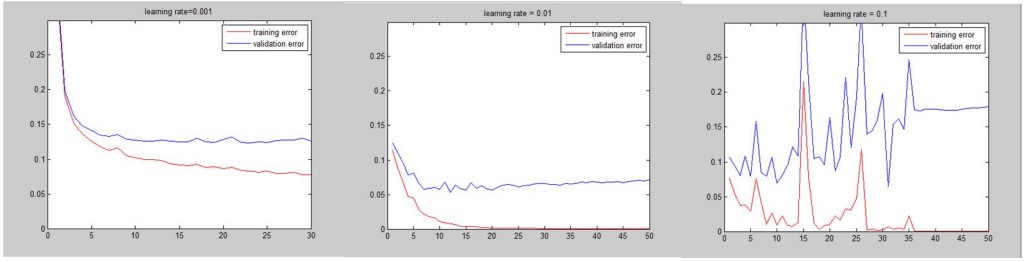
1.2.1 Early Stopping

For the best parameter choices, we found the classification error on the validation set would increase after several epochs (i.e. overfitting happens). So we stop training once the validation error starts increasing.

1.2.2 Model Selection

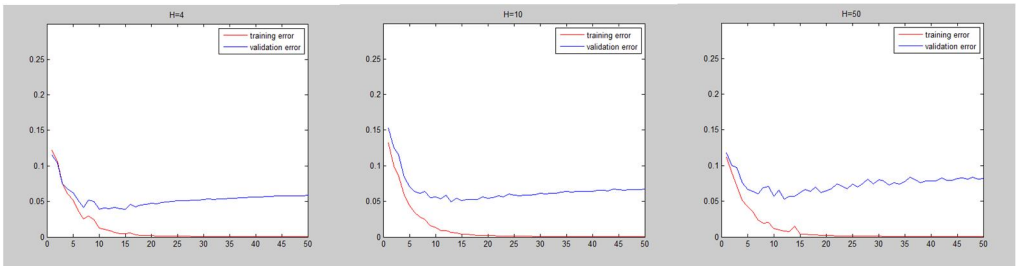
In our network architectures, we always use 1 hidden layer and 1 output layer. But we try to find the best choice of number of hidden units, learning rate and momentum term in the two different binary sub-tasks.

a) Learning rate (H1=10, μ=0.2)



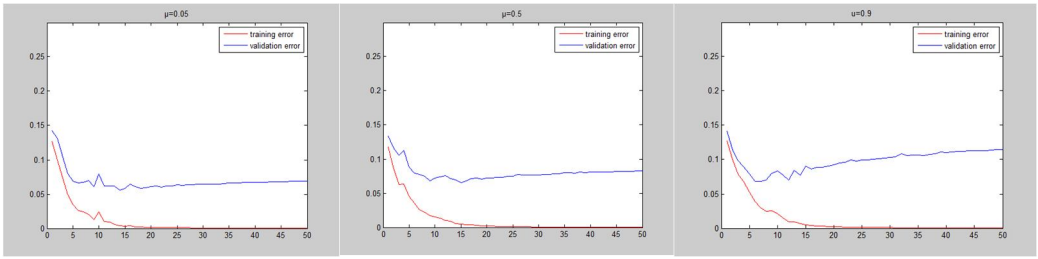
From the figures we can point out that as learning rate increases, the network convergence becomes faster. But not larger learning rate always leads to better performance. In fact, when learning rate=0.1 as shown in the figure, the performance of network starts fluctuation and the update trend becomes irregular. We know the larger learning rate is, the larger update step is, when Δw, Δb are around the target point, if it takes one step wrongly, large learning rate may cause a big step away from the target, thus larger errors may occur during this procedure. So in practice, we should not take a too large learning rate to avoid bursting errors.

b) Hidden units (learning rate=0.01, μ=0.2)



From the figures, we see large size of networks did not significantly improve the convergence speed. In fact, too large layer sizes cause much more complexity of network, which are not only slow for convergence, but also slow for computation. While for small size layers, the computation is fast enough, but due to the simplicity of network, the convergence won't be as fast as expected. Furthermore, in all the figures, we could find the validation error starts increasing after several iterations.

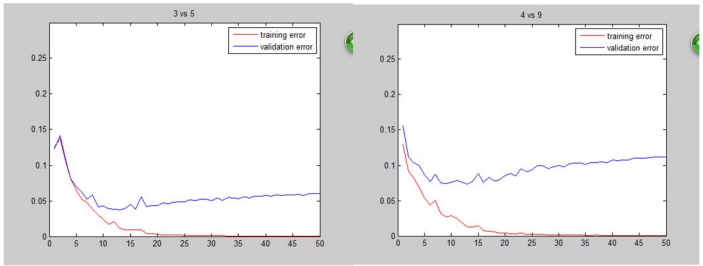
c) Momentum (learning rate=0.01, H=10)



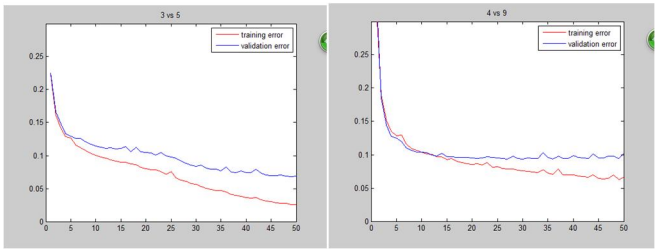
Considering the update formula $\Delta w_k = -\eta(1-\mu)\nabla_{w_k}E + \mu\Delta w_{k-1}$, the function of momentum term is to average over previous steps and smooth out the erratic output direction. Since Δw_k is a convex combination of steepest descent direction and previous update step, the larger μ is, the smoother update direction it will be. In our case, for smaller μ, the update term will converge to target area slower but more accurate and faster to our target (the gap between training error and validation error is much smaller before the validation error stop decreasing); while for lager μ, the term will quickly go to the right direction and converge to the target area but oscillating around the target, so we will have a slower convergence speed with respect to larger μ.

1.2.3 Choice of parameters for the two different binary sub-tasks

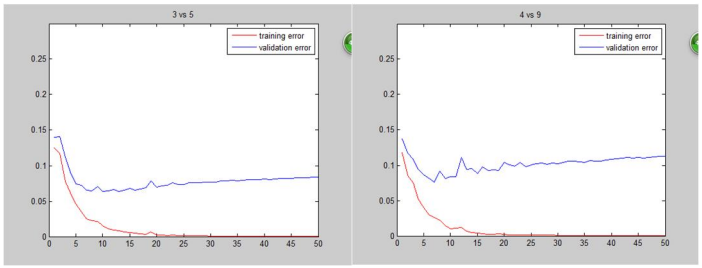
3 vs. 5 and 4 vs. 9 with same parameters: learning rate = 0.01, μ=0.1, number of hidden units = 10



3 vs. 5 and 4 vs. 9 with same parameters: learning rate = 0.001, μ=0.2, number of hidden units = 20



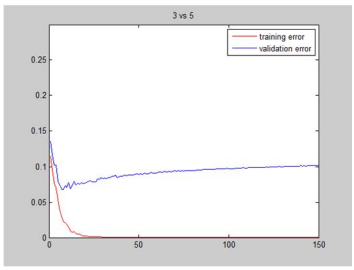
3 vs. 5 and 4 vs. 9 with same parameters: learning rate = 0.01, μ=0.1, number of hidden units = 10



Compared all figures above, with the same parameters our MLP code for the two different binary sub-tasks works in the same time. But there still are some differences between the results of these two sub-tasks. So we can use the same parameters for the two different sub-tasks without making too many errors in one sub-task. However, it is still best to pick different parameters for each task to get the best evaluation.

1.2.4 Overfitting

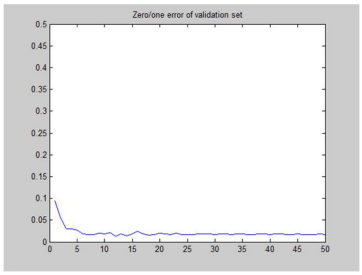
Learning rate =0.01, μ=0.1, number of hidden units = 100



In this example, we can find the validation error starts to increase after several epochs, i.e. overfitting happens.

1.2.5 Zero/One error of validation set

The Zero/One error of validation set is always close to 0.02 and the curve of this error is always like that:



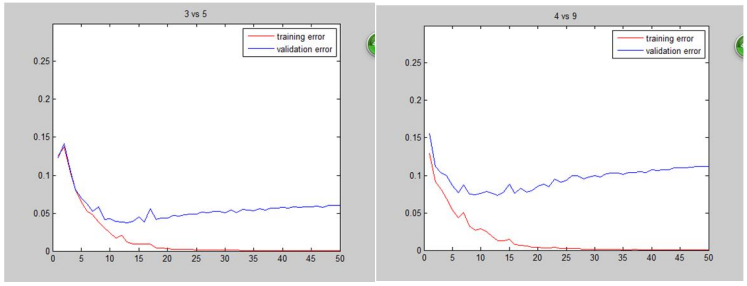
We could find through the figure above that after one epoch, most of the data is classified correctly; the error is always very close to zero.

1.2.6 Results & Discussion

From all the parameter selection analysis above, we figure that the best suitable configuration is as learning rate of 0.01, size of hidden layer equals 10 and we use a momentum of 0.1, the update stops at looping epoch 8 with early stopping.

We use the final classifier to compute the error on the test set and get:

a) Task 3 vs. 5 b) Task 4 vs. 9



By using this configuration, we use the trained classifier on our test set several times and get:

a) with Test logistic error around 0.04 and standard deviation=0.0105

b) with Test logistic error around 0.05 and standard deviation=0.0132

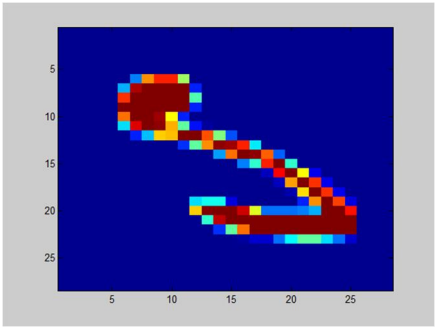
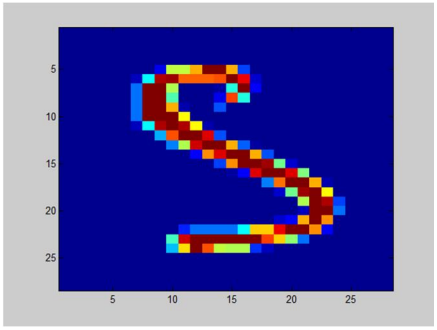
In addition, the test set (zero/one) errors of these two sub-tasks are 0.012 and 0.013.This result shows that our trained classifier has a fairly good and stable performance.

1.2.7 Misclassified Patterns

We choose sub-task 3 vs. 5 to plot examples of patterns which are misclassified:

		Predicted Class	
		Digit 3	Digit 5
Actual Class	Digit 3	1001	9
	Digit 5	21	871

So there are 30 patterns which are misclassified. And we find in these misclassified patterns, there are 5 patterns with $t_i a^{(2)}(x_i) < -2$ (large negative) and 2 patterns with $-0.1 < t_i a^{(2)}(x_i) <= 0$(close to zero). Then we check the value of largest negative $t_i a^{(2)}(x_i)$ is -16.2854, at position 1165 of the test data set. Then we choose another value of $t_i a^{(2)}(x_i)$, which is -0.0141, very close to zero, at position 1381 of the test data set. The images of these two digits are show as following:

1165:  1381: 

The former one is image of 1165th data, which is very clear but hard to distinguish it from digit 5. The other one is the image of 1381th data, which is vague Therefore we could conclude that small, but negative value of $t_i a^{(2)}(x_i)$ indicates very vague image. This kind of images is easily misclassified. Large negative value of $t_i a^{(2)}(x_i)$implies clear image but the object is weird-shaped, which is easily misclassified as well. This conclusion fits our intuitive judgments.

2. Support Vector Machines

2.1 Methods:

2.1.1 Preprocessing

In this part we use cross validation (We will describe later) to optimize the parameters. So it is unnecessary to split the data at the beginning. Just normalize the training data to [0,1].

$$Xtrain = \frac{1}{\max - \min}(x - \min)$$

Computing the kernel function on the whole data:

We consider the Gaussian kernel here. $K(X, X') = e^{-\frac{\tau}{2}\|X - X'\|^2}$. About the method for computation, firstly we compute the $A = \|x - x'\|^2$, let $d = diag(XX^T)$, In Matlab we compute it as: sum(X.*X,2);

Then $A = \|x - x'\|^2 = \mathrm{d1n}^T + 1n * \mathrm{d}^T - XX^T$. Then apply A to the function: $f = e^{-\tau a}\tau$

2.1.2 Theory Foundations

The problem we are going to solve is the dual of SVM problem.

$$\min_{\alpha}\{\Phi(\alpha) = \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j t_i t_j K_{ij} - \sum_{i=1}^{n}\alpha_i\} \quad \text{Subj. to} \quad \alpha_i \in [0, C], i = 1.....n, \sum_{i=1}^{n} t_i\alpha_i = 0;$$

We use the Sequential Minimal Optimization to optimize SVM problem. In order to get the most fitted alpha, we iterate w.r.t i, j to minimize $\Phi(\alpha)$ on $\alpha_i\alpha_j$.

The solution of dual problem is determined by the KKT conditions. Here we transform the KKT conditions to an equivalent problem: $b_{low} \le b_{up}$, the we will be satisfied with an approximately fulfilled conditions: $b_{low} \le b_{up} + 2\tau, (\tau > 0)$. τ is a small positive number(we choose 10^{-8} here). For convenience we omit the derivation of b_{low} and b_{up} here.

2.1.3 Implementation Procedures

The major idea of SMO is using '*while*' to minimization w.r.t $\alpha_i\alpha_j$ every iteration. Do test inside '*while*' for every iteration to see if the approximately convergence condition is fulfilled. If not, update alpha w.r.t $\alpha_i\alpha_j$. If fulfilled, jump out the *while* statement and return the optimized alpha. Then apply the gained alpha to Validation data and compute the validation error. For the validation part, we adapt the M fold Cross validation (M=10). We equally divide the training data into ten sections. Every time we choose one different section for validation and the other nine sections to train alpha. That means we run SMO ten times and return mean of the ten validation errors.

The major free parameters matters in the above computation are C and τ. C is hyperparameter defined in the maximum soft margin perception learning problem. And τ is used to compute the Gaussian kernel function. To choose a relatively optimal C and τ, we run the above algorithm to C and τ ranging between 0.5*[1 2 4 8 16 32 64 128 256 512] and 0.05*[1 2 4 8 16 32 64 128 256 512] respectively (100 times), and choose the pair with minimum validation error. Afterwards run the algorithm on the full training set with parameters found above. After convergence, apply the trained

alpha to test data to get the test error.

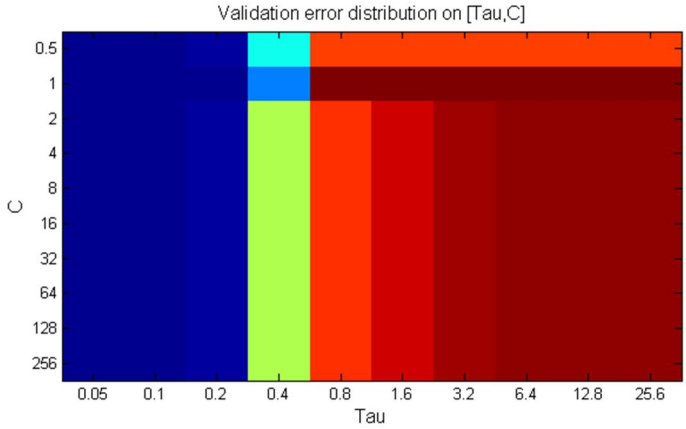
2.2 Results and Discussion

2.2.1 Matrix of 10*10 CV scores.

Do test on C ranging between 0.5*[1 2 4 8 16 32 64 128 256 512] and τ ranging between 0.05*[1 2 4 8 16 32 64 128 256 512]. Compute each group's validation error. The result table is as follow:

	tau=0.05	tau=0.1	tau=0.2	tau=0.4	tau=0.8	tau=1.6	tau=3.2	tau=6.4	tau=12.8	tau=25.6
C=0.5	0.0157	0.0123	0.0240	0.2033	0.4123	0.4123	0.4123	0.4123	0.4123	0.4123
C=1	0.0148	0.0100	0.0160	0.1297	0.5067	0.5123	0.5123	0.5123	0.5123	0.5123
C=2	0.0135	0.0098	0.0210	0.2797	0.4195	0.4708	0.4927	0.4977	0.5000	0.5000
C=4	0.0132	0.0097	0.0210	0.2797	0.4195	0.4708	0.4927	0.4977	0.5000	0.5000
C=8	0.0125	0.0097	0.0210	0.2797	0.4195	0.4708	0.4927	0.4977	0.5000	0.5000
C=16	0.0125	0.0097	0.0210	0.2797	0.4195	0.4708	0.4927	0.4977	0.5000	0.5000
C=32	0.0125	0.0097	0.0210	0.2797	0.4195	0.4708	0.4927	0.4977	0.5000	0.5000
C=64	0.0125	0.0097	0.0210	0.2797	0.4195	0.4708	0.4927	0.4977	0.5000	0.5000
C=128	0.0125	0.0097	0.0210	0.2797	0.4195	0.4708	0.4927	0.4977	0.5000	0.5000
C=256	0.0125	0.0097	0.0210	0.2797	0.4195	0.4708	0.4927	0.4977	0.5000	0.5000

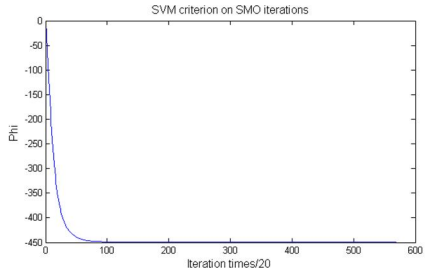
The imagesc plot of the distribution:



From the figure we can see that compared with C the validation error depends more on the value of Tau. If C becomes big enough, it then causes little influence on validation error. As for tau, the major trend is when Tau increases, validation error would increase with tau. From the result table, the winning groups are [τ ,C] $\in$ [0.1,4],[0.1,8]…..[0.1,256]. We use[τ ,C]=[0.1,4] for final evaluation.

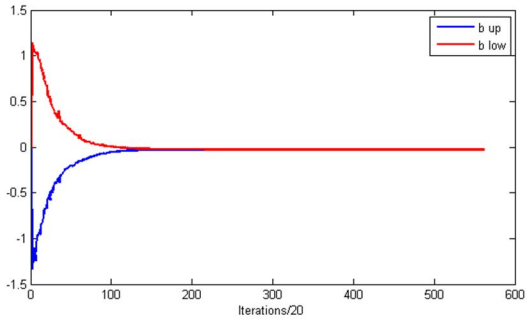
2.2.2 SVM criterion

Run SMO Algorithm on the full training set, with the optimal parameters found by cross validation above. The SVM criterion as function of SMO iterations are plotted as follows:

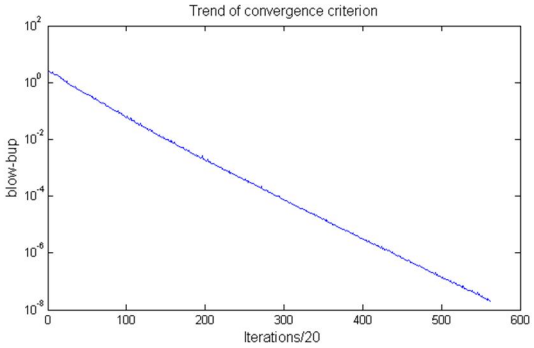


As it shows, the SVM criterion decreases fast in the beginning then it slows down to approach the convergence condition.

The figure of convergence criterion on iterations is as following. We plot the value of b_up and b_low respectively here:



The decrease trend of (b_low-b_up):



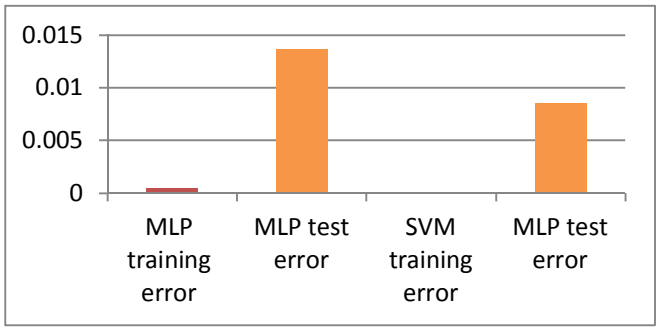
As the iteration times increase, the value of b_up and b_low all approach to zero. The difference between them approaches zero exponentially at the same time.

2.2.3 Final test results of SVM

Run the final classier after SMO convergence, the training error is **zero** and test error is **0.008538422903064**.

3. Comparing SVM and MLP

The training and the test set (zero/one) error of MLP classifier is 0.0005 and 0.0137 respectly. Compare the performance of final MLP classifier versus the final SVM classifier and we get a bar plot of test and training error of each algorithm:



So we find SVM has better performance on our task, as it produce less training error and test error. But we need to pay attention to the computational complexity of SVM which takes more computation time than MLP.