## [Introduction](http://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html" \l "id12)

Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It’s used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog). There are two main types:

**Simple regression**

Simple linear regression uses traditional slope-intercept form, where m and b are the variables our algorithm will try to “learn” to produce the most accurate predictions. xx represents our input data and yy represents our prediction.

y=mx+by=mx+b

## [Simple regression](http://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#id13)

Let’s say we are given a [dataset](http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv) with the following columns (features): how much a company spends on Radio advertising each year and its annual Sales in terms of units sold. We are trying to develop an equation that will let us to predict units sold based on how much a company spends on radio advertising. The rows (observations) represent companies.

|  |  |  |
| --- | --- | --- |
| **Company** | **Radio ($)** | **Sales** |
| Amazon | 37.8 | 22.1 |
| Google | 39.3 | 10.4 |
| Facebook | 45.9 | 18.3 |
| Apple | 41.3 | 18.5 |

### [Making predictions](http://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#id14)

Our prediction function outputs an estimate of sales given a company’s radio advertising spend and our current values for Weight and Bias.

Sales=Weight⋅Radio+Bias

**Weight**

the coefficient for the Radio independent variable. In machine learning we call coefficients weights.

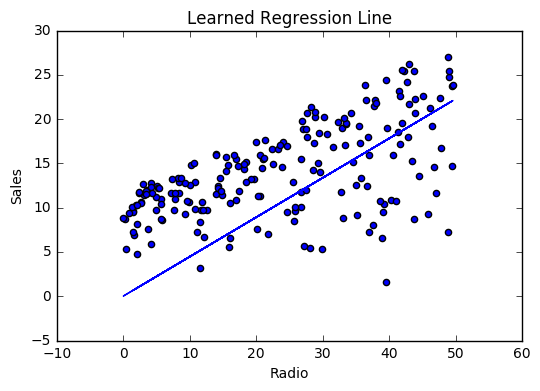
**Radio**

the independent variable. In machine learning we call these variables features.

**Bias**

the intercept where our line intercepts the y-axis. In machine learning we can call intercepts bias. Bias offsets all predictions that we make.

Our algorithm will try to learn the correct values for Weight and Bias. By the end of our training, our equation will approximate the line of best fit.



**Code**

**def** predict\_sales(radio, weight, bias):

**return** weight\*radio + bias

### [Cost function](http://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#id15)

The prediction function is nice, but for our purposes we don’t really need it. What we need is a [cost function](http://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html) so we can start optimizing our weights.

Let’s use [MSE (L2)](http://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#mse) as our cost function. MSE measures the average squared difference between an observation’s actual and predicted values. The output is a single number representing the cost, or score, associated with our current set of weights. Our goal is to minimize MSE to improve the accuracy of our model.

**Math**

Given our simple linear equation y=mx+by=mx+b, we can calculate MSE as:

MSE=1N∑i=1n(yi−(mxi+b))2MSE=1N∑i=1n(yi−(mxi+b))2

**Note**

* NN is the total number of observations (data points)
* 1N∑ni=11N∑i=1n is the mean
* yiyi is the actual value of an observation and mxi+bmxi+b is our prediction

**Code**

**def** cost\_function(radio, sales, weight, bias):

companies = len(radio)

total\_error = 0.0

**for** i **in** range(companies):

total\_error += (sales[i] - (weight\*radio[i] + bias))\*\*2

**return** total\_error / companies

### [Gradient descent](http://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#id16)

To minimize MSE we use [Gradient Descent](http://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html) to calculate the gradient of our cost function. [TODO: Slightly longer explanation].

**Math**

There are two [parameters](http://ml-cheatsheet.readthedocs.io/en/latest/glossary.html#glossary-parameters) (coefficients) in our cost function we can control: weight mm and bias bb. Since we need to consider the impact each one has on the final prediction, we use partial derivatives. To find the partial derivatives, we use the [Chain rule](http://ml-cheatsheet.readthedocs.io/en/latest/calculus.html#chain-rule). We need the chain rule because (y−(mx+b))2(y−(mx+b))2 is really 2 nested functions: the inner function y−mx+by−mx+b and the outer function x2x2.

Returning to our cost function:

f(m,b)=1N∑i=1n(yi−(mxi+b))2f(m,b)=1N∑i=1n(yi−(mxi+b))2

We can calculate the gradient of this cost function as:

f′(m,b)=⎡⎣dfdmdfdb⎤⎦=[1N∑−2xi(yi−(mxi+b))1N∑−2(yi−(mxi+b))]f′(m,b)=[dfdmdfdb]=[1N∑−2xi(yi−(mxi+b))1N∑−2(yi−(mxi+b))]

**Code**

To solve for the gradient, we iterate through our data points using our new weight and bias values and take the average of the partial derivatives. The resulting gradient tells us the slope of our cost function at our current position (i.e. weight and bias) and the direction we should update to reduce our cost function (we move in the direction opposite the gradient). The size of our update is controlled by the [learning rate](http://ml-cheatsheet.readthedocs.io/en/latest/glossary.html#glossary-learning-rate).

**def** update\_weights(radio, sales, weight, bias, learning\_rate):

weight\_deriv = 0

bias\_deriv = 0

companies = len(radio)

**for** i **in** range(companies):

*# Calculate partial derivatives*

*# -2x(y - (mx + b))*

weight\_deriv += -2\*radio[i] \* (sales[i] - (weight\*radio[i] + bias))

*# -2(y - (mx + b))*

bias\_deriv += -2\*(sales[i] - (weight\*radio[i] + bias))

*# We subtract because the derivatives point in direction of steepest ascent*

weight -= (weight\_deriv / companies) \* learning\_rate

bias -= (bias\_deriv / companies) \* learning\_rate

**return** weight, bias

### [Training](http://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#id17)

Training a model is the process of iteratively improving your prediction equation by looping through the dataset multiple times, each time updating the weight and bias values in the direction indicated by the slope of the cost function (gradient). Training is complete when we reach an acceptable error threshold, or when subsequent training iterations fail to reduce our cost.

Before training we need to initializing our weights (set default values), set our [hyperparameters](http://ml-cheatsheet.readthedocs.io/en/latest/glossary.html#glossary-hyperparameters)(learning rate and number of iterations), and prepare to log our progress over each iteration.

**Code**

**def** train(radio, sales, weight, bias, learning\_rate, iters):

cost\_history = []

**for** i **in** range(iters):

weight,bias = update\_weights(radio, sales, weight, bias, learning\_rate)

*#Calculate cost for auditing purposes*

cost = cost\_function(features, targets, weights)

cost\_history.append(cost)

*# Log Progress*

**if** i % 10 == 0:

print "iter: "+str(i) + " cost: "+str(cost)

**return** weight, bias, cost\_history

### [Model evaluation](http://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#id18)

If our model is working, we should see our cost decrease after every iteration.

**Logging**

iter=1 weight=.03 bias=.0014 cost=197.25

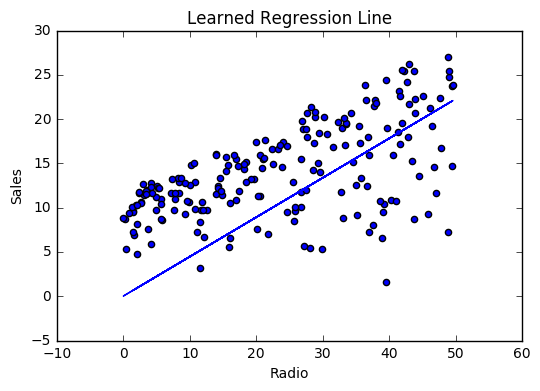
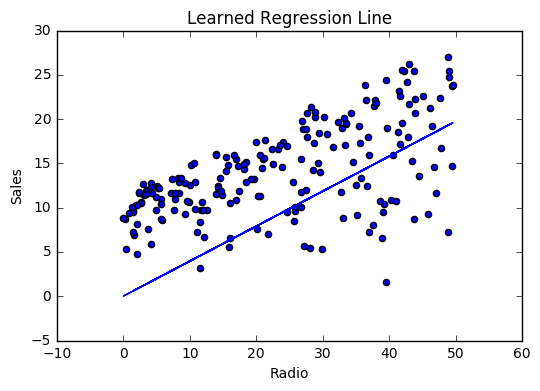
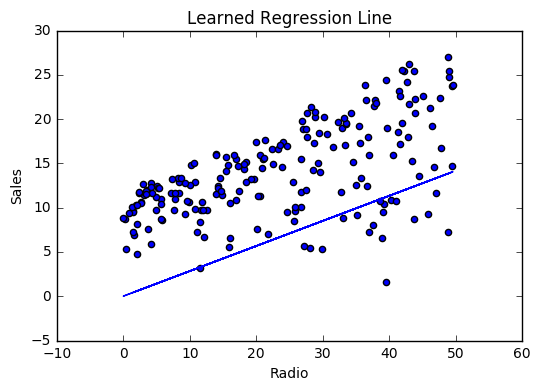
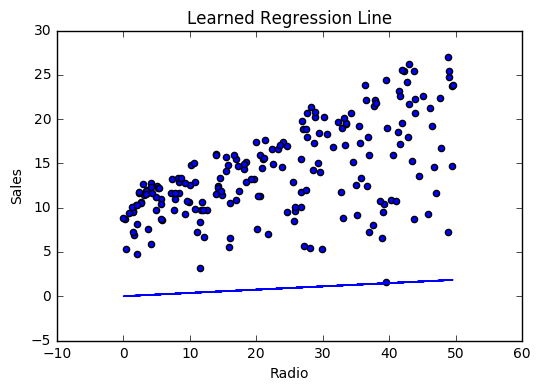
iter=10 weight=.28 bias=.0116 cost=74.65

iter=20 weight=.39 bias=.0177 cost=49.48

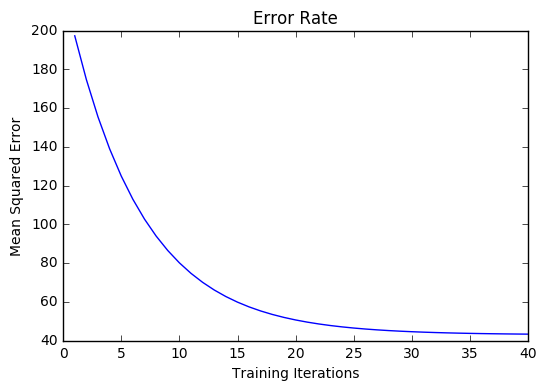
iter=30 weight=.44 bias=.0219 cost=44.31

iter=30 weight=.46 bias=.0249 cost=43.28

**Visualizing**



**Cost history**



### [Summary](http://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#id19)

By learning the best values for weight (.46) and bias (.25), we now have an equation that predicts future sales based on radio advertising investment.

Sales=.46Radio+.025Sales=.46Radio+.025

How would our model perform in the real world? I’ll let you think about it :)