Machine Learning by Stanford University

Study Sheet by Reah Miyara ✉ mail@reah.me

Intro to Machine Learning

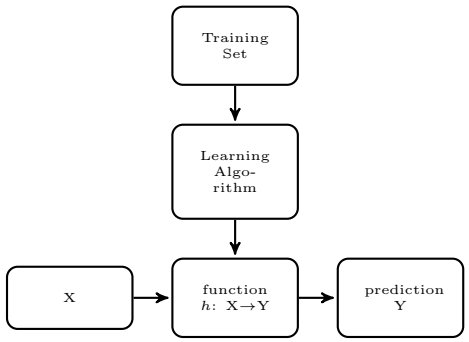
• **ML** – a computer program with increased performance P at some class of tasks T with experience E.
• **Supervised** – given a ['ground truth'] data set, predict output given the input. Types of prediction:

1. **Regression** – continuous, numerical
2. **Classification** – discrete, categorical

• **Unsupervised** – derive structure from data based on relationships among variables (with no prior knowledge as to what the results should look like)

Linear Regression with One Variable

• **Learning Goal** – given a training set, learn a function h: X→Y so $h(x)$ is a good y predictor



• **Hypothesis** – $h_\theta(x) = \theta_0 + \theta_1 x$
• **Cost Function** – takes an average difference of all results of the hypothesis with inputs from the x values and the actual y values. Goal: minimize θ_0, θ_1

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \qquad (1)$$

(1) Squared Error function or Mean Squared Error function

• **Gradient Descent Algorithm** repeat until convergence

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (2)$$

Multivariate Linear Regression

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

• **Gradient Descent Algorithm** repeat until convergence

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)}) \cdot x_j^{(i)} \ ; \ \text{j:= 0...n} \qquad (3)$$

• **Feature Scaling** – divide the input values by the range (max – min). Input values in roughly the same range speed up the convergence of gradient descent.
• **Mean Normalization** – subtract the mean for an input variable from the values for that input variable.

$$x_i := \frac{x_i - \mu_i}{s_i} \qquad (4)$$

(4) μ_i is the mean & s_i is the range, $(max - min)$, of all values for feature i

• **Learning Rate** – α too small $\implies$ slow convergence; α too large $\implies$ may not converge.

Normal Equation

• **Normal Equation** – non-iterative algorithm for minimizing $J(\theta)$; *note* : $O(n^3)$ to calculate $X^T X$

$$\theta = (X^T X)^{-1} X^T y \qquad (5)$$

m examples $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$; n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \ | \ \text{X} = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(n)})^T \end{bmatrix} \ | \ \text{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

$x^{(i)}$ = training vector i (containing values from all features); $X \rightarrow m$x(n+1)

• **If $\mathbf{X^T X}$ is noninvertible**, common causes include:

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

Classification Logistic Regression

• **Logistic Regression Model** – want $0 \leq h_\theta(x) \leq 1$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad (6)$$

• **Sigmoid/Logistic Function**

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T(x)}} \qquad (7)$$

$$h_\theta(x) = P(y = 1 | x; \theta) = 1 - P(y = 0 | x; \theta) \qquad (8)$$

• **Decision Boundary**

1. y = 1 if $\theta^T(x) \geq 0$
2. y = 0 if $\theta^T(x) < 0$

• **Cost Function**

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases} \qquad (9)$$

$$Cost(h_\theta(x), y) = -y log(h_\theta(x)) - (1-y) log(1 - h_\theta(x)) \qquad (10)$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) \qquad (11)$$

a vectorized implementation is:

$$h = g(X\theta); \quad J(\theta) = \frac{1}{m} \cdot (-y^T log(h) - (1-y)^T log(1-h)) \qquad (12)$$

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad (13)$$

a vectorized implementation is:

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y}) \qquad (14)$$

Multiclass Classification

$$\text{y} \in 0, 1, \dots n$$
$$\text{h}_\theta^{(0)}(x) = \text{P(y=0|x;}\theta\text{)}$$
$$\text{h}_\theta^{(1)}(x) = \text{P(y=1|x;}\theta\text{)}$$
$$\vdots$$
$$\text{h}_\theta^{(n)}(x) = \text{P(y=n|x;}\theta\text{)}$$
$$\text{prediction} = \max_i (h_\theta^{(i)}(x))$$

Under/Over –fitting

• **Underfitting** – hypothesis shows structure not captured by the model; does not fit the data well

• **Overfitting** – hypothesis corresponds too closely to data ∴ fail to predict future results reliably

• Addressing the problem of overfitting:

1. **Reduce the number of features** – manually/algorithmically select subset of features.
2. **Regularization** – keep all features, but reduce the magnitude of parameters θ_j by λ (regularization parameter).

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right] \quad (15)$$

Gradient Descent regularized linear regression does not penalize θ_0

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot x_0^{(i)} \quad (16)$$

$$\theta_j := \theta_j - \alpha\left[\left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right] \quad (17)$$

$$\theta_j := \theta_j(1-\alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot x_j^{(i)} \quad (18)$$

Normal Equation

$$\theta = (X^TX + \lambda\cdot L)^{-1}X^Ty \quad (19)$$

where L = $\begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$

Neural Networks

• **Activation Function**

$$h_\theta(x) = g(\theta^Tx) = \frac{1}{1+e^{-\theta^T(x)}} \quad (20)$$

$a_j^{(i)}$ – 'activation' of unit i in layer j

$\Theta^{(j)}$ – matrix of weights controlling function mapping from layer j to layer $j+1$

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} \rightarrow h_\theta(x)$$

$h_\theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$

Each layer gets its own matrix of weights, $\Theta^{(j)}$. If network has s_j units in layer j and s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1}\times(s_j+1)$.

• **Forward Propagation:** vectorized implementation –
for layer $j=2$ & node k, the variable z will be:

$$z_k^{(2)} = \Theta_{k,0}^{(1)}x_0 + \Theta_{k,1}^{(1)}x_1 + \Theta_{k,n}^{(1)}x_n$$

therefore it follows that:

$$a_1^{(2)} = g(z_1^{(2)}),\ a_2^{(2)} = g(z_2^{(2)}),\ a_3^{(2)} = g(z_2^{(3)})$$

a vectorized representation of x and $z^{(j)}$ is:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}\quad z^{(j)} = \begin{bmatrix} z_1^j \\ z_2^j \\ \vdots \\ z_n^j \end{bmatrix}$$

setting $x = a^{(1)}$ we can write it as $z^{(j)} = \Theta^{(j-1)}a^{(j-1)}$ $a^{(j)} = g(z^{(j)})$ where g is applied element-wise to our vector $z^{(j)}$ We can then add a bias unit (equal to 1) to layer j after we have computed $a^{(j)}$. This will be $a_0^{(j)}$ and will be equal to 1. To compute our final hypothesis, the last theta matrix $\Theta_{(j)}$ will have only one row which is multiplied by one column $a^{(j)}$ so that our result is a single number. Final result: $h_\Theta(x) = a^{(j+1)} = g(z^{(j+1)})$

Neural Networks

• **Cost Function & Backpropagation**

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[y_k^{(i)}log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})log(1-(h_\Theta(x^{(i)}))_k)] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2 \quad (21)$$

'Backpropagation' is neural-network term for minimizing our cost function. Given a training set $\{(x^{(1)}, y^{(1)})\cdots(x^{(m)}, y^{(m)})\}$:

$\Delta_{i,j}^{(l)} := 0$ for all (l,i,j), (end up with a matrix full of zeros)

for training example t=1 to m:

1. Set $a^{(1)} := x^{(t)}$
2. Perform forward propagation to compute $a^{(l)}$ for l = 2,3,…,L
3. Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$
   L is the total number of layers & $a^{(L)}$ is the vector of outputs of the activation units for the last later. So our δ [error values], are the differences of the actual results in the last layer & the correct outputs in y. For the delta values of the layers before last, use the following equation that steps back from right to left:
4. $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ w/ $\delta^{(l)} = ((\Theta^{(l)})^T\delta^{(l+1)}).\star a^{(l)}.\star(1-a^{(l)})$ delta values of layer l are calculated by multiplying the delta values in the next layer with the theta matrix of layer l. Next, perform element-wise multiplication with function g', the derivative of the activation function g evaluated with the input values given by $z^{(l)}$. $g'(z^{(l)}) = a^{(l)}.\star(1-a^{(l)})$
5. $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)}\delta_i^{(l+1)}$ vectorized as $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$
   Hence we update our new Δ matrix:
   - $D_{i,j}^{(l)} := \frac{1}{m}(\delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(l)})$ if j ≠ 0
   - $D_{i,j}^{(l)} := \frac{1}{m}(\delta_{i,j}^{(l)})$ if j = 0

   The capital-delta matrix D is used as an 'accumulator' to add up our values as we go along & eventually compute our partial derivative ∴ $\frac{\partial}{\partial\Theta_{i,j}^{(l)}}J(\Theta) = D_{i,j}^{(l)}$