Introduction to Machine Learning *Intekhab Naser, ZC11577*

HW2

Please note that only PDF submissions are accepted. We encourage using LaTeX to produce your write-ups. You'll need *mydefs.sty* and *notes.sty* which can be downloaded from the course page.

1. Explain in what case and why we need a regularizer and why l_2 norm regularizer is a reasonable one.
   (a) *In what case*? : In training the model, when unnecessary parameters are considered it tends to over-fit due to training of noisy data.
   (b) *Why*? : Regularization is a technique which discourages learning a highly complex and malleable model, so as to avoid the risk of overfitting and promote generalization so that the model makes better predictions. A regularizer does this by adding a penalty on the different parameters as the model complexity increases.
   (c) *Why l_2 norm* :
   - L_1 regularizer also known as Lasso Regression model, adds the absolute value of coefficient as a penalty term to the loss function which shrinks some parameters to zero which then do not play any role in the model. This is useful in feature selection.
   - Whereas L_2 regularizer also known as Ridge Regression model, adds the squared value of the coefficient as a penalty term to the loss function i.e. it does not completely cut down any parameters, but forces their value to be relatively small. As lambda increases, the value of the coefficients progressively decreases but are not cut down to zero. This makes L_2 work really well while dealing with overfitting:

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

2. What values of λ in the regularized loss objective will lead to overfitting? What values will lead to under-fitting? Note that λ is a multiplier for the regularizer term.
   (a) Overfitting will occur when there is no regularization being done. As λ is a multiplier for the regularizer term, overfitting will occur when $\lambda = 0$ i.e. the model is unregularized.
   (b) Underfitting occurs when there is over generalization and the model does not align along the given points. This happens when λ is very large.

3. Explain why the squared loss is not suitable for binary classification problems.
   (a) Suppose we have a sample x with correct classification as y and predicted classification as $\hat{y}$. Then, the squared loss function can be denoted as: $L2(x) = (1 - \hat{y}y)^2$
   (b) Squared loss is more commonly used in regression. However, it is not suitable for binary classification problems because:
   - Correct classifications are also penalized in situations where $\hat{y}y$ is very positive, even when y and $\hat{y}$ match.
   - It penalizes outliers severely. If a training sample is misclassified badly, it can have a great impact and shift the training too much.

4. One disadvantage of the squared loss is that it has a tendency to be dominated by outliers – the overall loss $\sum_n (y_n - \hat{y}_n)^2$, is influenced too much by points that have high $|y_n - \hat{y}_n|$. Suggest a modification to the squared loss that remedies this.
   (a) Squared loss can be modified to avoid inaccuracy by outlier data by adding weight normalization.
   (b) A regularization function can be added to the loss function so that the model created can be shrinked and effect of outlier data is reduced.

5. Perceptron algorithm:

   (a) Implement the perceptron algorithm for binary classification. Train and test it for classifying digits "1" and "6" in MNIST dataset. Note that we don't need the data of other digits in this part. Please use 1000 training examples and 1000 testing examples. Plot the accuracy on the test set w.r.t. the number of iterations. Here, processing each data-point is considered one iteration, so 1000 iterations means one pass over all training data.

   - Figure 1 shows Accuracy for classifying digits 1 and 6 is: **98.7**

   (b) Visualize the learned model in the image form to see if it makes sense. Note that the weight vector has both positive and negative values, so you should plot those on two separate planes. Note that you should use exactly the same testing data to be able to compare the results.

   - Figure 2 and 3 shows Learned model in image form for digits 1 and 6

   (c) For each class, visualize the 20 best scoring and 20 worst scoring images that are classified as that class. The worst ones are close to the boundary and may be wrong.

   - Figure 4 shows 20 Best scoring images for number 1.
   - Figure 5 shows 20 Worst scoring images for number 1.
   - Figure 6 shows 20 Best scoring images for number 6.
   - Figure 7 shows 20 Worst scoring images for number 6.

   (d) Randomly flip the label (add labeling error) for 10% of the training data and repeat (b) and (c).

   - Figure 8 shows Accuracy for classifying digits 1 and 6 with 10 percent random flip is: **95.0**
   - Repeating (b), Figure 17 and 18 shows Learned weight vectors (with 10% data with error-labels) in image form for digits 1 and 6 respectively.
   - Repeating (c) with 10% data with error-labels, Figures 19,20,21 and 22 shows 20 Best scoring images for number 1, 20 Worst scoring images for number 1, 20 Best scoring images for number 6 and 20 Worst scoring images for number 6 respectively.

   (e) Sort the data before training so that all "1"s appear before "6"s and plot the accuracy w.r.t. the number of iterations. Is this faster or slower in training? Explain why.

   - Figure 9 shows Accuracy Plot for classifying digits 1 and 6 with sorted data
   - From the plotted graph we can see that sorted data is slower in training.
   - In sorted training data the model achieves its best accuracy at around the 15^{th} epoch or in 15000 iterations.
   - While in shuffled training data the model gets learned or achieves it's best accuracy after the 1^{st} epoch or in 1000 iterations.

   (f) Repeat (a) with 10 training examples per class and compare with the result of (a).

   - Figure 10 shows Accuracy with 10 training examples is **70.0**
   - Figure 11 shows Accuracy with 1000 training examples is **99.2**
   - We can infer that increasing the number of training examples will increase the accuracy

6. SVM algorithm:

   (a) Implement the gradient descent algorithm for binary SVM. Train and test it for classifying digits "1" and "6" in MNIST dataset. Plot the accuracy on the test set w.r.t. the number of iterations. Here, processing each data-point is considered one iteration. You don't need to use all training data if it takes a long time to run.

   - Figure 12 shows accuracy plot with respect to number of epochs for digits 1 and 6.
   - To check how the weight vectors were formed, Figure 13 and 14 shows the visualization for 1 and 6 respectively

   (b) Play with the hyper-parameter C to see its effect in the accuracy and overfitting. For instance, try very large and very small values for it. Choosing the learning rate may be a little tricky. One popular strategy is to reduce it proportional to $\frac{1}{t}$ where t is the iteration number.

- After trying a number of values for C, I can infer the following:
  i. For very large values of Hyper-parameter C the accuracy is decreased drastically.
  ii. Small values of C does not have any effect on accuracy of test set.
  iii. Optimal value of C can be stated as 1/(number of epochs)

(c) Implement a function for 1-vs-all multi-class SVM that calls the binary SVM function.

- Figure 15 shows accuracy plot with respect to number of epochs for digits 1 vs all.

(d) Train and test it on all 10 digits in MNIST and report the confusion matrix as well as the average precision. $conf(i, j)$ is the number of images that are from category i and are classified into category j. You should normalize this so that all rows sum to one and then average accuracy is the average of its diagonal values.

- Average Precision is: 0.96399
- The confusion matrix can be seen below:
  [**0.984**, 0.0, 0.003, 0.001, 0.001, 0.002, 0.002, 0.001, 0.001, 0.004]
  [0.001, **0.986**, 0.003, 0.002, 0.0, 0.002, 0.002, 0.001, 0.004, 0.001]
  [0.003, 0.004, **0.95**, 0.01, 0.006, 0.006, 0.007, 0.003, 0.006, 0.005]
  [0.003, 0.003, 0.005, **0.955**, 0.003, 0.012, 0.003, 0.006, 0.007, 0.004]
  [0.001, 0.001, 0.004, 0.001, **0.975**, 0.001, 0.003, 0.003, 0.004, 0.007]
  [0.004, 0.003, 0.005, 0.008, 0.005, **0.952**, 0.008, 0.003, 0.007, 0.004]
  [0.005, 0.002, 0.004, 0.004, 0.005, 0.004, **0.969**, 0.002, 0.002, 0.002]
  [0.004, 0.003, 0.005, 0.005, 0.006, 0.002, 0.004, **0.957**, 0.006, 0.009]
  [0.005, 0.002, 0.004, 0.009, 0.004, 0.01, 0.004, 0.003, **0.952**, 0.006]
  [0.002, 0.002, 0.004, 0.005, 0.008, 0.004, 0.002, 0.006, 0.007, **0.958**]
- We can seen from the above matrix that the diagonal values indicate correct prediction and as expected the accuracy is high for those cells.

(e) Look at top mistakes and show images along with ground truth label and the predicted label to see if they make sense.

- Figure 16 shows top mistakes for every number (0-9).
- Label for every image is presented as (i,j) where i is actual label of that image and j in predicted label.
- This output makes some sense as numbers 1,2,6,8 are shown to be mis-classified as 7,7,0,3 respectively.

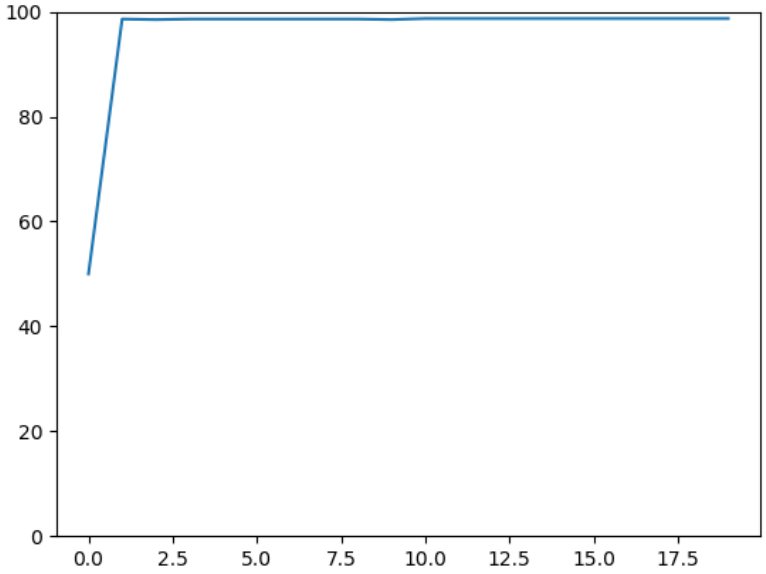

Figure 1: **Plot for accuracy with respect to number of epochs for number 1 and 6 (1 epoch is 1000 iterations)**

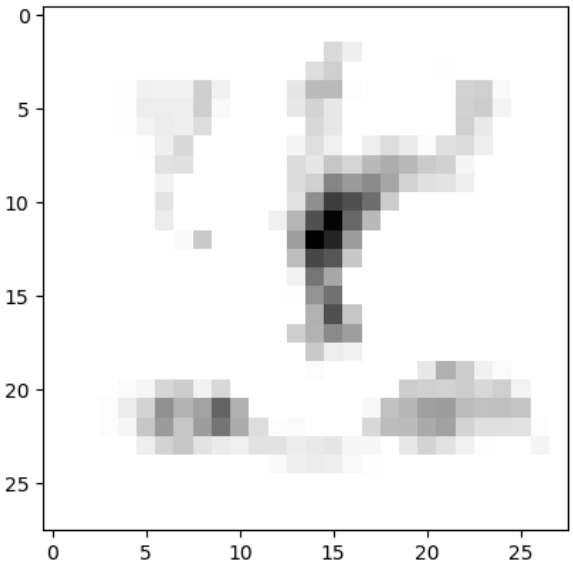

Figure 2: **Learned Weight Vector for image 1**

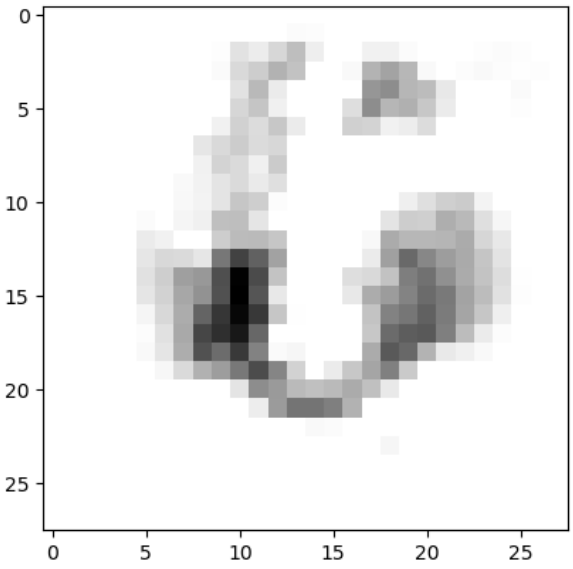

Figure 3: **Learned Weight Vector for image 6**

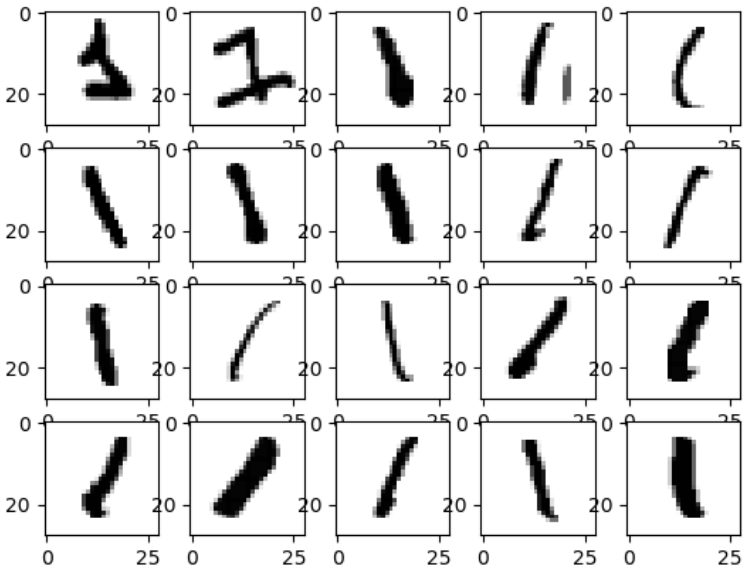

Figure 4: **20 Best scoring images for number 1**

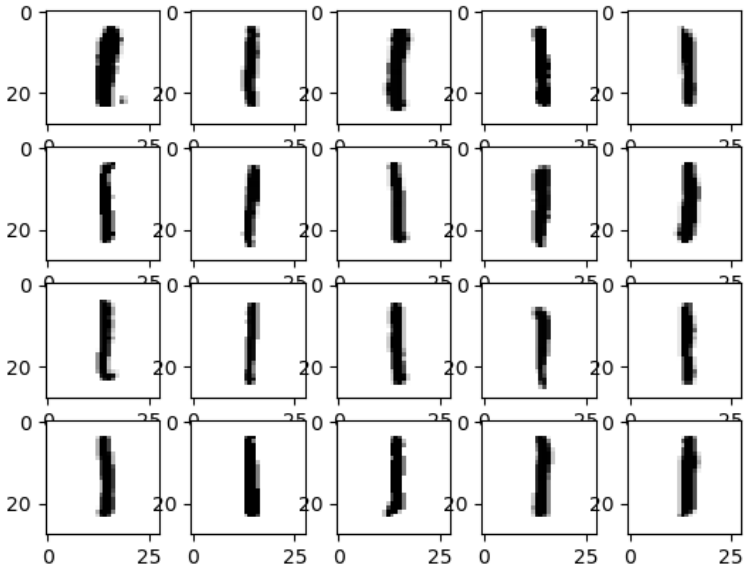

Figure 5: **20 Worst scoring images for number 1**

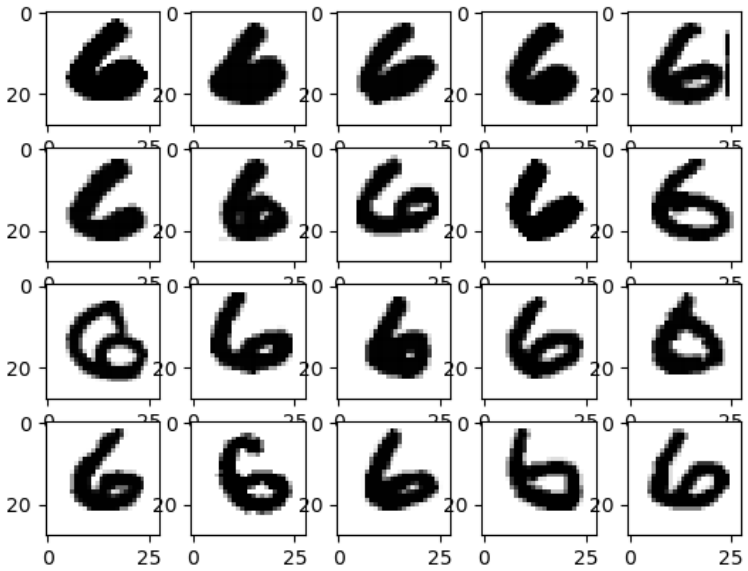

Figure 6: **20 Best scoring images for number 6**

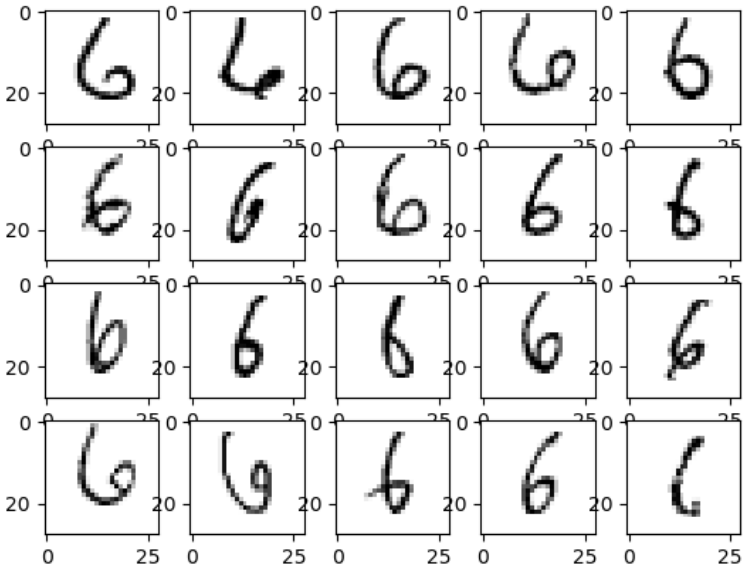

Figure 7: **20 Worst scoring images for number 6**

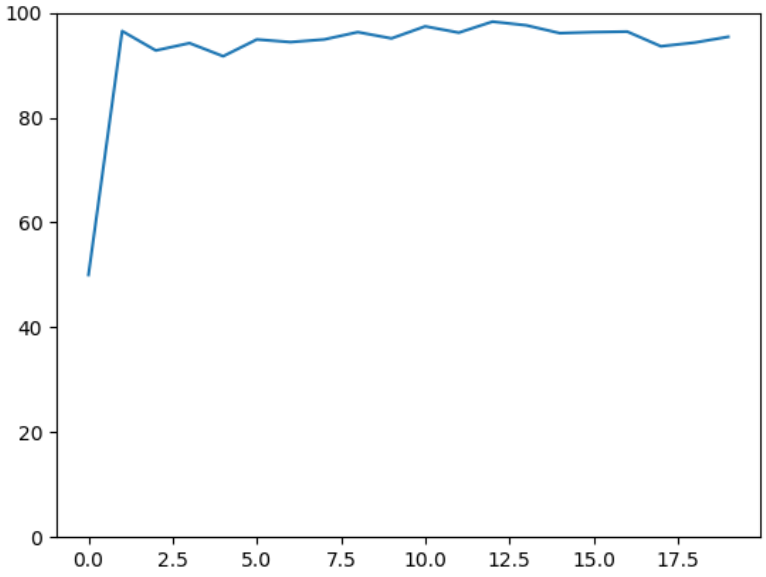

Figure 8: **Accuracy Plot for classifying digits 1 and 6 with 10 percent random flip of data**

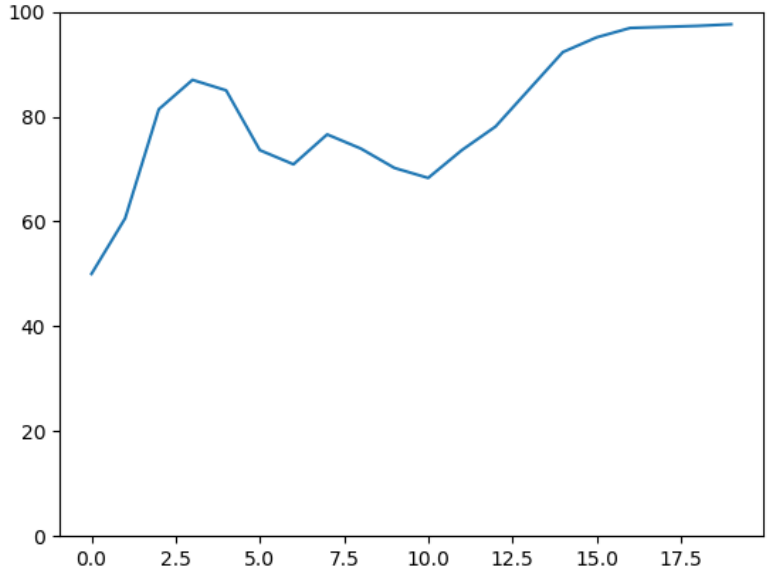

Figure 9: **Accuracy for classifying digits 1 and 6 with sorted data**

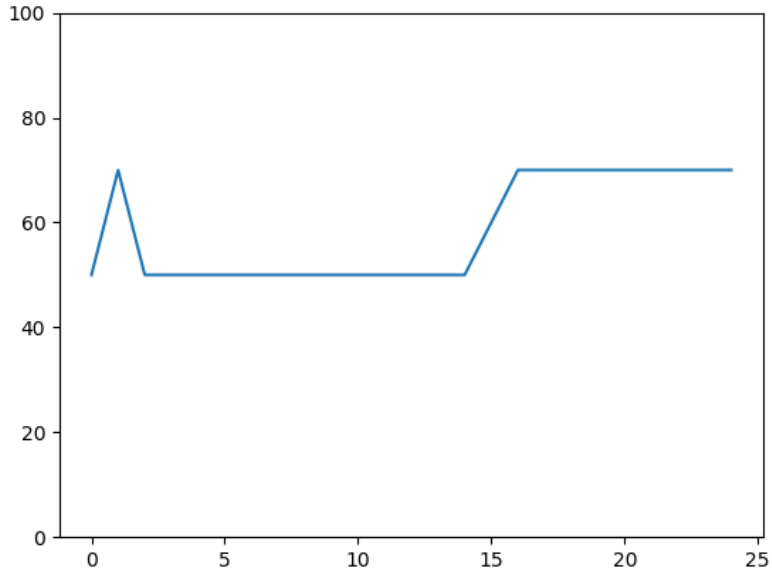

Figure 10: **Accuracy Plot with 10 training examples**

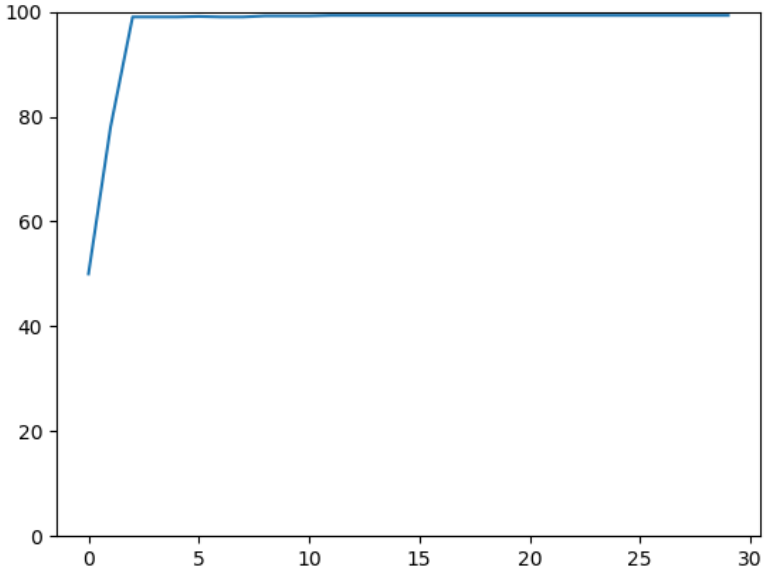

Figure 11: **Accuracy Plot with 1000 training examples**

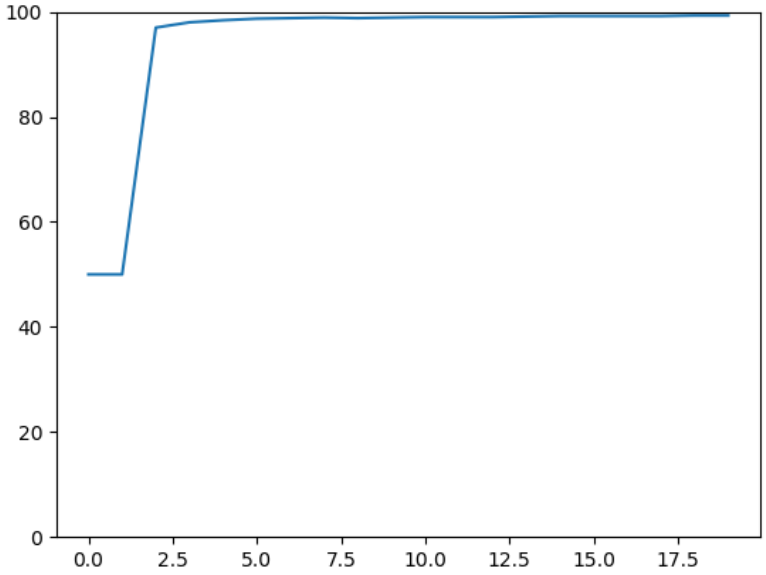

Figure 12: **Accuracy Plot with respect to number of epochs for digits 1 and 6 with SVM (1 epoch is 1000 iterations)**

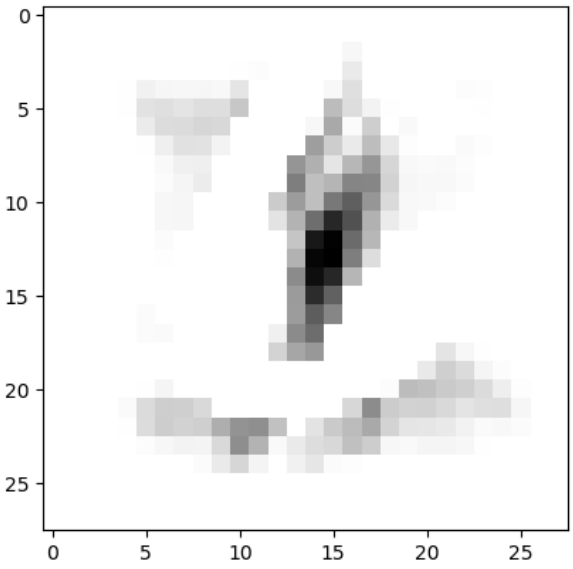

Figure 13: **Weight vector visualization for 1 with SVM**

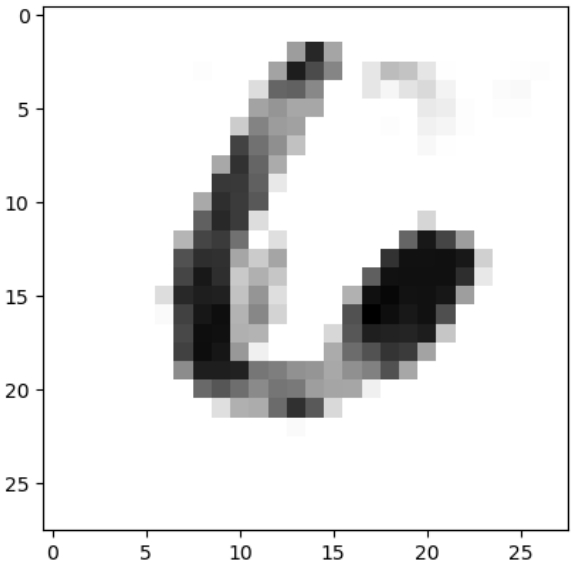

Figure 14: **Weight vector visualization for 6 with SVM**

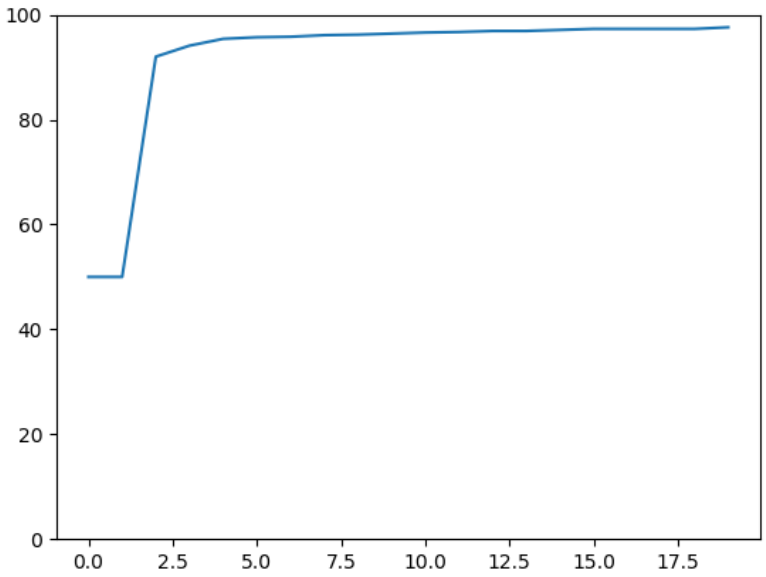

Figure 15: **Accuracy Plot with respect to number of epochs for digits 1 vs all with SVM (1 epoch is 1000 iterations)**

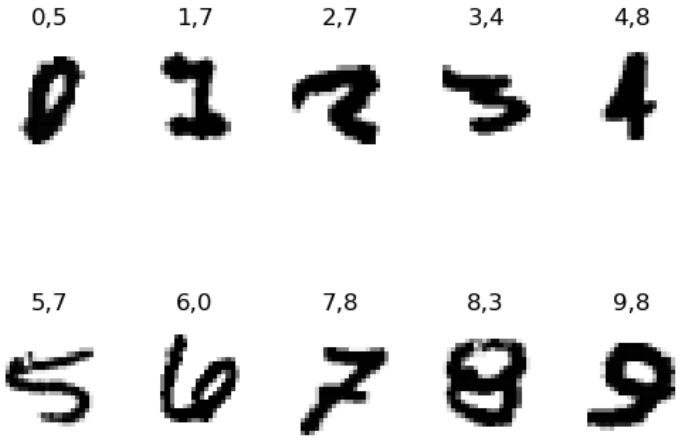

Figure 16: **Plot of Worst set of images with their Actual and Predicted labels**

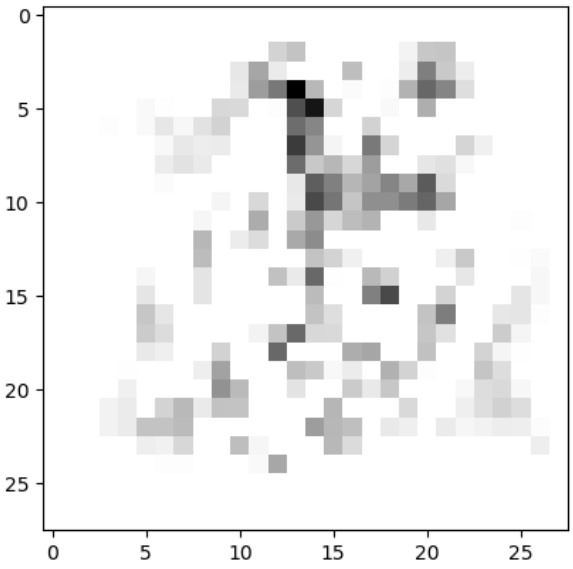

Figure 17: **Plot of Weight vector for number 1 with 10 percent error labels**

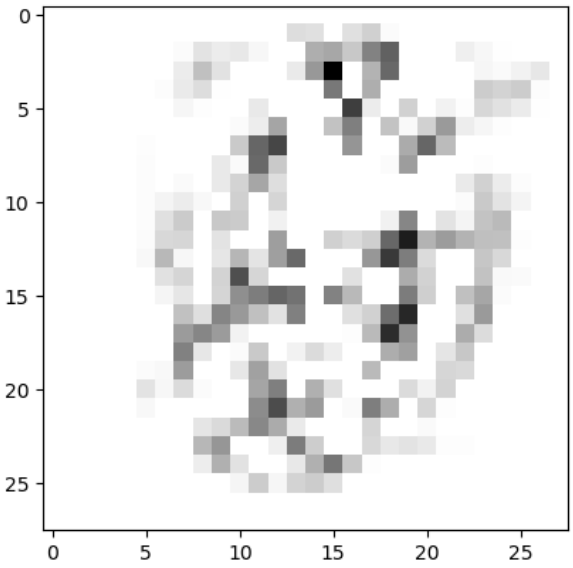

Figure 18: **Plot of Weight vector for number 6 with 10 percent error labels**

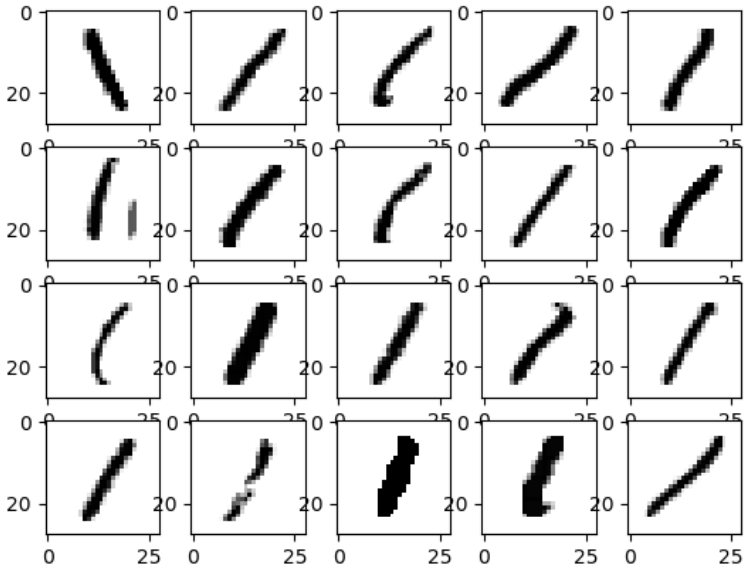

Figure 19: **Plot with 10 percent error labels for 20 Best scoring images for number 1**

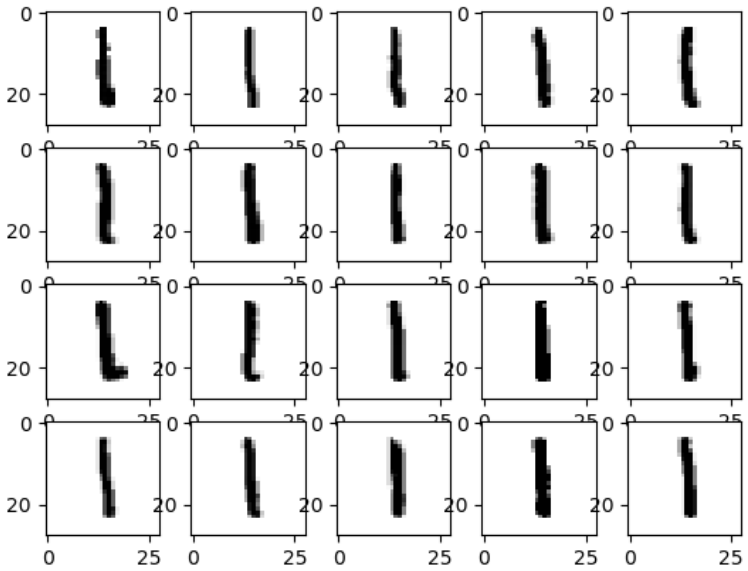

Figure 20: **Plot with 10 percent error labels for 20 Worst scoring images for number 1**

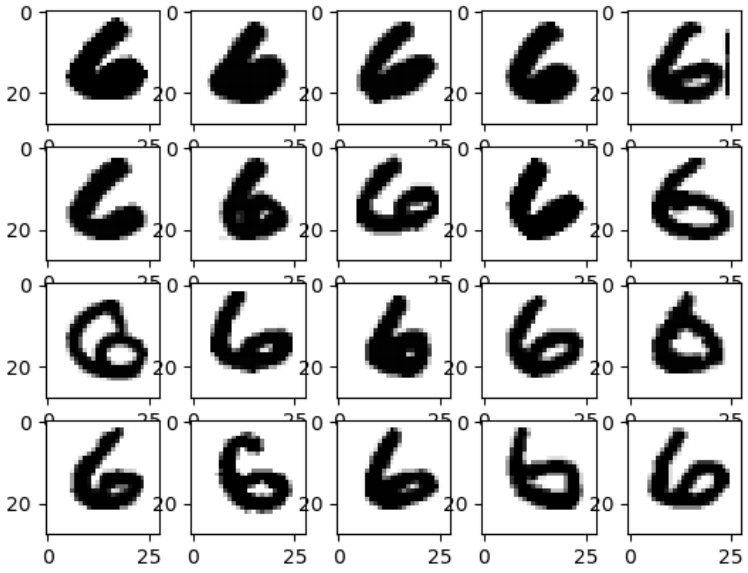

Figure 21: **Plot with 10 percent error labels for 20 Best scoring images for number 6**

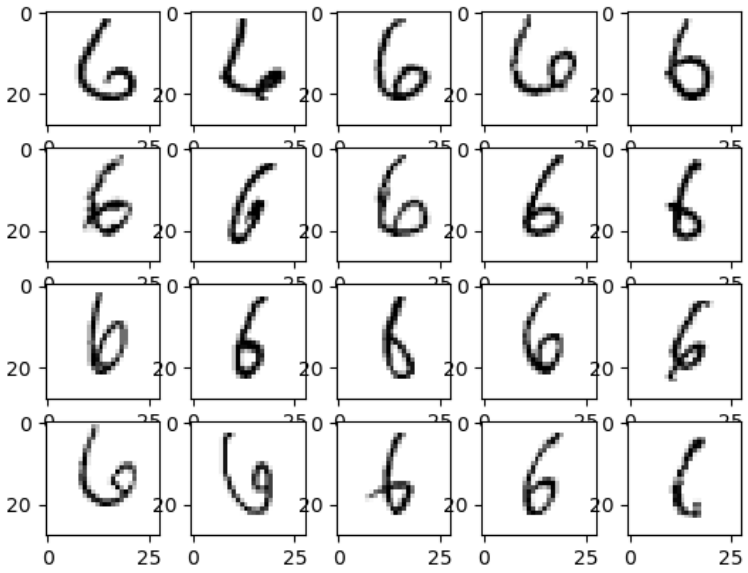

Figure 22: **Plot with 10 percent error labels for 20 Worst scoring images for number 6**