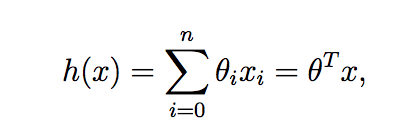
How to Implement Linear Regression

h(x) is the function for linear regression. It is the sum of Thetai times xi where n is the number of Input variables.



For this implementation I will use python. Let’s start by defining a fit function where X and y are in the input and output variables

def fit(self, X, y):

X = np.insert(X, 0, 1, axis=1)

self.initialize\_weights(n\_features=X.shape[1])

Lets start by defining our htheta(xi) for each iteration

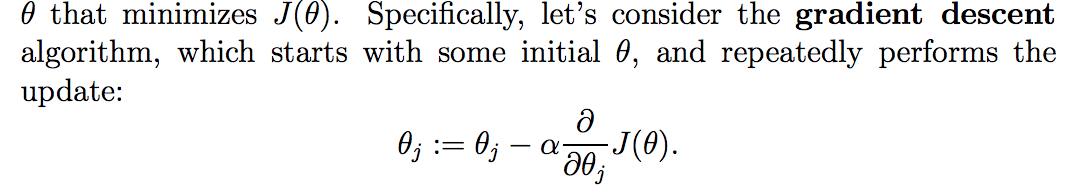
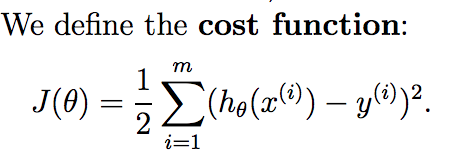
for i in range(self.n\_iterations):

y\_pred = X.dot(self.w)

x is the input xi, while self.w is the theta,i. We do the dot product so that we can find a sum of the products

We want to find the optimum Weight (theta) so we need to find a Theta that minimizes the Cost, the J(Theta)

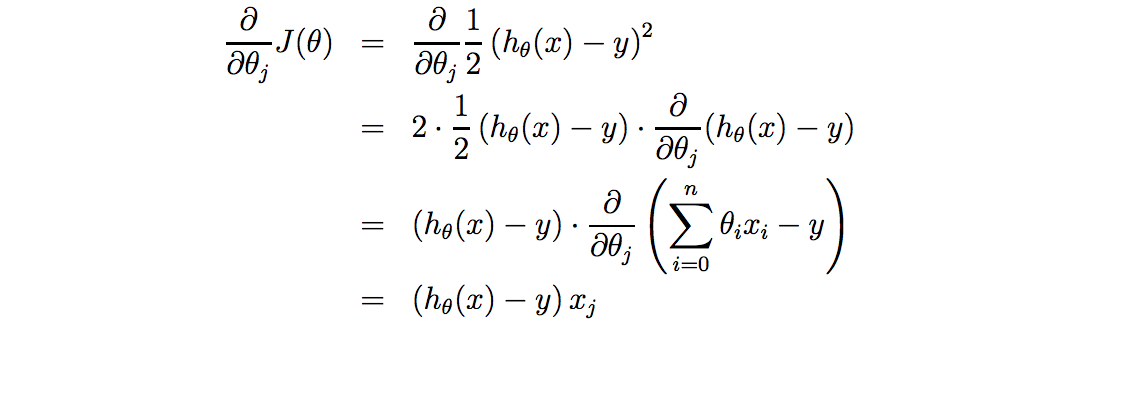
As we want to choose θ so as to minimize J(θ), let’s use a search algorithm that starts with some “initial guess” for θ, and that repeatedly changes θ to make J(θ) smaller, until hopefully we converge to a value of

Lets define the gradient descent algorithm. Self.learning rate is the alpha and grad\_w is the derivative of J(theta)

self.w -= self.learning\_rate \* grad\_w

In order to implement this algorithm, we have to work out what is the partial derivative term on the right hand side. Let’s first work it out for the case of if we have only one training example (x, y), so that we can neglect the sum in the definition of J. We have:

Here we need to code what grad\_w is.

grad\_w = (y\_pred - y).dot(X)

y\_pred is htheta(x). We do the dot product to multiply them

By the end of this loop for n\_iterations we will have updated the Theta or self.w

Here is the full code for the fitting function

def fit(self, X, y):

""" Insert constant ones for bias weights """

X = np.insert(X, 0, 1, axis=1)

self.initialize\_weights(n\_features=X.shape[1])

for i in range(self.n\_iterations):

""" See h(x): LinearRegression.png

self.w = Theta -^ i

X = x -^ i

Dot product sums the the product of each weight(theta) times the input(x)

h,theta(x^i)

"""

y\_pred = X.dot(self.w)

""" Least Squares Cost Function:

See LMS\_one\_example.png

y\_pred = h,theta(x)

y = y

.dot(X) = x -^j

"""

grad\_w = (y\_pred - y).dot(X)

""" Update the Weights: Gradient Descent.png

learning\_rate = alpha

grad\_w = derivative of J -^ Theta

"""

self.w -= self.learning\_rate \* grad\_w

To predict we calculate htheta(x) with the updated weight

def predict(self, X):

""" Insert constant ones for bias weights """

X = np.insert(X, 0, 1, axis=1)

"""

Calculate htheta(x) with updated weight

Dot product, sums the the product of each weight(theta) times the input(x)

h,theta(x^i)

"""

y\_pred = X.dot(self.w)

return y\_pred