Supervised Learning

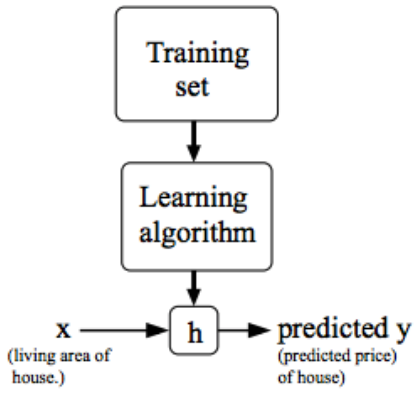
- We are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.
- "right answers" given.
- Regression problem: predict results with a continuous valued output (map input values to some continuous function).
- Classification problem: predict results in a discrete valued output (0, 1, 2, 3). Map input variables into discrete categories.
- Training Set
  - (x, y) : one training example
  - (x^i, y^i) : ith training example

Unsupervised Learning

- Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.
- We can derive this structure by clustering the data based on relationships among the variables in the data.
- With unsupervised learning there is no feedback based on the prediction results.
- Clustering: Take a collection of data and find a way to automatically group this data into groups that are somehow similar or related by different variables.
- Non-clustering: find structure in a chaotic environment

Notation:
m = number of training examples
x = input variable/feature
y = output variable/target variable
(x, y) = one training example
(x^i, y^i) = ith training example

Our goal is, given a training set, to learn function
h : X → Y so that h(x) is a "good" predictor for the corresponding value of y. For historical reasons, 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.

Linear regression:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Cost Function

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

$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^i) - y^i\right)^2$$

cost function

*Square error cost function

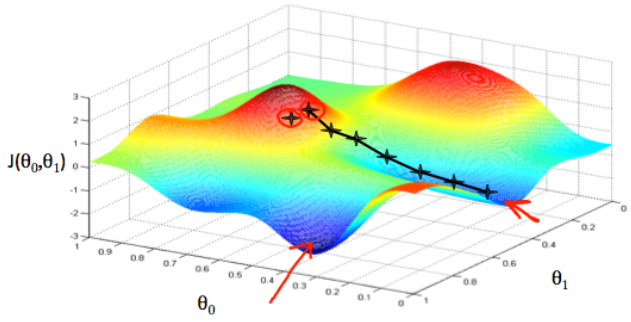
Goal: minimize θ_0, θ_1

To break it apart, it is $\frac{1}{2}\bar{x}$ where $\bar{x}$ is the mean of the squares $h_\theta(x_i)-y_i$ or the difference between the predicted value and the actual value.
This function is otherwise called the "Squared error function" or "Mean squared error". The mean is halved as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the ½ term.

Our objective is to get the best possible line. The best possible line will be such that the average squared vertical distances of the scattered points from the line will be the least.

So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in the hypothesis function. That's where gradient descent comes in. Imagine that we graph our hypothesis function based on its fields θ_0 and θ_1 (actually we are graphing the cost function as a function of the parameter estimates). We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters.

We put θ_0 on the x axis and θ_1 on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters. The graph depicts such a setup.



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 red arrows show the minimum points in the graph.

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.

For example, the distance between each 'star' in the graph above represents a step determined by our parameter α. 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 $J(\theta_0,\theta_1)$. Depending on where one starts on the graph, one could end up at different points. The image above shows us two different starting points that end up in two different places.

The gradient descent algorithm:

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{for } j = 0 \text{ and } j = 1$$ (feature index number)

}

* Simultaneously update θ_0 and θ_1

Gradient descent automatically converges towards a minimum value. If the derivative (slope) is negative, the value of θ_1 increases. When the derivative (slope) is positive, the value of θ_1 decreases.

As the gradient descent approaches a local minimum, the step inherently gets smaller and smaller since the derivative is approaching zero. At the local minimum, the derivative is zero.

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) \quad \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) \cdot x^i \quad \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

}

* update θ_0 and θ_1 Simultaneously

If we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.
This is simply gradient descent on the original cost function J. This method looks at every example in the entire training set on every step, and is called batch gradient descent.

Linear Algebra

A x B matrix, A rows, B columns

Vector = n x 1 matrix
Matrix addition: matrices must have same dimensions

Matrix-Vector Multiplication

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}} \quad \underset{v}{\begin{bmatrix} x \\ y \end{bmatrix}} = \underset{A \times v}{\begin{bmatrix} ax+by \\ cx+dy \\ ex+fy \end{bmatrix}}$$

$3 \times \underline{2} \quad \underline{2} \times 1 \quad 3 \times 1$ (must match)

A_{ij} = element in i^{th} row, j^{th} column

Data values:

a
b
c
d

$\xrightarrow{\text{matrix}}$ $\begin{bmatrix} 1 & a \\ 1 & b \\ 1 & c \\ 1 & d \end{bmatrix}$

$h_\theta = \theta_0 + \theta_1 x$

$\begin{bmatrix} 1 & a \\ 1 & b \\ 1 & c \\ 1 & d \end{bmatrix} \times \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} \theta_0 + \theta_1 a \\ \theta_0 + \theta_1 b \\ \theta_0 + \theta_1 c \\ \theta_0 + \theta_1 d \end{bmatrix}$

prediction = Data matrix x parameters

Matrix-Matrix Multiplication

$A \times B = C$

$m \times n \quad n \times o \quad m \times o$

The i^{th} column of the matrix C is obtained by multiplying A with the i^{th} column of B.

$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} * \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw+by & ax+bz \\ cw+dy & cx+dz \\ ew+fy & ex+fz \end{bmatrix}$

Data values:

a
b
c
d

Competing hypothesis:

$h_\theta(x) = \theta_{0u} + \theta_{1u} x$

$h_\theta(x) = \theta_{0v} + \theta_{1v} x$

$h_\theta(x) = \theta_{0w} + \theta_{1w} x$

prediction = $\begin{bmatrix} 1 & a \\ 1 & b \\ 1 & c \\ 1 & d \end{bmatrix} \times \begin{bmatrix} \theta_{0u} & \theta_{0v} & \theta_{0w} \\ \theta_{1u} & \theta_{1v} & \theta_{1w} \end{bmatrix}$

Matrix Multiplication Properties:

Let A and B be matrices. Then in general, A x B $\neq$ B x A. (not commutative)

A x (B x C) = (A x B) x C (associative)

Identity Matrix I (or $I_{n \times n}$)

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \ldots$

A x I = I x A

Matrix Inverse:

If A is an m x m matrix, and if it has an inverse:

$A(A^{-1}) = A^{-1}A = I$

Matrix Transpose:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \qquad A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} \qquad A^T_{ij} = A_{ji}$$