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{

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

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

}

- α 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
$$h_\theta(x) = \theta^T x \text{ or } < \theta, x >$$
- The parameter we're estimating here is θ
- 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.
$$x_i \leftarrow \frac{x_i - u_i}{s_i}$$
- Note that J should always decresase w.r.t to the number of iteration. If it ever increases, that means our α, the step param, is too large. We would want to decrease α.

- Pick ϵ for the convergence threshold value.

- Tip: in order to choose α, try a range of values. Example: choosing based on a logarithmic scale:

$$0.001, 0.003, 0.01, 0.03, \ldots$$

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 θ given by $\theta = (X^T X)^{-1} X^T y$

- With normal equation, you don't need feature scaling.

Gradient descent:	*Normal equation*:
α needs to be chosen	no need to choose α
needs many iterations	no iterations needed
works well even when n is large	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.

1 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

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

- 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.
-

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

- For example if $y = 1$ then if $x = 1$ we have cost=0. And as hypothesis approach 0, cost approaches ∞ 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 unction J:

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

- The gradient descent algorithm is essetially the same but referring to a different hypothsis 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). θ is a $n \times 1$ vector, which makes $X^T\theta$ a $m \times 1$ vector which yields the answer.
- Left off at the spot where we have two theta equations.