

**Coursera- Machine Learning**  
**May 2019**  
**Taught by Prof. Andrew Ng**  
**Janeshi99**

## **Summary**

Supervised learning

- linear regression, logistic regression, neural network, SVMs

Unsupervised learning

- k-means, PCA, Anomaly detection

Special applications/special topics

- Recommender systems, large scale machine learning

Advice for building a machine learning system

- bias/variance, regularization, deciding what to work next, evaluation of a learning algorithm, learning curves, error analysis, ceiling analysis

# Week 1

## Intro

### Definition of ML

- A program learns from experience (E) w.r.t task(T) and performance measure (P) if its performance on T improves with more E.
- With supervised learning, we know what our answers are as a relation of input and output. But with unsupervised learning, we have little idea about the result.

## Cost function

-

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

our goal is to minimize the cost function, which is calculated as square error

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

- where the error function is defined as

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

## Linear regression

- Repeat until converge{  $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$  for  $j = 0, 1$  }
- Note that the update is simultaneous :

-

$$\text{temp}_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{temp}_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp}_0$$

$$\theta_1 := \text{temp}_1$$

- if we compute the derivative we get Repeat until converge{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m ((h_{\theta}(x_i) - y_i)x_i)$$

}

- $\alpha$  is the learning rate.
- we use linear regression algorithm to updates the parameters until we arrive at the minimal cost.

## Week 2

### Multi-feature linear regression

- Hypothesis

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \dots \theta_n x_n$$

- convenience  $\forall x, x_0 = 1$ , so that  $h_{\theta} = \sum_{i=0}^n \theta_i x_i$

-

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \text{ and that } \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

- Hypothesis can be represented as

$$h_{\theta}(x) = \theta^T x \text{ or } \langle \theta, x \rangle$$

- The parameter we're estimating here is  $\theta$

- Cost function

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

- Gradient descent

$$repeat\{\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}, \text{simultaneously update } \theta_0 \dots \theta_j\}$$

- When working with gradient descent in practice, we should... consider

- Feature scaling:

Make sure features are in a similar scale, so that each values are roughly between  $[-3, 3]$

- Mean normalization: Replace all  $x_i$  (except for  $x_0$ ) with  $x_i - \mu_i$  so that the mean is roughly 0.

$$x_i \leftarrow \frac{x_i - \mu_i}{s_i}$$

- Note that  $J$  should always decrease w.r.t to the number of iteration. If it ever increases, that means our  $\alpha$ , the step param, is too large. We would want to decrease  $\alpha$ .

- Pick  $\epsilon$  for the convergence threshold value.
- Tip: in order to choose  $\alpha$ , try a range of values. Example: choosing based on a logarithmic scale:

$$0.001, 0.003, 0.01, 0.03, \dots$$

### feature & polynomial regression

- We can combine multiple features into one and change the behaviour of the hypothesis.
- For example we can combine  $x_1, x_2$  into a polynomial term by defining that  $x_3 = x_1 * x_2$ .
- polynomial regression, instead of linear, we make it quadratic or cubic to tune the hypothesis

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 \sqrt{x}$$

Keep in mind that feature scaling is still very important.

### Normal equation: computing parameters analytically

- Define  $X$  as the design matrix. That is

$$\text{if } x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \text{ then we have } X = \begin{bmatrix} - & (x^{(1)})^T & - \\ - & (x^{(2)})^T & - \\ & \vdots & \\ - & (x^{(n)})^T & - \end{bmatrix}$$

- Optimum  $\theta$  given by  $\theta = (X^T X)^{-1} X^T y$
- With normal equation, you don't need feature scaling.

*Gradient descent:*

$\alpha$  needs to be chosen

needs many iterations

works well even when  $n$  is large

*Normal equation:*

no need to choose  $\alpha$

no iterations needed

computing  $(X^T X)^{-1}$  takes  $O(n^3)$

performs slow with large  $n$  ( $n \geq 10,000$ )

- Note: what if  $X^T X$  is non-invertible? Then we use the *pinv* to generate the pseudo-inverse.

vectorization helps to compute vectors faster.

## Week 3

### Binary classification problems

- each element  $y$  belongs to negative class (0) or positive class (1).

### Logistic regression

- We want  $0 \leq h_\theta(x) \leq 1$   
 $h_\theta(x) = g(\theta^T x)$  where  $g$  is the sigmoid function  
 $g(z) = \frac{1}{1+e^{-z}}$

### decision boundary

- The decision boundary is the line that separates area where  $y = 0$  or  $y = 1$ .
- Note that

$$h_\theta(x) \geq 0.5 \iff \theta^T X \geq 0 \rightarrow y = 1$$

$$h_\theta(x) < 0.5 \iff \theta^T X < 0 \rightarrow y = 0$$

- we can also work with non-linear decision boundaries

### Logistic regression model

- The training set will be  $\{((x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)}))\}$   
There are  $m$  examples, and for  $\forall x, x \in \mathbb{R}^{n+1}$ ,  $x_0 = 1$ ,  $y \in \{0, 1\}$   
 $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$
- We realize that lin.reg. will not give you a convex function but we want a convex function. This brings us to construct a good cost function.

-

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

- For example if  $y = 1$  then if  $x = 1$  we have cost=0. And as hypothesis approach 0, cost approaches  $\infty$  so we're penalized. Similar with the other situation.
- This gives us a convex and local optimum free function.
- The uncompressed cost function is:

$$\text{cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

- The total cost function  $J$ :

$$j(\theta) = \frac{1}{m} \sum_{i=1}^m \text{cost}(h_{\theta}(x), y)$$

- The gradient descent algorithm is essentially the same but referring to a different hypothesis which is  $h_{\theta}(x)$  that now refers to the sigmoid function

### The vectorized implementation

- $h = g(X\theta)$ , which computes the quantity  $h_{\theta}(x^{(i)})$
- $J(\theta) = \frac{1}{m}(-y^T \log(h) - (1 - y)^T \log(1 - h))$

### Gradient descent

- Idea is to re-arrange the vectors until it's easier to type into matlab.
- Reminder that the matrix  $X$  looks like this:

$$X = \begin{bmatrix} - & (x^{(1)})^T & - \\ - & (x^{(2)})^T & - \\ & \vdots & \\ - & (x^{(m)})^T & - \end{bmatrix}$$

- $X$  is a  $m \times n$  matrix (ignoring the extra leftmost column of 1s).  $\theta$  is a  $n \times 1$  vector, which makes  $X^T\theta$  a  $m \times 1$  vector which yields the answer.

-

$$\theta := \theta - \frac{\alpha}{m} \sum_{i=1}^m [(h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}]$$

note that the  $x$  is a column vector.

- the vectorized version is:

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

- Advanced optimization: Given the cost function  $J(\theta)$  and gradient  $\frac{\partial}{\partial \theta_j} J(\theta)$  we can compute  $\min_{\theta} J(\theta)$ .
- These following algorithms are more complex but compute  $\theta$  at a faster rate, and there is no need to compute  $\alpha$ .

- Conjugate gradient, BFGS, L-BFGS
- Use the function "fminunc()"

### Logistic optimization for multiple classes

- Multiple classification is the situation when  $y = \{0, \dots, n\}$ . i.e. we have different classes of outcomes. Our strategy is to assign one class as 'positive' and the rest of classes as 'the rest'.
- $y \in \{0, 1, 2, \dots, n\}$
- $h_{\theta}^{(0)}(x) = P(y = 0 \mid x; \theta)$
- $h_{\theta}^{(1)}(x) = P(y = 1 \mid x; \theta)$
- $\vdots$
- $h_{\theta}^{(n)}(x) = P(y = n \mid x; \theta)$
- The prediction is  $\max_i h_{\theta}^{(i)}(x)$

### The problem of over-fitting

- Underfitting: the hypothesis function fits too poorly onto the trend of the data, the function is either too simple or accounting too little features.
- Overfitting: the hypothesis function is not generalized enough. It fits well with given data but presents unnecessary corners/ angles.
- To resolve overfitting, we can either 1) reduce the number of features, i.e. have an algorithm that ditches unimportant features. Or 2) apply regularization to reduce magnitude of some features.

### Cost function with regularization

-

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^m m(h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n n\theta_j^2$$

- We want a big  $\lambda$  so that bumps up and forces  $\theta_j$  to be small as we penalize bit  $\theta_j$ .

### Gradient descent



-

$$\begin{aligned} \text{repeat } \{ \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)} \text{ and} \\ \theta_j &:= \theta_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \\ \text{for } j &= \{1, \dots, n\} \end{aligned}$$

- or we can equivalently write this in one equation:

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

### Normal Equation

-

$$\theta = (X^T X + \lambda L)^{-1} X^T y$$

- where  $L$  is the same as  $I \in M_n^{(n+1) \times (n+1)}$  with the first 1 replaced with 0. Note that if  $m < n$  then  $X^T X$  is non-invertible but adding  $L$  makes it invertible. Hence regularization also solves non-invertability.

### Regularized logistic equation

(note that advanced optimization method me mentioned earlier also works here)

- Regularized cost function for logistic regression is:

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

- the last term is newly added for the regularization

- Gradient descent is exactly the same

-

$$\begin{aligned} \text{repeat } \{ \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)} \text{ and} \\ \theta_j &:= \theta_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \\ \text{for } j &= \{1, \dots, n\} \end{aligned}$$

Advanced functions (regularization)

-  $Jval$  does not change while the gradients do change.

- Gradient 1(index 0):

$$\frac{1}{m} \sum_{i=1}^m (h_{\theta}(X^{(i)}) - y^{(i)}) x_0^{(i)}$$

- Gradient 2(index 1, 2, ... n)

$$\left( \frac{1}{m} \sum_{i=1}^m (h_{\theta}(X^{(i)}) - y^{(i)}) x_0^{(i)} \right) + \frac{\lambda}{m} \theta_j$$

note that the last term is a newly added item.

## Week 4

### Neural Networks model representation

- We use matrices and vectors to model neurons and layers. [I'm too lazy to use tikz to draw it]
- Neural network is consisted of multiple layers. There is an input-layer, lots of hidden layer in the middle and the output layer wich is one node.
- $a_i^{(j)}$  = "activation" of unit  $i$  in layer  $j$ .
- $\Theta^{(j)}$  = matrix of weights controlling function mapping from layer  $j$  to layer  $j + 1$ .
- Vec representation

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

- Layer 1 to layer 2:

$$\begin{aligned} a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\ a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\ a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \end{aligned}$$

- Layer 2 to layer 3:

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

- What is the dimension of  $\Theta$ ?
- Note that each layer has its own matrix of weights. If network has  $S_j$  layers in level  $j$  and  $S_{j+1}$  layers in level  $j + 1$  then  $\Theta^{(j)}$  has dimensions  $S_{j+1} \times (S_j + 1)$  where the extra 1 comes from the bias node.
- the intuition is that NN allows nodes in its hidden layer to 'learn' its own features.

### Vectorization of Computation

- $a_1^{(2)} = g(z_1^{(2)})$

- $a_2^{(2)} = g(z_2^{(2)})$
- $a_3^{(2)} = g(z_3^{(2)})$
- For layer  $j$ , node  $k$ ,  $z$  is  $z_k^{(j)} = \Theta_{k0}^{(j-1)}x_0 + \Theta_{k1}^{(j-1)}x_1 + \dots + \Theta_{kn}^{(j-1)}x_n$

-

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

- note that

$$z^{(j)} = \Theta^{(j-1)}a^{(j-1)}$$

since dimensions of  $\Theta$  is  $S - j \times n + 1$  and that dimensions of  $a$  is  $n + 1 \times 1$ .

- We add the bias unit to layer  $j$  after computing  $a^{(j)}$  i.e.  $a_0^{(j)} = 1$ .
- In order to compute the final hypothesis, we compute the final  $z$  vector, where the last matrix of  $\Theta$  only has 1 row, which multiplied by a column will result in a real number.

$$h_{\Theta}(x) = a^{(j+1)} = g(z^{(j+1)})$$

- Multi-class classification- we still use the one-vs all method. We set  $h_{\Theta}(x) \in \mathbb{R}^4$  if there are 4 classes.

-

$$h_{\Theta}(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

with different input  $x$ . It returns one of the standard vectors  $e_i$ s given a particular input.

## Week 5

Our goal is to learn how to train NNs.

### The cost function for NN

- $L$ : is the total number of layers in the NN
- $S_l$ : is the number of units not counting the bias unit in layer  $l$
- $k$ : is the number of output unit/ classes
- The cost function is:

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

### The back propagation algorithm

- The goal is to compute  $\min_{\Theta} J(\Theta)$ .
- look at the partial derivative of  $J(\Theta)$ , which is

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta)$$

, The back propagation algorithm works as follows:

- given the training sets  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$
- we set  $\Delta_{i,j}^{(l)} := 0 \forall i, j$
- For training exmples  $t = 1 \sim m$ :
  - 1. Set  $a^{(1)} := x^{(t)}$
  - 2. Perform forward propagation to compute  $a^{(l)}$  for  $l = 1, 2, \dots, L$   
(i.e. set up  $z$ (intermediate value) and use  $g(z)$  to calculate next layers, so on and so on.
  - 3.  $\delta^{(L)} = a^{(L)} - y^{(t)}$

- 4. compute  $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$  using

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$$

where the first value is the calculated value and that the second value is derivative of  $g$  which is  $g'$ .

$$g'(z^{(l)}) = a^{(l)} .* (1 - a^{(l)})$$

Note:  $.*$  means product form direct vector multiplication. i.e. resulting vector takes  $i^{th}$  value from the product of the two  $i^{th}$  values from the multiplicands.

- 5.  $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_j^{(l+1)}$   
with vectorization we have  $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + \delta_j^{(l+1)} (a_j^{(l)})^T$
- Hence the update is

$$D_{i,j}^{(l)} := \begin{cases} \frac{1}{m} \Delta_{i,j}^{(l)} + \lambda_{\Theta_{i,j}}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{m} \Delta_{i,j}^{(l)} & \text{if } j = 0 \end{cases}$$

- In here,  $D$  is the accumulator for the gradient where  $\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta) = D_{i,j}^{(l)}$
- Very useful resources to read about:  
”<http://neuralnetworksanddeeplearning.com/chap2.html>”
- Tips for implementation:
  1. For this part, the homework assignment did not ask you to implement these steps.
  2. Unrolling is basically changing a matrix into a single column vector, i.e. unrolling
  3. Gradient checking ensures bug-free implementation
  4. Initial theta is set randomly

# Week 6

## Evaluating a learning algorithm

- What are some ways to arrive at a better hypothesis?  
More examples  
more / less # of features  
more / less values of  $\lambda$
- To evaluate a hypothesis, we split the data into training set and the test sets. They usually have 70% and 30% respectively.
- Then we learn  $\Theta$ , minimize  $j_{train}(\Theta)$  using the training set. Next step we evaluate how good our hypothesis is by calculating the test set error with  $J_{test}(\Theta)$ .
- to compute the test set error for lin.reg., it is half of the square error, whereas for log.reg. it is the average of the *err* function where it is 1 if hypothesis is in the right range, 0 otherwise.

## Model selection

- We can break down the data-set into three data-sets:  
  
training set, cross validation set, and test set  
  
taking up 60%, 20%, 20% respectively.
- We can test different degree of polynomial, evaluate their error functions.
  1. optimize params in  $\Theta$  using training set for each degree.
  2. find the polynomial degree  $d$  that produce least error by cross validation set.
  3. estimate generalized error with test set.

## Diagnosing bias, variance and regularization, learning curves

- Better explanation seen in the handwritten notes- it has graph examples.

## debugging a learning algorithm

problem	try
high var	get more training data
high var	less features
high var	increase $\lambda$
high bias	get more features
high bias	add polynomial features
high bias	decrease $\lambda$

Note that small NN are computationally cheap but prone to underfitting.

Large NN are computationally expensive but prone to overfitting. Note that  $\lambda$  regularization can be use to fix this problem.

### Error analysis

- Always implement a quick implementation, and use that to decide how to spend your time.
- Plot learning curves and use that to decide about your next steps. You can manually examine your errors or impementa metric that returns your performance.
- For skewed classes, for example one class has very large size and another has very little, we use another error metric that accounts true and false positives and negatives. ( $F$  scores)



## Week 7

### SVMs

- SVM for logistic regression works with different cost function, that is easier to compute for the computer. It allows a large margin classifier that draws the decision boundary that stays naturally far apart from the dataset. kernels take data as input and transform them into the required forms.
- When using a SVM, we can use nice libraries such as *liblinear* or *libsvm*. We need to choose kernel (linear kernel a.k.a. no-kernel or gaussian kernel) as well as parameter  $c$ .
- Other choices of kernel must satisfy Mercer's theorem so that it does not diverge.
- for multi-classification, we can use the built-in SVM package or the one-vs-all methods.

# Week 8

## Unsupervised learning

- The given input has no labels
- the algorithm finds clustering data, and it identifies structures. It can be used for identifying market segmentations, social network analysis, organize computer clusters, and galaxy formation/astronomical data analysis.

## k-means algorithm

- This is used to identify clustering.
- Step 1. Initialize cluster centroids randomly
- Step 2. assign each point to centroids that is the closest
- Step 3. move the centroids, to new means of assigned points
- Step 4. repeat