Machine Learning - Andrew Ng

Matteo Esposito

November 15, 2018

0 Introduction

This document is a compilation of notes from the machine learning coursera MOOC taught by Prof. Andrew Ng of Stanford University offered by Coursera.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

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.

From this point, we will use X to denote the space of input values, and Y to denote the space of output values. In this example, $X = Y = \mathbb{R}$.

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function $h : X \longrightarrow Y$ so that $h(x)$ is a "good" predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis.

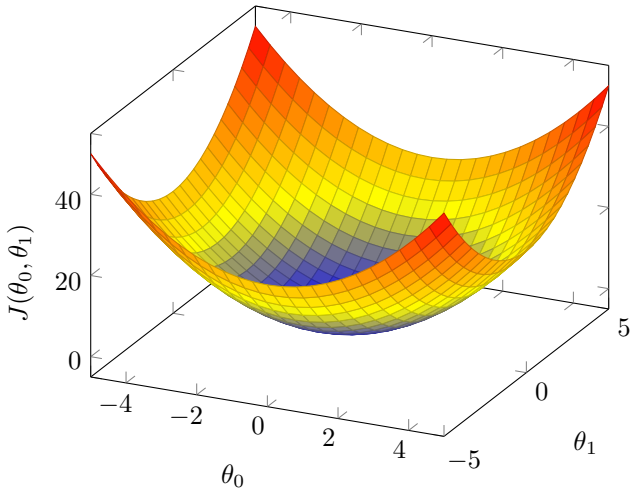
0.1 Cost Function

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.

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

This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved $\left(\frac{1}{2}\right)$ as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the $\left(\frac{1}{2}\right)$ term. J will be 0 if we can achieve a perfect fit (not always possible, and when possible, will probably mean overfitting). We can use a contour plot to visualize the minimizing of the cost function.

Convex Cost Function



0.2 Gradient Descent

Algorithm 1 General Gradient Descent

for $n >= 1$ **do**
  n =number of features, m =number of samples
  repeat until convergence:
    $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)})$
end for

We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum.

The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter α, which is called the learning rate.

1 Linear Regression

The standard form of the hypothesis function for linear regression is the following:

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

In the case of linear regression we use MSE as the cost function $J(\theta)$.

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= (h_\theta(x) - y) x_j
\end{aligned}$$

Algorithm 2 Gradient Descent for Linear Regression

n =number of features, m =number of samples
repeat until convergence:
  $\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$
  $\theta_1 \leftarrow \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)})$

The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

Note: In the general case, we know that there is a possibility for gradient descent to fall into a local minimum trap, this is however impossible in the case of linear regression. There is a single global minimum and no local minima.

1.1 Multivariate Linear Regression

The multivariable form of the hypothesis function accommodating these multiple features is as follows:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_1 x_2 + \ldots + \theta_n x_n$$

This can be represented as follows:

$$\begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \ldots & \theta_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

Here we use the general gradient descent algorithm.

1.2 Feature Scaling

We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven. Scaling features becomes increasingly important when creating features (i.e. in the case of polynomial regression).

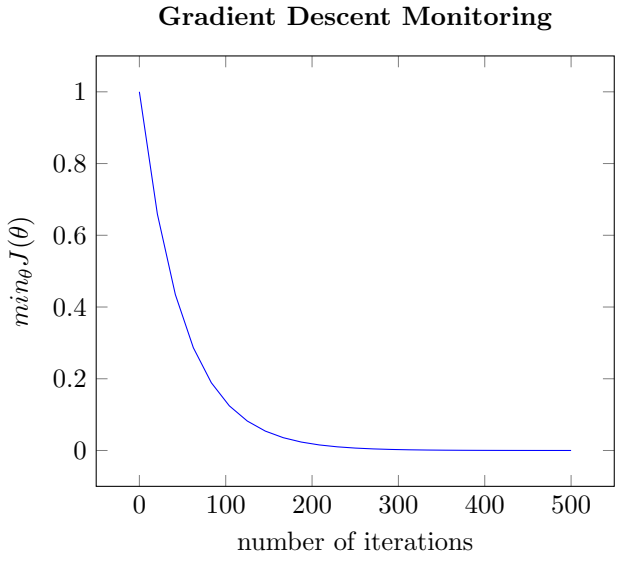
Two techniques to help with this are **feature scaling** and **mean normalization**.

- 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.
- 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. To implement both of these techniques, adjust your input values as shown in this formula, where s_i is the range our standard deviation.

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

1.3 Learning Rate

To monitor/make sure gradient descent is working correctly, we can plot the value of $min_\theta J(\theta)$ vs. the number of iterations and hope to see a reasonably paced decrease in the minimum value achieved.



Properties of the learning rate α:

- It has been proven that if learning rate α is sufficiently small, then $J(\theta)$ will decrease on every iteration.
- If α is too small: slow convergence.
- If α is too large: may not decrease on every iteration and thus may not converge.

1.4 Normal Equation

Gradient descent gives one way of minimizing J. Let's discuss a second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm. In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θ_j's, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:

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

There is no need to do feature scaling with the normal equation. With the normal equation, computing the inversion has complexity $\mathcal{O}(n^3)$, whereas gradient descent has time complexity $\mathcal{O}(kn^2)$. So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.

Note: Non-invertibility/singularity of the feature/design matrix X can become an issue when using the normal equation, but only in cases where the number of samples is less than or equal to the number of features ($m \leq n$).

2 Logistic Regression

3 Neural Networks

4 Neural Networks/Backpropagation