DASC 6363
Machine Learning

by

Alex John Dillhoff

UNIVERSITY OF TEXAS ARLINGTON

Assignment I
Linear Regression

Spring 2023

By:
Nikhil Das Karavatt
1002085391
nxk5391@mavs.uta.edu

Table of Contents

Overview:

This assignment covers Linear Regression and Gradient Descent. Linear Regression is a method of predicting real values given some input. Gradient descent is the algorithm we will use to optimize our linear model.

Task 1: Creating LinearRegression class

I made a fit, predict, score(mean square error), and rmse(root mean square error) functions in order to evaluate data for the upcoming tasks.

1.1 Fit function

The fit method accepts 7 parameters:

1. The input data (X)
2. the target values (y)
3. batch_size, int - The size of each batch during training, default to 32
4. regularization, int - The factor of L2 regularization to add, default to 0
5. max_epochs, int - The maximum number of times the model should train through the entire training set, default to 100
6. patience, int - The number of epochs to wait for the validation set to decrease, default to 3
7. learning rate – hyperparameter for gradient descent, default to .01

I have initialized the bias to zero and weights to small random numbers drawn from a gaussian distribution with mean 0 and a small standard deviation.

I have used gradient descent to optimise the model parameter using mean square error as the loss function. Early stopping is used by using patience. I have set aside 10% of the training data as a validation set, in which at each step I have evaluated the loss and if the loss on validation set increase consecutively for three times, we stop the training and save the parameters to set it as model parameters.

1.2 Predict Function

The predict method accepts 1 parameter, which is input data (X).

$$\mathbf{y} = XW + \mathbf{b} \in \mathbb{R}^{n \times m}$$

We use the above formula to predict the target value.

1.3 Score Function

The score method accepts 2 parameters, which is input data (X) and the target values (y).

$$\text{MSE} = \frac{1}{nm} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

We use the above mean square error formula to calculate the loss.

1.4 RMSE Function

The RMSE function is similar to score function with similar parameters.

As the name suggests root mean square error, the only difference between score and rmse function is that the rmse is square root value of the score calculated using the score function.

Task 2: Regression with Single Output

We are using Iris dataset in order to perform regression with single output.
Each sample in the Iris dataset has 4 features:

- sepal length
- sepal width
- petal length
- petal width

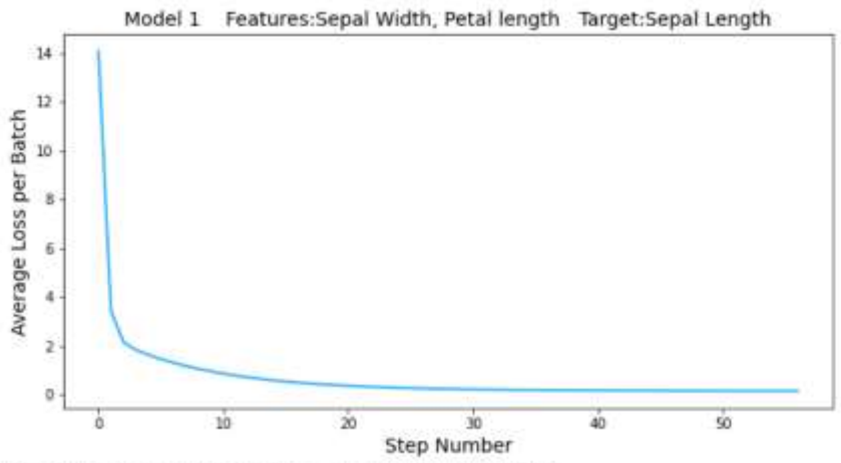
In order to understand which features is most predictive of another, I have fixed one target for all my four different models. I have kept my target as sepal length.

I have split the dataset by keeping 10% of the dataset for test set while remaining 90% for training set and also made sure that there is even split of each class with the help of stratify parameter. Below are the four different models that I have made by using different input features.

Model 1:

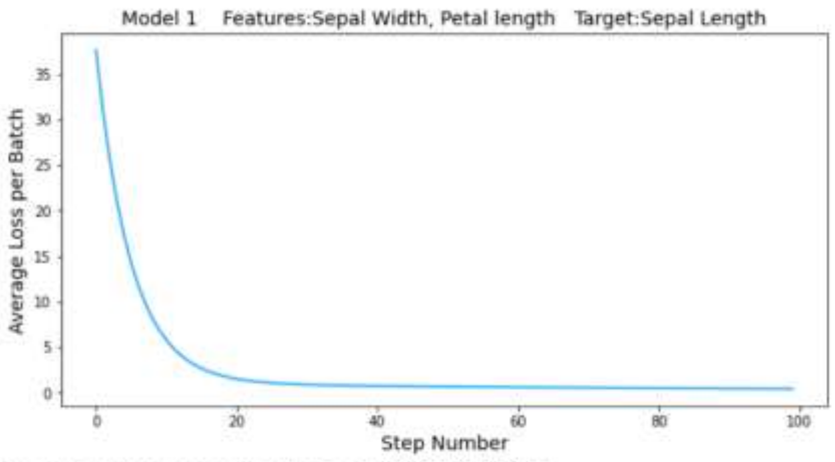
Input Features: Sepal Width, and Petal Length
Target: Sepal Length

α = 0.01 and λ = 0 (α=learning rate and λ= regression coefficient)



```
The mean square error of model 1: 0.17487883383172395
```

α = 0.001 and λ = 0

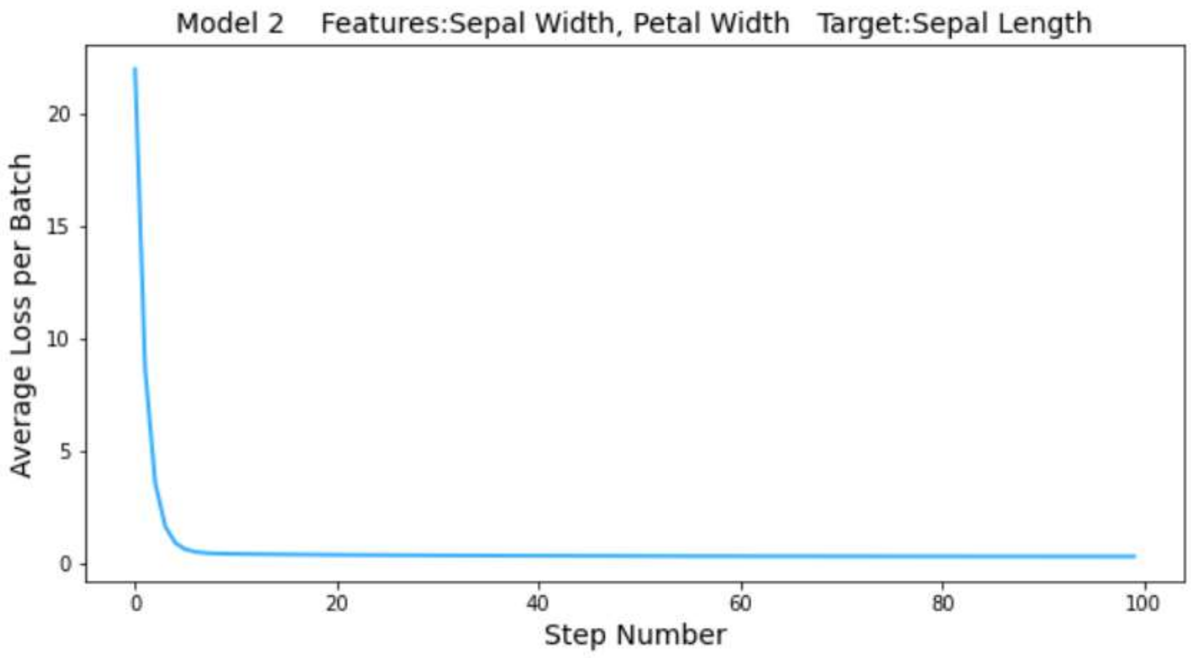


```
The mean square error of model 1: 0.4724999337953034
```

Model 2:

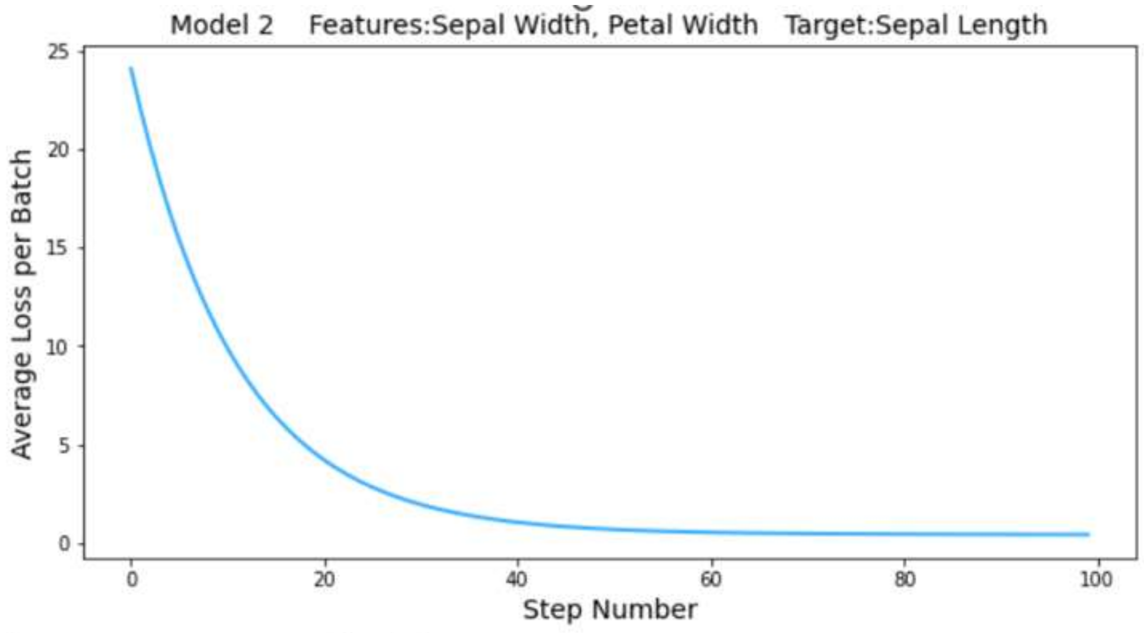
Input Features: Sepal Width, and Petal Width
Target: Sepal Length

α = 0.01 and λ = 0



```
The mean square error of model 2: 0.38540760813930347
```

α = 0.001 and λ = 0

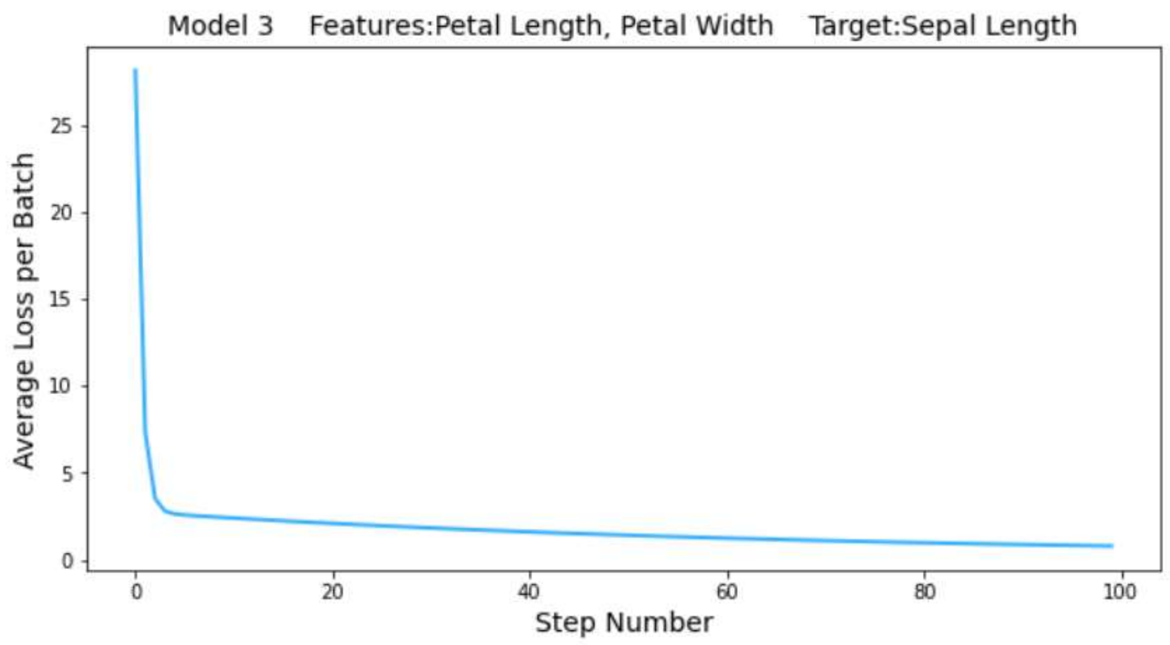


```
The mean square error of model 2: 0.5595109726839085
```

Model 3:

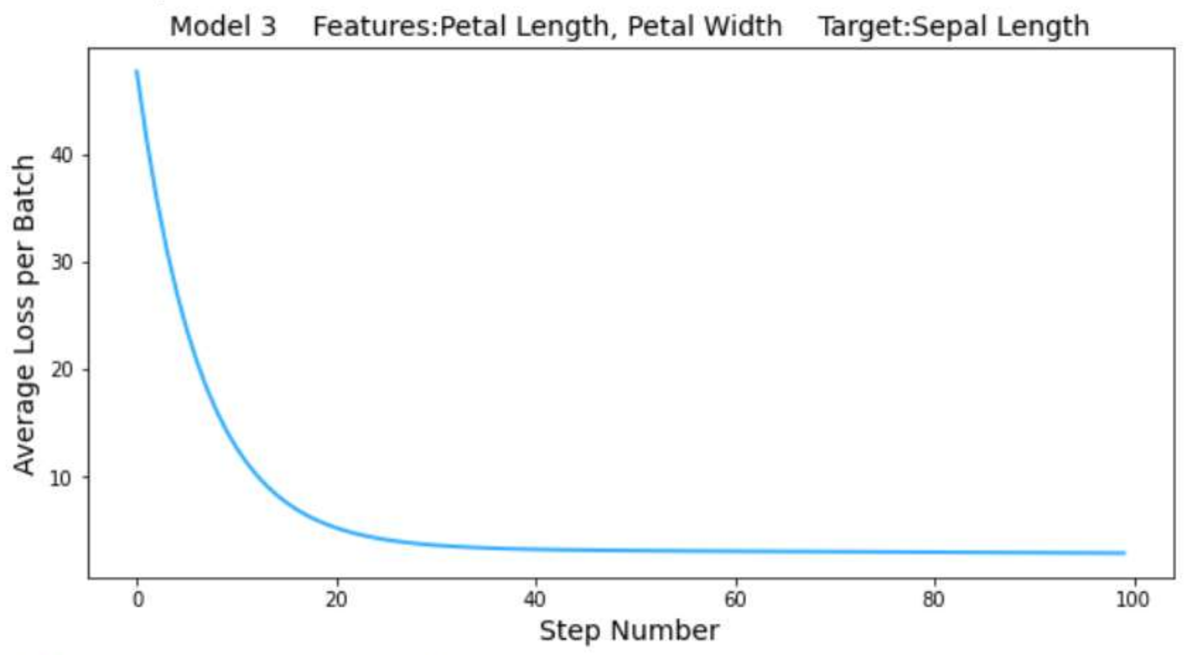
Input Features: Petal Length, and Petal Width
Target: Sepal Length

α = 0.01 and λ = 0



```
The mean square error of model 3: 0.9268143562036268
```

α = 0.001 and λ = 0

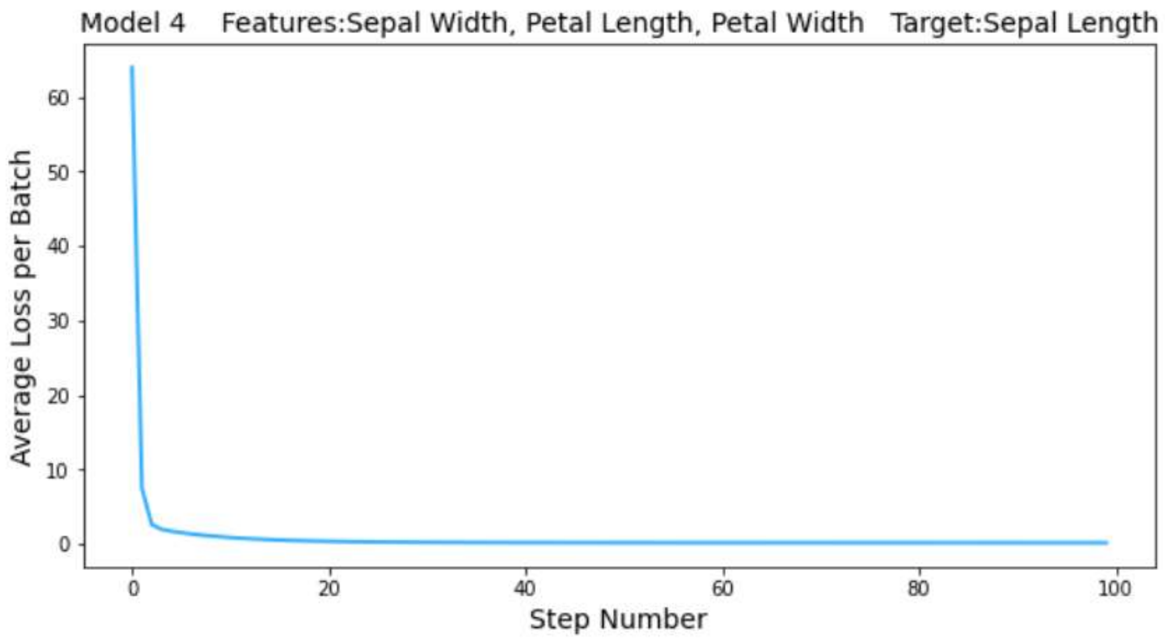


```
The mean square error of model 3: 3.280697602924927
```

Model 4:

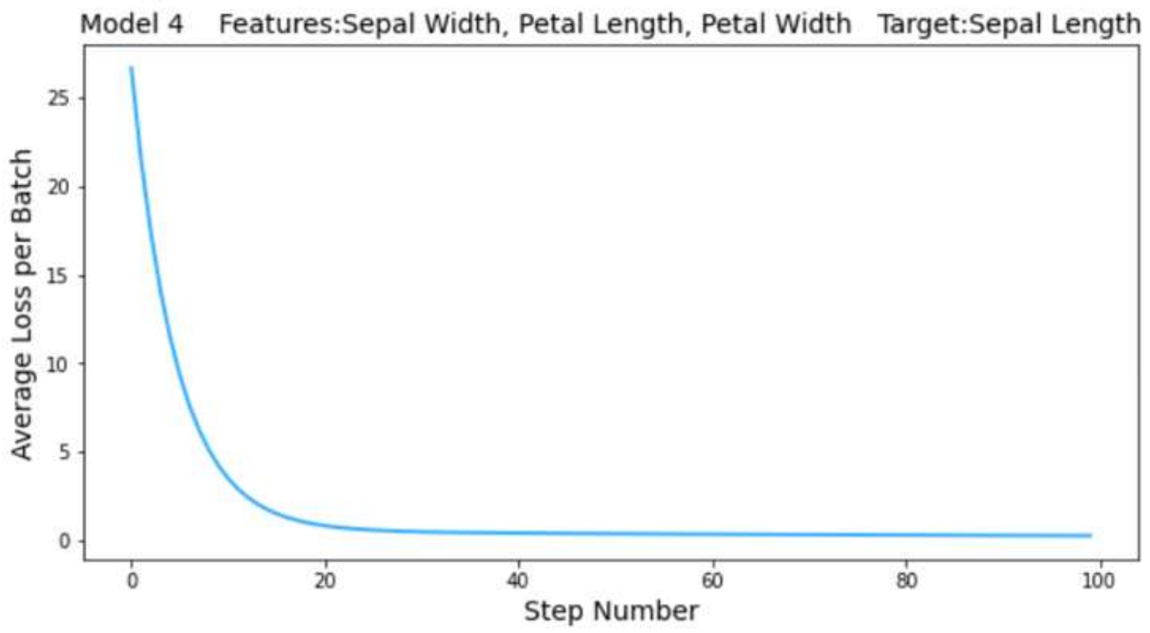
Input Features: Sepal Width, Petal Length, and Petal Width
Target: Sepal Length

α = 0.01 and λ = 0



```
The mean square error of model 4: 0.20603489367947922
```

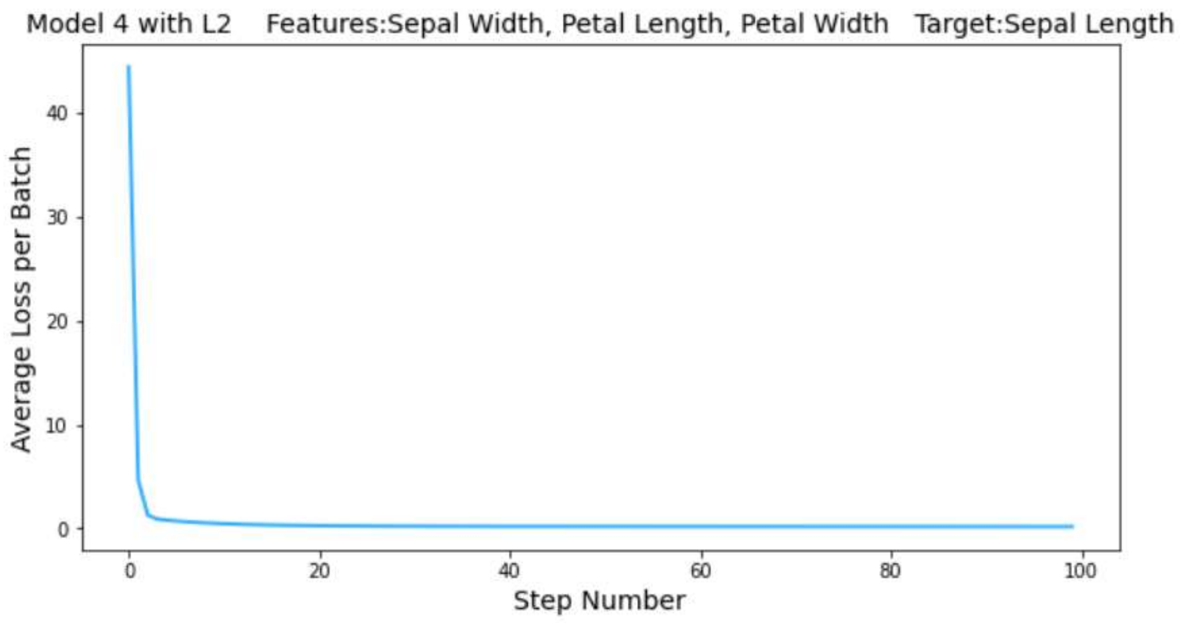
α = 0.001 and λ = 0



```
The mean square error of model 4: 0.2870051965346632
```

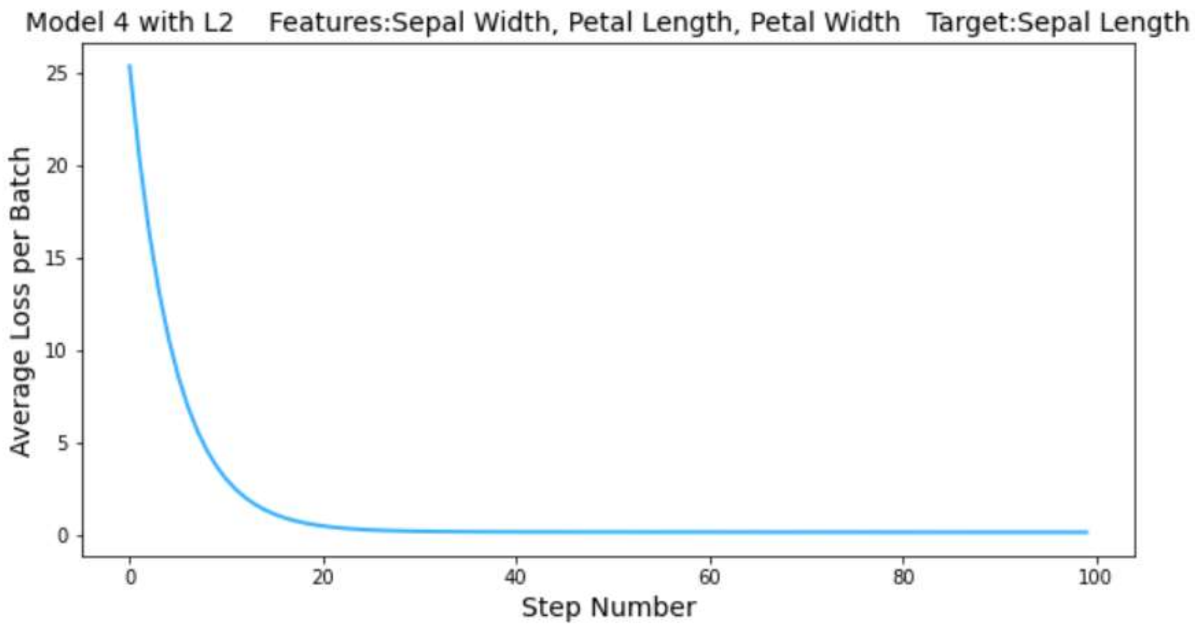
Model 4 with Ridge Regularization:

α = 0.01 and λ = 0.001



```
The mean square error of model 4 with regularization: 0.20386457422063362
```

α = 0.001 and λ = 0.001



```
The mean square error of model 4 with regularization: 0.14159220013852283
```

Note: The change in learning rate is creating changes in slope of the line.

Weight Matrix Comparison of Model 4 with and without regularization

Without regularization:

α = 0.01 and λ = 0

Weight Matrix:

```
[[1.05129045]
 [0.38290833]
 [0.34585329]]
```

Bias Matrix:

```
[[0.70240028]]
```

α = 0.001 and λ = 0

Weight Matrix:

```
[[ 0.90611172]
 [ 0.75080838]
 [-0.10803741]]
```

Bias Matrix:

```
[[0.23638609]]
```

With regularization:

α = 0.01 and λ = 0.001

Weight Matrix:

```
[[1.08537724]
 [0.4126927 ]
 [0.28638637]]
```

Bias Matrix:

```
[[0.55566589]]
```

α = 0.001 and λ = 0.001

Weight Matrix:

```
[[ 1.00799147]
 [ 0.90884774]
 [-0.77260595]]
```

Bias Matrix:

```
[[0.19938918]]
```

Difference between regularised and unregularised models:

Learning Rate	Weights and Bias	Non-Regularized λ = 0 (a)	Regularized λ = 0.001 (b)	Difference (a-b)
α = 0.01	w1	1.05129045	1.08537724	-0.034087
	w2	0.38290833	0.4126927	-0.029784
	w3	0.34585329	0.28638637	0.059467
	b	0.70240028	0.55566589	0.146734
α = 0.001	w1	0.90611172	1.00799147	-0.10188
	w2	0.75080838	0.90884774	-0.158039
	w3	-0.10803741	-0.77260595	0.664569
	b	0.23638609	0.19938918	0.036997

We can see that regularization significantly reduces the variance of the model without substantial increase in its bias. Tuning of λ controls the impact on bias and variance.

Conclusion

From the various models and changes in hyperparameters, I can conclude that in order to find the sepal length (target), the three Input Features: Sepal Width, Petal Length, and Petal Width, helps to train the model accurately with Ridge regularization. Also adjusting the hyperparameter plays a crucial role in making the best model as extreme changes in hyperparameters can lead to underfitting or overfitting. The best model to find sepal length is by using all three remaining features and training the model with Ridge regularization by keeping the following hyperparameters: α = 0.001 and λ = 0.001.

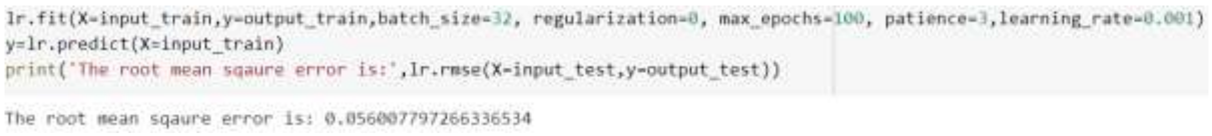
Task 3: Regression with Multiple Outputs

Preparation of the data:

As we are using the same class which we made in task one and used for task 2, I made changes to the shape of the features data to make sure it fits in the various function that I made. The matrix inside the matrix (1261, 36, 48) was made to one matrix (1261, 1728), thereby making this data available to be used in our function to give an output data of (1261, 36) which is a 36-dimensional vector representing the relative traffic congestion of each location.

Training the data:

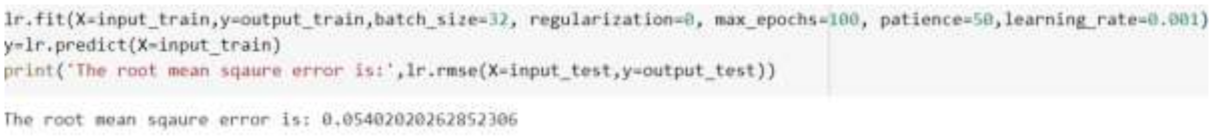
I changed various parameters and hyperparameters to get a better root mean square error



```
lr.fit(X=input_train,y=output_train,batch_size=32, regularization=0, max_epochs=100, patience=3,learning_rate=0.001)
y=lr.predict(X=input_train)
print('The root mean sqaure error is:',lr.rmse(X=input_test,y=output_test))

The root mean sqaure error is: 0.056007797266336534
```

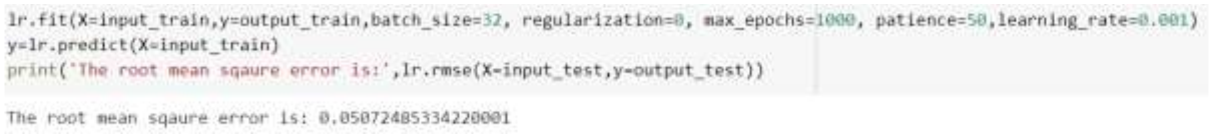
First, I started with keeping the α = 0.001 which is the learning rate and no regularization and I see a good root mean square error value, but changing the hyperparameters and parameters could help me to reduce this value. So next I changed the patience value to 50, as I think by keeping it to 3 the model was getting restricted to find a better local minima of the gradient descent or even a possibility to find a global minima which may help to optimize our weights and bias magnitude for a really good model



```
lr.fit(X=input_train,y=output_train,batch_size=32, regularization=0, max_epochs=100, patience=50,learning_rate=0.001)
y=lr.predict(X=input_train)
print('The root mean sqaure error is:',lr.rmse(X=input_test,y=output_test))

The root mean sqaure error is: 0.05402020262852306
```

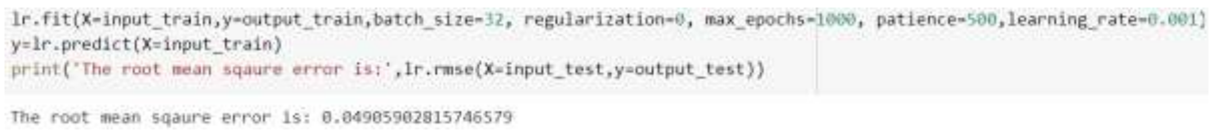
Then I changed the max epoch to 1000, so the model can learn about the data really well and perform even better cause of extensive training.



```
lr.fit(X=input_train,y=output_train,batch_size=32, regularization=0, max_epochs=1000, patience=50,learning_rate=0.001)
y=lr.predict(X=input_train)
print('The root mean sqaure error is:',lr.rmse(X=input_test,y=output_test))

The root mean sqaure error is: 0.05072485334220001
```

Now again by increasing the patience I was able to reduce the root mean square error.

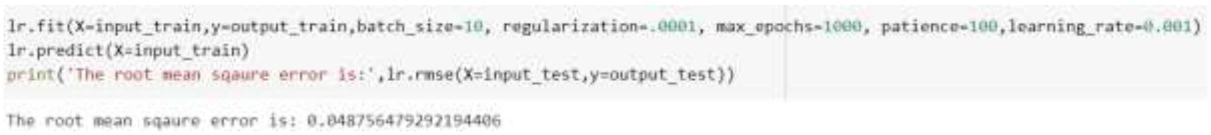


```
lr.fit(X=input_train,y=output_train,batch_size=32, regularization=0, max_epochs=1000, patience=500,learning_rate=0.001)
y=lr.predict(X=input_train)
print('The root mean sqaure error is:',lr.rmse(X=input_test,y=output_test))

The root mean sqaure error is: 0.04905902815746579
```

Now I have introduced regularization, to make sure that data doesn't overfit or underfit as we have trained the data extensively. The regularization helps to reduce the magnitudes of the weights which helps to reduce the variance thereby preventing overfitting, but we need to choose the α value carefully as the model starts losing important properties, giving rise to bias in the model and thus underfitting.

The below is the least error that I have got, but on an average, I have been getting around .050 root mean square error.



```
lr.fit(X=input_train,y=output_train,batch_size=10, regularization=.0001, max_epochs=1000, patience=100,learning_rate=0.001)
lr.predict(X=input_train)
print('The root mean sqaure error is:',lr.rmse(X=input_test,y=output_test))

The root mean sqaure error is: 0.048756479292194406
```

Test:

As seen in the above the root mean square error is 0.0487. This close to 0.0024 mean square error, which is a great score to define it as a good model.