Introduction to Machine Learning | *CMSC 678, Fall 2019*

HW 2

Divya Natolana Ganapathy

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 regularizer constraints the coefficient estimates towards zero. It avoids over learning a very flexible model and prevents overfitting. If a regularizer is not used, we might minimize the training error so much that it would work great on the training data but would perform poorly for any new test data. A regularizer reduces variances without too much increase in bias.
   L2 more stable,L2 has better resistance to horizontal adjustments. If a data point is moved horizontally the shape of L2 would not really change much but L1 could completely change the prediction and boundaries. L2 norm is a convex function and when added on a convex loss function, the combined function is still convex. As L2 norm uses Euclidean distance, there is always just one way of going from point a to point B. Whereas L1 norm uses Manhattan(taxicab) distance which means there are multiple ways to get from point A to Point B , leaving us with multiple solutions, which is not desirable always. L1 norm gives sparse coefficient vector whereas l2 norm gives non-sparse coefficient which improves the prediction performance as it makes use of more features instead of ignoring them.
   As L2 is a square of a value, we can use matrix math to compute it .In case of L1 as it is not a closed form , it is a non-differential piecewise function, hence is more expensive to compute. Hence it is reasonable to use l2 norm regularizer.

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.

   The impact of the regularization term is tuned by multiplying it by the scalar (lambda) also called the regularization rate. Increasing this would increase the regularization which means the model would be very simple , would not learn enough about the training data to make predictions leading to underfitting . Overfitting of the model happens when lambda value is too low, making the model very complex which learns a lot from the training data and would not generalize with new data.

3. Explain why the squared loss is not suitable for binary classification problems.

   The double derivative of squared loss in not >=0 which means it is not a convex function. If such a function is applied on a binary classification which is a convex function , it is not guaranteed that global minima would be reached , we could get stuck with a local minimum. It leads to low convergence rate. Functions that give high values for some points will perform poor with squared loss function. Therefore, its preferred not to use Squared loss on binary classification. If we use Squared loss (actual value=1 and predicted is 0) squared loss gives $(1-0)^2$ =1 , if we instead use some other loss function like log loss, we get ***-(1 * log(0) + 0 * log(1))***=-infinity**.** This gives a better picture of the error compared to squared loss.

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.

   This major disadvantage of squared loss is that it heavily penalizes outliers, because of squaring the difference. To overcome this we could use Absolute error , $\left|y_n - \hat{y}_n\right|$ (as here we only take the absolute diff between the actual and predicted) and not the square of the diff. So if the difference is large it would not blow up by squaring the error. We could also use truncated loss, where the upper values are truncated after a threshold or use Hubber loss $\frac{1}{2}\left[y_n - \hat{y}_n\right]^2$.

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.

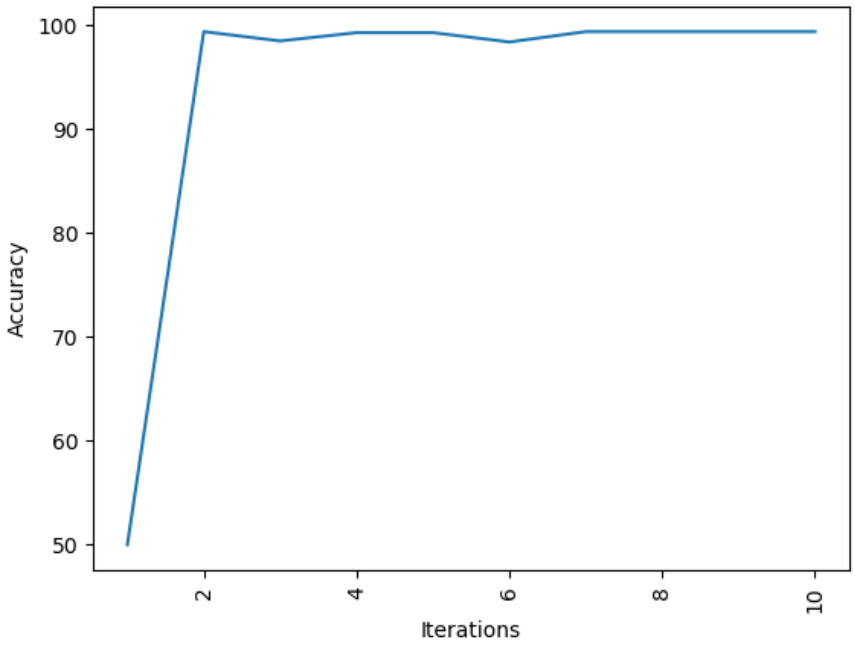
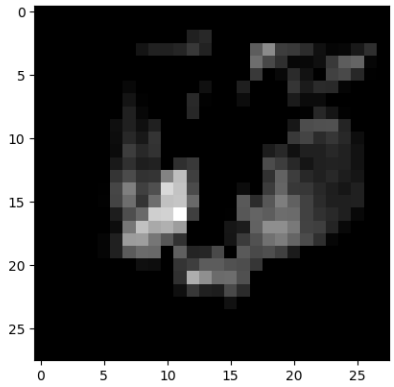


Figure 5 (a):Plot of accuracy Vs Number of iterations

We see that as the number of iterations increase, we are getting better with accuracy. With just 1 iteration the model has not learned enough to make good predictions so the

accuracy is between 50-60%. When we complete 10 iterations the model is giving close to 100 % accuracy.

(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 the same testing data to be able to compare the results.



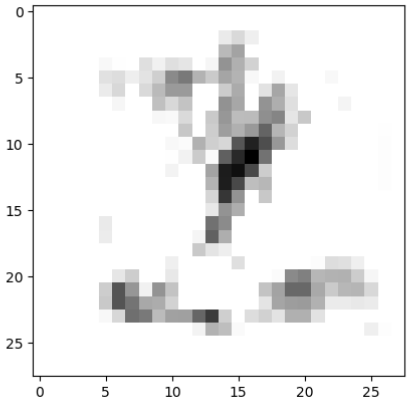


Figure 5 (b) 1, 2 Plot of learned negative and positive weights

(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.

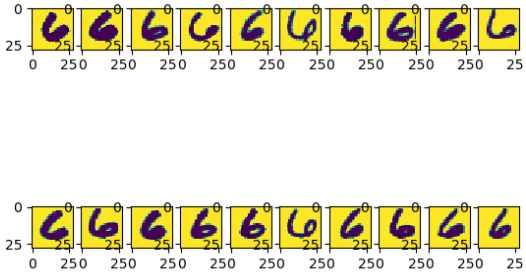


Figure 5 (c1), - 20 Best 6s

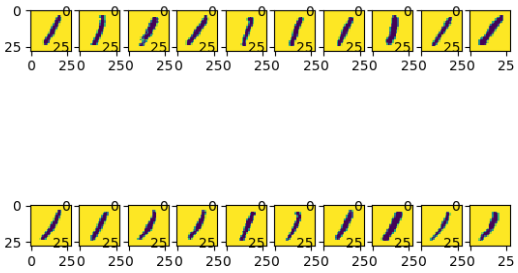


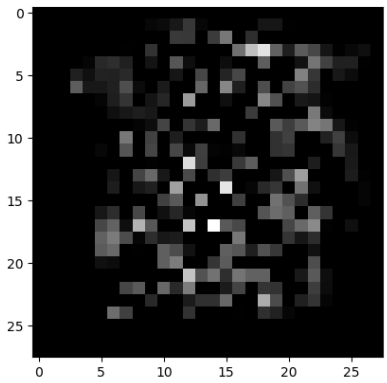
Figure 5(c2) – 20 Best 1s

Figure 5(c3)- 20 worst 6s (we can see a few 1s being classifies as 6s here, or not too clear 6s)

Figure 5(c4)- 20 worst 1s (we can see a few 6s being classified as 1s here, or not too clear 1s)

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

Flipping the labels in training data will make the model learn a 6 as 1 or a 1 as 6.



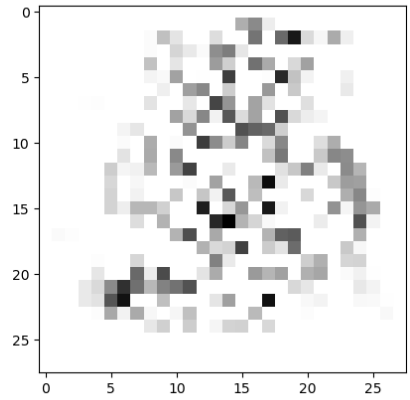
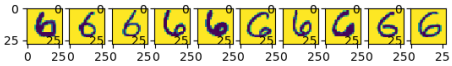
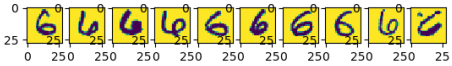
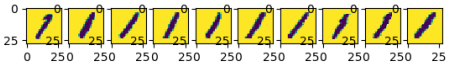


Figure (5d) Weights 1and weights 2- The learned model weights look different from the ones learnt in 5(b) because of label flipping

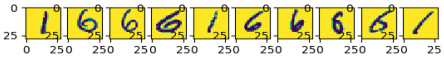


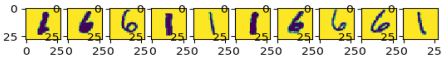


Best 6

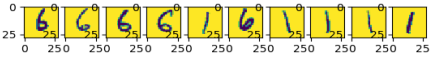


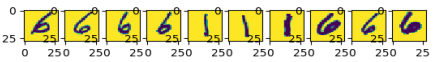
Best 1





Worst 6



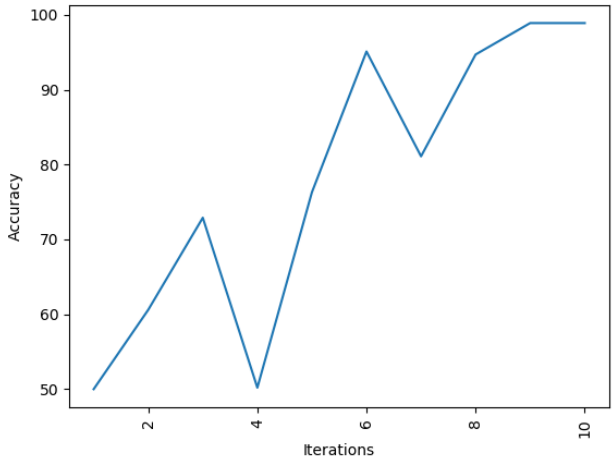


Worst 1

Figure (5d) Best 6s, Best 1s, Worst 6s and wors1s- We see here that there is more error in prediction because the learning itself is errored

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

```
Total time without orderly arrangement: 0.2632889747619629

Process finished with exit code 0
```



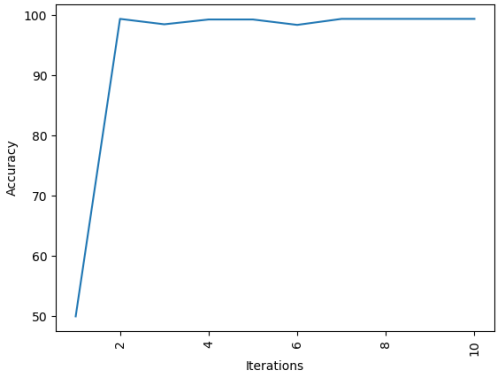

```
Total time with sorted labels: 0.2687351703643799

Process finished with exit code 0
```

Figure 5(e) – 1) Time(in sec) take for learning and predicting if the labels are arranged in random order 2) Plot of Accuracy Vs Iterations when the training model is arranged all 1s first and all 6s first. 3) Time taken for learning and predicting if the labels are arranged in the above-mentioned order of all 1s first and all 6s first.

The time taken is lesser or almost the same for unordered compared to ordered training. An example of 3 being lower than 1 is also attached in the documents. It varies depending on the number of times it’s run. However, the accuracy of the predictions is changing very sporadically as the iterations increase , we see a dip at 4 iterations , it could be because the

model has learnt only one kind of labels till then but if the test labels are of a diff kind , the model cannot predict well. However, with increased iterations (10 iterations) the model would have seen both the kinds of labels and adjusts the weights accordingly. Shuffling makes sure there is a uniform distribution of data. This method is not the best as if we stop at lower iterations, we could get very low accuracy.

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



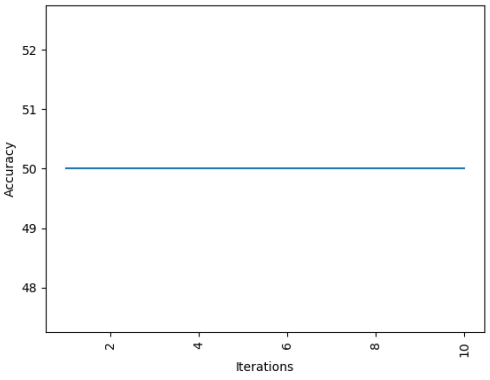


Figure 5 (f) Plot of accuracy vs Iterations for 1000 training and 10 training examples of each label.

We can see that the accuracy is very poor when we use a smaller number of training data but same number of test data as 5 a. This is because the model has not seen too many varying kinds of features, giving it very little to train on. The model is tested on labels of features whose kind it has not learnt yet. As training data points are very few , any increase in iterations will not make a impact on the accuracy.