Machine Learning by Andrew Ng

Ramandeep Farmaha

Summer 2016

Chapter 1

Week 1

1.1 Introduction

1.1.1 Supervised Learning

Examples of learning algorithms include straight and quadratic lines of best fit that go through a scatterplot.

Supervised Learning: We are told what is the "correct" solution

Regression: Predicting continuous value output

Classification: Predicting a discreet value output (such as, but not limited to 0 and 1)

Machine learning problems may depend on 0, 1 or up to infinite number of features. How can we deal with an infininte number of features using a computer with a finite amount of memory?

1.1.2 Unsupervised Learning

Unlike supervised learning, the data set in unsupervised learning does not have any labels or structure. An example of an unsupervised learning algorithm is clustering, such as in Google News, clustering is used to aggregate news articles about similar topics.

Another application of a clustering alogrithm is determining relationships between friends on a social network.

Cocktail Party Problem: Two speakers talking into microphones that are both at arbitrary distances from each other and each speaker. The cocktail party algorithm is able to separate the two individual voices.

1.2 Linear Regression with One Variable

1.2.1 Model Representation

In supervised learning, there are data sets, and our job is to learn from this data. For example, a supervised learning algorithm can be used to predict the price of a house in Portland, Oregon based on its square footage. In this course, there will be a set of key notation:

m = Number of training examples

x = Input variables/features
y = Output variables/features
(x, y) = One training example
$(x^{(i)}, y^{(i)})$ = ith training example

Start with a training set and feed that to a learning algorithm, which outputs to a function denoted by a h, i.e. h is a function that maps from x's to y's. The term h refers to the word hypothesis. When designing a learning algorithm, the next step is determining how to represent this hypothesis h. For example, h could be a linear regression with one variable, represented by:

$$h_\theta = \theta_0 + \theta_1 x \tag{1.1}$$

1.2.2 Cost Function

In equation 1.1, θ_0 and θ_1 are the two parameters of the model. The next step is determining the values of these two parameters which would give an h(x) that is closest to a value y for a given x. The best way to find these values is to minimize the squared difference between the estimated values and they y-values for every point in the training set. This cost function can be summarized as:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta x^{(i)} - y^{(i)})^2 \tag{1.2}$$

1.3 Gradient Descent

Gradient descent can be used to minimize the cost function J, as well as other functions. The main idea behind gradient descent is to guess θ_0 and θ_1 and then finding the error, then moving each up or down to minimize this error. One common guess is 0 for both values. Below is the equation for gradient descent:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \tag{1.3}$$

Equation 1.3 should be repeated until convergence for j = 0 and j = 1. Below is pseudo-code for the iteration, demonstrating simultaneous updates of the values:

$$\begin{aligned} temp0 &:= \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ temp1 &:= \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ \theta_0 &:= temp0 \\ \theta_1 &:= temp1 \end{aligned} \tag{1.4}$$

1.3.1 Gradient Descent for Linear Regression

Gradient descent can be used to get both θ_0 and θ_1, which can then be applied to the linear regression hypothesis and cost functions, 1.1 and 1.2 The first step is to calculate the derivative of the cost function and then subbing into the gradient descent function:

$$\begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta x^{(i)} - y^{(i)}) \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta x^{(i)} - y^{(i)}) x^{(i)} \end{aligned} \tag{1.5}$$

The main drawback to gradient descent in this form is that if there are any local minima, the function may fail to find the absolute minimum. However, in the case of linear regression, the cost function takes the form of a convex function, i.e. the local minimum is always the global minimum.

"Batch" gradient descent: Loooking at all training examples at each step of gradient descent

Chapter 2

Week 2

2.1 Multivariate Linear Regression

2.1.1 Multiple Features

Notation:
n = number of features
$x^{(i)}$ = input (features) of i^{th} training example
$x_j^{(i)}$ = value of feature j in i^{th} training example
The form of the hypothesis will now change because of the multiple features. i.e.:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n \tag{2.1}$$

For convenience of notation, define $x_0 = 1$. This is the additional zero feature. In vector notation:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \epsilon \Re^{n+1} \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ . \\ . \\ . \\ \theta_n \end{bmatrix} \epsilon \Re^{n+1} \tag{2.2}$$

$$h_\theta(x) = \theta^T x$$

2.1.2 Gradient Descent for Multiple Variables

For more than one feature, the gradient descent algorithm is summarized as:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta x^{(i)} - y^{(i)}) x_j^{(i)} \tag{2.3}$$

Where θ_j is being simulatneously updated for j = 0, ..., n.

2.1.3 Gradient Descent and Feature Scaling

If the features in your gradient descent contain a large range of values, it can become very tedious and time consuming for the algorithm to find the global minimum. One shortcut would be to scale all features so that all of their values lie as close between -1 and 1 as you can.

Similarly, mean normalization makes features have approximately zero mean using the following equation:

$$x_1 = \frac{x_1 - \mu_1}{s_1} \tag{2.4}$$

Where s is the standard deviation and μ is the mean of the feature set.

2.1.4 Learning Rate

To make sure that gradient descent is running smoothly, one tip would be to plot the algorithm while it is running, with the number of iterations on the x-axis and the minimum of the cost function on the y-axis. At some point, the cost function decrease will eventually flatten out (i.e. its slope will approach zero), at which point gradient descent converges. One tip would be to have a check that checks whether gradient descent produces a cost function value that decreases by less than some threshold.

If your cost function is increasing, maybe use a smaller alpha value. Try values of α at a few orders of magnitude and then pick the largest reasonable value.

2.1.5 Features and Polynomial Regression

You are not constrained to the features presented to you when doing a linear regression model. In the case of a house for example, you can use a combination of depth and frontage to calculate the housing price. You could multiply the two values and use a third feature called Area which is the product of the two.

Similarily, polynomial regression refers to using models that have different powers of influence over the hypothesis function. One thing to be careful of is the range of valeus for these new features, as they will increase exponentially, thus it's important to make sure feature scaling is being used to optimize the algorithm.

2.2 Computing Paramaters Analytically

2.2.1 Normal Equation

Rather than run an iterative algorithm for minimizing the cost function J, we can take an analytical approach to solve for θ. In 1-D application, if the cost function is quadtratic, we can take the derivative of it and set it to 0, which would give us the minimum.

For m **examples** $(x^{(1)}, y^{(1)})$, ..., $(x^{(m)}, y^{(m)})$; n **features**.

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ . \\ . \\ . \\ x_n^{(i)} \end{bmatrix} \epsilon \Re^{n+1} \tag{2.5}$$

NOTE: Remember that x_0 always equals to 1. We are now going to create a matrix X, dubbed the design matrix, which contains the transpose of all elements in the x matrix, thus making it an m x (n+1) matrix. We do the same for the Y matrix.

The calculation of θ can be analytically given as:

$$\theta = (X^T X)^{-1} X^T y \tag{2.6}$$

Where $(X^T X)^{-1}$ is the inverse matrix of $X^T X$

For m training examples and n features, the advantage of the analytical approach over gradient descent is that we don't need to choose a value for α and we don't need to iterate. However, gradient descent works pretty well when n is large, which can get pretty slow for the analytical approach as it needs to compute the inverse of an n x n matrix, which is roughly $O(n^3)$ in computation costs. Normal equation should probably be used until around 10,000 features.

2.2.2 Normal Equation and Noninvertibility

What happens when $X^T X$ is non-invertible? (i.e. it is singular or degenerate). The most common causes for this would be if you have redundant features, such as the size of a house in two different dimensions. Another possible cause is if you have too many features, i.e. the value of n is greater than n. Solution to this would be to either aggregate more data (ideal) or delete some features or use a technique called regularization.