Linear Regression

Training Linear Models

Presented by
Sai Harshinee Roopakula
19577

Table of Contents

Introduction - Linear Regression

Linear regression is used for finding linear relationship between target and one or more predictors.
There are two types of linear regression- Simple and Multiple.

Simple Linear Regression

Simple linear regression is useful for finding relationship between two continuous variables. One is predictor or independent variable and other is response or dependent variable.

It looks for statistical relationship but not deterministic relationship. Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other.
For example, using temperature in degree Celsius it is possible to accurately predict Fahrenheit.

Statistical relationship is not accurate in determining relationship between two variables.
For example, relationship between height and weight.

The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.

Introduction - Linear Regression

For example, in a simple regression problem (a single x and a single y), the form of the model would be:

$$y = b0 + b1*x$$

The values b0 and b1 must be chosen so that they minimize the error. If sum of squared error is taken as a metric to evaluate the model, then goal to obtain a line that best reduces the error.

$$\text{Error} = \sum_{i=1}^{n}(actual\_output - predicted\_output) ** 2$$

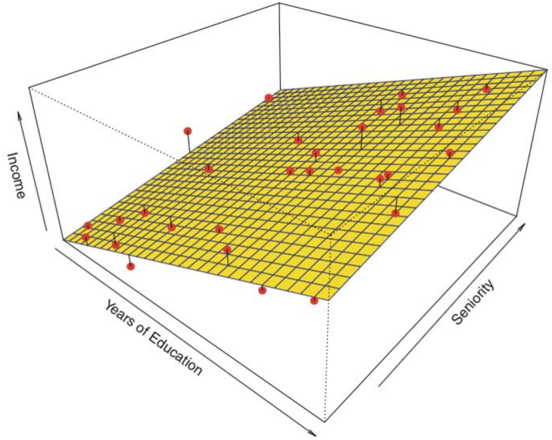
If we don't square the error, then positive and negative point will cancel out each other. For model with one predictor,

$$b_0 = \bar{y} - b_1\bar{x}$$

Intercept Calculation

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Coefficient Formula



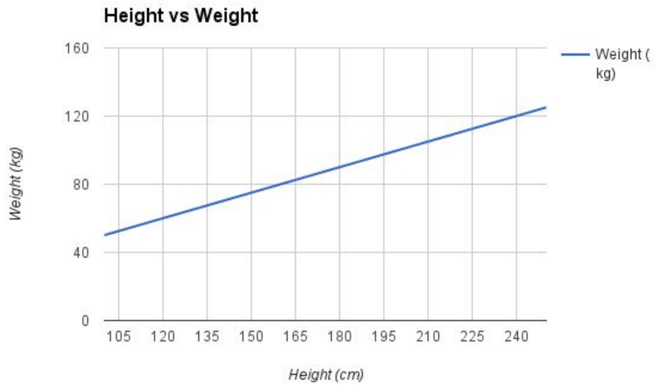
Introduction - Linear Regression

Apart from above equation coefficient of the model can also be calculated from normal equation.

$$\text{Theta} = (X^T X)^{-1} X^T Y$$

In the above equation,
θ : hypothesis parameters that define it the best.
X : Input feature value of each instance.
Y : Output value of each instance.

Theta contains co-efficient of all predictors including constant term 'b0'. Normal equation performs computation by taking inverse of input matrix. Complexity of the computation will increase as the number of features increase. It gets very slow when number of features grow large.



Project-Understanding the project

The Abalone dataset contains the physical measurements of abalones, which are large, edible sea snails.

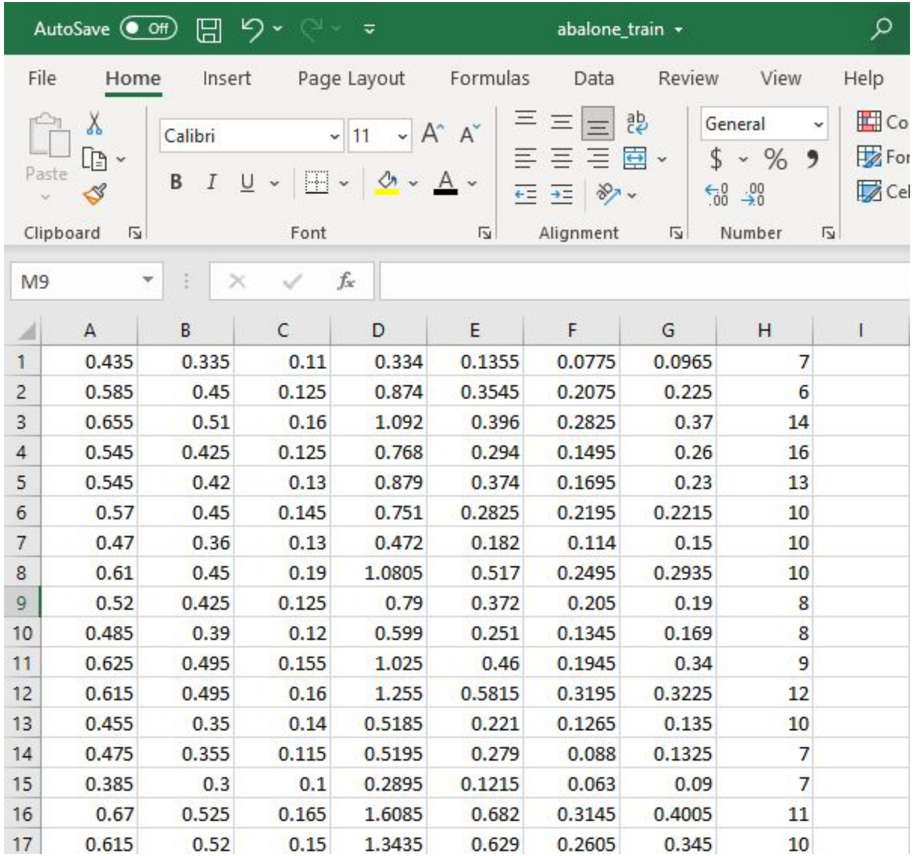
There are 3320 rows and 8 columns.

The columns include Length, Diameter, Height, Whole weight, Shucked weight, Viscera weight, Shell weight and Age.

There are no missing values.

The dataset is stored in csv file named abalone_train,csv

Project-Project Dataset



	A	B	C	D	E	F	G	H	I
1	0.435	0.335	0.11	0.334	0.1355	0.0775	0.0965	7	
2	0.585	0.45	0.125	0.874	0.3545	0.2075	0.225	6	
3	0.655	0.51	0.16	1.092	0.396	0.2825	0.37	14	
4	0.545	0.425	0.125	0.768	0.294	0.1495	0.26	16	
5	0.545	0.42	0.13	0.879	0.374	0.1695	0.23	13	
6	0.57	0.45	0.145	0.751	0.2825	0.2195	0.2215	10	
7	0.47	0.36	0.13	0.472	0.182	0.114	0.15	10	
8	0.61	0.45	0.19	1.0805	0.517	0.2495	0.2935	10	
9	0.52	0.425	0.125	0.79	0.372	0.205	0.19	8	
10	0.485	0.39	0.12	0.599	0.251	0.1345	0.169	8	
11	0.625	0.495	0.155	1.025	0.46	0.1945	0.34	9	
12	0.615	0.495	0.16	1.255	0.5815	0.3195	0.3225	12	
13	0.455	0.35	0.14	0.5185	0.221	0.1265	0.135	10	
14	0.475	0.355	0.115	0.5195	0.279	0.088	0.1325	7	
15	0.385	0.3	0.1	0.2895	0.1215	0.063	0.09	7	
16	0.67	0.525	0.165	1.6085	0.682	0.3145	0.4005	11	
17	0.615	0.52	0.15	1.3435	0.629	0.2605	0.345	10	

The .csv file has 3320 rows and 8 columns namely

- Length
- Diameter
- Height
- Whole weight
- Shucked weight
- Viscera weight
- Shell weight
- Age.

Implementation - Linear Regression using Normal Equation

Step 1 - Setup - Importing the necessary modules

Setup

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. We also check that Python 3.5 or later is installed (although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead), as well as Scikit-Learn ≥0.20.

```
[106] # Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "training_linear_models"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

In Step 1, we are importing the necessary modules.

- matplotlib
- numpy
- sklearn

Checking system and sklearn versions.
Writing a function to save the figures.

Implementation - Linear Regression using Normal Equation

Step 2 - Uploading the datafile (.csv file) and preparing X and y values.

```
[107] import numpy as np
      import pandas as pd
      from google.colab import files
      uploaded  = files.upload()

      import io
      abalone = pd.read_csv(
          io.BytesIO(uploaded['abalone_train.csv']),
          names=["Length", "Diameter", "Height", "Whole weight", "Shucked weight",
                 "Viscera weight", "Shell weight", "Age"])

      X1 = abalone["Length"]
      X2 = np.array(X1)
      X = X2.reshape(-1, 1)

      y1 = abalone["Height"]
      y2 = np.array(y1)
      y = y2.reshape(-1, 1)
```

```
Choose Files  abalone_train.csv
• abalone_train.csv(application/vnd.ms-excel) - 145915 bytes, last modified: 5/26/2021 - 100% done
Saving abalone_train.csv to abalone_train (3).csv
```

In Step 2, we are reading the dataset from a csv file.

The columns names are also added to understand the data better.

X (input) and y (output) values are prepared.

Here, X value is Length and y value is Height.

np.array(X1) converts the data into 1D array.

X2.reshape(-1,1) is used to reshape the data. It reshapes the data from 1D array to 2D array.

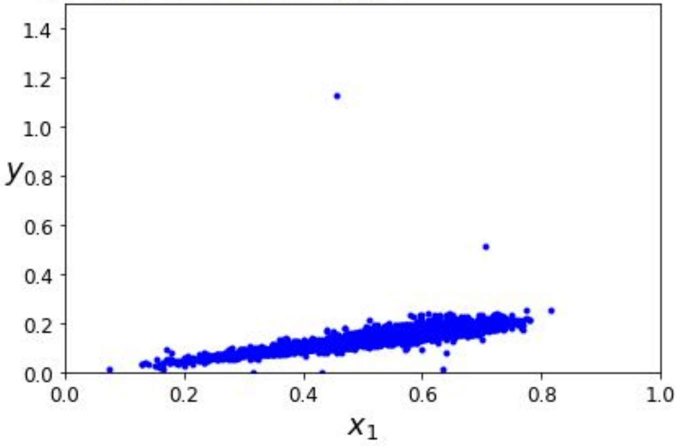
The same holds good for y values.

Implementation - Linear Regression using Normal Equation

Step 3 - Plotting the X and y values

```
[108] plt.plot(X, y, "b.")
      plt.xlabel("$x_1$", fontsize=18)
      plt.ylabel("$y$", rotation=0, fontsize=18)
      plt.axis([0, 1, 0, 1.5])
      save_fig("generated_data_plot")
      plt.show()
```

```
Saving figure generated_data_plot
```



In Step 3, X and y values are plotted on a graph using matplotlib.pyplot.

The x and y axis names can be set using plt.xlabel and plt.ylabel

plt.axis is used to change the x and y axis lengths to be displayed.

plt.show() is used to show the plot on the screen as output.

Implementation - Linear Regression using Normal Equation

Step 4 - Finding the theta best value using normal equation formula

```
[109] X_b = np.c_[np.ones((X.size, 1)), X]  # add x0 = 1 to each instance
      theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)

[110] theta_best

      array([[-0.0108267 ],
             [ 0.28716253]])

[111] X_new = np.array([[0], [2]])
      X_new_b = np.c_[np.ones((2, 1)), X_new]  # add x0 = 1 to each instance
      y_predict = X_new_b.dot(theta_best)
      y_predict

      array([[-0.0108267 ],
             [ 0.56349837]])
```

In Step 4, we are finding the theta best value using Normal Equation

$$\text{Theta} = (X^TX)^{-1}X^TY$$

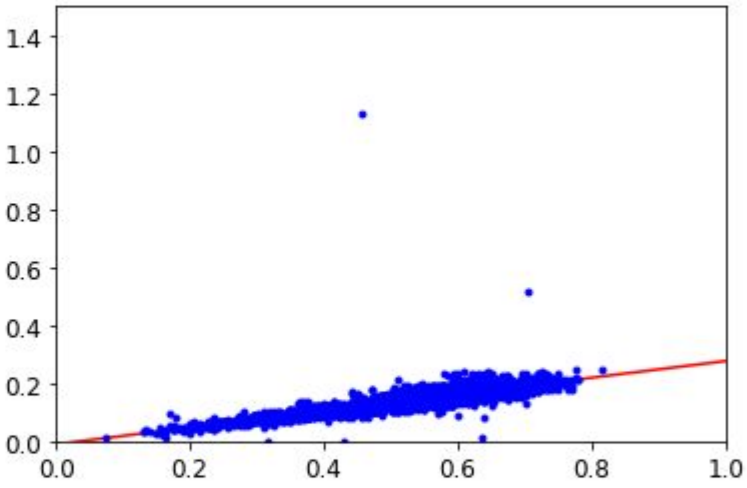
np.linalg.inv(a) computes the multiplicative inverse of a matrix.

We can predicting y values using X_new_b and theta_best

Implementation - Linear Regression using Normal Equation

Step 5 - Plotting X, y and X_new and y_predict

```
[112] plt.plot(X_new, y_predict, "r-")
      plt.plot(X, y, "b.")
      plt.axis([0, 1, 0, 1.5])
      plt.show()
```



In Step 5, we are plotting X and y in blue color and plotting the X_new and y_predict obtained by theta_best in red color.

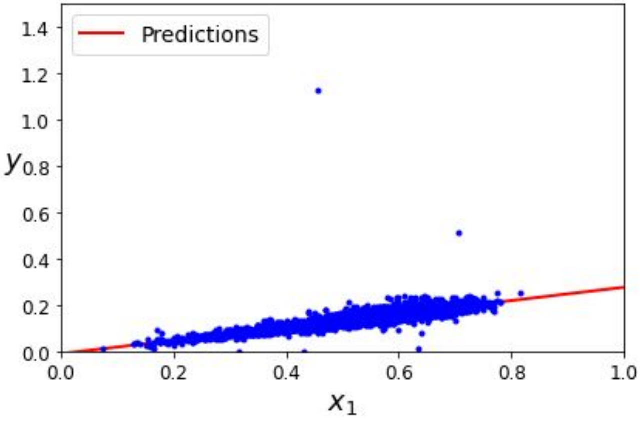
The plot between X_new and y_pred results in a linear regression line.

Implementation - Linear Regression using Normal Equation

Step 6 - Adding features to the plot obtained in above step.

```
[113] plt.plot(X_new, y_predict, "r-", linewidth=2, label="Predictions")
      plt.plot(X, y, "b.")
      plt.xlabel("$x_1$", fontsize=18)
      plt.ylabel("$y$", rotation=0, fontsize=18)
      plt.legend(loc="upper left", fontsize=14)
      plt.axis([0, 1, 0, 1.5])
      save_fig("linear_model_predictions_plot")
      plt.show()
```

Saving figure linear_model_predictions_plot



Here, Legends and axis labels are added to the plot.

In plt.plot(), label can be added. Here label = "Predictions"

Location, font size for the legend can be specified using the plt.legend()

The x and y axis names can be set using plt.xlabel and plt.ylabel

plt.axis is used to change the x and y axis lengths to be displayed.

plt.show() is used to show the plot on the screen as output.

Implementation - Linear Regression using scikit-learn

Step 1 - Creating a Linear Regression model

```
[114] from sklearn.linear_model import LinearRegression

      lin_reg = LinearRegression()
      lin_reg.fit(X, y)
      lin_reg.intercept_, lin_reg.coef_

      (array([-0.0108267]), array([[0.28716253]]))
```

Here, we are creating a Linear Regression model by importing LinearRegression from sklearn.linear_model

We are then creating an object of the Linear Regression model.

We can now fit the data and then find the intercept and coefficient by using .intercept_ and .coef_

y = b0 + b1*x
where,
b0 is intercept
b1 is coefficient.

Implementation - Linear Regression using scikit-learn

Step 2 - Predicting new dataset value

```
[115] lin_reg.predict(X_new)

      array([[-0.0108267 ],
             [ 0.56349837]])
```

We can predict new y value using the .predict() by passing the X_value whose y value needs to be predicted.

The value of y_predict obtained by using the scikit-learn method is same as the value of y obtained when normal equation formula is used above.

Implementation - Linear Regression using scikit-learn

Step 3 - Getting the theta value

The `LinearRegression` class is based on the `scipy.linalg.lstsq()` function (the name stands for "least squares"), which you could call directly:

```
[116] theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y, rcond=1e-6)
      theta_best_svd

      array([[-0.0108267 ],
             [ 0.28716253]])
```

This function computes $\mathbf{X}^{+}\mathbf{y}$, where $\mathbf{X}^{+}$ is the *pseudoinverse* of $\mathbf{X}$ (specifically the Moore-Penrose inverse). You can use `np.linalg.pinv()` to compute the pseudoinverse directly:

```
[117] np.linalg.pinv(X_b).dot(y)

      array([[-0.0108267 ],
             [ 0.28716253]])
```

The theta_best value can be found directly by calling the scipy.linalg.lstsq() function (which is the "least squares") which is available in the LinearRegression.

Also, np.linalg.pinv() which computes the pseudoinverse can also be used to directly find the theta_best value.

Conclusion

We have implemented Linear Regression algorithm using 2 methods namely:

- Normal Equation
- scikit-learn

It is found that the 2 methods above, gave the same result.

Normal equation formula is bit confusing to implement each time,

While, the best method to be used to implement Linear Regression algorithm is using scikit learn. We can import LinearRegression from sklearn.linear_model

Bibliography

https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86

https://npu85.npu.edu/~henry/npu/classes/data_science/algorithm/slide/index_slide.html

https://colab.research.google.com/github/ageron/handson-ml2/blob/master/04_training_linear_models.ipynb

Link to view the presentation
https://docs.google.com/presentation/d/1KEGFgRT4DlK_qwmVSmyhBt4A0qwIA6nkbJ5BKzrexMk/edit?usp=sharing