Machine Learning Standford course - Summary

Gi Wah Dávalos Loo

April 6, 2018

1 Introduction

1.1 Definition

Modern definition by Tom Mitchell:

> "A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**."

1.2 Types

In general, any machine learning problem can be assigned to one of two broad classifications: Supervised and Unsupervised learning.

1.2.1 Supervised Learning

We are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output. Supervised learning problems are categorized into:
1.*Regression*: tries to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function.
2.*Classification*: tries to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

1.2.2 Unsupervised Learning

Allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.
We can derive this structure by clustering the data based on relationships among the variables in the data.
With unsupervised learning there is no feedback based on the prediction results.

2 Linear Regression

The main idea of linear regression is to predict some value from input (x_i) using a function that has a linear behaviour. The **goal** is to find the most accurate coefficients for the linear function we must reduce the error as much as possible, in other words, we gotta minimize the cost function.

2.1 Types

2.1.1 Univariable

When we have only 1 feature. The linear function for univariable regression should look like this:

$$h_\theta(x) = \theta_0 + \theta_1 x \tag{1}$$

2.1.2 Multivariable

When we have more than 1 feature. The multivariable linear regression function should look like this:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n \tag{2}$$

2.2 Cost Function

We can measure the accuracy of our hypothesis function by using a cost function. This takes an average difference (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's and the actual output y's.

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

2.3 Notation (Matrixes and Vector approaches)

In order to make operations more code/cpu "friendly" we represent our set of data and variables on matrixes and vectors.

$$\begin{aligned} x_j^{(i)} &= \text{value of feature } j\text{in the } i^{th} \text{ training example} \\ x^{(i)} &= \text{the input (features) of the } i^{th} \text{ training example} \\ m &= \text{The number of training examples} \\ n &= \text{The number of features} \end{aligned}$$

$$\theta_{(n\times 1)} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \quad x_{(n\times 1)} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad X_{(m\times n)} = \begin{bmatrix} x_0^1 & x_1^1 & x_2^1 \dots x_n^1 \\ x_0^2 & x_1^2 & x_2^2 \dots x_n^2 \\ x_0^3 & x_1^3 & x_2^3 \dots x_n^3 \\ \vdots & \vdots & \vdots \\ x_0^m & x_1^m & x_2^m \dots x_n^m \end{bmatrix} \quad y_{(m\times 1)} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

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

$$\text{solution format: } h_{(\theta)} = \begin{bmatrix} x_0^1 & x_1^1 & x_2^1 \dots x_n^1 \\ x_0^2 & x_1^2 & x_2^2 \dots x_n^2 \\ x_0^3 & x_1^3 & x_2^3 \dots x_n^3 \\ \vdots & \vdots & \vdots \\ x_0^m & x_1^m & x_2^m \dots x_n^m \end{bmatrix} \begin{bmatrix} \theta_0 & \theta_1 \dots \theta_n \end{bmatrix} = X\theta \qquad (3)$$

$$\text{cost function: } J(\theta) = \frac{1}{2m}(X\theta - y)^2 \qquad (4)$$

Note 1: x_0 is a *"hack"* and it's value is always 1. It's for making both (θ and x) same size in order to multiply them.

Note 2: The univariable and multivariable $h_{(\theta)}$ can be solved using $X\theta$

2.4 How to obtain θ_j

The idea is to minimize the **Cost Dunction** (minimize the error). In order to do that, we need to build an equation by comparing the derivative of J to cero.

$$min(J(\theta)) = \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} = 0$$

We got 2 ways to approach this problem:

1. *Gradient Descent*
2. *Normal Equation*

2.4.1 Gradient Descent

It is an iterative algorithm that, with a fixed learning rate (α), updates all values of θ simultaneously and decreases the Cost Function. The learning rate (α) is adjusted by try/error. The initial values to test should be on a log-scale, at multiplicative steps of about 3 times the previous value (i.e., 0.3, 0.1, 0.03, 0.01 and so on).

repeat until converge {

$$\theta_j = \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j}$$

}

Since it's an iterative algorithm we want the steps to be really small in order to make the computation time a lot faster. We care about the relationship between entries, not the actual values; that why we tend to make the mean zero. For that we got 2 methods:

1. *Feature Scaling*: Involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1.

2. *Mean normalization*: Involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero.

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

Where μ_i is the average of all the values for feature (i) and s_i is the range of values (max - min), or s_i is the standard deviation.

Note that dividing by the range, or dividing by the standard deviation, give different results.

2.4.2 Normal Equation

This is the analytical way to tackle this problem. We actually solve the equation $\frac{\partial J(\theta)}{\partial \theta_j} = 0$. There is no need to do feature scaling with the normal equation.

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

The main differences between this two approaches:

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
$O(kn^2)$	$O(n^3)$, need to calculate inverse of $X^T X$
Works well when n is large	Slow if n is very large