Linear Regression From Scratch in Python WITHOUT Scikit-learn

Sindhu Seelam · Follow

Published in Geek Culture · 7 min read · May 18, 2021

In this tutorial, I'll go over a brief introduction to one of the most commonly used machine learning algorithms, Linear Regression, and then we'll learn how to implement it using the least-squares method from scratch in python without sci-kit-learn. We'll also look at the interpretation of R squared in regression analysis and how it can be used to measure the goodness of the regression model.

Linear Regression is a type of predictive analysis algorithm that shows a linear relationship between the dependent variable(x) and independent variable(y).

Based on the given data points, we try to plot a straight line that fits the points the best. The equation of a straight line is shown below:

$$y = c + m^{*}x$$

where,
x: input data points
y: predicted value, dependent variable (supervised learning)

The model gets the best-fit regression line by finding the best m, c values.
m: bias or slope of the regression line
c: intercept, shows the point where the estimated regression line crosses the *y* axis

Cost Function (J)-

As explained above our goal is to find a regression line or the best fit line which has the least difference (error/residual) between the predicted value and the actual value. This is where the cost function comes into the picture as we use the cost function extensively to calculate the values of (c, m) to reach the best value that minimizes the error between predicted y value (y^) and true y value (y).

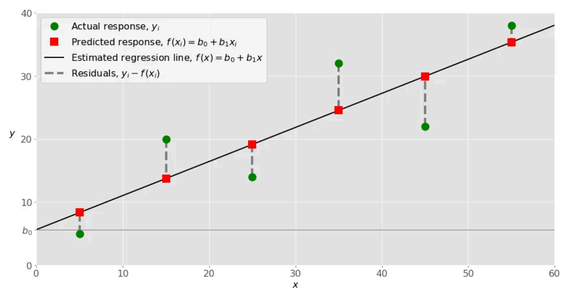


Image Source: Linear Regression By Real Python

> *Cost function(J) of Linear Regression is the* ***Root Mean Squared Error (RMSE)*** *between predicted y value (y^) and true y value (y).*

Mean Squared Error (MSE)-

Given our simple linear equation y = c + m*x, we can calculate MSE as:

$$J = MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

Where,

- *N* is the total number of observations (data points)
- y_i is the actual value of an observation and y^ is the predicted value
- J is the cost function which is the mean squared error in this case

Python Implementation from Scratch:

It's time to learn the mathematical implementation of the algorithm. For this tutorial, I'll be working with a simple data set of x and corresponding y values as shown below.

x	y
1	3
2	4
3	2
4	4
5	5

Let's calculate the mean of x and y, we'll denote them as $\bar{x}$ & $\bar{y}$.

x	y
1	3
2	4
3	2
4	4
5	5
mean: $\bar{x}$ 3	3.6

See, our goal is to predict the best-fit regression line using the least-squares method. So to find that we've to first find the equation of such a line. So if y = c+ m*x, where 'm' is slope/bias which is denoted by a change in x divided by change in y.

> *Change in x is the difference between actual input value x_i and $\bar{x}$, and similarly change in y is the difference between label y_i and $\bar{y}$.*

Below is the mathematical representation of **m**,

$$m = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2}$$

So moving ahead, according to the formula of 'm', what we're gonna do is calculate (x-$\bar{x}$)& (y-$\bar{y}$) for each data point in our very simple dataset.

x	y	$x-\bar{x}$	$y-\bar{y}$	$(x-\bar{x})^2$	$(x-\bar{x})(y-\bar{y})$
1	3	-2	-0.6	4	1.2
2	4	-1	0.4	1	-0.4
3	2	0	-1.6	0	0
4	4	1	0.4	1	0.4
5	5	2	1.4	4	2.8

Image Source: Linear Regression by Edureka

Now that we've all the individual elements in our formula ready, we will calculate the summation of numerator and denominator and find the final value of 'm'.

x	y	$x-\bar{x}$	$y-\bar{y}$	$(x-\bar{x})^2$	$(x-\bar{x})(y-\bar{y})$
1	3	-2	-0.6	4	1.2
2	4	-1	0.4	1	-0.4
3	2	0	-1.6	0	0
4	4	1	0.4	1	0.4
5	5	2	1.4	4	2.8
3	3.6			$\Sigma = 10$	$\Sigma = 4$

$$m = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sum (x-\bar{x})^2} = \frac{4}{10}$$

Image Source: Linear Regression by Edureka

Now we have the final equation :

3.6 = 0.4 * 3 + c

c = 2.4

Now, for given m = 0.4 & c=2.4, let's predict y for for all input values x = {1,2,3,4,5}

y = 0.4*1 + 2.4 = 2.8

y = 0.4*2 + 2.4 = 3.2

y = 0.4*3 + 2.4 = 3.6

y = 0.4*4 + 2.4 = 4.0

y = 0.4*5 + 2.4 = 4.4

Now if we plot them, the line passing through all these predicted y values and cutting the y-axis at 2.4 is our regression line.

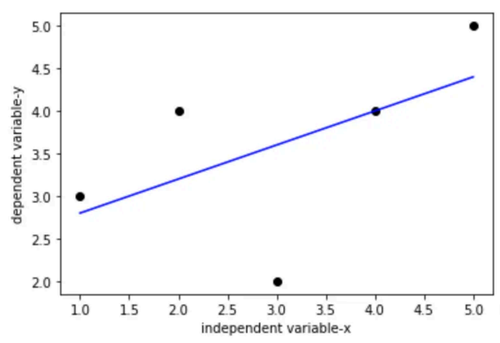


Image by Author

Now our job is to calculate the distance between actual and predicted values and reduce this distance. Or in other words, we've to reduce the error between the actual and the predicted value. The line with the least error will be the line of linear regression.

So this is how it works internally:

1. *It does a number of iterations for different values of 'm',*
2. *then calculates the equation of line y = c + m*x. As the value of m changes, the line will change.*
3. *After every iteration, it will calculate the predicted value according to the line and compares the distance between actual & predicted values.*
4. *And the line for which the predicted value has the least distance from the actual value will be determined as the regression line.*

R-squared value:

Now that we've found the best-fit regression line, it's time to measure the goodness of it or to check how good our model is performing.

> *R-squared value is a statistical measure of how close the data are to the fitted regression line.*

It is also known as the coefficient of determination or coefficient of multiple determination.

Here's how we calculate R-squared value,

$$R^2 = \frac{\sum (y_{pred} - \bar{y})^2}{\sum (y - \bar{y})^2}$$

Where y_{pred} is the predicted y value and $\bar{y}$ is the mean and y is the actual value

Basically, we're calculating the difference between the predicted value and the mean, then dividing it by the difference between the actual value and the mean.

Now below will be the final R-squared value after summating all the differences between predicted and actual values and

x	y	$y-y$	$(y-y)^2$	y_p	(y_p-y)	$(y_p-y)^2$
1	3	– 0.6	0.36	2.8	-0.8	0.64
2	4	0.4	0.16	3.2	-0.4	0.16
3	2	–1.6	2.56	3.6	0	0
4	4	0.4	0.16	4.0	0.4	0.16
5	5	1.4	1.96	4.4	0.8	0.64
mean y	3.6		Σ 5.2			Σ 1.6

$$R^2 = \frac{1.6}{5.2} = \frac{\sum \left((y_p - \bar{y}\right)^2}{\sum (y-\bar{y})^2}$$

Image Source: Linear Regression by Edureka

And that is approximate,

$$R^2 \approx 0.3$$

It means that our data points are far away from the regression line. Well, that's no good, is it?

Basically the higher the R-squared value the better our model performance will be. So as the R-squared value gradually increases, the distance of actual points from the regression line decreases, and the performance of the model increases.

Implementation in Python:

Now that we've learned the theory behind linear regression & R-squared value, let's move on to the coding part. I'll be using python and Google Colab.

I'll be working with a simple dataset called head brain from Kaggle. It has 237 rows and 4 columns which means 237 observations and 4 labels. We have to predict the brain weight of an individual based on given head size(cm).

Step-1:

Import necessary libraries like pandas, NumPy & matplotlib

```
#import necessary libraries
import pandas as pd
import numpy as np
import math
import operator
import matplotlib.pyplot as plt
%matplotlib inline
```

linreg.py hosted with ❤ by GitHub view raw

Step-2:

Import CSV file as a pandas data frame.

```
# uplading data file from local drive in google colab
from google.colab import files
uploaded = files.upload()

import io

# reading csv file as pandas dataframe
data = pd.read_csv(io.BytesIO(uploaded['headbrain.csv']))
print(data.head())
```

upload.py hosted with ❤ by GitHub view raw

This is our sample dataset:

	Gender	Age Range	Head Size(cm^3)	Brain Weight(grams)
0	1	1	4512	1530
1	1	1	3738	1297
2	1	1	4261	1335
3	1	1	3777	1282
4	1	1	4177	1590

Step-3:

We can find a linear relationship between head size and brain weight. The next step is to collect our X and Y values. X would consist of head size values and Y would consist of brain weight values. We also need to find the values of 'm' and 'c', so for that, we need to find the mean of X & Y values.

```
# collecting x & y
X = data['Head Size(cm^3)'].values
Y = data['Brain Weight(grams)'].values

# calculate mean of x & y using an inbuilt numpy method mean()
mean_x = np.mean(X)
mean_y = np.mean(Y)
```

linreg2.py hosted with ❤ by GitHub view raw

Step-4:

Calculate m & c using the formulas, if you remember we've discussed previously in this article,

$$y = c + m^* x$$

$$m = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2}$$

```
# total no.of input values
m = len(X)

# using the formula to calculate m & c
numer = 0
denom = 0
for i in range(m):
  numer += (X[i] - mean_x) * (Y[i] - mean_y)
  denom += (X[i] - mean_x) ** 2
m = numer / denom
c = mean_y - (m * mean_x)

print (f'm = {m} \nc = {c}')
```

form1.py hosted with ❤ by GitHub view raw

```
m = 0.26342933948939945
c = 325.57342104944223
```

Step-5:

Now that we've our m & c, let's plot the input points and the regression line.

```
# plotting values and regression line
max_x = np.max(X) + 100
min_x = np.min(Y) - 100

# calculating line values x and y
x = np.linspace (min_x, max_x, 100)
y = c + m * x

plt.plot(x, y, color='#58b970', label='Regression Line')
plt.scatter(X, Y, c='#ef5423', label='data points')

plt.xlabel('Head Size in cm')
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()
```

plot1.py hosted with ❤ by GitHub view raw

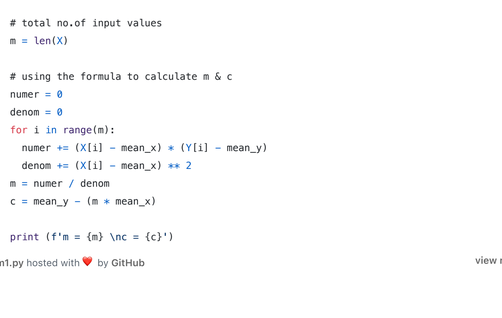


Image by Author

Step-6:

Now it's time to measure how good our model is. For this, as discussed above, we will calculate the R-squared value and evaluate our linear regression model. If you need a refresher on the formula of R-squared:

$$R^2 = \frac{\sum(y_{pred} - \bar{y})^2}{\sum(y - \bar{y})^2}$$

```
# calculating R-squared value for measuring goodness of our model.

ss_t = 0 #total sum of squares
ss_r = 0 #total sum of square of residuals

for i in range(int(val_count)): # val_count represents the no.of input x values
  y_pred = c + m * X[i]
  ss_t += (Y[i] - mean_y) ** 2
  ss_r += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_r/ss_t)

print(r2)
```

rsquared.py hosted with ❤ by GitHub view raw

The above code generates the R-squared value,

```
0.6393117199570003
```

And that was the linear regression implemented from scratch without using sklearn library.

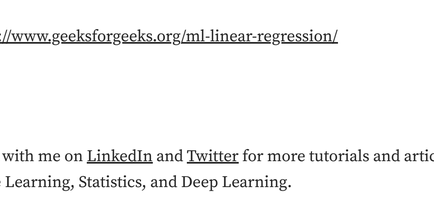


Image Source: Google

If you can't be bothered with all this mathematics and theory and would very much like to go for a neater method, sklearn library has an amazing inbuilt linear regressor function you can use. Here's the code snippet for that:

```
# Linear regression implementation using sklearn

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X = X.reshape(m, 1)
reg = LinearRegression()
reg = reg.fit(X,Y)

Y_pred = reg.predict(X)
r2_square = reg.score(X, Y)

print (r2_square)
```

linreg_sklearn.py hosted with ❤ by GitHub view raw

End Notes:

In this tutorial, we've learned the theory behind linear regression algorithm and also the implementation of the algorithm from scratch without using the inbuilt linear model from sklearn.

You can find the data and iPython notebook here on my GitHub.

References:

[1] Linear Regression by Edureka on YouTube

[2] https://www.geeksforgeeks.org/ml-linear-regression/

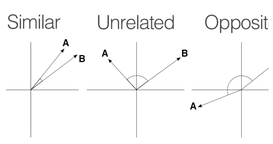
Connect with me on LinkedIn and Twitter for more tutorials and articles on Machine Learning, Statistics, and Deep Learning.

Machine Learning · Linear Regression · Statistics · Scikit Learn · Algorithms

Responses (3)

What are your thoughts?

Respond



In Geek Culture by Sindhu Seelam

Machine Learning Fundamentals: Cosine Similarity and Cosine...

Cosine similarity is a metric that measures the cosine of the angle between two vectors...

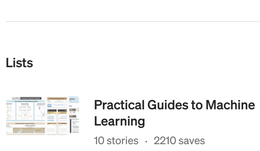
May 26, 2021 214 4

In Geek Culture by Hasitha Subhashana

Circuit Breaker Pattern (Design Patterns for Microservices)

In a distributed system we have no idea how other components would fail. Network issue...

Jun 12, 2021 834 10

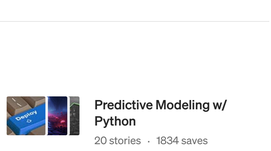


In Geek Culture by Anshul Borawake

React Native Generate APK—Debug and Release APK

Generate Debug and Release APK in React Native; Windows, iOS and Linux

Apr 3, 2021 1.8K 15



Sindhu Seelam

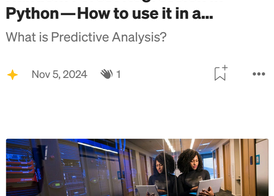
How To Improve Your Model's Performance Using Cross-...

Machine Learning models often fail to generalize well on data it has not been traine...

May 26, 2021 227

See all from Sindhu Seelam | See all from Geek Culture

Recommended from Medium



CodexRushi

Gradient Descent: Explanation with Python Code

Gradient descent is one of the algorithms considered to form the basis of machine...

Oct 24, 2024

Hugman Sangkeun Jung

Understanding Backpropagation and Vanishing Gradients

A Comprehensive Guide to Backpropagation and the Vanishing Gradient Problem

Nov 14, 2024 16

Lists

Practical Guides to Machine Learning
10 stories · 2210 saves

Predictive Modeling w/ Python
20 stories · 1834 saves

Natural Language Processing
1954 stories · 1601 saves

The New Chatbots: ChatGPT, Bard, and Beyond
12 stories · 553 saves

Rita Angelou

Random Forest Regression in Python—How to use it in a...

What is Predictive Analysis?

Nov 5, 2024 1

In CodeX by Apoorva

Cracking DSA Problems with Sliding Window: A Minimalist's...

Solving in Python

Nov 25, 2024 2

Saro

Scikit-learn in python-1

Hi,

Aug 25, 2024 14

ishan

Creating Our Own Linear Regression Algorithm From...

Languages used: Python

Jan 31

See more recommendations