LINEAR REGRESSION MANUAL IMPLEMENTATION

Submitted by

Nelson Joseph (1002050500)

in partial fulfilment for the award of the degree of

M.S. IN DATA SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

CSE 6363 – MACHINE LEARNING
INSTRUCTOR: Prof. Alex Jon Dillhoff

10 FEBRUARY 2023

Honor Code:
I Nelson Joseph did not give or receive any assistance on this project, and the report submitted is wholly on our own.

Table of Contents

Contents

Introduction

The primary objective was to create a Linear regression class. The class includes different methods as functions. The NumPy library was utilized in the implementation of each function. A class which is inherited from the Linear regression which has the regularization feature as well is also implemented. The data taken into consideration is iris dataset which consist of 4 different parameters as features. 4 different combinations of models are made and compared with each other to evaluate the performance. The model with the lowest performance is further assessed both with and without regularization. The linear regression class is also used here to predict the output in regression with multiple output as well.

File Description

File Type	File Name
.py Files	Linear_regression.py
Jupyter Notebook	Linear_regression.ipynb
. docx	Assignment_Report_Nelson_Joseph.docx

Table 1.1 displays information about the various files used in the overall analysis.

Pre-Processing

- The iris data was spitted into 4 different combinations.
- For determining the traffic volume, the traffic flow forecasting data was reshaped from (1261, 36, 48) to (1261, 1728).
- Models were created on each of the 4 combinations.

Implementation of Linear Regression Class

The Linear regression class includes 4 methods inside it those are fit (), predict (), score (), and. RMSE()

Fit method

The fit method should accept 6 parameters:

1) Input data
2) Target values
3) batch size, int - The size of each batch during training
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
6) patience, int - The number of epochs to wait for the validation set to decrease.
7) Learning rate, float – The rate at which the model learns
8) Loss_per_cycle, list – The list in which the averaged loss per batch is stored.
9) Weights – weights to be updated.
10) Bias – The bias which is to be updated.

Predict Method

The Predict method accepts 1 parameter.

1) Input test data

The weights and bias obtained from the fit method is used to predict on the test data.

Score Method

The score method accepts 2 parameters:

1) Input test data
2) Target test data

The Input test data is first used to predict the outputs. The results are then compared with the Target test data to obtain the Mean Square Error (MSE).

RMSE Method

The RMSE method accepts 2 parameters:

1) Y_pred – The prediction results from the Input test data
2) Target test data

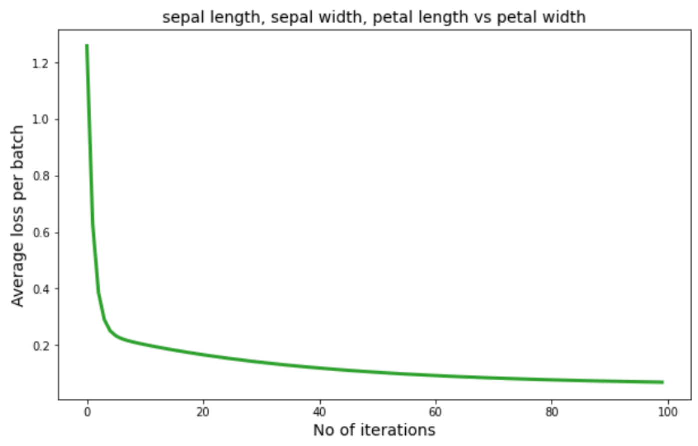
Here the Root Mean Square Error (RMSE) is calculated by taking the root of mean square error.

Analysis and Visualization on Iris Dataset

Training

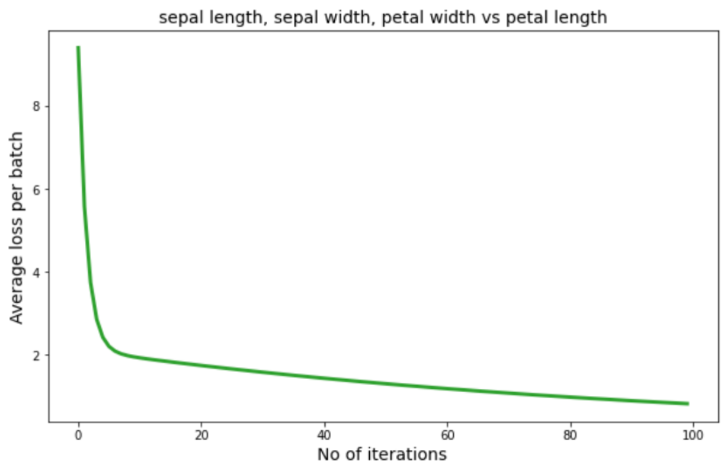
The model is trained on the given parameters that includes a batch size = 32, max_epochs = 100, and patience = 3. The learning rate and regularization parameters are taken manually.

First Model (sepal length, sepal width, petal length vs petal width)



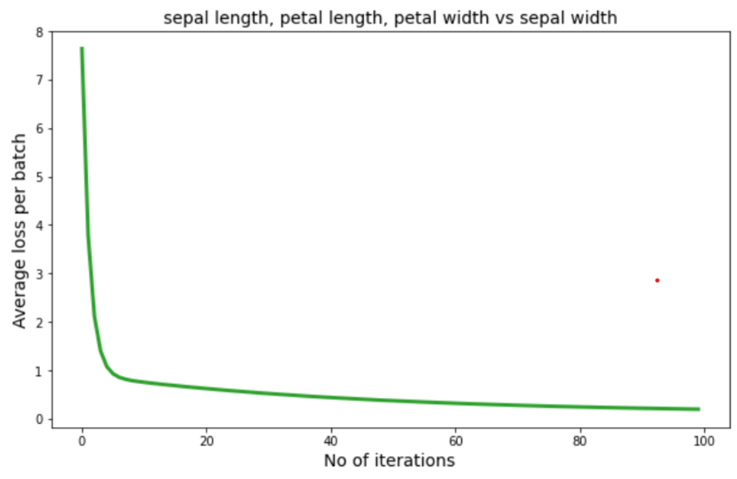
```
Weights =  [[0.08131942]
 [0.04003096]
 [0.32979491]]
MSE = 11.503328260897764
```

Second Model (sepal length, sepal width, petal width vs petal length)



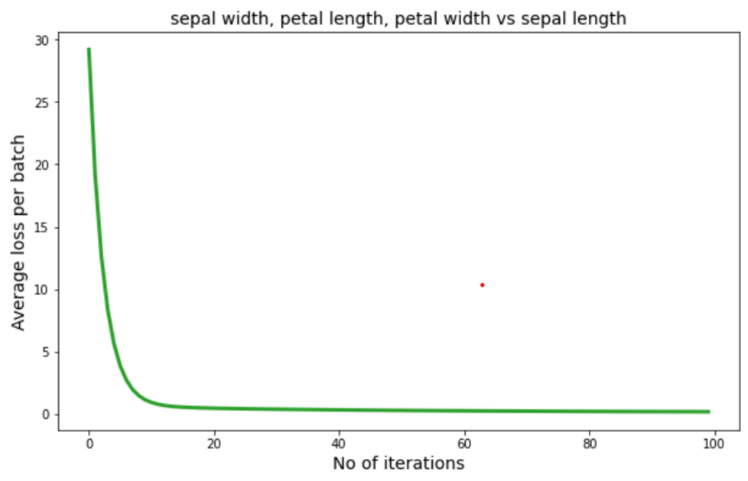
```
Weights =  [[ 0.49872485]
 [-0.20239839]
 [ 0.69012694]]
MSE = 4.46014357431666
```

Third Model (sepal length, petal length, petal width vs sepal width)



```
Weights =  [[ 0.53204194]
 [-0.13260097]
 [-0.10552455]]
MSE = 5.308233408473235
```

Fourth Model (sepal length, sepal width, petal width vs petal length)



```
Weights =  [[0.8324532 ]
 [0.56855992]
 [0.07387295]]
MSE = 11.612592013918821
```

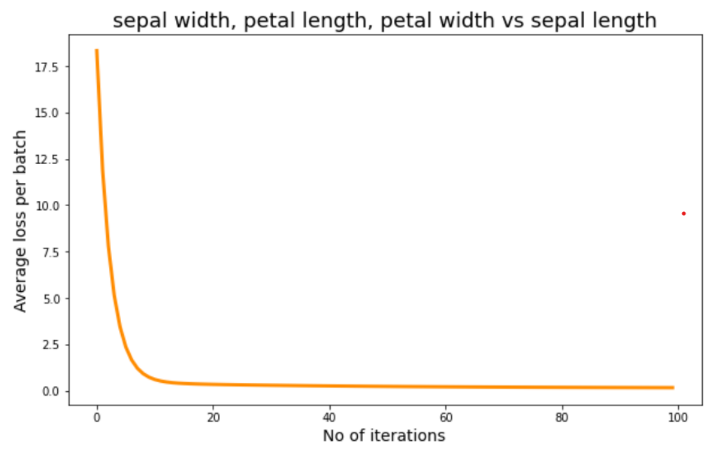
Summarized Results of all Models

Independent Features	Dependent feature	MSE
sepal length, sepal width, petal length	petal width	11.503328260897764
sepal length, sepal width, petal width	petal length	4.46014357431666
sepal length, petal length, petal width	sepal width	5.308233408473235
sepal width, petal length, petal width	sepal length	11.612592013918821

The results suggest that the combination of sepal length, sepal width, and petal width as independent features and petal length as the dependent feature had the lowest mean squared error (MSE). The second lowest MSE was observed when using sepal length as the dependent feature. The combination of sepal length, sepal width, and petal length as independent features and petal width as the dependent feature had the highest MSE. The combination of sepal width, petal length, and petal width as independent features and sepal length as the dependent feature had the second highest MSE.

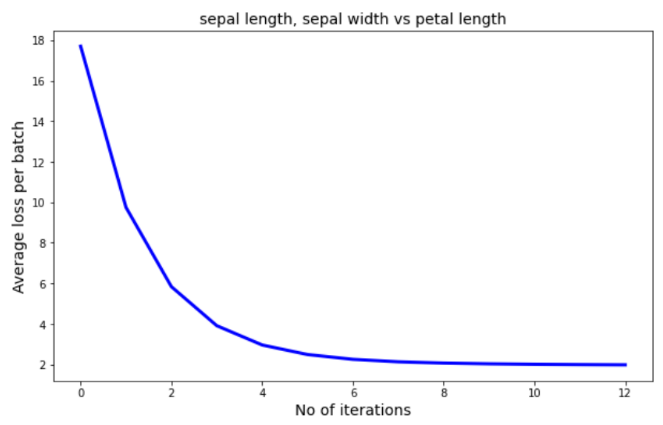
The Regularization is applied over the model with the highest MSE. The MSE was reduced to 11.529922888355234.

Weights	Initial weights	After Regularization	Initial MSE	Final MSE
W0	0.8324532	0.6884453	11.612592013918821	11.529922888355234
W1	0.56855992	0.53363156		
W2	0.07387295	0.1062753		

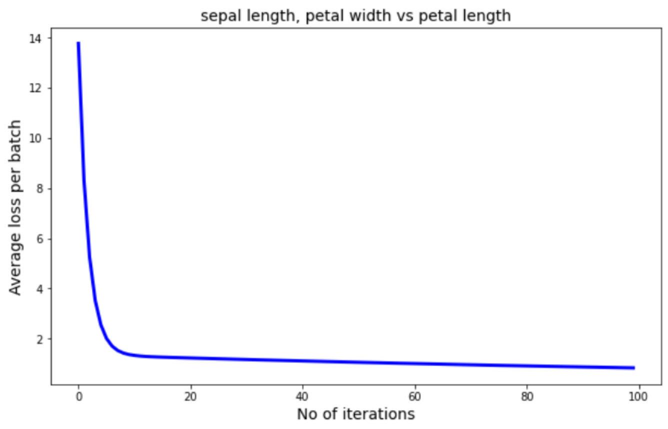


```
Weights [[0.6884453 ]
 [0.53363156]
 [0.1062753 ]]
MSE = 11.529922888355234
```

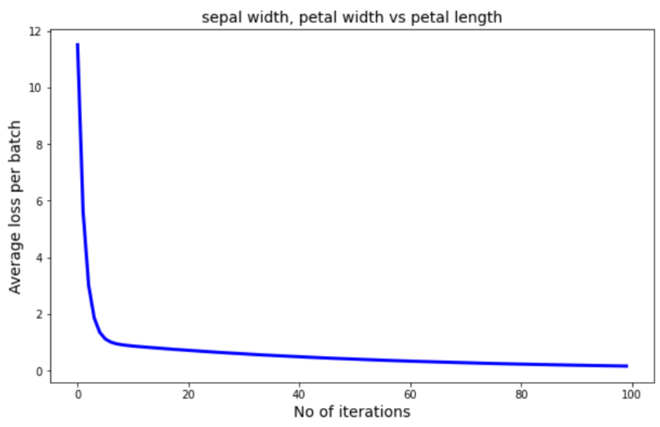
The models with 2 features and 1 Target is also considered into account.



```
Weights =  [[0.56953443]
 [0.11846559]]
MSE = 2.6449023493970136
```



```
Weights =  [[0.60105147]
 [0.63010562]]
MSE = 5.772730229904171
```

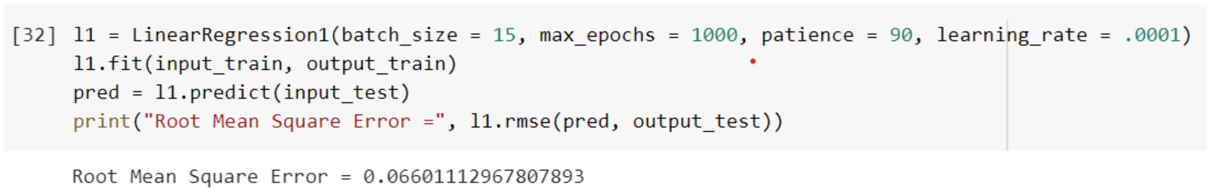


```
Weights =  [[0.08915734]
 [0.74852026]]
MSE = 2.613443198091329
```

The Model with sepal width, petal width as independent features and petal length as dependent feature showed the best performance.

Regression with Multiple Output

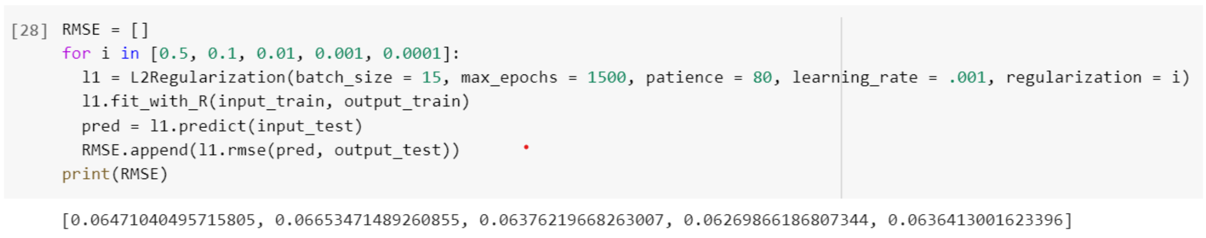
The data was pre-processed using the reshape method of NumPy library. The data is first trained using the normal Linear regression class and it was obtained a RMSE of 0.06601112967807893 by changing different values for batch size, max_epochs, patience, and learning rate.



```
[32] l1 = LinearRegression1(batch_size = 15, max_epochs = 1000, patience = 90, learning_rate = .0001)
     l1.fit(input_train, output_train)
     pred = l1.predict(input_test)
     print("Root Mean Square Error =", l1.rmse(pred, output_test))

     Root Mean Square Error = 0.06601112967807893
```

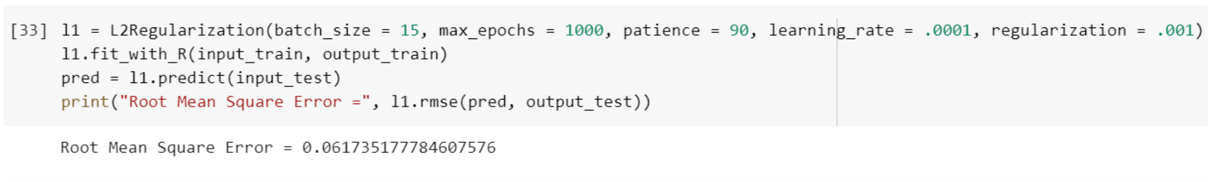
The same set of data was trained with applying regularization and resulted in a decreased RMSE = 0.06325649318163183
The regularization value was taken after considering different ranges of values as shown below.



```
[28] RMSE = []
     for i in [0.5, 0.1, 0.01, 0.001, 0.0001]:
       l1 = L2Regularization(batch_size = 15, max_epochs = 1500, patience = 80, learning_rate = .001, regularization = i)
       l1.fit_with_R(input_train, output_train)
       pred = l1.predict(input_test)
       RMSE.append(l1.rmse(pred, output_test))
     print(RMSE)

     [0.06471040495715805, 0.06653471489260855, 0.06376219668263007, 0.06269866186807344, 0.0636413001623396]
```

The regularization value is taken as .0001 as it shows the least RMSE among the list of values.



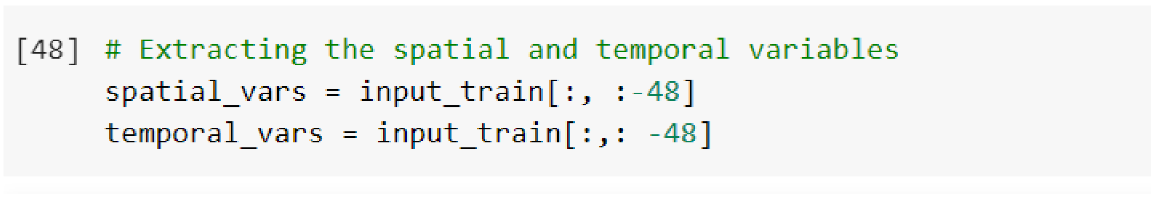
```
[33] l1 = L2Regularization(batch_size = 15, max_epochs = 1000, patience = 90, learning_rate = .0001, regularization = .001)
     l1.fit_with_R(input_train, output_train)
     pred = l1.predict(input_test)
     print("Root Mean Square Error =", l1.rmse(pred, output_test))

     Root Mean Square Error = 0.061735177784607576
```

The traffic flow forecasting comes with an adjacency matrix which is used to represent the relationships between different road segments in a transportation network. The current linear regression created doesn't take the adjacency matrix in action.

SADIL Approach

1) The first step is extracting the spatial and temporal features from the data.



```
[48] # Extracting the spatial and temporal variables
     spatial_vars = input_train[:, :-48]
     temporal_vars = input_train[:,: -48]
```

2) Computing the spatial dependency term using the adjacency matrix
3) Combining the spatial variables and dependence term
4) Training the combined features
5) Making predictions and finding the RMSE

Conclusion

Linear regression algorithm was implemented manually using the NumPy library. The datasets involved were iris data set and traffic flow forecasting data. The iris dataset different features were taken into different combinations to predict the one of the features. It was found to be the combination of sepal length, sepal width, and petal width as independent features and petal length as the dependent feature had the lowest mean squared error (MSE) which is equal to 4.46. This was a regression example with a single output. The regression with multiple output was done using traffic flow forecasting. This data was pre-processed by reshaping it to 2D array. The training obtained with different hyperparameters resulted in a RMSE of 0.06. The RMSE can further be minimized using SADIL approach by considering adjacency matrix along the model.