Linear Regression Model Algorithm

IS3117 – Machine Learning and Neural Computing

Isuru Harischandra | 19020333
UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING

Table of Contents

List of Tables

1 IMPORT LIBRARIES

```
In [8]: import pandas as pd
        import numpy as np
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
        import matplotlib.pyplot as plt
```

Figure 1: Import libraries

1.1 USES OF IMPORTED LIBRARIES

1. ***pandas***
    - For data manipulation and analysis.
    - It offers data structures and operations for manipulating numerical tables and time series.
2. ***numpy***
    - Adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
3. ***train_test_split***
    - Split arrays or matrices into random train and test subsets.
4. ***LinearRegression***
    - Ordinary least squares Linear Regression.
    - LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.
5. ***r2_score***
    - R^2 (coefficient of determination) regression score function.
    - Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).
    - In the general case when the true y is non-constant, a constant model that always predicts the average y disregarding the input features would get a R2 score of 0.0.
6. ***mean_absolute_error***
    - Mean absolute error regression loss.
7. ***mean_squared_error***
    - Mean squared error regression loss.
8. ***pyplot***
    - Mainly intended for interactive plots and simple cases of programmatic plot generation.

2 IMPORT DATASET

```
In [9]: data_df = pd.read_csv('Medicalpremium.csv')
        data_df
```

Out[9]:

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily	NumberOfMajorSu
0	45	0	0	0	0	155	57	0	0	
1	60	1	0	0	0	180	73	0	0	
2	36	1	1	0	0	158	59	0	0	
3	52	1	1	0	1	183	93	0	0	
4	38	0	0	0	1	166	88	0	0	
...	...	...	...	...	...	...	...	...	...	
981	18	0	0	0	0	169	67	0	0	
982	64	1	1	0	0	153	70	0	0	
983	56	0	1	0	0	155	71	0	0	
984	47	1	1	0	0	158	73	1	0	
985	21	0	0	0	0	158	75	1	0	

986 rows × 11 columns

Figure 2: Import dataset

- First, we have to save the dataset (excel file) in the same directory and import it into our notebook file.
- We can use ***read_csv*** function in ***pandas*** library to import the dataset and create a data frame.

3 DEFINE 'X' AND 'Y'

- Then we can separate the x columns (independent variables), and y column (dependent variable).
- X columns:
    - Age
    - Diabetes
    - BloodPressureProblems
    - AnyTransplants
    - AnyChronicDiseases
    - Height
    - Weight
    - KnownAllergies
    - HistoryOfCancerInFamily
    - NumberOfMajorSurgeries
- Y column:
    - PremiumPrice

```
In [10]: x = data_df.drop(['PremiumPrice'], axis=1).values
         print(x)

[[45  0  0 ...  0  0  0]
 [60  1  0 ...  0  0  0]
 [36  1  1 ...  0  0  1]
 ...
 [56  0  1 ...  0  0  1]
 [47  1  1 ...  1  0  1]
 [21  0  0 ...  1  0  1]]
```

Figure 3: Define 'x'

```
In [11]: y = data_df['PremiumPrice'].values
         print(y)

[25000 29000 23000 28000 23000 23000 21000 15000 23000 23000 28000 25000
 15000 35000 15000 23000 30000 23000 25000 15000 28000 15000 32000 23000
 35000 21000 15000 28000 23000 21000 15000 19000 15000 15000 28000 28000
 23000 25000 30000 15000 15000 28000 15000 29000 15000 23000 32000 35000
 25000 15000 23000 28000 28000 32000 25000 23000 29000 28000 24000 23000
 23000 25000 15000 23000 28000 15000 28000 15000 23000 21000 15000 30000
 25000 38000 28000 28000 15000 23000 28000 38000 23000 25000 15000 23000
 25000 25000 38000 15000 31000 21000 25000 28000 31000 28000 28000 15000
 23000 23000 25000 23000 15000 38000 31000 15000 15000 23000 25000 23000
 19000 28000 29000 23000 15000 23000 23000 19000 28000 25000 25000 21000
 28000 29000 23000 15000 23000 38000 30000 31000 29000 15000 15000 28000
 23000 15000 23000 15000 23000 28000 29000 38000 31000 28000 21000 21000
 28000 29000 23000 38000 28000 25000 28000 35000 29000 23000 15000 35000
 25000 15000 23000 25000 35000 15000 23000 29000 15000 23000 15000 15000
 23000 38000 30000 23000 25000 38000 15000 35000 23000 23000 23000 28000
 23000 23000 23000 23000 23000 30000 15000 15000 23000 23000 23000 35000
 28000 23000 31000 15000 23000 23000 35000 15000 23000 23000 26000 26000
 39000 35000 23000 15000 23000 35000 30000 19000 24000 22000 28000 15000
 28000 24000 38000 26000 28000 35000 35000 15000 23000 35000 28000 31000
 15000 28000 25000 23000 28000 21000 21000 15000 23000 25000 28000 30000
 35000 15000 15000 28000 23000 28000 15000 29000 25000 15000 15000 25000
 23000 35000 28000 23000 15000 15000 35000 23000 23000 15000 25000 23000
 15000 15000 25000 23000 30000 29000 25000 28000 30000 23000 28000 23000
```

Figure 4: Define 'y'

4 SPLIT THE DATASET AS 'TRAINING SET' AND 'TESTING SET'

- We need to separate our data set as **training set** and **testing set.**
- **Training set** will be used to train (create) the ML model.
- **Testing set** will be used to test the performance (accuracy of the predictions) of the ML model.
- I took the ratio between testing data set and training data set as 3:7.

```
In [12]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
         x_train

Out[12]: array([[42,  0,  0, ...,  0,  0,  0],
                [49,  1,  0, ...,  0,  0,  2],
                [62,  0,  1, ...,  1,  0,  1],
                ...,
                [41,  0,  0, ...,  0,  0,  0],
                [22,  0,  1, ...,  0,  0,  0],
                [24,  1,  0, ...,  1,  1,  1]], dtype=int64)
```

Figure 5: x_train dataset

```
In [13]: print(x_test)

[[36  1  0 ...  0  0  0]
 [46  1  1 ...  1  0  1]
 [60  0  1 ...  0  0  2]
 ...
 [57  1  1 ...  1  0  2]
 [62  1  1 ...  0  0  0]
 [51  0  1 ...  0  0  1]]
```

Figure 6: x_test dataset

```
In [14]: print(y_train)

[30000 28000 25000 23000 23000 23000 30000 15000 23000 15000 15000 28000
 23000 15000 25000 15000 23000 15000 28000 23000 28000 28000 28000 15000
 23000 15000 23000 25000 38000 15000 25000 15000 15000 31000 32000 15000
 28000 15000 32000 15000 29000 15000 28000 15000 25000 25000 23000 23000
 15000 25000 25000 23000 30000 30000 21000 15000 15000 30000 15000 30000
 15000 15000 28000 27000 38000 28000 25000 15000 35000 28000 23000 23000
 15000 15000 23000 15000 15000 28000 39000 28000 25000 25000 18000 23000
 28000 19000 28000 29000 23000 15000 28000 31000 23000 15000 23000 25000
 25000 15000 25000 19000 23000 23000 28000 23000 23000 29000 23000 23000
 15000 15000 28000 35000 23000 25000 15000 38000 28000 23000 23000 23000
 30000 15000 25000 23000 23000 15000 25000 23000 15000 23000 15000 28000
 34000 29000 35000 35000 25000 28000 28000 23000 23000 25000 28000 15000
 23000 23000 23000 28000 25000 21000 23000 29000 21000 15000 31000 15000
 31000 25000 28000 23000 15000 28000 25000 23000 15000 15000 32000 15000
 21000 35000 38000 23000 25000 29000 30000 28000 23000 35000 23000 23000
 23000 15000 23000 28000 30000 29000 28000 31000 30000 28000 31000 15000
 25000 23000 21000 38000 23000 23000 23000 15000 23000 15000 28000 15000
 23000 29000 23000 23000 23000 15000 15000 25000 28000 21000 30000 28000
 34000 25000 23000 15000 15000 23000 15000 31000 15000 23000 31000 15000
 23000 22000 15000 15000 25000 15000 15000 23000 28000 31000 15000 23000
 23000 29000 15000 28000 23000 28000 23000 23000 28000 19000 23000 35000
 29000 38000 29000 25000 23000 15000 25000 23000 23000 15000 31000 25000
 15000 21000 29000 30000 23000 28000 15000 28000 15000 28000 28000 30000
```

Figure 7: y_train dataset

```
In [15]: print(y_test)

[23000 23000 28000 23000 38000 23000 15000 29000 25000 35000 23000 23000
 15000 25000 30000 15000 23000 29000 15000 40000 15000 28000 23000 15000
 19000 38000 23000 25000 29000 35000 35000 15000 15000 15000 25000 23000
 23000 25000 25000 19000 28000 19000 23000 19000 23000 15000 28000 39000
 29000 28000 15000 21000 15000 28000 30000 28000 25000 28000 35000 23000
 29000 28000 30000 30000 25000 31000 15000 23000 29000 23000 23000 23000
 30000 15000 28000 28000 23000 25000 29000 23000 30000 28000 15000 29000
 23000 28000 15000 23000 30000 23000 38000 38000 31000 23000 21000 19000
 25000 25000 38000 28000 25000 23000 21000 15000 28000 25000 25000 23000
 23000 23000 15000 28000 28000 35000 38000 35000 26000 23000 25000 15000
 23000 28000 23000 38000 23000 29000 21000 15000 23000 23000 23000 23000
 23000 25000 28000 25000 15000 35000 15000 15000 29000 15000 28000 15000
 35000 29000 15000 15000 30000 28000 23000 21000 15000 15000 19000 29000
 36000 23000 23000 15000 28000 23000 28000 25000 23000 24000 23000 15000
 31000 15000 28000 28000 23000 30000 31000 23000 28000 29000 23000 35000
 23000 28000 23000 23000 25000 23000 28000 23000 28000 15000 28000 15000
 23000 25000 25000 28000 39000 20000 23000 15000 15000 35000 23000 23000
 25000 30000 23000 38000 25000 35000 36000 30000 15000 15000 15000 38000
 28000 23000 23000 29000 23000 23000 23000 25000 15000 15000 23000 15000
 30000 23000 29000 25000 23000 28000 30000 23000 28000 25000 21000 31000
 29000 39000 23000 23000 23000 35000 23000 25000 31000 28000 25000 29000
 15000 23000 21000 19000 15000 39000 21000 23000 23000 30000 23000 23000
 29000 21000 25000 15000 23000 15000 23000 23000 15000 17000 25000 35000
```

Figure 8: y_test dataset

5 TRAIN THE MODEL WITH 'TRAINING SET'

```
In [16]: ml = LinearRegression()
         ml.fit(x_train, y_train)

Out[16]: LinearRegression()
```

Figure 9: Training the model

- We initialize a linear regression ML model by using ***LinearRegression*** function.
- We use the ***fit*** function to feed training data set and train out ML model.

6 PREDICT THE 'TESTING SET' RESULTS

```
In [17]: y_pred = ml.predict(x_test)
         print(y_pred)

[20876.01149832 24416.8042804  29043.0096839  25133.66293852
 30940.30194271 22294.16605122 19859.38133309 29196.60416663
 26144.19512571 30860.84210117 19353.44606745 19360.15393381
 18779.41831573 26600.16618646 29357.73785907 22362.06043146
 24499.23375845 28451.54744784 18596.82874676 33305.78203113
 15772.39396825 30627.07715624 24212.67679329 20258.12471786
 22293.39888697 31271.61207354 25952.05487343 25891.7009764
 29248.58675284 31866.24620222 31893.39071888 16624.35525806
 18089.8478003  16013.8420137  31398.29587926 21965.81003703
 22365.37058913 25740.79124579 28093.6461732  21505.32302697
 30182.67211147 19746.44130828 19619.51713111 21155.43786727
 23200.04945983 18702.93465046 24291.72508886 23512.28859871
 26851.1758127  29651.37223567 17002.23080827 20523.80403228
 19108.05208737 27502.9783518  25951.85381479 28231.05658074
 25008.19829939 29955.3951682  28156.12721174 22690.76496977
 27038.52290307 28217.16215911 31220.59942069 27399.60573547
 24559.45753497 26552.14598841 17242.34346687 20585.10262378
 28805.22407751 24635.58989568 20704.0239329  22702.3903547
 27789.83474357 16622.10110161 30880.42451545 34371.42454449
 23393.7264436  27017.43274719 26695.64318048 23012.83008083
 34108.3228926  30201.5795362  15893.25277184 25237.99872924
 24569.51231724 30024.59978572 17458.5588314  21821.2693572
```

Figure 10: Predicted results

- We use the ***predict*** method to get predicted results for testing data set.

7 EVALUATE THE MODEL

- **r2_score**.

```
In [18]: r2_score(y_test, y_pred)

Out[18]: 0.5086541413229406
```

Figure 11: Evaluate the model using r2_score

- There are three error metrics that are commonly used for evaluating and reporting the performance of a regression model;
    - Mean Squared Error (MSE).
    - Root Mean Squared Error (RMSE).
    - Mean Absolute Error (MAE).

7.1 MEAN SQUARED ERROR (MSE)

- The MSE is calculated as the mean or average of the squared differences between predicted and expected target values in a dataset.
- MSE = ***(1 / N) **** sum for ***i*** to ***N (y_i – yhat_i) ^ 2***
    - ***y_i*** is the ***i***'th expected value in the dataset.
    - ***yhat_i*** is the ***i***'th predicted value.
    - The difference between these two values is squared, which has the effect of removing the sign, resulting in a positive error value.
- The squaring also has the effect of inflating or magnifying large errors (the larger the difference between the predicted and expected values, the larger the resulting squared positive error).
- This has the effect of "punishing" models more for larger errors when MSE is used as a loss function.

In [28]:
```python
# calculate errors
errors = list()
for i in range(len(y_test)):
    # calculate error
    err = (y_test[i] - y_pred[i]) ** 2
    # store error
    errors.append(err)
# plot errors
plt.figure(figsize = (10, 5))
plt.plot(errors)
plt.xticks(ticks = [i for i in range(len(errors))], labels = y_pred)
plt.xlabel('Predicted Value')
plt.ylabel('Mean Squared Error')
plt.show()
```

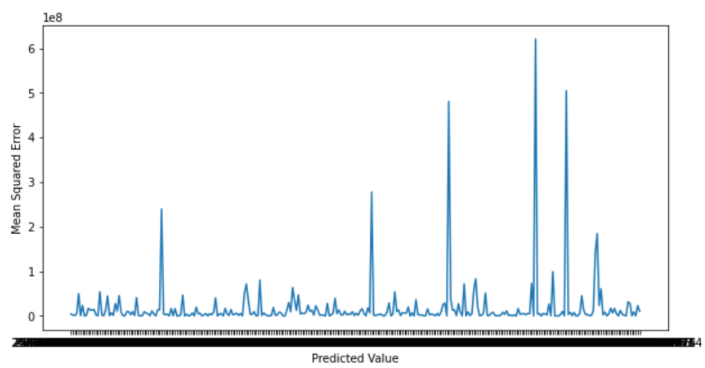

Figure 12: Plot MSE against Predicted value

- A line plot is created showing the curved or super-linear increase in the squared error value as the difference between the expected and predicted value is increased.
- The units of the MSE are squared units.

- The mean squared error between your expected and predicted values can be calculated using the ***mean_squared_error*** function.

```
In [29]: mean_squared_error(y_test, y_pred)
Out[29]: 18526121.127886258
```

Figure 13: Mean squared error between expected and predicted values

- A perfect mean squared error value is 0.0, which means that all predictions matched the expected values exactly.
- This is almost never the case, and if it happens, it suggests your predictive modeling problem is trivial.
- A good MSE is relative to your specific dataset.

7.2 Root Mean Squared Error (RMSE)

- Importantly, the square root of the error is calculated, which means that the units of the RMSE are the same as the original units of the target value that is being predicted.
- As such, it may be common to use MSE loss to train a regression predictive model, and to use RMSE to evaluate and report its performance.
- RMSE = ***sqrt(1 / N **** sum for ***i*** to ***N (y_i – yhat_i) ^ 2)***
    - ***y_i*** is the ***i***'th expected value in the dataset.
    - ***yhat_i*** is the ***i***'th predicted value.
    - ***sqrt()*** is the square root function.
- We can restate the RMSE in terms of the MSE as:
    - RMSE = ***sqrt(MSE)***

```
In [38]: from math import sqrt

         # calculate errors
         errors = list()
         for i in range(len(y_test)):
             # calculate error
             err = sqrt((y_test[i] - y_pred[i]) ** 2)
             # store error
             errors.append(err)
         # plot errors
         plt.figure(figsize = (10, 5))
         plt.plot(errors)
         plt.xticks(ticks = [i for i in range(len(errors))], labels = y_pred)
         plt.xlabel('Predicted Value')
         plt.ylabel('Root Mean Squared Error')
         plt.show()
```

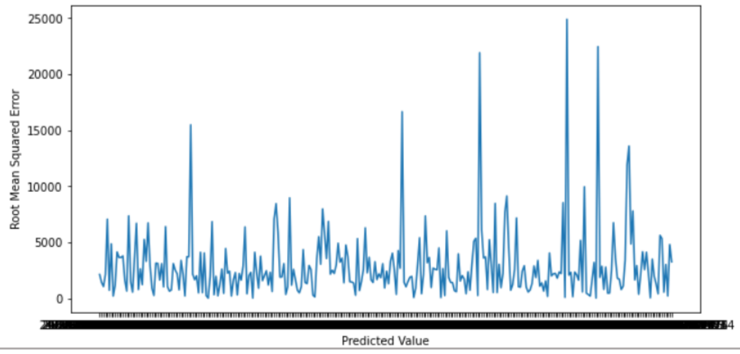

Figure 14: Plot RMSE against Predicted value

- The root mean squared error between your expected and predicted values can be calculated using the ***mean_squared_error*** function.
- By default, the function calculates the MSE, but we can configure it to calculate the square root of the MSE by setting the "squared" argument to False.

```
In [40]: mean_squared_error(y_test, y_pred, squared = False)
Out[40]: 4304.198081859879
```

Figure 15: Root mean squared error between expected and predicted values

- A perfect RMSE value is 0.0, which means that all predictions matched the expected values exactly.
- This is almost never the case, and if it happens, it suggests your predictive modeling problem is trivial.
- A good RMSE is relative to your specific dataset.

7.3 Mean Absolute Error (MAE)

- Mean Absolute Error, or MAE, is a popular metric because, like RMSE, the units of the error score match the units of the target value that is being predicted.
- Unlike the RMSE, the changes in MAE are linear and therefore intuitive.

- That is, MSE and RMSE punish larger errors more than smaller errors, inflating or magnifying the mean error score.
- The MAE score is calculated as the average of the absolute error values.
- The difference between an expected and predicted value may be positive or negative and is forced to be positive when calculating the MAE.
- MAE = ***(1 / N)*** ***** sum for ***i*** to ***N abs(y_i – yhat_i)***
    - ***y_i*** is the ***i***'th expected value in the dataset.
    - ***yhat_i*** is the ***i***'th predicted value.
    - ***abs()*** is the absolute function.
- We can create a plot to get a feeling for how the change in prediction error impacts the MAE.

In [42]:
```
# calculate errors
errors = list()
for i in range(len(y_test)):
    # calculate error
    err = abs((y_test[i] - y_pred[i]))
    # store error
    errors.append(err)
# plot errors
plt.figure(figsize = (10, 5))
plt.plot(errors)
plt.xticks(ticks=[i for i in range(len(errors))], labels=y_pred)
plt.xlabel('Predicted Value')
plt.ylabel('Mean Absolute Error')
plt.show()
```

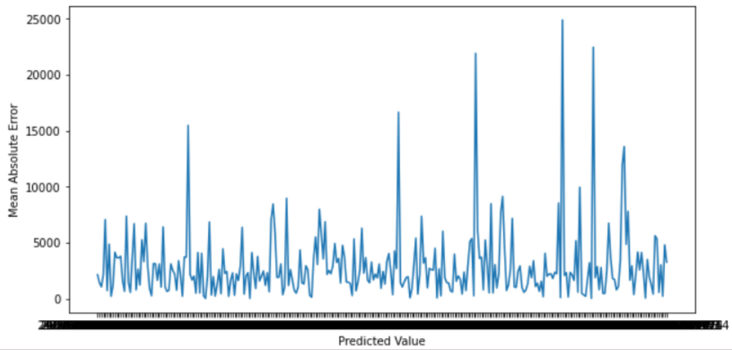

Figure 16: Plot MAE against Predicted value

- The mean absolute error between your expected and predicted values can be calculated using the ***mean_absolute_error*** function.

In [43]:
```
mean_absolute_error(y_test, y_pred)
```
Out[43]:
```
2902.212708668325
```

Figure 17: Mean absolute error between expected and predicted values

- A perfect mean absolute error value is 0.0, which means that all predictions matched the expected values exactly.

- This is almost never the case, and if it happens, it suggests your predictive modeling problem is trivial.
- A good MAE is relative to your specific dataset.

8 PLOT THE RESULTS

- We can view the difference between the actual data points and their predicted data points in a scatter plot.

```
In [31]: plt.figure(figsize=(12, 6))
         plt.scatter(y_test, y_pred)
         plt.xlabel('Actual')
         plt.ylabel('Predicted')
         plt.title('Actual vs. Predicted')

Out[31]: Text(0.5, 1.0, 'Actual vs. Predicted')
```

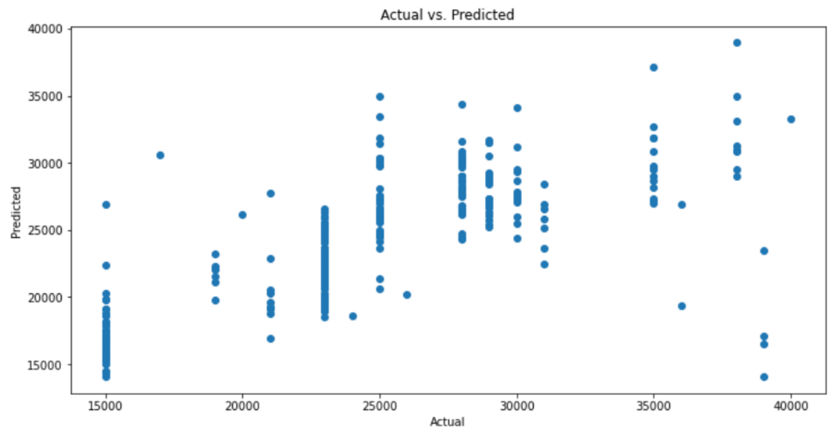

Figure 18: The difference between actual values and their predicted values (scatter plot view)

9 PREDICTED RESULTS

- As the final step of the study we can view the difference between actual results and predicted results in a data frame.

```
In [32]: pred_y_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred, 'Difference': (y_test-y_pred)})
         pred_y_df
```

Out[32]:

	Actual	Predicted	Difference
0	23000	20876.011498	2123.988502
1	23000	24416.804280	-1416.804280
2	28000	29043.009684	-1043.009684
3	23000	25133.662939	-2133.662939
4	38000	30940.301943	7059.698057
...	...	...	...
291	25000	24457.591886	542.408114
292	19000	22010.702072	-3010.702072
293	28000	28199.365742	-199.365742
294	25000	29795.386750	-4795.386750
295	29000	25746.656927	3253.343073

296 rows × 3 columns

Figure 19: The difference between actual values and their predicted values (data frame view)

10 Source Code

https://github.com/IsuruVihan/ML-Regression-Model