Coursera ML Notes

Stanford: Andrew Ng's "Machine Learning"

Lectures by Andrew Ng
Notes by Matthew Low `mattchrlw`

Updated December 24, 2019

Contents

CHAPTER 1 Week 1

1.1 Introduction

Pretty simple stuff. House price example, news classification example etc. etc.

1.2 Model and cost function

Let $x^{(i)}$ denote the "input" variables/features, and $y^{(i)}$ denote the "output" or target variable we wish to predict. A pair $(x^{(i)}, y^{(i)})$ is called a **training example**, and the dataset that we'll be using to learn:

$$(x^{(i)}, y^{(i)}), \quad i = 1, \ldots, m,$$

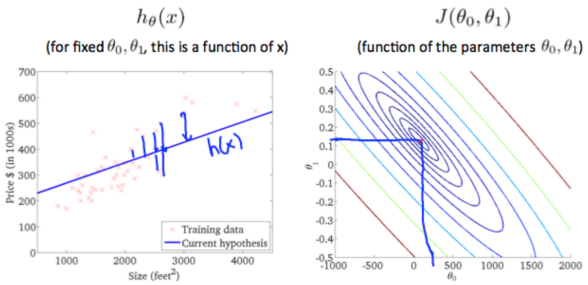
is called a **training set**. Superscript denotes an index into the training set, which isn't too bad notation when you start looking at gradient descent etc. For the purposes of this introductory course, we use X to denote the space of input values and Y to denote the space of output values, and $X = Y = \mathbb{R}$.

In supervised learning, the goal is given a training set to learn a function $h : X \to Y$ such that $h(x)$ is a good predictor for y. h is called a **hypothesis** for historical reasons. Discrete $\implies$ classification, continuous $\implies$ regression (but it can be more complicated than that.

To measure the accuracy of our hypothesis function (in linear regression), we can use a **cost function**. In the context of linear regression, this cost function is least squares.

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

The function is called the "squared error function", "mean squared error" or MSE. The mean is halved as a convenience for gradient descent (because the squared carries over to the fraction, useful for the normal equation). Since this function represents the *cost*, we wish to minimise it.



1.3 Parameter learning

We want to estimate the parameters in the hypothesis function, which is where **gradient descent** comes in. Graphing our hypothesis function as a function of the parameter estimates gives a surface. Gradient descent uses the fact that the gradient is the direction towards the minimum (very roughly speaking) to perform an iterative movement towards the minimum. I would write more detail about gradient descent but I'm literally doing a summer project on optimisation so I don't think there's much point, refer to my optimisation notes for more details. This algorithm gives an iterate

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

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

Update the parameters simultaneously, or else gradient descent won't work!

(Note: this fact would be obvious if we employed the gradient operator $\nabla J(\theta_0, \theta_1)$ instead of Ng's notation, but I digress.)

When specially applied to linear regression, we can obtain a new form of gradient descent. Replacing the cost function and hypothesis function, we obtain the following iterate:

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

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

These can both be verified with some calculus.

CHAPTER 2 Week 2

2.1 Multivariate linear regression

Linear regression with multiple variables is also known as **multivariate linear regression**. In the more general case, we can have the following:

$x_j^{(i)}$	value of feature j in the ith training example
$x^{(i)}$	the input of the ith training example
m	the number of training examples
n	the number of features

The multivariate form of this is

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n \quad \equiv \quad h_\theta(x) = \theta^\top x$$

where $\theta = \begin{pmatrix}\theta_0 & \theta_1 & \ldots & \theta_n\end{pmatrix}$ and $x = \begin{pmatrix}x_0 & x_1 & \ldots & x_n\end{pmatrix}^\top$.

For gradient descent in the multivariate case using Ng's notation we have the following iterate:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}, \quad \forall j \in \{0, \ldots, n\}.$$

We introduce a practical trick for gradient descent called **feature scaling**. The idea is to ensure that features are on a similar scale, making gradient descent faster. We get every feature into *approximately* a $-1 \leq x_i \leq 1$ range. You can also perform **mean normalisation**, where we replace x_i with $x_i - \mu_i$ to make features have approximately zero mean, making sure not to apply to $x_0 = 1$ or anything like that. You can also divide by the standard deviation s_i instead of the range.

For gradient descent, we can also use something called the **automatic convergence test**. Declare convergence if $J(\theta)$ decreases by less than ε in one iteration, where ε is some small value. (It's difficult to choose this threshold value). If α is too small, we have a slow convergence rate, but if α is too large, we may not decrease on every iteration and thus may not converge.

We may also have a situation where we have polynomial regression. We can change the behaviour 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 $h_\theta(x) = \theta_0 + \theta_1 x$, we can create additional features based on x_1 to get a quadratic with $\theta_2 x_1^2$ or even a cubic with both $\theta_2 x_1^2$ and $\theta_3 x_1^3$ added.

2.2 Computing parameters analytically

We have been using iterative algorithms so far such as gradient descent, but for some problems we can solve for θ analytically. The **normal equation** formula (the Moore-Penrose pseudoinverse times y for the enlightened) is the solution to the problem

$$\theta = (X^\top X)^{-1} X^\top y.$$

There is no need to do feature scaling or any other bollocks with the normal equation. However, the normal equation has complexity $\mathcal{O}(n^3)$ and you may need to compute the inverse of $X^\top X$, so it can be really slow for large n.

Gradient Descent	Normal Equation
Need to choose α	No need to choose α
Needs many iterations	No need to iterate
$\mathcal{O}(kn^2)$	$\mathcal{O}(n^3)$ and inverse
Works well when n is large	Slow if n is large

But look at the normal equation; we have an inverse. Not every matrix has an inverse. Using `pinv` in MATLAB gets around this in some fancy ways, but we should still be alarmed if $X^\top X$ is noninvertible, as there may be:

- Redundant features, where two features are very closely related (linearly dependent)
- Too many features ($m \leq n$). In this case, delete some features or use "regularisation".

CHAPTER 3 Week 3

3.1 Classification and representation

The problem of classifying a tumour as either malignant or benign. Assign 0 to benign and 1 to malignant. We can use a **threshold classifier**, which says

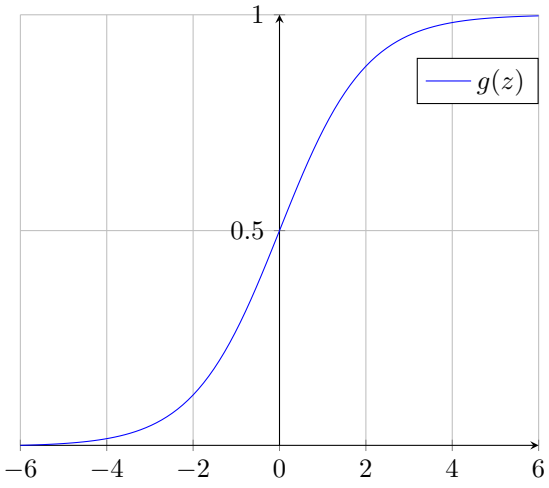
$$y = \begin{cases} 1 & h_\theta(x) \geq 0.5 \\ 0 & h_\theta(x) < 0.5. \end{cases}$$

But linear regression has some flaws. If we have some output way out of the linear regression line, the threshold classifier doesn't really work as it may misclassify based on the outlier. Linear regression to a dataset for classification is generally really bad. Another issue is that for linear regression, the hypothesis is not bounded to purely $\{0, 1\}$.

In order to get around this, we develop a new technique called **logistic regression** (for purely historical reasons, it is considered regression, but it is really classification). We define the **sigmoid or logistic function**

$$g(z) = \frac{1}{1+e^{-z}}, \quad h_\theta(x) = g(\theta^\top x) \quad \implies \quad h_\theta(x) = \frac{1}{1+e^{-\theta^\top x}}$$

The sigmoid function appears as follows:



We need to fit the parameters θ given a training set. We can interpret the hypothesis output as follows. $h_\theta(x)$ is the **estimated probability** that $y = 1$ on input x.

Example 3.1.1 If $x = (x_0, x_1)^\top = (1, \text{tumor size})^\top$ and we obtain $h_\theta(x) = 0.7$, we can interpret that as the patient "has a 70% chance of the tumour being malignant". We know that

$$\begin{aligned} \mathbb{P}(y = 0 \mid x; \theta)\mathbb{P}(y = 1 \mid x; \theta) &= 1 \\ \mathbb{P}(y = 0 \mid x; \theta) &= 1 - \mathbb{P}(y = 1 \mid x; \theta) \end{aligned}$$

(I recognise that this probability stuff is a bit rough.)

For logistic regression, we can use the threshold to predict $y = 1$ if $h_\theta(x) \geq 0.5$, and $y = 0$ if $h_\theta(x) < 0.5$. This is equivalent to predicting $y = 1$ if $\theta^\top x \geq 0$ and $y = 0$ if $\theta^\top x < 0$. Suppose we have a training set which looks like the following, and that our hypothesis is

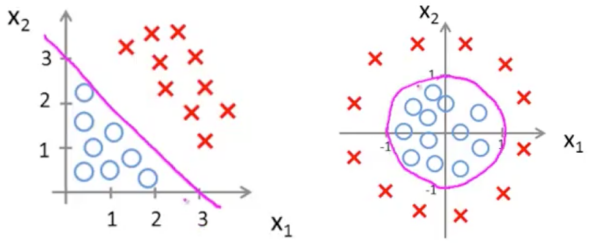
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2), \quad \theta = (-3, 1, 1)^\top.$$

Given this choice of hypothesis parameters, we predict $y = 1$ if $-3 + x_1 + x_2 \geq 0$, and $y = 0$ otherwise. The **decision boundary** is this function $-3+x_1+x_2$; it is the function that separates the area where $y = 0$ and $y = 1$. If the decision boundary is positive, we predict 1 and negative vice-versa.

We can also add higher-order terms to the features. For this decision boundary, we choose

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2), \quad \theta = (-1, 0, 0, 1, 1)^\top.$$

We predict $y = 1$ if $-1 + x_1^2 + x_2^2 \geq 0$ (this is the equation of a circle when rearranging).



3.2 Logistic regression model

Given the training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ of m examples, with each example represented by a feature vector $x \in (x_0, x_1, \ldots, x_n)^\top (\mathbb{R}^{n+1}$, $x_0 = 1, y \in \{0, 1\}$ and a hypothesis function $h_\theta(x) = \frac{1}{1+e^{-\theta^\top x}}$, how do we choose parameters θ?

Back in linear regression land, we formulated the cost function as

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

This function worked fine for linear regression, but we are interested in logistic regression. This function is **nonconvex** for parameter θ in logistic regression, which means we can't run gradient descent and expect any sort of convergence.

Instead, we use the function

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

If our correct answer $y = 0$, then the cost function will be 0 if our hypothesis function also outputs 0. If our hypothesis approaches 1, then the cost function will tend to infinity. Conversely, if our correct answer $y = 1$, then the cost function will be 0 if our hypothesis function outputs 1, and if our hypothesis approaches 0, then the cost function will tend to infinity. This function gives a convex and local-optima-free cost function $J(\theta)$.

We can actually rewrite this cost function more cleanly than before. Since $y = 0$ or 1 always, we can rewrite this function using a convex combination:

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

We can fully write out our entire cost function as

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

or the equivalent vectorised

$$h = g(X\theta), \quad J(\theta) = \frac{1}{m}(-y^\top \log(h) - (1-y)^\top \log(1-h)).$$

Gradient descent is also easy to implement; in fact, we obtain the exact same iterate

$$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \quad \equiv \quad \theta := \theta - \frac{\alpha}{m}X^\top(g(X\theta) - y).$$

Feature scaling also applies to gradient descent with logistic regression.

For optimisation problems, we also have more sophisticated algorithms such as conjugate gradient, BFGS and L-BFGS. They have a number of advantages, such as not needing to manually pick α and being faster than gradient descent, but this comes at the cost of being complicated (??). In MATLAB, use the `fminunc` function (minimise unconstrained).

3.3 Multiclass classification

Instead of $y = \{0, 1\}$, consider the situation where $y = \{0, 1, \ldots, n\}$. We divide our problem into $n+1$ binary classification problems. In each one, we predict the probability that y is a member of one of our classes. To make a prediction on a new x, pick the class which maximises $h_\theta(x)$:

$$\begin{aligned} h_\theta^{(0)}(x) &= \mathbb{P}(y = 0 \mid x; \theta) \\ h_\theta^{(n)}(x) &= \mathbb{P}(y = n \mid x; \theta) \\ \text{Prediction} &= \max_i(h_\theta^{(i)}(x)) \end{aligned}$$