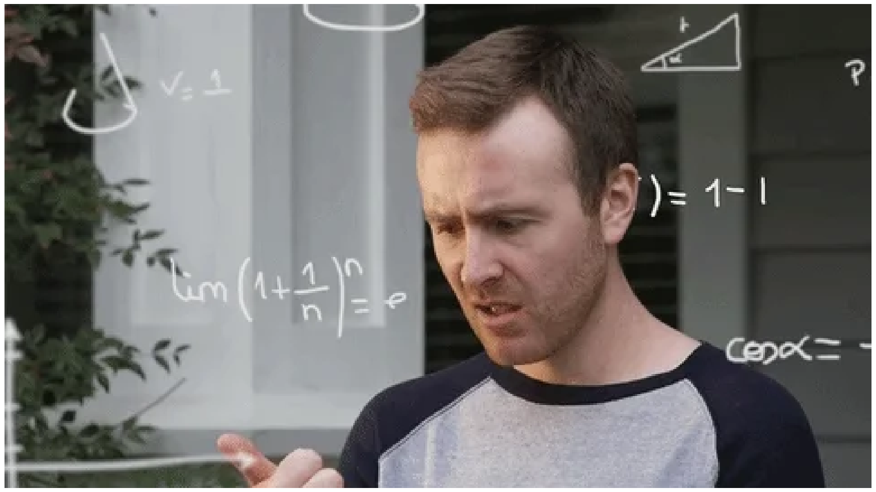
Introduction to Machine Learning Algorithms: Linear Regression

Build your own model from scratch

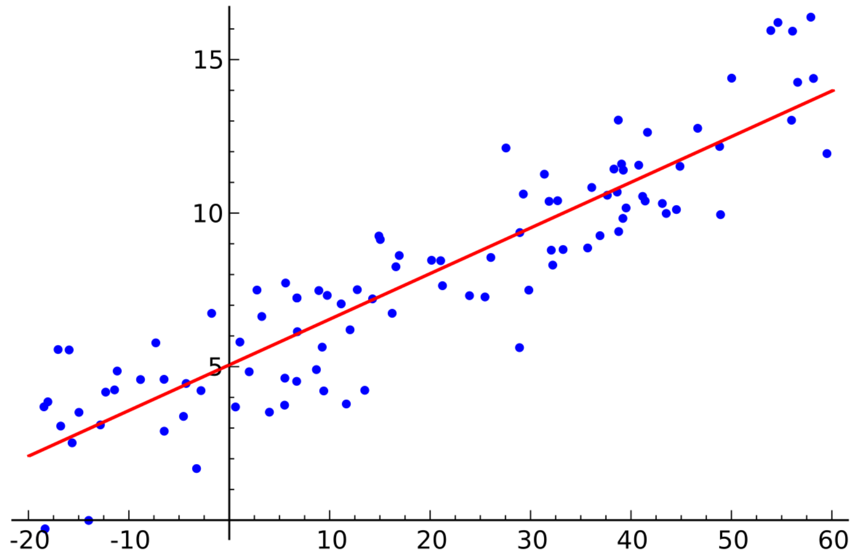
Rohith Gandhi Follow
May 27, 2018 · 7 min read

Artificial Intelligence has become prevalent recently. People across different disciplines are trying to apply AI to make their tasks a lot easier. For example, economists are using AI to predict future market prices to make a profit, doctors use AI to classify whether a tumor is malignant or benign, meteorologists use AI to predict the weather, HR recruiters use AI to check the resume of applicants to verify if the applicant meets the minimum criteria for the job, etcetera. The impetus behind such ubiquitous use of AI is machine learning algorithms. For anyone who wants to learn ML algorithms but hasn't gotten their feet wet yet, you are at the right place. The rudimental algorithm that every Machine Learning enthusiast starts with is a linear regression algorithm. Therefore, we shall do the same as it provides a base for us to build on and learn other ML algorithms.

What is linear regression??



Before knowing what is linear regression, let us get ourselves accustomed to regression. Regression is a method of modelling a target value based on independent predictors. This method is mostly used for forecasting and finding out cause and effect relationship between variables. Regression techniques mostly differ based on the number of independent variables and the type of relationship between the independent and dependent variables.



Linear Regression

Simple linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable. The red line in the above graph is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best. The line can be modelled based on the linear equation shown below.

```
y = a_0 + a_1 * x      ## Linear Equation
```

The motive of the linear regression algorithm is to find the best values for a_0 and a_1. Before moving on to the algorithm, let's have a look at two important concepts you must know to better understand linear regression.

Cost Function

The cost function helps us to figure out the best possible values for a_0 and a_1 which would provide the best fit line for the data points. Since we want the best values for a_0 and a_1, we convert this search problem into a minimization problem where we would like to minimize the error between the predicted value and the actual value.

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

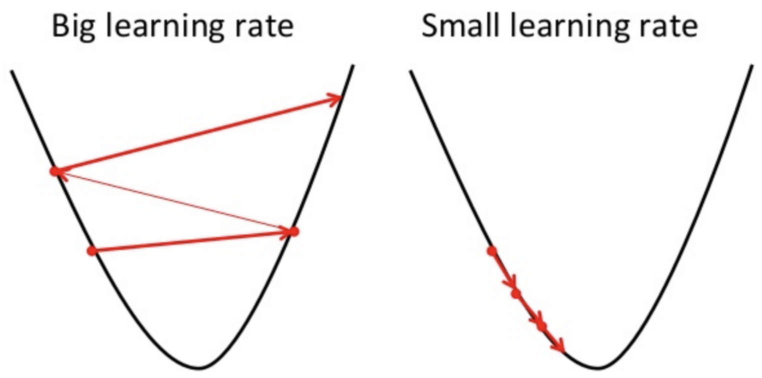
$$J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

Minimization and Cost Function

We choose the above function to minimize. The difference between the predicted values and ground truth measures the error difference. We square the error difference and sum over all data points and divide that value by the total number of data points. This provides the average squared error over all the data points. Therefore, this cost function is also known as the Mean Squared Error(MSE) function. Now, using this MSE function we are going to change the values of a_0 and a_1 such that the MSE value settles at the minima.

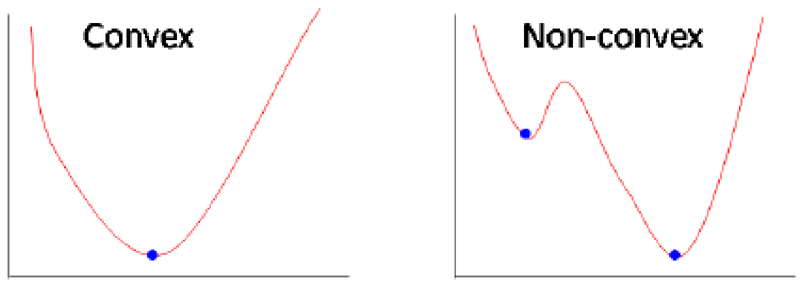
Gradient Descent

The next important concept needed to understand linear regression is gradient descent. Gradient descent is a method of updating a_0 and a_1 to reduce the cost function(MSE). The idea is that we start with some values for a_0 and a_1 and then we change these values iteratively to reduce the cost. Gradient descent helps us on how to change the values.



Gradient Descent

To draw an analogy, imagine a pit in the shape of U and you are standing at the topmost point in the pit and your objective is to reach the bottom of the pit. There is a catch, you can only take a discrete number of steps to reach the bottom. If you decide to take one step at a time you would eventually reach the bottom of the pit but this would take a longer time. If you choose to take longer steps each time, you would reach sooner but, there is a chance that you could overshoot the bottom of the pit and not exactly at the bottom. In the gradient descent algorithm, the number of steps you take is the learning rate. This decides on how fast the algorithm converges to the minima.



Convex vs Non-convex function

Sometimes the cost function can be a non-convex function where you could settle at a local minima but for linear regression, it is always a convex function.

You may be wondering how to use gradient descent to update a_0 and a_1. To update a_0 and a_1, we take gradients from the cost function. To find these gradients, we take partial derivatives with respect to a_0 and a_1. Now, to understand how the partial derivatives are found below you would require some calculus but if you don't, it is alright. You can take it as it is.

$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

$$J = \frac{1}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i)^2$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i) \implies \frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i) \cdot x_i \implies \frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

$$a_0 = a_0 - \alpha \cdot \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

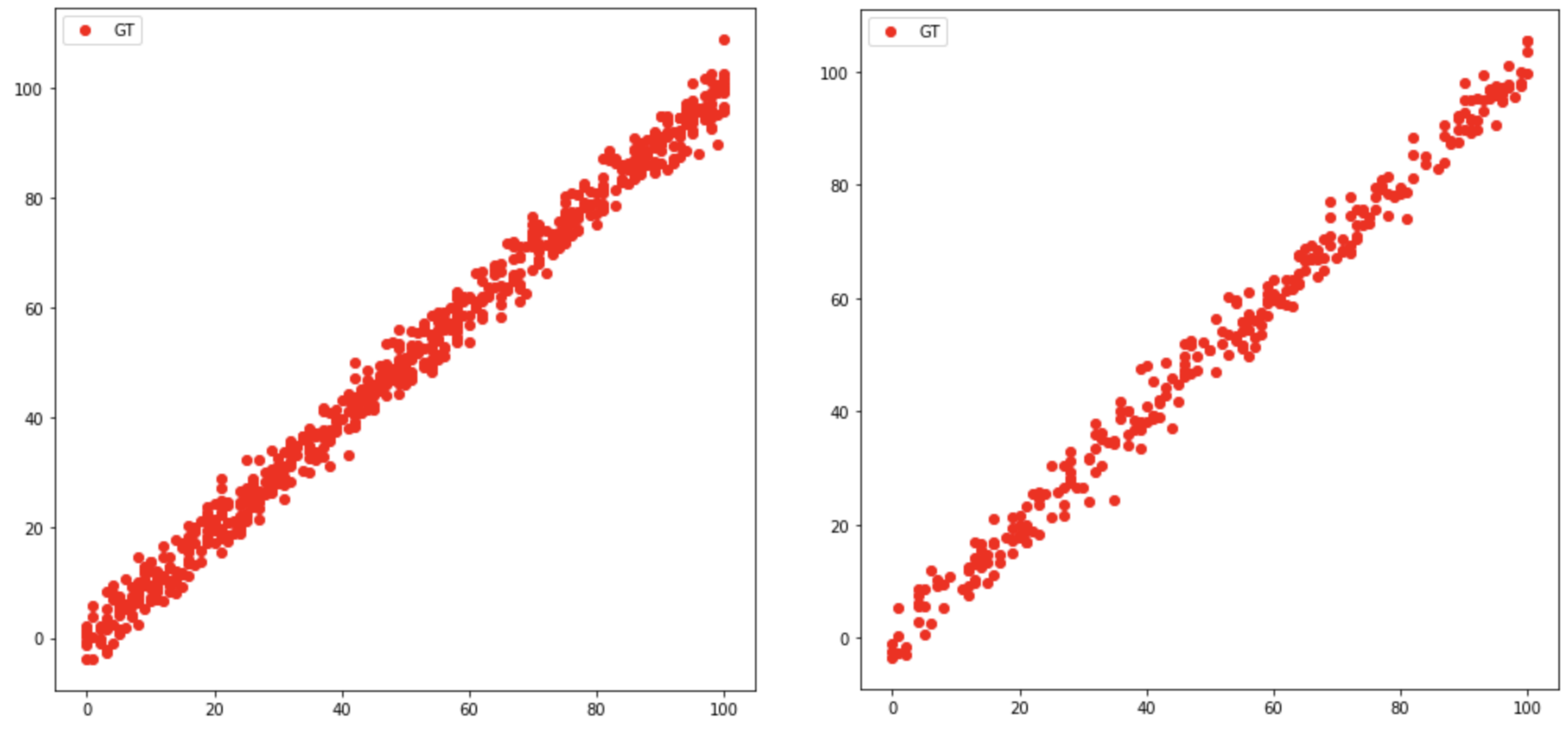
$$a_1 = a_1 - \alpha \cdot \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

The partial derivates are the gradients and they are used to update the values of a_0 and a_1. Alpha is the learning rate which is a hyperparameter that you must specify. A smaller learning rate could get you closer to the minima but takes more time to reach the minima, a larger learning rate converges sooner but there is a chance that you could overshoot the minima.

Code

Let’s get to the code. We have two choices, we can either use the scikit learn library to import the linear regression model and use it directly or we can write our own regression model based on the equations above. Instead of choosing one among the two, let’s do both :)

There are many datasets available online for linear regression. I used the one from this link. Let's visualise the training and testing data.



Training(left) and Testing(right) data

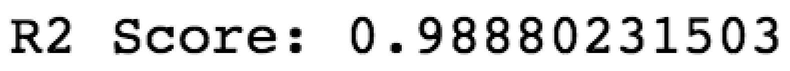
Let’s start with the easiest of the two methods, i.e using scikit learn library to build our linear regression model.

```python
import pandas as pd
import numpy as np

df_train = pd.read_csv('/Users/rohith/Documents/Datasets/Li
df_test = pd.read_csv('/Users/rohith/Documents/Datasets/Lin

x_train = df_train['x']
y_train = df_train['y']
x_test = df_test['x']
y_test = df_test['y']

x_train = np.array(x_train)
```

We use pandas library to read the train and test files. We retrieve the independent(x) and dependent(y) variables and since we have only one feature(x) we reshape them so that we could feed them into our linear regression model.

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

clf = LinearRegression(normalize=True)
clf.fit(x_train,y_train)
```

We use scikit learn to import the linear regression model. we fit the model on the training data and predict the values for the testing data. We use R2 score to measure the accuracy of our model.



```
R2 Score: 0.98880231503
```

R2 score on testing data

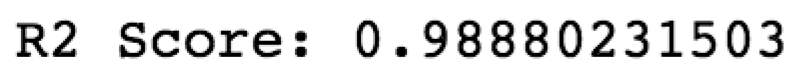
Now, let's build our own linear regression model from the equations above. We will be using only numpy library for the computations and the R2 score for metrics.

```
## Linear Regression
import numpy as np

n = 700
alpha = 0.0001

a_0 = np.zeros((n,1))
a_1 = np.zeros((n,1))

epochs = 0
while(epochs < 1000):
    y = a_0 + a_1 * x_train
    error = y - y_train
```

We initialize the value 0.0 for a_0 and a_1. For 1000 epochs we calculate the cost, and using the cost we calculate the gradients, and using the gradients we update the values of a_0 and a_1. After 1000 epochs, we would've obtained the best values for a_0 and a_1 and hence, we can formulate the best fit straight line.

```python
import matplotlib.pyplot as plt

y_prediction = a_0 + a_1 * x_test
print('R2 Score:',r2_score(y_test,y_prediction))

y_plot = []
for i in range(100):
    y_plot.append(a_0 + a_1 * i)
plt.figure(figsize=(10,10))
```

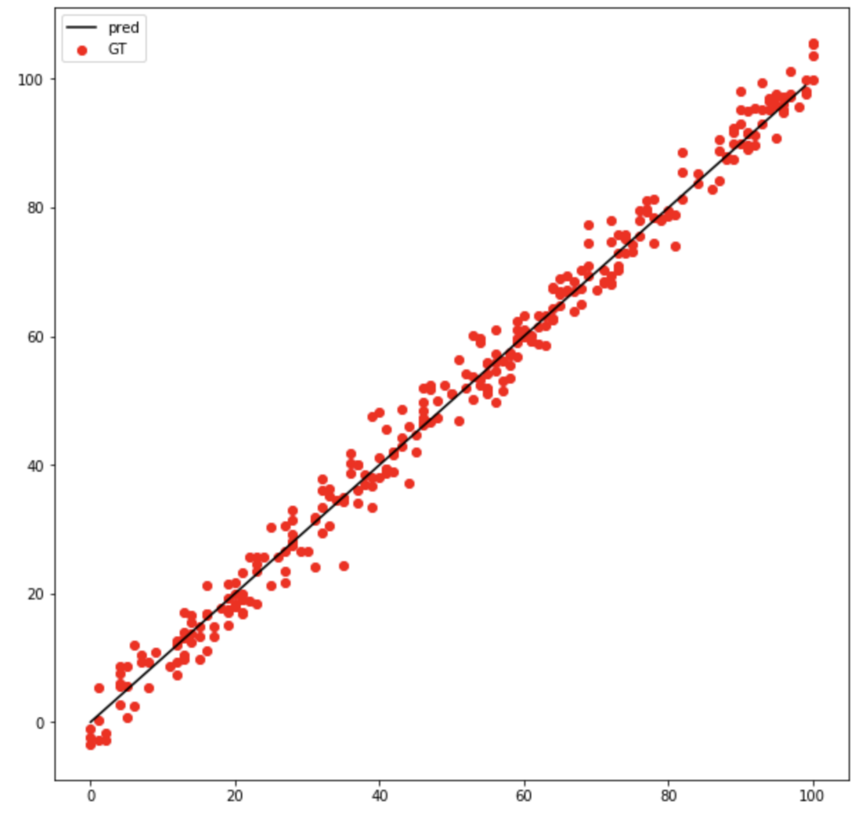
The test set contains 300 samples, therefore we have to reshape a_0 and a_1 from 700x1 to 300x1. Now, we can just use the equation to predict values in the test set and obtain the R2 score.



```
R2 Score: 0.98880231503
```

R2 score on testing data

We can observe the same R2 score as the previous method. We also plot the regression line along with the test data points to get a better visual understanding of how good our algorithm works.



Regression line — Test data

Conclusion

Linear Regression is an algorithm that every Machine Learning enthusiast must know and it is also the right place to start for people who want to learn Machine Learning as well. It is really a simple but useful algorithm. I hope this article was helpful to you.