Image Classification with SVMs and MLPs

William Murray

October 2023

Introduction

This report presents and discusses results from 4 different experiments which test the performance of Soft Margin Support Vector Machines (SMSVM) and Multi Layer Perceptron (MLP). These experiments will make use of the `sklearn` library. From this library the functions `SVC()` and `MLPClassifier()` will be used as the implementations for SMSVM and MLP respectively.

Data Pre-processing

This set of experiment uses the Fashion MNIST data set. Unless otherwise specified the data is preprocessed in the following way:

- From the 60000 training examples 3000 are selected with label 7 and 3000 with label 5
- Flip the labels of the 6000 selected examples with probability of 0.2

Experiment 1

Experiment 1 tunes the regularization parameter C for a SMSVM. The performance of the model is tested setting the regularization parameter $C = 10^i$ and ranging i over 20 evenly spaced values from $[10^{-3}, 10^2]$. In terms of programming $C \in$ `[10**i for i in np.linspace(-3,2, 20)]`.

Figure 1 plots the 5-fold cross validation error of the model as the value of C increases over this range:

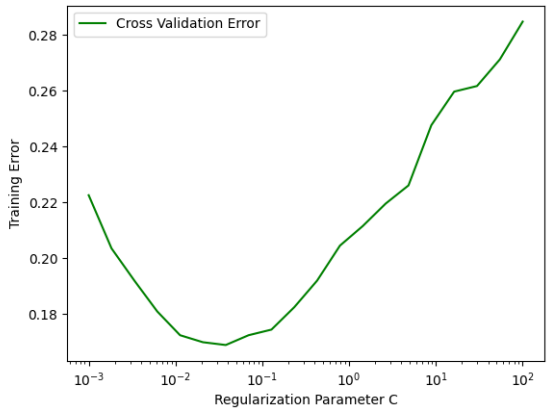

Figure 1: Line Plot of C vs training error (log_{10} scale)

From Figure 1 we can see that training error is minimized when C is close to 0.1. The results from Figure 1 suggests that the optimal choice of C might be found by looking at 10 evenly spaced values 10^i where i ranges from -1.2 to -0.8. In terms of programming $C \in$ `[10**i for i in np.linspace(-1.5,-0.5, samples)]` .

Figure 2 exhibits plots the value of C over the range specified above against the 5 -fold cross validation error.

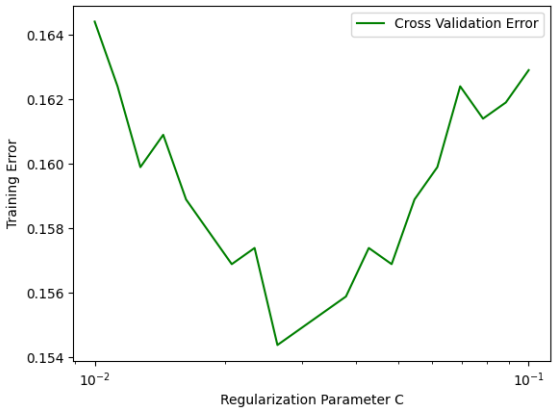

Figure 2: Line plot of C vs training error (log_{10} scale) on narrowed range of C values

The minimum 5-fold cross validation error is achieved at $C = 0.026366$.

Finally we will plot the training and test error as C varies. For a particular value of C, a SMSVM will be fitted on the training data and then the error of the model is calculated for the training data and test data. Note here that k-fold cross validation is **not** being used. For this experiment $C \in$ `[10**i for i in np.linspace(-3,3, 40)]`.

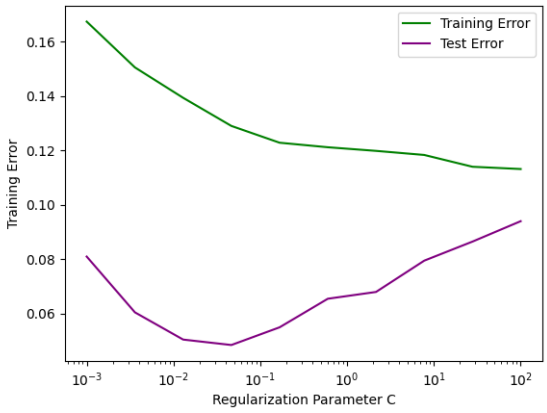

Figure 3: Line plot of C vs training error (log_{10} scale) on narrowed range of C values

Firstly, notice that the minimal test error in Figure 3 occurs at a value of C which is close to 0.026366, the value which minimizes the error in Figure 2. The figure above exhibits a test error which is significantly lower than the training error. Based on the low test error achieved around $C = 10^{-1}$ it is clear that the SVM was able to separate the data in a way that generalizes to new examples. Despite this the noise added to the training data makes it inherently harder to correctly classify thus it scores a much higher error.

Consider $C = 0.026366$, the value which minimized the 5-fold cross validation error. This is a (relatively) small value which implies that the noisy training data is best fitted to the SVM when the noisy examples on the wrong side of the hyperplane are given a small penalty. This can bee seen in Figure 3 since when C increases, the hyperplane becomes over fitted to the noise in the training data which increases the test error.

Experiment 2

This experiment tunes the "bandwidth parameter" γ for a SVM with a gaussian kernel.

The x axis plots the bandwidth parameter (γ). In this experiement $\gamma = 10^i$, for 10 evenly spaced values of $i \in [-3, 0]$. For each γ we consider 15 values for the regularization parameter C. In this case $C = 10^j$ where j takes on 15 evenly spaced values in $[-2, 1]$. For a particular γ_i, let C_{γ_i} be the C value which achieves the minimum cross validation error among the 15 choices for C.

This methodology finds an optimal value of γ and its corresponding C_γ. Figure 4 plots $(\gamma_1, \gamma_2, ..., \gamma_{10})$ on the x axis and $(C_{\gamma_1}, C_{\gamma_2}, ..., C_{\gamma_{10}})$ on the y axis.

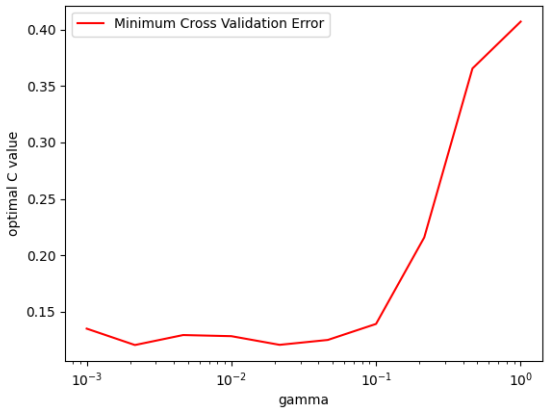

Figure 4: Gamma vs Minimum Error

From the same experiment we obtained the value of γ along with the optimal choice of the regularization parameter C_γ. Figure 5 plots γ against C_γ

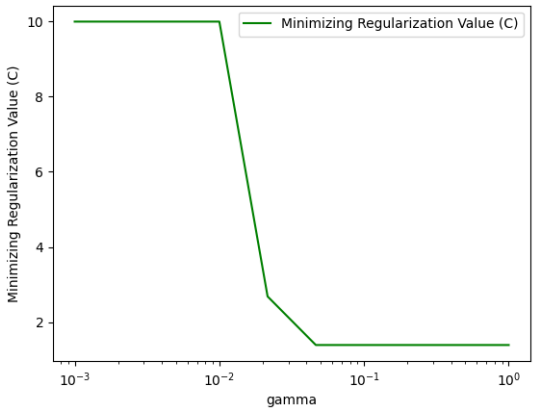

Figure 5: Gamma vs Minimum Error

Clearly this experiment did not test on an optimal range of C values, since smallest 3 values of γ took on the largest possible C value in the range and conversely, the largest γ values took on the smallest C values.

In any case, the optimal (γ, C_γ) is found at (0.02154, 2.68269). Since the range of C values tested was sparse as evidenced by figure 5 we will tune C_γ once more on a denser range of values.

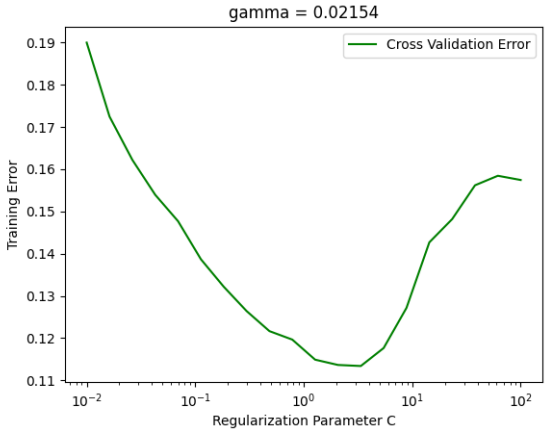

Figure 6: Tuning C_γ on a denser range

We find that optimal regularization parameter for $\gamma = 0.02154$ $C = 3.3598$ performs better than our initial value.

Finally, Figure 7 plots the training and test error for each of (γ, C_γ) using the entire training set (12000 examples)

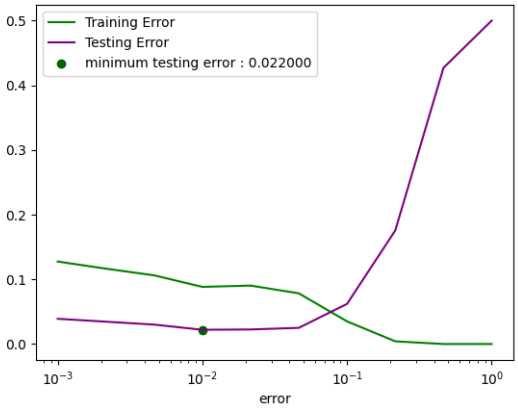

Figure 7: Training and Test Error for each (γ, C_γ)

In Figure 7 we see

- training error decreasing as gamma increases
- test error decreases until $\gamma = 0.022$, then plateaus until it rapidly increases for $\gamma > 0.1$

We can observe that as γ increases the model becomes increasingly more fitted to the test set and drives the training error to zero. Also we can see that the best choice of γ is close to the point where the training error decreases and the test error rapidly increases. This suggests that a good choice for the bandwidth parameter is one which fits a model as close to the training data as possible while still allowing for generalization to data outside the training set. Lastly we can observe the testing error increases rapidly compared to the decreased training error. This shows that the only information is only gained about the training set for $\gamma > 0.1$.

Experiment 3

In this section we will tune the hyper parameters of a multilayer perceptron (MLP) using the sklearn implementation `sklearn.neural_network._multilayer_perceptron.MLPClassifier`. In the interest of time. I have fixed the following hyper parameters:

- `solver = 'adam'`
- `learning_rate = 'adaptive'`
- `max_iterations = 500`
- 1 hidden layer

Another thing to note is I wont be doing 'grid search' to configure my hyper parameters. Instead tuning will be done by considering a range of values for the hyper parameter and choosing the value which minimizes the 5-fold cross validation error.

Activation Function

Here we will plot the 5-fold cross validation error of the MLP. Figure ... uses only 1 layer of hidden nodes. The number of nodes within the hidden layer will range over $\{2, 3, 4, 5, 6, 7, 8, 9\}$. Figure 8 shows how the error changes with the number of hidden nodes using 3 different activation functions, hyperbolic tangent (tanh), sigmoid (logistic), and rectified linear unit (relu).

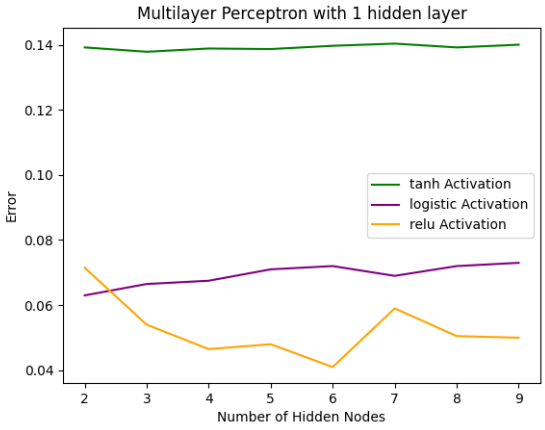

Figure 8: Activation Functions vs Number of Hidden Nodes

Clearly tanh is not a candidate. For 2 hidden nodes, the logistic function performs better than relu. As the number of nodes increases the error under the relu function decreases while the error under sigmoid increases. We can see that the cross validation error is minimized with 6 hidden nodes.

L2 Regularization

Figure 9 tests a range of values for the regularization parameter. For this experiment the number of nodes in the hidden layer is fixed at 6 since this was a adequate choice from Figure 8. We consider values α in the range `[10**i for i in np.linspace(-2,2,15)]`

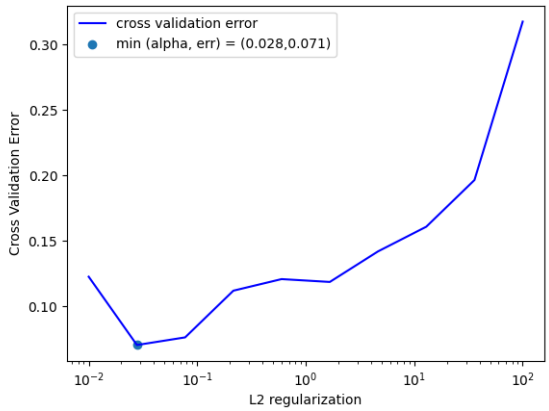

Figure 9: Testing L2 Regularization Parameter

We obtain an optimal value at $\alpha = 0.028$. This hyperparameter will be used for subsequent tuning experiments.

Hidden Nodes

Next we tune the number of hidden nodes. The number of hidden layers is fixed at one and Figure 10 considers hidden nodes h is the range $\{20, 40, 60, 80, 100, 120, 140\}$

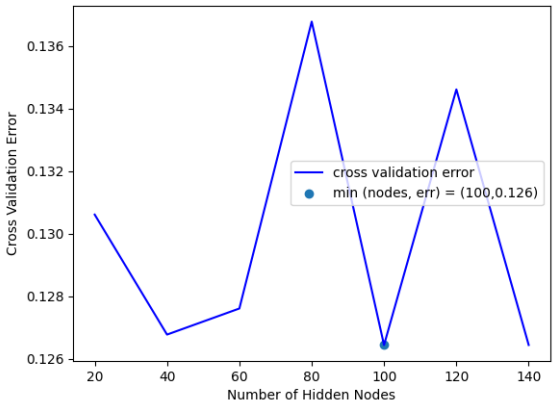

Figure 10: Testing L2 Regularization Parameter

We obtain a minimum error with 100 nodes. Although this graph has considerable variation the range of errors found are all within the 12-14% range.

To summarize,we have obtained an optimal configuration for our Multi Layer Perceptron using the following hyper parameters:

- `solver = 'adam'`
- `learning_rate = 'adaptive'`

- `max_iterations = 500`
- 1 hidden layer
- `activation = 'relu'`
- `alpha = 0.028`
- 100 hidden nodes

We will now experiment with the test set. Firstly Figure 11 ranges the maximum number of epocs over $\{1, 10, 25, 50, 75, 100, 250, 500, 1000, 2500, 5000\}$

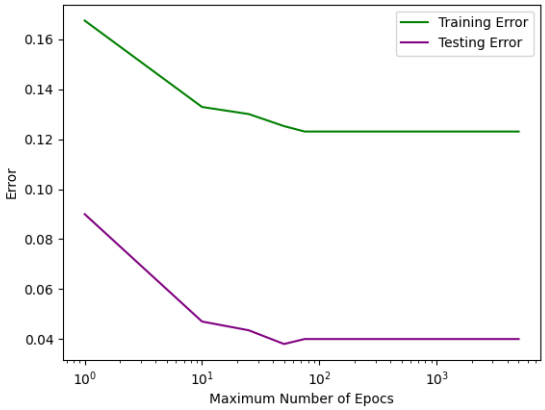

Figure 11: Max Iterations vs Error

We see both the training and testing error decreases rapidly as max iterations increases to . This (along with the warning `ConvergenceWarnings` printed by sklearn) indicates that the optimization process fails to converge and as such fails to minimize the error. As the max number of iterations increases the decrease in error plateaus. The plateau around max iterations = 100 is evidence that with this hyper parameter configuration, 100 iterations is sufficient to completely minimize the error.

Lastly we will look at the number of hidden nodes in the hidden layer. This is similiar to a previous experiment however this time we are considering training and test error instead of cross validation error of the training set.

Figure 11 this experiemnt we will look at hidden nodes in $\{2, 4, 6, 10, 20, 40, 80, 160\}$ and we will plot the training and test error.

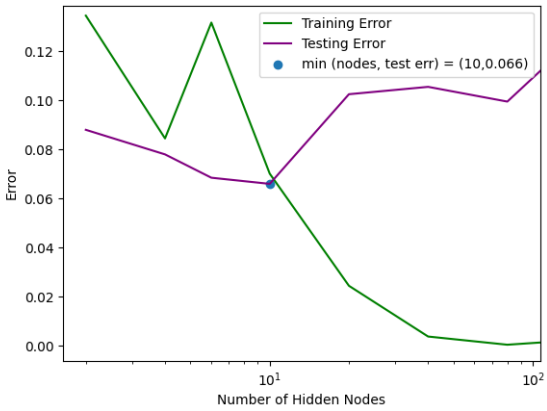

Figure 12: Max Iterations vs Error

We can see the training error converges to zero as the number of hidden nodes increases. This shows that more nodes in the hidden layer overfits to the training data, and in this case 0.2 percent of the training data is mis labelled.

The test error achieves its minimum with 10 nodes which is interesting when compared with the "optimal" value of 100 nodes we found when tuning the model with cross validation error in Figure 10. In Figure 12 we can see that 10 nodes is a clear minimum, unlike the in Figure 10 where the cross validation error varied heavily between 12 to 14%. It might've been a poor choice to pick 100 as the number of hidden nodes because we could've picked a much smaller number at a very little cost in error.

Experiment 4

With our 3 tuned models, we will determine if there's a significant difference in between their test errors. For each model we use the test error E to calculate the 95% confidence interval of the true error:

$$\left[\frac{-1.36}{\sqrt{n}} + E, \frac{1.36}{\sqrt{n}} + E\right]$$

We achieve the following test errors with our tuned models:

- Linear Kernel SVM : 0.0450
- Gaussian Kernel SVM: 0.0180
- MLP: 0.0414

Figure 12 visualizes the 95% CI's of our 3 tuned models. Note that that lower bound of each interval is clamped to zero.

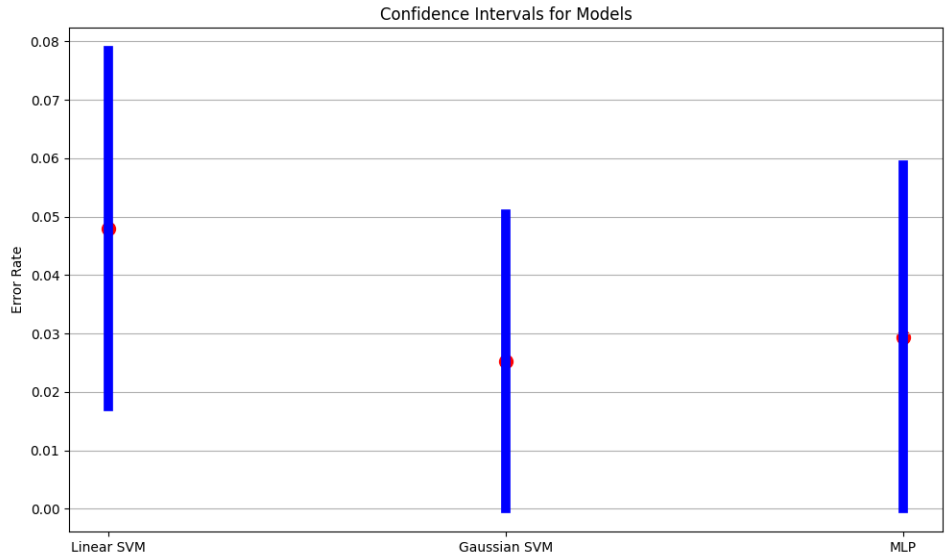

Figure 13: True Risk 95% CI

We can see that these confidence intervals are overlapping, which indicates that there's no significant difference between the true testing error of these 3 models. This suggests that the non linearity introduced by the Gaussian kernel did not offer significantly better performance on the available training set. With a larger testing set we would expect the CI's to become smaller and a significant difference might be observed between the Gaussian and Linear SVM.

Despite the lack of significant difference the testing error achieve on the Gaussian Kernel SVM was at 1.8% which in a practical sense is much lower than the Linear Kernel and MLP errors at 4.5% and 4.1% respectively.