CSE343: Machine Learning

Assignment-1

Madhav Kansil (2022270)

September 14, 2024

1 Section A (Theoretical)

Problem (a) (2 marks)

You are developing a machine-learning model for a prediction task. As you increase the complexity of your model, for example, by adding more features or by including higher-order polynomial terms in a regression model, what is most likely to occur? Explain in terms of bias and variance with suitable graphs as applicable.

Solution

When developing a model for prediction task as we increase the complexity of the model the ***variance*** for the model will ***increase*** and **`bias`** would **`decrease`**. This is because by definitions

- ***variance***: How much the model varies on test data when ran with different parameters.
- ***Bias***: It means the loss/error in the training model. It means how well the model fits the dataset.

There is always a tradeoff between Bias and Variance. One should need to maintain a balance between them i.e. Both Bias and Variance should be low for a model to be accurate enough on the unseen data as well as optimize loss in such a way that training loss in minimized.

This can be visualised by the following image.

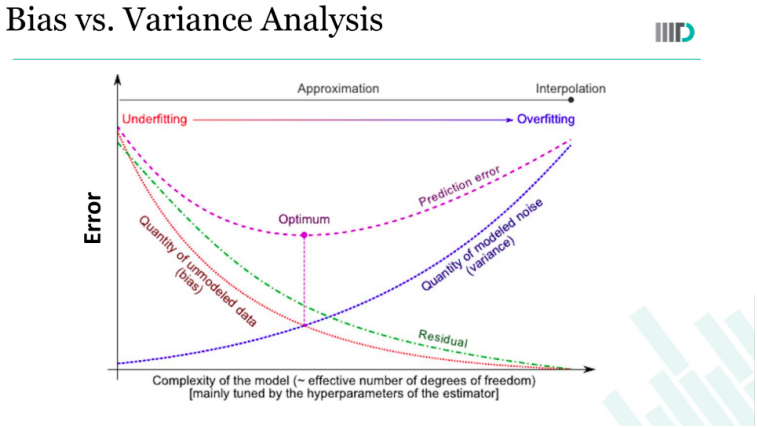

Figure 1: Bias Variance Tradeoff (Credits **Lecture 5 Slide 43**)

Graphs and figures that explain bias and variance are presented as follows.

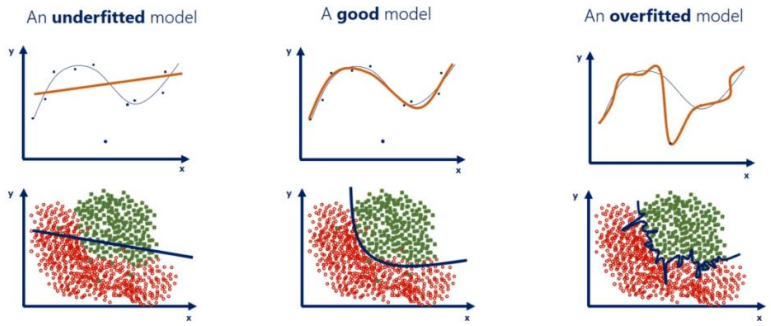

Figure 2: Bias Variance Explaination (Credits Image Site Link)

The underfitted model represents that high bias low variance. And in case of overfitting it is low bias and high variance.

Problem (b) (3 marks)

You're working at a tech company that has developed an advanced email filtering system to ensure users' inboxes are free from spam while safeguarding legitimate messages. After the model has been trained, you are tasked with evaluating its performance on a validation dataset containing a mix of spam and legitimate emails. The results show that the model successfully identified 200 spam emails. However, 50 spam emails managed to slip through, being incorrectly classified as legitimate. Meanwhile, the system correctly recognised most of the legitimate emails, with 730 reaching the users' inboxes as intended. Unfortunately, the filter mistakenly flagged 20 legitimate emails as spam, wrongly diverting them to the spam folder. You are asked to assess the model by calculating an average of its overall classification performance across the different categories of emails.

Solution

Getting the values from the question in terms of `True Positive(TP)`, `True Negative(TN)`, `False Positive(FP)`, `False Negative(FN)`. Here for the classes of binary classification we take, being identified as `spam as belonging to class 1` and `legitimate email as belonging to class 0`. Extracting the values for the following from the text and making a confusion matrix:

1. `TP = 200`: We get this information by *model successfully identified 200 spam emails.*
2. `TN = 730`: We get this information by *recognised most of the legitimate emails, with 730 reaching the users' inboxes*
3. `FP = 20`: We get this information by *filter mistakenly flagged 20 legitimate emails as spam*
4. `FN = 50`: We get this information by *50 spam emails managed to slip through, being incorrectly classified as legitimate*

Consfusion Matrix is given by:

	Predicted Spam	**Predicted Legitimate**
Actual Spam	200(`TP`)	50(`FN`)
Actual Legitimate	20(`FP`)	730(`TN`)

Table 1: Confusion Matrix

For getting the model's overall classification performance. We'll use the metrics as **`Accuracy`**, **`Recall`**, **`Precision`**, **`F1-Score`** and **`Specificity`**. The formulae for the following is given by:

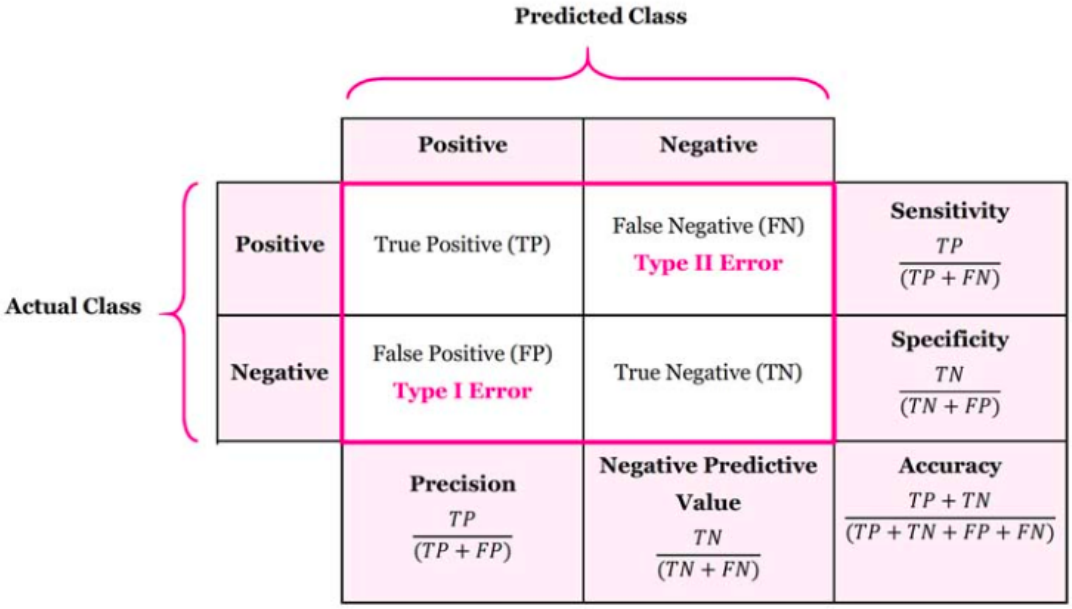

Figure 3: Corresponding Formulae (Credits: **Lecture 1 Slide No. 11**)

1. **`Accuracy:`** Accuracy depicts how accurate our model is in identifying correctly classified emails. It gives an overall performance measure of the model.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{200 + 730}{200 + 730 + 20 + 50} = \frac{930}{1000} = 0.93$$

   Therefore the model would predict a given email in it's actual class with an accuracy of 93%.

2. **`Recall/Sensitivity:`** Recall tells us the ability of the model to classify True Positive data points out of all the original Positive data points in the data. In this context it will give the ratio which would describe the ability of model of correctly classify True spam emails out of all the actual spam emails in the dataset.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{200}{200 + 50} = \frac{200}{250} = 0.80$$

   This means that the model correctly identifies 80% of the spam email in the given dataset.

3. **`Precision:`** Precision metric gives us an idea of how precise our model is in predicting the True Positives for the data points out of all the data points classified in Positive class. In this context it will give the proportion of emails classified as spam that are actually spam. It is also called positive predictive value.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{200}{200 + 20} = \frac{200}{220} \approx 0.909$$

   Therefore out of all the emails classified as spam approx 91% are correctly classified as spam.

4. **`Specificity:`** Specificity tells us the ability of the model to classify True Negative data points out of all the original Negative data points in the data. In this context it will give the ratio which would describe the ability of model of correctly classify True Legitimate emails out of all the actual Legitimate emails in the dataset.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} = \frac{730}{730 + 20} = \frac{730}{750} \approx 0.973$$

This means that out of the dataset and original Negative data points in the dataset. Model is able to predict legitimate emails with probability of 97.3%.

5. **`F1-Score:`** F1-Score act as a harmonic mean between Recall and Precision. It represents both precision and recall into one metric.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times 0.909 \times 0.8}{0.909 + 0.80} = \frac{1.4544}{1.709} \approx 0.851$$

Overall the Model has quite good performance as per the metrics. But a few improvements could be done as Recall is 80%. This would mean that 20% of the spam emails would be classified as legitimate which would clutter the inbox of a user with spam emails.

Problem (c) (3 marks)

Consider the following data where y(units) is related to x(units) over a period of time: Find the equation of the regression line and, using the regression equation obtained, predict the value of y when $x = 12$.

x	**y**
3	15
6	30
10	55
15	85
18	100

Table 2: Table of x and y values

Solution

The problem is solved by Linear Regression with loss function taken as Mean Squared Error or `MSE`. Equation of the **True** Model is given by

$$y = \beta_0 + \beta_2 X + \epsilon_i \tag{1}$$

where ϵ_i is the error for ith sample which is randomly drawn from an unknown normal distribution $\epsilon \sim N[0, \sigma^2]$.
Since this is uni-variate analytical solution for the problem exists.
We take an estimator to the parameters as

$$\hat{y_i} = \hat{\beta_0} + \hat{\beta_1}$$

This becomes the equation of our regression lines.
We first start by deriving the formulas for $\hat{\beta_0}$ and $\hat{\beta_1}$.

First for $\hat{\beta}_0$ we have to minimize the squared sum of residuals or error terms. We find error term in terms of y_i and $\hat{y}_i$. As error would be original - predicted.

$$\epsilon_i = y_i - \hat{y}_i$$

$$\epsilon_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$$

The sum of squared residuals is given by

$$S = \sum (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2$$

Partial Derivative w.r.t $\hat{\beta}_0$

$$\frac{\partial S}{\partial \hat{\beta}_0} = -2 \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)$$

Setting to zero and doing algebraic manipulation we get

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Similarly for $\hat{\beta}_1$

$$\frac{\partial S}{\partial \beta_1} = -2 \sum x_i (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)$$

Setting to zero and doing algebraic manipulation and substituting $\hat{\beta}_0$ we get

$$\hat{\beta}_1 = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2}$$

Now we go to the calculation. The following Table shows the computation done.

Now summing everything and putting in the formula for $\hat{\beta}_1$ we get

x	y	xy	x^2
3	15	45	9
6	30	180	36
10	55	550	100
15	85	1275	225
18	100	1800	324

Table 3: Computations

$$n = 5, \quad \sum x = 52, \quad \sum y = 285, \quad \sum xy = 3850, \quad \sum x^2 = 694$$

$$\hat{\beta}_1 = \frac{5 \times 3850 - 52 \times 285}{5 \times 694 - 52 \times 52}$$

we get $\hat{\beta}_1 = 5.78$. Similarly we compute for $\hat{\beta}_0$ as $\hat{\beta}_0 = -3.11$.

The final equation comes out to be:

$$\hat{y_i} = 5.78X_i - 3.11$$

Now at $x = 12$ we get as prediction as

$$\hat{y} = 5.78(12) - 3.11$$

Giving us the value of $\hat{y}$ at $x = 12$ as **66.25**.

Problem (d) (2 marks)

Given a training dataset with features X and labels Y , let $\hat{f}(X)$ be the prediction of a model f and $L(\hat{f}(X), Y)$ be the loss function. Suppose you have two models, f_1 and f_2, and the empirical risk for f_1 is lower than that for f_2. Provide a toy example where model f_1 has a lower empirical risk on the training set but may not necessarily generalize better than model f_2.

Solution

We can take **True** Model f any function. Say f is $\cos 2\pi x$. We generate data data with noise on the true model. We get say these dummy points.

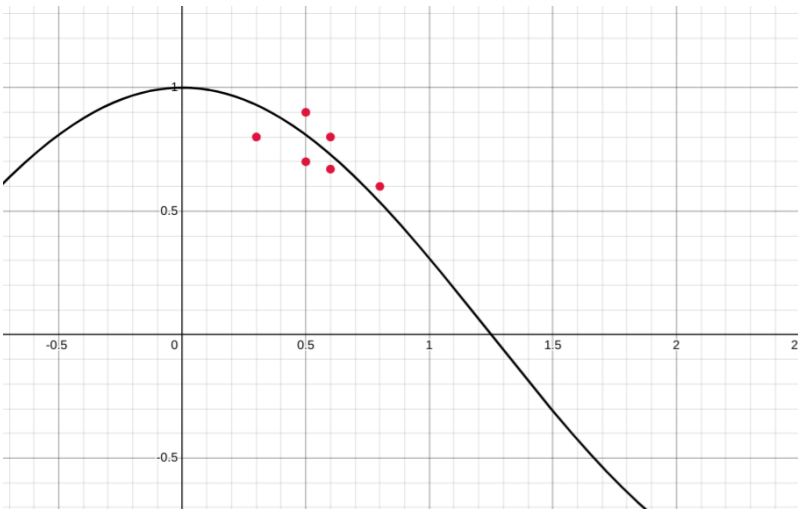

Figure 4: True f and 5 dummy points with noise

From this we take f_1 to be as some combination of values of polynomial

$$x^9 + bx^8 + cx^6 + dx^6 + fx^5 + s + qx^3 + wx$$

which fits the model as Figure 5. Since it is not representative of the true model. It will have low empirical risk but won't be able to generalise well. Take f_2 as any linear combination of line

$$y = mx + c$$

Now we take a green "test" point. We will clearly observe that test point is better estimated by f_2 even though the empirical risk of f_1 is lower.

Hence not always the model with low empirical can generalise. Sometimes it can be the case that it over fits the data.

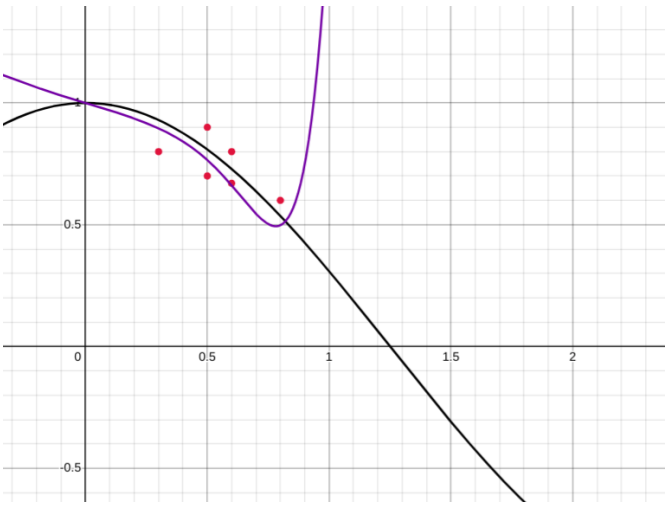

Figure 5: $f1$ its fitting

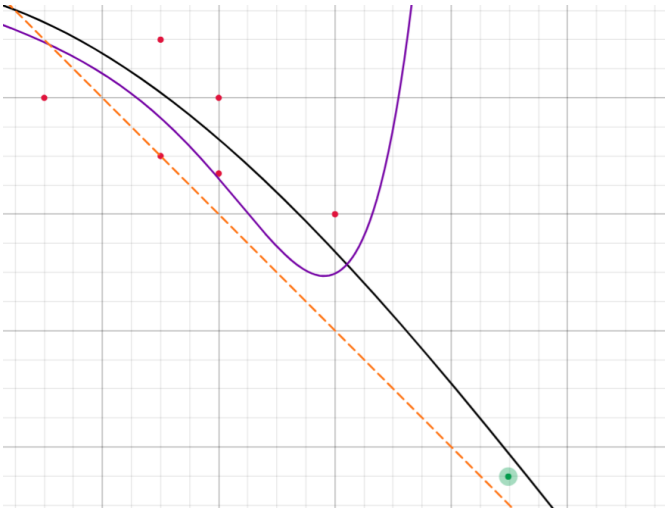

Figure 6: $f2$ its fitting and Green test point

2 Section B (Scratch Implementation)

This problem required us to implement Logistic Regression from scratch. Using the **Heart Disease** Dataset was provided. Corresponding implementation is provided in `main.ipynb`. Some pre-processing is done for NA values which were replaced by median values and mode values for continuous and categorical variables respectively.

We split the dataset in 70:15:15 for (train : test : validation).

Problem (a) and (b) Batch GD (3+2 marks)

For this part we used Batch Gradient Descent using cross entropy as loss function. In Batch Gradient Descent we use per iteration all the training set and then update the weights. We get these following graphs.

Convergence of the Model

The model is ran on only 1000 and could reach the max accuracy of approx 73% and gives a minimum loss of little less than 4 - 3.65 and it had loss initially between 18 and 16. At first the

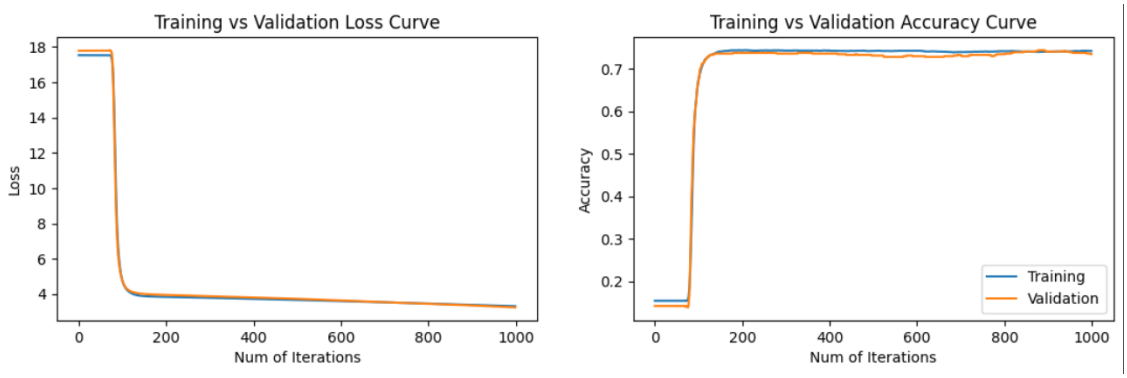

Figure 7: Graphs for the unscaled data (500 epochs and 0.0001 eta)

graph shows no drop in loss but after a few hundred iterations it steeply drops and then reduces very slowly. But after applying `min-max` scaling. The graph is as follows

Scaled Data

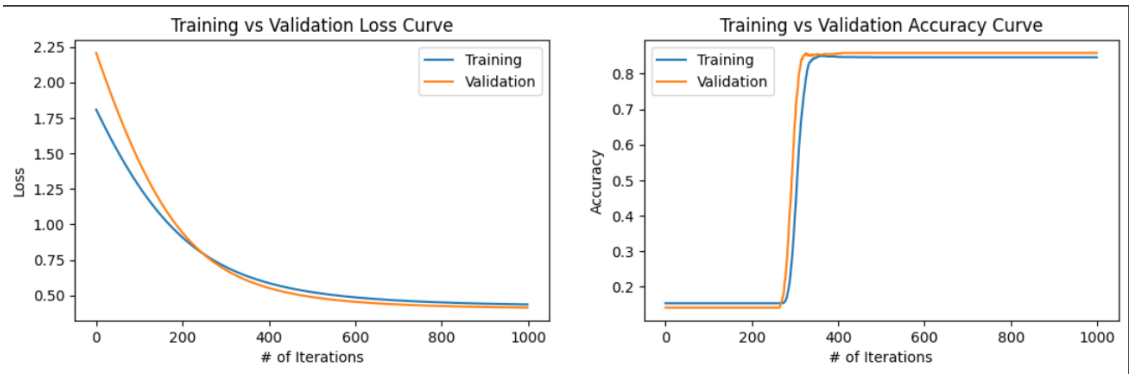

Figure 8: Graphs for the scaled data (1000 epochs and 0.01 eta)

As we can see the after scaling data the loss for the model reduces significantly and accuracy is increased by almost 10%. The graph is alot smoother and due to scaling is also converges faster and all the values are normalized by `min-max`. This is because Logistic Regression is affect alot by the values and outliers in the data and scaling helps to reduce down all values in the variables on more suitable number scale.

Problem (c) (2 marks)

In Figure 10 we can see the provided Confusion Matrix for the Validation set. There is disparity in True positive detection because people those who were diagnosed with heart disease were way less than those who were. In Figure 9 We can see the metrics for the data.

The metrics that were to be analysed provide insights as follows

- **`Recall:`** Recall tells us the ability of the model to classify True Positive data points out of all the original Positive data points in the data. In this context it will give the ratio which would describe the ability of model of correctly classify True people diagonsed with Heart Disease out of all the actual people who had Heart disease in the dataset.

- **`Precision:`** Precision metric gives us an idea of how precise our model is in predicting the True Positives for the data points out of all the data points classified in Positive class. That means it will provide proportion of people classified as having heart disease that do have the heart disease.
- **`ROC AUC:`** It shows how well the model distinguishes between the two classes (positive and negative) as the decision threshold is varied.
- **`F1 Score:`** F1-Score act as a harmonic mean between Recall and Precision. It represents both precision and recall into one metric.

```
Recall :0.15555555555555556
Precision :0.1308411214953271
ROC-AOC :0.45470085470085464
F1 :0.14213197969543145
```

Figure 9: Metrics

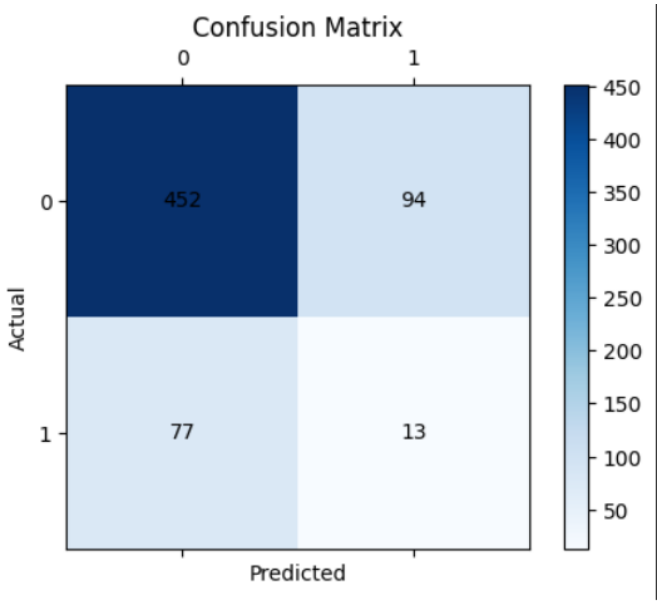

Figure 10: Confusion Matrix

Problem (d) (3 marks)

For this section 2 new optimization techniques where introduced. Following is the brief of there methodology.

- **`Stochastic Gradient Descent:`** It changes weights by running on each of the single data point in the data set. It is computationally very expensive. It converges by approximating its way through each data point.
- **`Mini Batch Gradient Descent:`** It is an upgrade to SGD as instead of a single data point is utilizes batch sizes and compute the weights after training on a subset of shuffled dataset.

As we can see from the graphs The loss gets very low in cases of Mini Batch Gradient Descent. As the batch size increase the stability in the accuracy this is because as the batch size increases model gets more and more information to train on but in return convergence of the loss gets a hit as less and less points there are to calculate loss for.

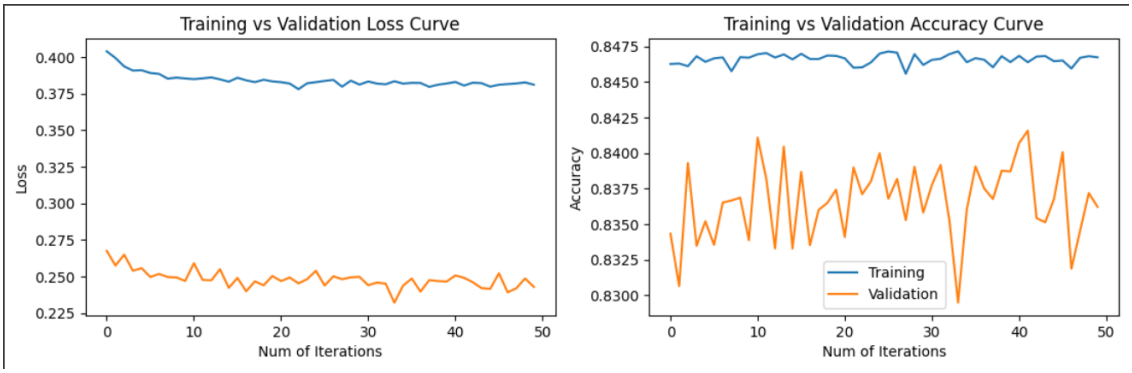

Figure 11: Stochastic GD (epochs 50)

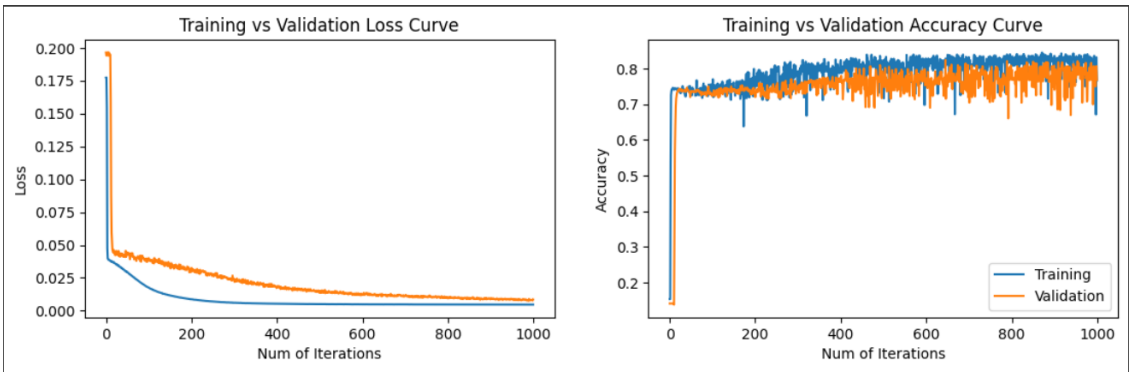

Figure 12: Mini Batch with Batch Size 100 (epochs 1000)

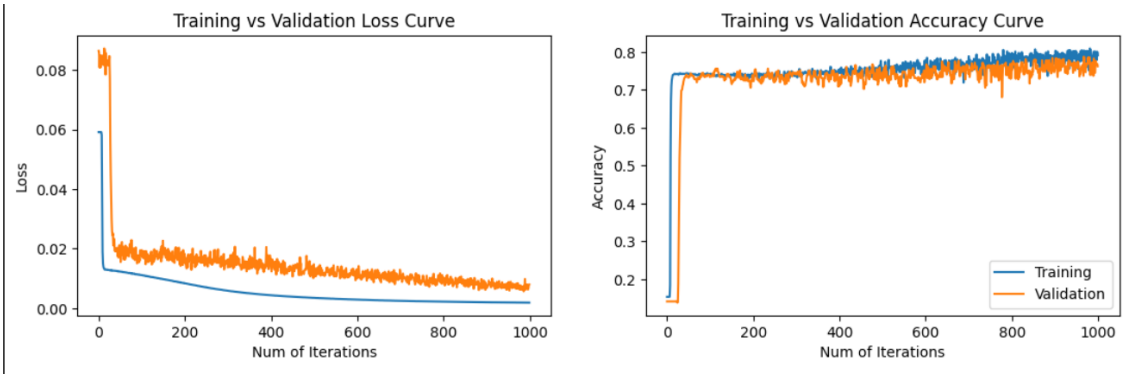

Figure 13: Mini Batch with Batch size 300 (epochs 1000)

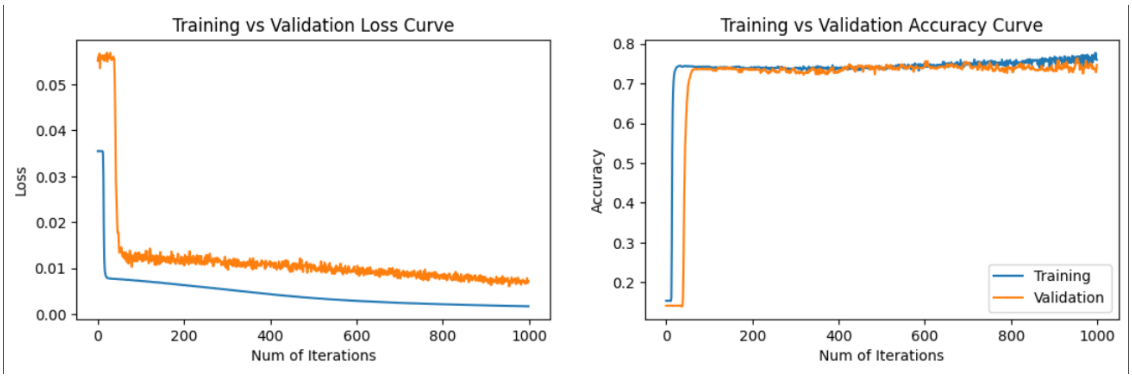

Figure 14: Mini Batch with Batch size 500 (epochs 1000)

Problem (e) (2 marks)

The Best Model will be Mini Batch Gradient Descent with batch size as 400. Performing K fold we will see that model has a stable accuracy across K folds and would not deviate much. K fold helps to test model's robustness and stability across different unseen data.

Problem (f) (3 marks)

For the stopping criteria can be added is of tolerance. This means that after a few iterations if the loss is not decreasing more than a precise tolerance. Then we stop training the model. In practice this can help in tackling overfitting. A graph without tolerance is provided as follows. It will be run for 50,000 iterations and we can see that it makes the loss almost equal to zero and overfits the data.

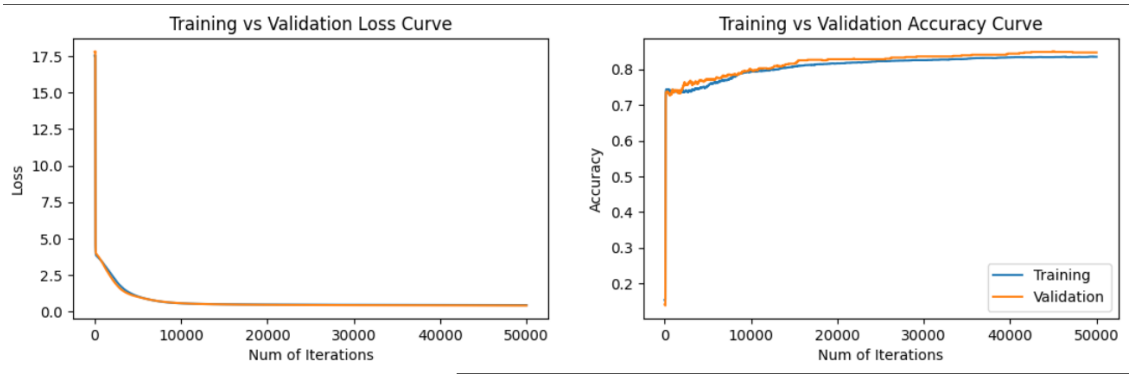

Figure 15: Overfitting without early stopping

As we can see loss is almost 0 and both validation and training graphs overlap. Thus causing overfitting.