Basic Implementation of Linear Regression

Stopping Criterion	Max Iter or Val Decrease Threshold	LR	Train MSE	Val MSE	Train MAE	Val MAE
maxit	1000	0.001	0.222	0.645	0.375	0.623
maxit	10 (Gradients Explode)	0.01	1.2e14	1.26e14	1.09e7	1.11e7
maxit	5 (Gradients Explode)	0.1	4.11e18	4.20e18	1.99e9	2.03e9
reltol	0.001	0.001	0.54	0.59	0.57	0.58
reltol	0.1 (Gradients Explode)	0.01	547.83	559.67	23.03	23.43
reltol	0.1 (Gradients Explode)	0.1	80199.80	81672.40	278.46	282.81

Observation: Increasing learning rate causes the gradients to explode because of large steps as seen in the following diagrams:

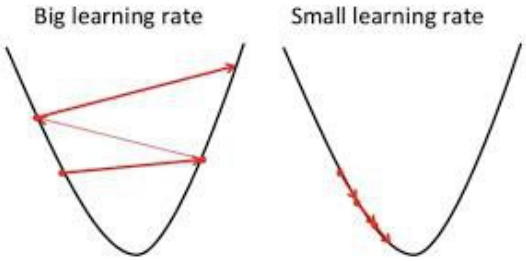

Figure 1: Effect of Large Learning Rate

Since this is a convex optimization problem, if the stopping criterion is based on the number of iterations, then the train loss can keep decreasing and tend to zero. In contrast, the validation loss can keep increasing due to overfitting; hence, relative tolerance is a better-stopping criterion. Relative Tolerance will stop the learning process as soon as the validation increases, which might not be desirable.

Ridge Regression

For this part I use *reltol* as the stopping criterion with a threshold of *0.001*. I fix the learning rate at *0.001*. Cost Function equation is as follows:

$$J(\theta) = \frac{1}{2N}(\sum_{i=1}^{N}(y^{(i)} - \sum_{j=0}^{d}(\theta_j x_j^{(i)}))^2) + \lambda(\sum_{j=1}^{d}(\theta_j^2)$$

Hence the update equation is as follows: (Note that bias term is not regularized)

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \sum_{j=0}^{d}(\theta_j x_j^{(i)}))x_0^{(i)}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \sum_{j=0}^{d}(\theta_j x_j^{(i)}))x_j^{(i)} + 2\lambda\theta_j, j > 0$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

λ	Train MSE	Val MSE	Train MAE	Val MAE
1	0.547	0.601	0.581	0.593
5	0.588	0.636	0.600	0.621
25	0.895	0.940	0.732	0.773

Observation: Increasing the λ parameter corresponds to increasing the bias of the model and reducing the variance. This can prevent overfitting and help in generalization. E_{in} will keep increasing as we increase λ (higher bias) but $E_{out} - E_{in}$ will keep decreasing due to better generalization capability of simpler models. There is thus a tradeoff if we increase λ too much ($\lambda = 25$) then we can perform poorly on the validation set since the E_{in} can be very high as seen even though $E_{out} - E_{in}$ is low. Slight regularization ($\lambda = 1$) performs better than simple linear regression though too much regularization ($\lambda = 25$) performs poorly compared to normal linear regression.

Scikit-Learn Implementation

I have used sci-kit-learn's default Linear regression implementation.

Train MSE	Val MSE	Train MAE	Val MAE
2.33e-29	1.05	3.85e-15	0.832

Observation: This implementation does not make use of the validation data and tends to overfit on the train dataset taking the loss to almost zero which is possible (convex optimization). Though the learnt model does not generalize well because of overfitting and performs poorly on the validation dataset as compared to 3.1 and 3.2

Feature Selection

For feature selection, I first used sci-kit-learn's modules to find a reduced subset of features. Then I used my ridge regression implementation with $\lambda = 5$, learning rate = 0.001 and *reltol* as the stopping criterion (threshold=1e-7). For Select from Model, I used sklearn's Linear Regression as the base estimator.

Selection Method	Train MSE	Val MSE	Train MAE	Val MAE
Select K Best	1.94	1.908	1.125	1.128
Select from Model	1.999	1.793	1.094	1.058

Observation: Using a subset of features corresponds to fitting simpler models (low variance and high bias). This helps the models generalize better to the validation set (lower $|E_{in} - E_{out}|$). But fitting to the training data also suffers (high E_{in}). In this case the combined effect is the model performs poorly compared to the baselines because we have used too few features.

Linear Classification

The loss function will be as follows:

$$J(\theta) = -(\sum_{x \in C_1} \log(\frac{e^{\theta_1^T x}}{1 + \sum_{r=1}^{8} e^{\theta_r^T x}}) + \sum_{x \in C_2} \log(\frac{e^{\theta_2^T x}}{1 + \sum_{r=1}^{8} e^{\theta_r^T x}}) + \cdots + \sum_{x \in C_9} \log(\frac{1}{1 + \sum_{r=1}^{8} e^{\theta_r^T x}}))$$

The update equations for each θ_r is as follows:

$$\frac{\partial J(\theta)}{\partial \theta_r} = (\sum_{x \in C_r} (\frac{e^{\theta_r^T x}}{1 + \sum_{r=1}^{8} e^{\theta_r^T x}} - 1)\boldsymbol{x} + \sum_{x \notin C_r} (\frac{e^{\theta_r^T x}}{1 + \sum_{r=1}^{8} e^{\theta_r^T x}} - 0)\boldsymbol{x}))$$

Gradient update:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

Train Accuracy	Val Accuracy	Train Log Loss	Val Log Loss
77.77%	52.38 %	0.828	1.158

Visualization

I have shifted the Validation plots upwards by 0.5 when the two lines overlap for ease of visualization.

Linear Regression Plots

Stopping Criterion: Max Iter

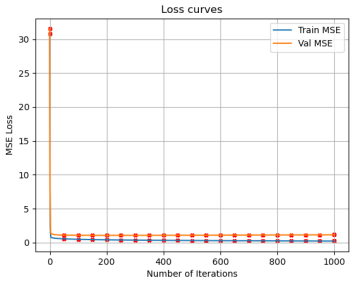

Figure 2: learning rate = 0.001

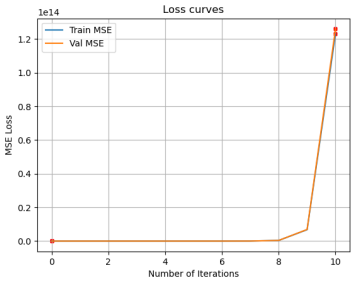

Figure 3: learning rate = 0.01

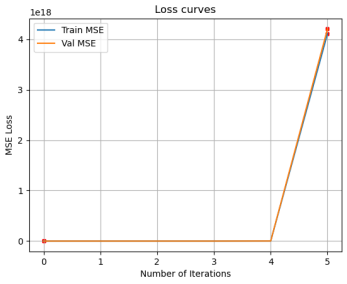

Figure 4: learning rate = 0.1

Stopping Criterion: Val Relative Tolerance

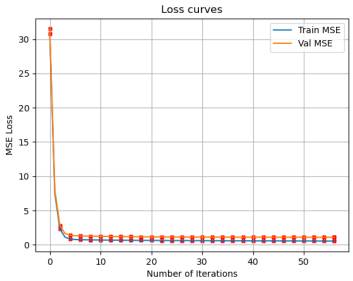

Figure 5: Tolerance = 0.001

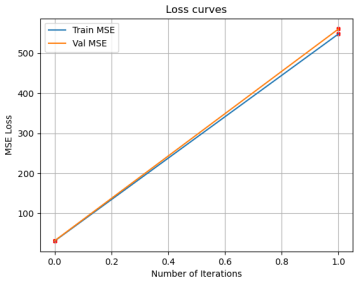

Figure 6: learning rate = 0.01

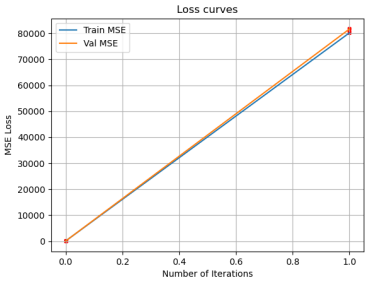

Figure 7: learning rate = 0.1

As explained for large learning rates the loss starts exploding. For smaller learning rates both train and validation loss decrease initially, but eventually the models starts overfitting and the validation loss keeps increasing while training loss keeps decreasing.

Ridge Regression Plots

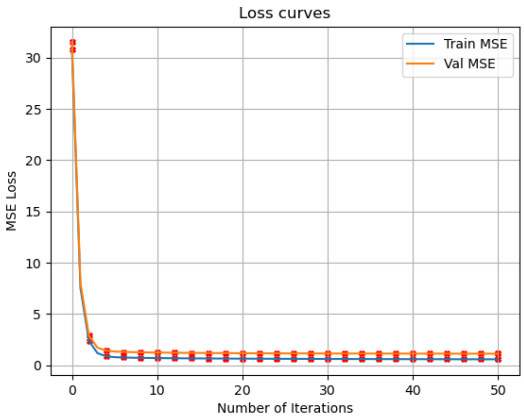

Figure 8: $\lambda = 5$

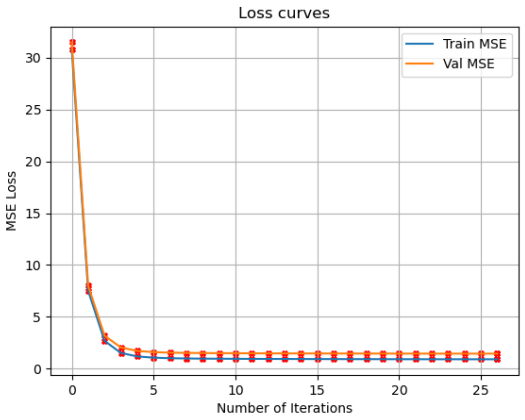

Figure 9: $\lambda = 25$

Feature Selection MSE plots

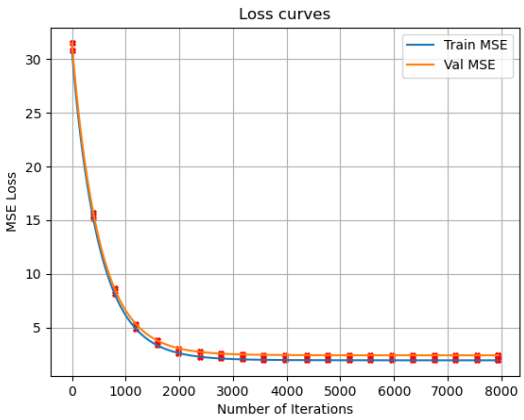

Figure 10: SelectKBest

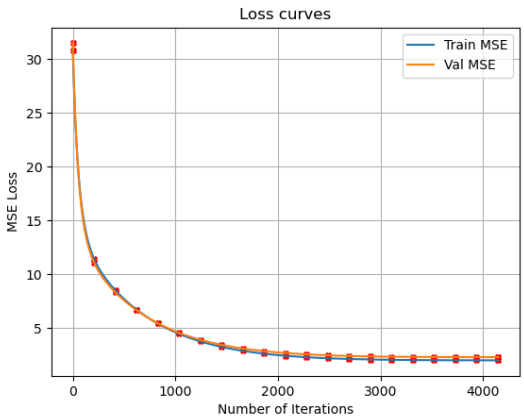

Figure 11: SelectFromModel

The loss functions are much smoother because we are using less number of features. Hence it takes the model more number of iterations to push the train loss to zero making the training process more stable.

Linear Classification Log Loss Plots

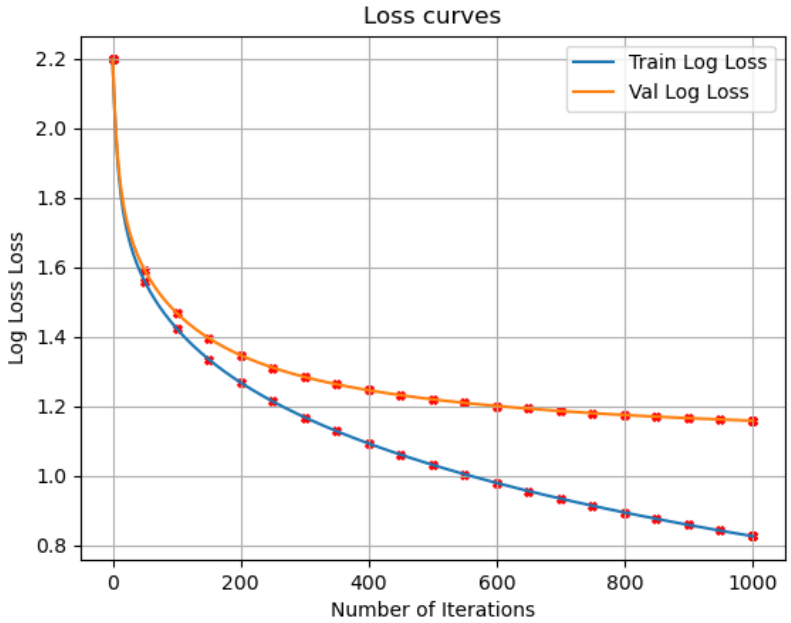

Figure 12: Log Loss Plots for Linear Classification

The loss function is again more smooth and the loss decreases more gradually because of the nature of the cost surface.

Data Normalization

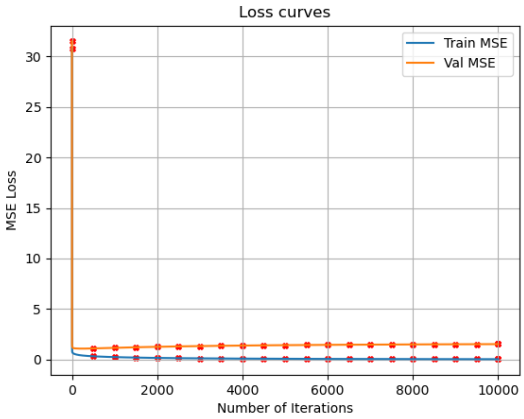

Figure 13: Without Normalization

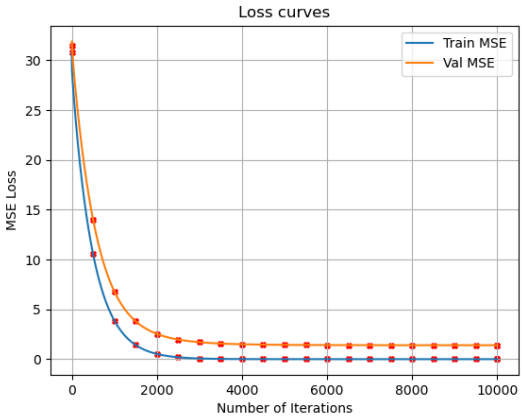

Figure 14: With Normalization

Normalizing data helps increasing training convergence, because it makes all features equally important (otherwise some weights are updated a lot and others are not updated much). For a large number of iterations normalizing the data thus ensures a greater decrease in train loss (at the end of 1e4 iterations, train loss was lower with normalized data). Although in the earlier iterations updating just a few number of weights(corresponding to features with larger values) can be updated can decrease the loss suddenly (steeper loss curves), normalizing the features corresponds to much smoother loss curves and robust training.

Size of Data

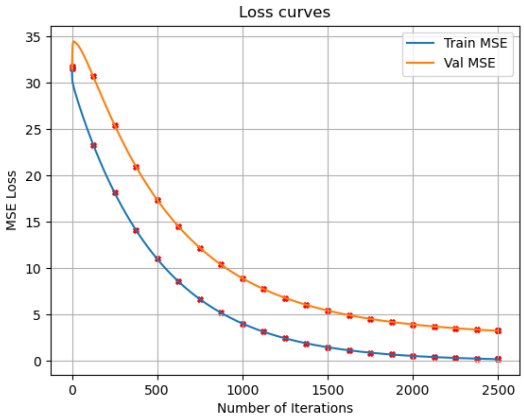

Figure 15: Data Fraction = 0.25

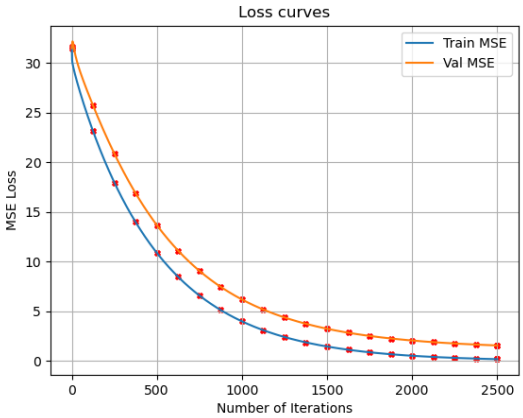

Figure 16: Data Fraction = 0.5

As we decrease the size of the data it becomes easier to fit to the train set (fewer points) and thus train MSE keeps decreasing. But this causes poor generalization results to the validation set ($E_{out} - E_{in} \propto \sqrt{\frac{d}{N}}$) and hence validation MSE keeps increasing. Training converges faster with smaller data-sets but validation loss also converges faster and hence overfitting starts quicker with smaller datasets.

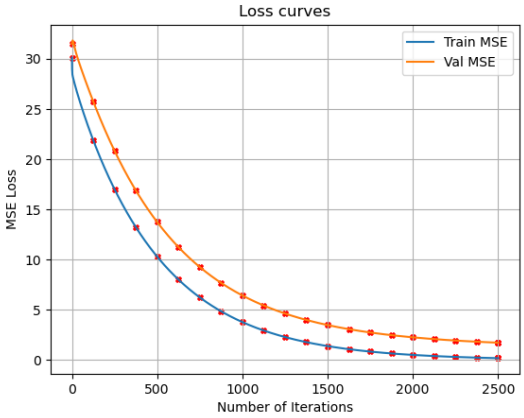

Figure 17: Data Fraction = 0.75

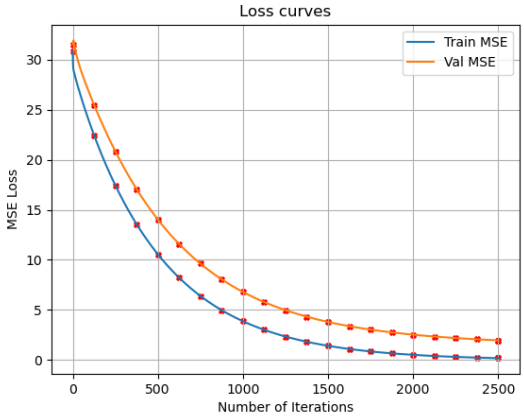

Figure 18: Data Fraction = 1

Train Data Fraction	Train MSE	Val MSE	Train MAE	Val MAE
0.25	0.200	2.76	0.448	1.201
0.5	0.198	1.079	0.445	0.870
0.75	0.1875	1.229	0.433	0.9020
1	0.192	1.470	0.437	0.989

Dividing Training Data

The way I interpret this experiment is that we divide the training data into two parts. Train 2 models using these two sets and compare how much their predictions differ on some other set (validation). We repeat the same for linear regression and ridge regression. Ridge Regression has a lower variance; hence, for different training data sets, its predictions have lower variance on some other test set (MAE of the two predictions is lower).

Model	MAE of the two predictions on validation set
Linear Regression	0.40
Ridge Regression	0.25

Class wise MSE plots

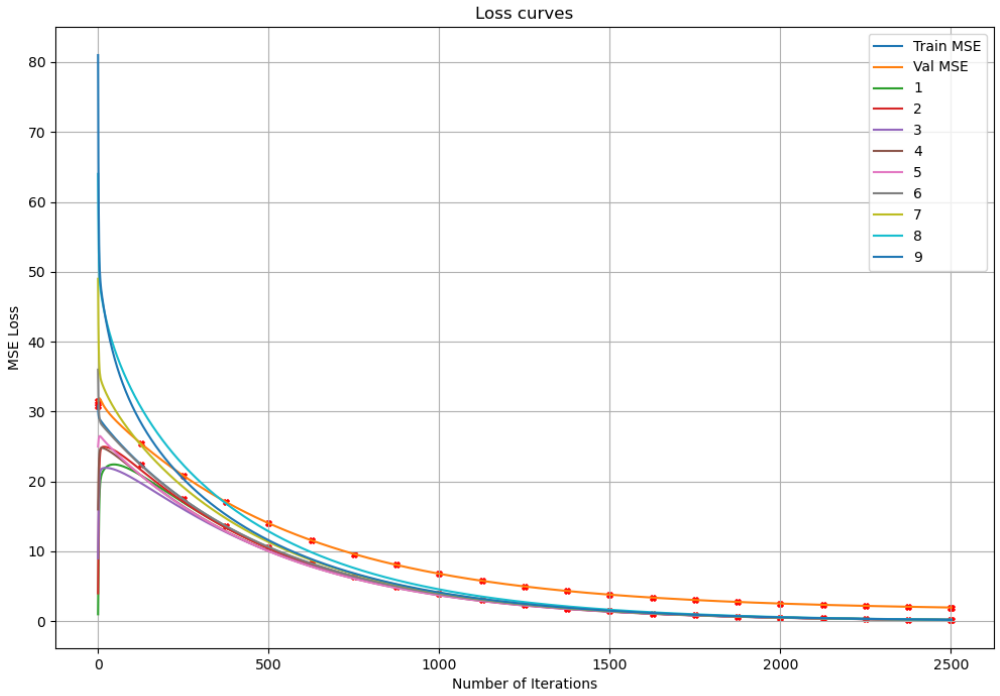

Figure 19: Class Wise MSE plots

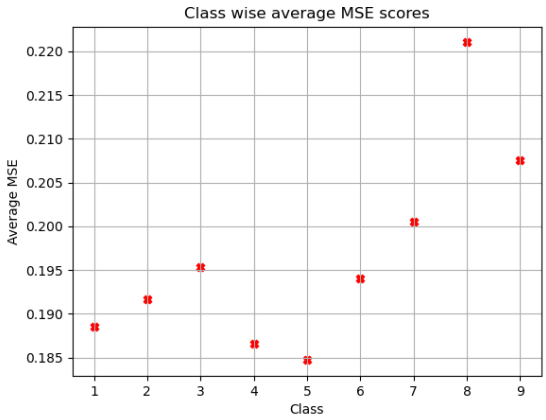

Figure 20: Classwise Average Loss

Figure 21: Classwise Total Loss

The dataset is heavily skewed, with some classes dominating the loss function classes 5 and 6 have (41/144 and 36/144) points respectively and hence dominate the loss function. The gradient descent algorithm tries to decrease their mean loss the most (since it corresponds to largest decrease in the total MSE). Although, the average MSE for all classes is almost similar. The total loss on the other hand remains the largest for classes 5 and 6 which have the largest number of points since it becomes difficult to fit all the points perfectly.

Feature Selection and Number of Features

For me, SelectFromModel worked better.

Number Features	Train MSE	Val MSE	Train MAE	Val MAE
10	2.28	1.99	1.18	1.145
100	1.03	1.033	0.820	0.854
1000	0.477	0.903	0.549	0.778
2048	0.477	0.903	0.549	0.778

Note that the maximum number of features selected by the selector is 835, and hence beyond that, values remain the same. The generalization $|E_{in} - E_{out}|$ is the best when we use a small number of features due to a smaller V_{dc} and hence better generalization capability. Although fitting the data becomes difficult with small number of features leading to a large E_{in}. Hence there is a tradeoff and increasing the features upto 1000 leads to best learning. Beyond that taking more features does not help since the features don't have additional predictive power and only lead to poor generalization because they start fitting to the random noise in the dataset.

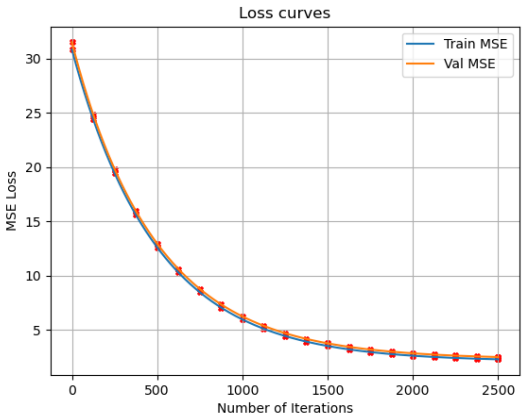

Figure 22: Number of Features = 10

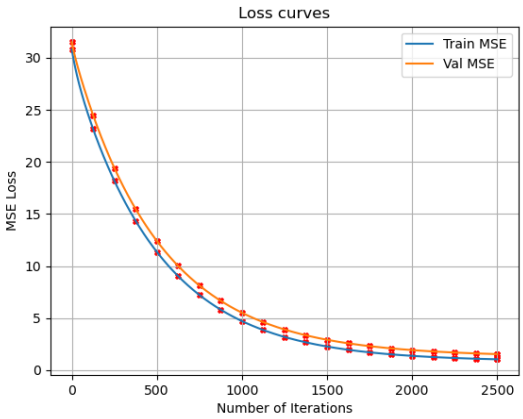

Figure 23: Number of Features = 100

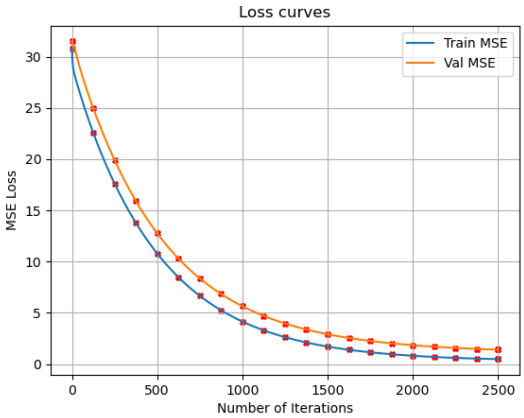

Figure 24: Number of Feature = 1000

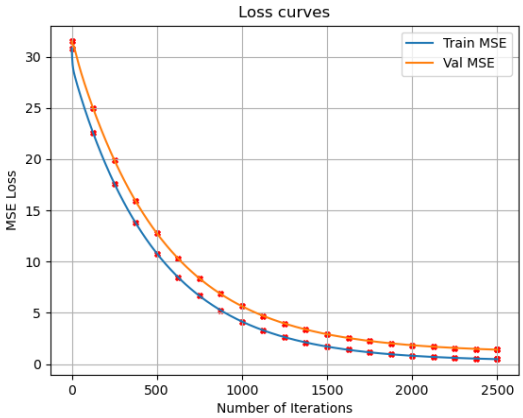

Figure 25: Number of Features = 2048

Figure 26: Train and Val loss vs Number of Features

Generalization Analysis

For this part, I have used 1000 iterations with a learning rate of 0.001 and my implementation of linear regression.

Number of Features	**Train MSE(E_{in})**	**Test MSE(E_{out})**	$\lvert E_{in} - E_{out} \rvert$
2	0.875	0.965	0.090
5	0.850	0.964	0.114
10	1.176	0.858	0.32
100	2.456	3.917	1.461

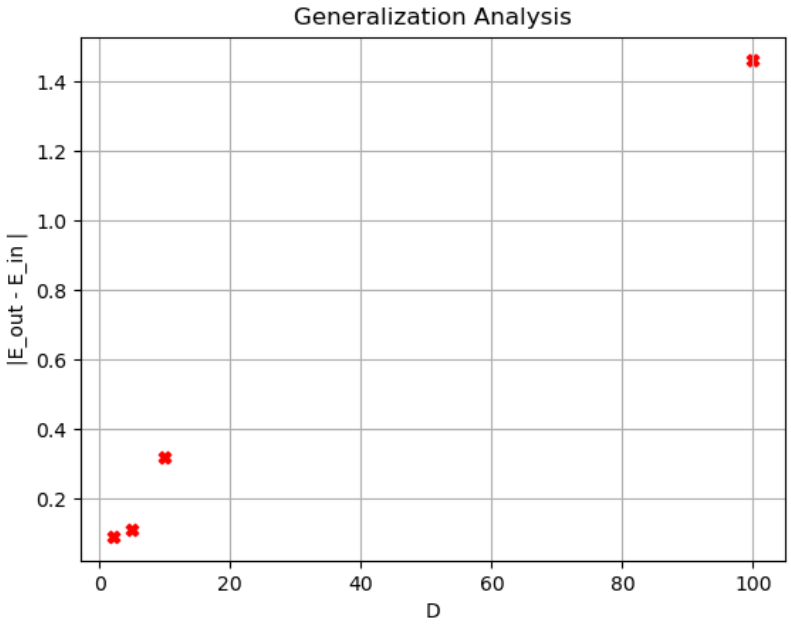

Figure 27: Generalization Analysis

For all these datasets the number of training points is the same and hence the generalization capability is decided by the number of dimensions we have to fit the data. We can see that as the number of dimensions increase the generalization capability of the model in terms $|E_{in} - E_{out}|$ keeps getting worse and roughly follows $\sqrt{\frac{d}{N}}$.

Bonus (One Vs All)

I have learnt 9 classifiers. The update equations for each classifier are the same as that of logistic regression, which is the same as that of Linear regression. The results for the same are as follows:

Train Log Loss	Val Log Loss	Train Accuracy	Val Accuracy
0.1233	0.2205	88.88 %	57.14 %

Training Loss Curve:

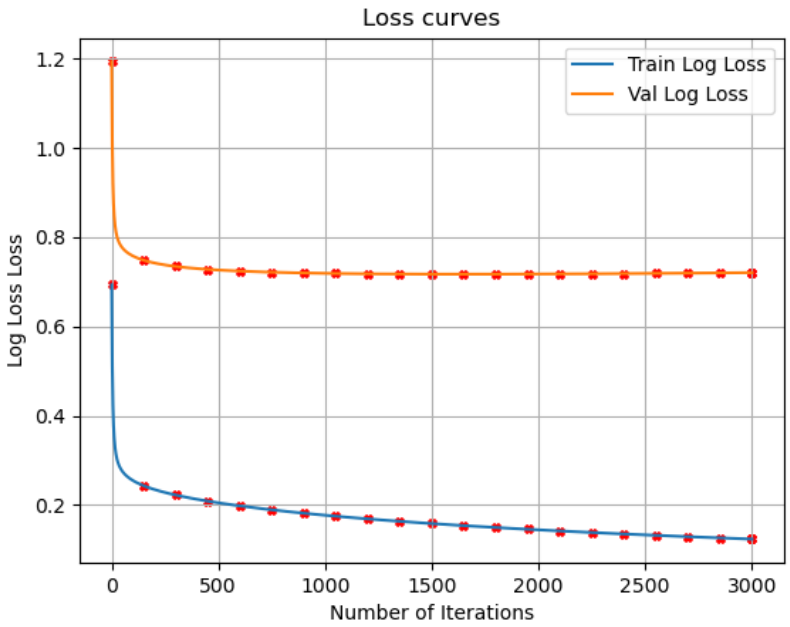

Figure 28: One vs All loss curve