**Parthivi Choubey**

**Progress Report 1 (10/12/18)**

Machine Learning: -

Andrew Ng course from Coursera

Completed all videos, readings and quizzes till week 3.

Week 2 & 3 programming assignments completed.

Univariate linear regression:

A pair (x^{(i)} , y^{(i)} )(x(i),y(i)) is called a training example, and the dataset that we’ll be using to learn—a list of m training examples (x(i),y(i));i=1,...,m—is called a training set. Given a training set, to learn a function h : X → Y so that h(x) is a “good” predictor for the corresponding value of y. This function h is called a hypothesis.

When the target variable that we’re trying to predict is continuous, we call the learning problem a regression problem. When y can take on only a small number of discrete values, we call it a classification problem.

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average difference of all the results of the hypothesis with inputs from x's and the actual output y's.

COST : J(θ0​,θ1​) = (1/2m) ​i=1∑m​ (y^​i​−yi​)2 = (1/2m) ​i=1∑m (hθ​(xi​)−yi​)^2

Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. Ideally, the line should pass through all the points of our training data set. In such a case, the value of J(θ0​,θ1​) will be 0.



We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum. The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter α, which is called the learning rate. A smaller α would result in a smaller step and a larger α results in a larger step. The direction in which the step is taken is determined by the partial derivative of J(θ0​,θ1​). Depending on where one starts on the graph, one could end up at different points.

The gradient descent algorithm is:

repeat until convergence:

θj​ := θj − α (∂/∂θj)​​J(θ0​,θ1​)

where j=0,1 represents the feature index number.

At each iteration j, one should simultaneously update the parameters \theta\_1, \theta\_2,...,\theta\_nθ1​,θ2​,...,θn​. Updating a specific parameter prior to calculating another one on the j^{(th)}j(th) iteration would yield to a wrong implementation.



Gradient descent always converges (assuming the learning rate α is not too large) to the global minimum. Indeed, J is a convex quadratic function.

Multivariate linear regression:

hθ​(x) = θ0​ + θ1​x1 ​+ θ2​x2 ​+ θ3​x3 ​+ ⋯ + θn​xn = θ’ x

repeat until convergence:{

θj := θj – α/m i=1 ∑ m (hθ(x(i))−y(i))⋅x(i)j

for j := 0...n }

Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero. To implement both of these techniques, adjust your input values as shown in this formula:

xi​ := (​xi​−μi)/ si​​​

Where μi​ is the **average** of all the values for feature (i) and si​ is the range of values (max - min), or si​ is the standard deviation.

**Debugging gradient descent.** Make a plot with number of iterations on the x-axis. Now plot the cost function, J(θ) over the number of iterations of gradient descent. If J(θ) ever increases, then you probably need to decrease α. It has been proven that if learning rate α is sufficiently small, then J(θ) will decrease on every iteration.

To summarize:

If α is too small: slow convergence.

If α is too large: ￼may not decrease on every iteration and thus may not converge.

We can improve our features and the form of our hypothesis function in a couple different ways.

We can combine multiple features into one. For example, we can combine x1​ and x2​ into a new feature x3​ by taking x1​⋅x2​.

Polynomial Regression

Our hypothesis function need not be linear (a straight line) if that does not fit the data well.

We can change the behavior or curve of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).

For example, if our hypothesis function is hθ​(x)=θ0​+θ1​x1​ then we can create additional features based on x1​, to get the quadratic function hθ​(x)=θ0​+θ1​x1​+θ2​x1​^2 or the cubic function  hθ​(x)=θ0​+θ1​x1​+θ2(​x1)^2+θ3(​x1)^3​

In the cubic version, we have created new features x2​ and x3​ where  x2​=x1​^2 and x3​=x1^3.

Normal Equation:

In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θj ’s, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:

θ=(XTX)-1XTy

There is no need to do feature scaling with the normal equation.

If XTX is noninvertible, the common causes might be having :

* Redundant features, where two features are very closely related (i.e. they are linearly dependent)
* Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization"

Classification:

To attempt classification, one method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. However, this method doesn't work well because classification is not actually a linear function.

The classification problem is just like the regression problem, except that the values we now want to predict take on only a small number of discrete values.

Binary classification

y∈{0,1}. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols “-” and “+.”

hθ(x)=g(θTx)

z= θTx

g(z)=1/(1+e-z)

"Sigmoid Function," also called the "Logistic Function":



hθ(x) = P(y=1|x;θ) = 1−P(y=0|x;θ)

P(y=0|x;θ)+P(y=1|x;θ) = 1

hθ(x)≥0.5 → y=1 when θTx≥0

hθ(x)<0.5 → y=0 when θTx<0

The **decision boundary** is the line that separates the area where y = 0 and where y = 1.

Cost:

J(θ)=1m∑i=1mCost(hθ(x(i)),y(i))

Cost(hθ(x),y)=−log(hθ(x)) if y=1

Cost(hθ(x),y)=−log(1−hθ(x)) if y = 0

We can fully write out our entire cost function as follows:

J(θ)=−(1/m) ​i=1∑m​ [y(i)log(hθ​(x(i)))+(1−y(i))log(1−hθ​(x(i)))]

Gradient descent:

Repeat{

θj:=θj−(α/m)∑i=1 m (hθ(x(i))−y(i)).x(i)j

}