Regression

based on a document by Andrew Ng

May 5, 2014

1 Linear regression

Lets consider a supervised learning example where data points are houses with 2 features (X^1= living area; X^2 = number of bedrooms) and labels are the prices.

Living area (ft^2)	#bedrooms	price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
...	...	...

Each datapoint is thus $(\mathbf{x}_t, y_t)$, where $\mathbf{x}_t = (x_t^1, x_t^2)$ are the living area and the number of bedrooms respectively, and y_t is the price. The main problem addressed by *regression* is: Can we predict the price (or label) y_t from input features $\mathbf{x}_t$? Today we study *linear regression*, that is if we can predict the price from the features using a linear regressor

$$h_w(\mathbf{x}) = w^0 + w^1 x^1 + w^2 x^2$$

where $w = (w^0, w^1, w^2)$ are the parameters of the regression function. Within the class of linear functions (regressors) our task shall be to find the best parameters w. When there is no risk of confusion, we will drop w from the h notation, and we will assume a dummy feature $x^0 = 1$ for all datapoints such that we can write

$$h(\mathbf{x}) = \sum_{d=0}^{D} w^d x^d$$

where d iterates through input features 1,2,... ,D (in our example $D = 2$).

What do we mean by the best regression fit? For today, we will measure the error (or cost) of the regression function by the *squared error*

$$J(w) = \sum_t (h_w(\mathbf{x}_t) - y_t)^2$$

and we will naturally look for the w that minimizes the error function. The regression obtained using the squared error function J is called *least square regression*, or *least square fit*. We present two methods for minimizing J.

2 Least mean squares via Gradient descent

2.1 Gradient descent

The gradient descent algorithm finds a local minima of the objective function (J) by guessing an initial set of parameters w and then "walking" episodically in the opposite direction of the gradient $\partial J/\partial w$. Since w is vector valued (3 components for our example) we need to perform the update for each component separately

$$w^j = w^j - \lambda \frac{\partial J(w)}{\partial w^j}$$

where λ is the *learning rate* parameter or the step of the update. To see this in practice, lets consider an example (other than the mean square error): say $J(x) = (x-2)^2 + 1$ and the initial guess for a minimum is $x_0 = 3$. The differential of J is

$$\frac{\partial J(x)}{\partial x} = 2x - 4$$

Assuming a fixed $\lambda = 0.25$, the first several episodes of gradient descent are:

$$x_1 = x_0 - \lambda \frac{\partial J(x_0)}{\partial x} = 3 - .25(2 * 3 - 4) = 2.5$$
$$x_2 = x_1 - \lambda \frac{\partial J(x_1)}{\partial x} = 2.5 - .25(2 * 2.5 - 4) = 2.25$$
$$x_3 = x_2 - \lambda \frac{\partial J(x_2)}{\partial x} = 2.25 - .25(2 * 2.25 - 4) = 2.125$$

Figure1 illustrates the episodic process.

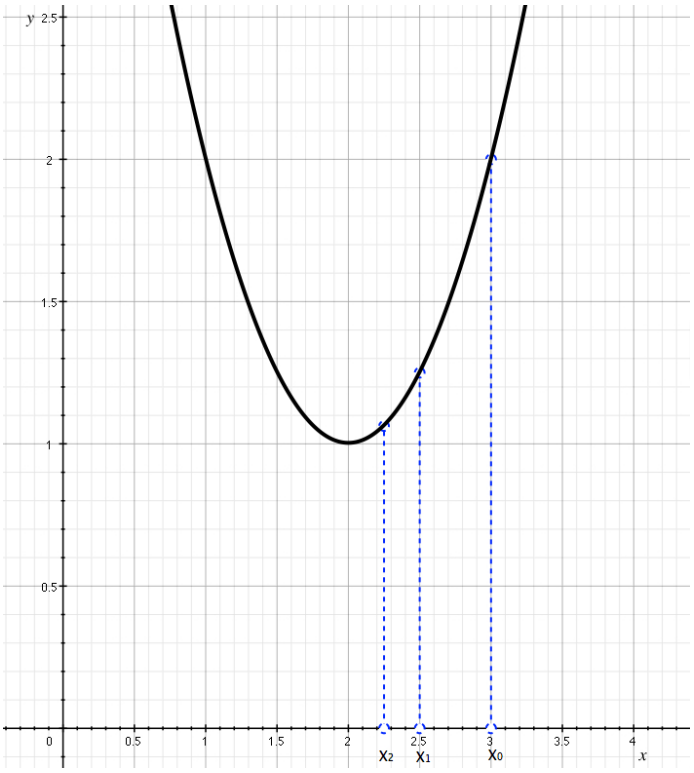

Figure 1: Gradient descent iterations

2.2 Learning rates

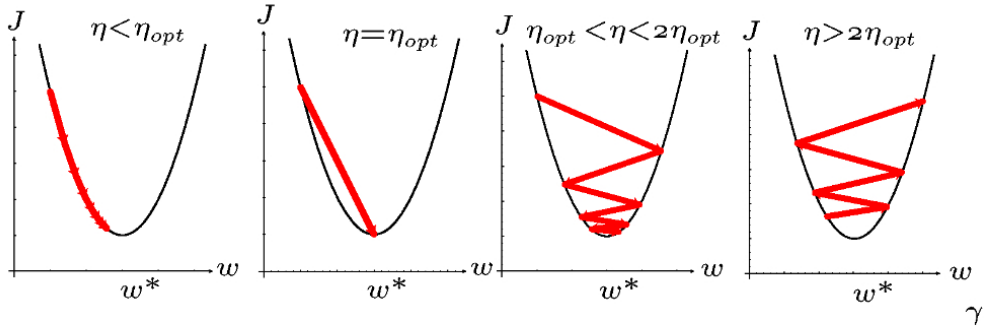

Figure 6.18: Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, the system will oscillate but nevertheless converge, but training is needlessly slow. If $\eta > 2\eta_{opt}$, the system diverges.

Figure 2: learning rates

2.3 Least mean squares

In order to apply gradient descent to our problem, we need first to differentiate our error (objective) function J.

$$J(w) = \frac{1}{2}\sum_t (h_w(\mathbf{x}_t) - y_t)^2$$

Here we put 1/2 in the formula to make the subsequent calculation easier. It doesn't change the optimal w.

Let's look at the differential for the case when there is only one datapoint (so there are no sums)

$$\begin{aligned}
\frac{\partial J(w)}{\partial w^j} &= \frac{\partial \frac{1}{2}(h_w(\mathbf{x}) - y)^2}{\partial w^j} \\
&= (h_w(\mathbf{x}) - y)\frac{\partial (h_w(\mathbf{x}) - y)}{\partial w^j} \\
&= (h_w(\mathbf{x}) - y)\frac{\partial (\sum_j w^j x^j - y)}{\partial w^j} \\
&= (h_w(\mathbf{x}) - y)x^j
\end{aligned}$$

Thus the gradient descent update rule (for one data point) is

$$w^j = w^j - \lambda(h_w(\mathbf{x}) - y)x^j$$

The update has to be performed for all components j = 1,2,..., D simultaneously (in parallel).

There are two ways to adapt the rule derived above for the case where many datapoints. The first one, *batch gradient descent*, is to use the error function J for all datapoints when differentiate (essentially make the update for all points at once). The sum inside J expression propagates to the differential expression and the update becomes

$$w^j = w^j - \lambda\sum_t (h_w(\mathbf{x}_t) - y_t)x_t^j$$

for all j=1,2,.., D. Note that in batch gradient descent, a single update require analysis of all datapoints; this can be very tedious if the data set is large.

The second way to involve all datapoints is to make the update separately for each datapoint (*stochastic gradient decent*). Each update step becomes then a loop including all datapoints:

LOOP for t=1 to m

$$w^j = w^j - \lambda(h_w(\mathbf{x}_t) - y_t)x_t^j \text{ for all } j$$

END LOOP

Stochastic (or online) gradient descent advances by updating based on one datapoint at a time. if the regression problem is not too complicated, often few iterations are enough to converge and so in practice this version of gradient decent is often faster. it is possible (although very unlikely) to have a problem where stochastic updates "dance" around the best w (that realizes minimum error) without actually converging to it.

To conclude regression via gradient descent, we make one final observation. The objective function J is convex, which means any local minima is in fact a global minima, thus the gradient descent (or any method that finds local minima) finds a global minima.

We can verify the convexity of J like this:

We already know that

$$\frac{\partial(J(w))}{\partial w^i} = \sum_t (\sum_d w^d x_t^d - y_t)x_t^i$$

We take partial derivative again

$$\frac{\partial^2 J(w)}{\partial w^i \partial w^j} = \sum_t x_t^i x_t^j$$

So the Hessian matrix of $J(w)$ is $X^T X$. Clearly, $\forall w, w^T X^T X w = (Xw)^T (Xw) \geq 0$. So Hessian matrix of $J(w)$ is positive semi-definite, which implies that J is convex.

3 Least mean square probabilistic interpretation

Why squared error? We show now that the objective J used is a direct consequence of a very common assumption over the data. Lets look at the errors

$$\epsilon_t = h(\mathbf{x}_t) - y_t$$

and lets make the assumption that they are IID according to a gaussian (normal) distribution of mean $\mu = 0$ and variance σ^2. That we write $\epsilon\, \mathcal{N}(0, \sigma^2)$ or

$$p(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right)$$

which implies

$$p(y|x; w) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(w\mathbf{x} - y)^2}{2\sigma^2}\right)$$

Note that above w is a parameter (array) and not a random variable. Given the input X, what is the probability of Y given the parameters w ? Equivalently, this is the *likelihood* that w is the correct parameter for the model

$$L(w) = L(w; X, Y) = p(Y|X; w)$$

Using the IID assumption the likelihood becomes

$$\begin{aligned} L(w) &= \prod_t p(y_t|\mathbf{x}_t); w \\ &= \prod_t \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(w\mathbf{x}_t - y_t)^2}{2\sigma^2}\right) \end{aligned}$$

Since we have now a probabilistic model over the data, a common way to determine the best parameters is to use *maximum likelihood*; in other words find w that realizes the maximum L. Instead of maximizing L we shall maximize the *log likelihood* $\log L(w)$ because it simplifies the math (and produces the same "best" w)

$$\begin{aligned} l(w) &= \log L(w) \\ &= \log \prod_t \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(w\mathbf{x}_t - y_t)^2}{2\sigma^2}\right) \\ &= \sum_t \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(w\mathbf{x}_t - y_t)^2}{2\sigma^2}\right) \\ &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_t (h_w(\mathbf{x}_t) - y_t)^2 \end{aligned}$$

Hence, maximizing the likelihood L produces the same w as minimizing the mean square error (since the front term does not depend on w). That is to say that, if we believe the errors to be IID normally, then the maximum likelihood is obtained for the parameters w that minimizes the mean square error.

4 Classification and logistic regression

In classification, the labels y are not numeric values (like prices), but instead *class labels*. For today, lets assume that we have two classes denoted by 0 and 1; we call this *binary classification* and we write $y \in \{0, 1\}$.

4.1 Logistic transformation

We could, in principle try to run the linear regression we just studied, without making use of the fact that $y \in \{0, 1\}$. (Essentially assume y are simply real numbers). There are several problem with this approach: first, the regression assumes the data supports a linear fit, which might not be true anymore for classification problems; second, our regressor $h(\mathbf{x})$ will take lots of undesirable values (like the ones far outside the interval [0,1]).

To make an explicit mapping between the real valued regressor h and the set {0,1}, we would like a function that preserves differentiability and has a easy interpretable meaning. We choose the *logistic function*, also called *sigmoid*

$$g(z) = \frac{1}{1 + e^{-z}}$$

Note that g acts like a indicator for {0,1}, but it is much more sensitive than a linear function. We can make it even more sensitive by, for example, doubling the input z before applying the function. Lets state

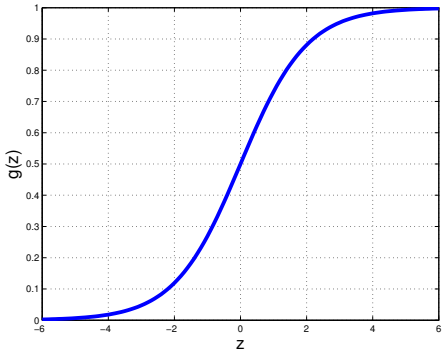

Figure 3: Logistic function

the derivative of g, since we are going to use it later on

$$\begin{aligned} g'(z) &= \frac{\partial g(z)}{\partial z} \\ &= \frac{1}{(1+e^{-z})^2} e^{-z} \\ &= \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}}\right) \\ &= g(z)(1-g(z)) \end{aligned}$$

4.2 Logistic regression

We apply g to the linear regression function to obtain a *logistic regression*. Our new hypothesis (or predictor, or regressor) becomes

$$h_w(\mathbf{x}) = g(w\mathbf{x}) = \frac{1}{1+e^{-w\mathbf{x}}} = \frac{1}{1+e^{-\sum_d w^d x^d}}$$

Because logistic regression predicts probabilities, we can fit it using likelihood. We assume

$$P(y=1|x;w) = h_w(x)$$

$$P(y=0|x;w) = 1 - h_w(x)$$

We can combine these two formulas into one:

$$P(y|x;w) = (h_w(x))^y (1-h_w(x))^{1-y}$$

The likelihood is therefore

$$\begin{aligned} L(w) &= p(y|X;w) \\ &= \prod_{i=1}^{m} p(y_i|x_i;w) \\ &= \prod_{i=1}^{m} (h_w(x_i))^{y_i} (1-h_w(x_i))^{1-y_i} \end{aligned}$$

To make the calculation easier, we take the log and get the log likelihood:

$$\begin{aligned} l(w) &= \log L(w) \\ &= \sum_{i=1}^{m} y_i \log h(x_i) + (1-y_i)\log(1-h(x_i)) \end{aligned}$$

Our goal here is to maximize this likelihood, so we calculate the derivative and use gradient ascent.

Like before, let's first consider just one data point.

$$\begin{aligned} \frac{\partial}{\partial w^j} l(w) &= (y\frac{1}{g(wx)} - (1-y)\frac{1}{1-g(wx)})\frac{\partial}{\partial w^j} g(wx) \\ &= (y\frac{1}{g(wx)} - (1-y)\frac{1}{1-g(wx)})g(wx)(1-g(wx))\frac{\partial}{\partial w^j} wx \\ &= (y(1-g(gx)) - (1-y)g(wx))x^j \\ &= (y - h(x))x^j \end{aligned}$$

We get the stochastic gradient ascent rule:

$$w^j := w^j + \lambda(y_i - h_w(x_i)))x_i^j$$

We also the batch gradient ascent rule:

$$w^j := w^j + \lambda \sum_i (y_i - h_w(x_i)))x_i^j$$

4.3 Newton's method

Newton's method is a widely used optimization method. Like gradient descent, Newton's method is an iterative method. It starts with some initial guess, and then iteratively improves the guess. The major difference is that, while gradient descent uses the first derivative to improve the guess, Newton method uses both the first and the second derivative to improve the guess.

Let's first consider a simple case where the function we want to minimize f has only one parameter w. Suppose we are at round n and our current guess about optimal w is w_n. If we approximate $f(w)$ around w_n using its second order Taylor expansion, we get

$$f(w) \approx f(w_n) + f'(w_n)(w - w_n) + \frac{1}{2}f''(w_n)(w - w_n)^2$$

This is a quadratic function. Now we wish to find the w which minimizes this quadratic function. We set the derivative with respect to w equal to zero, and get

$$w = w_n - \frac{f'(w_n)}{f''(w_n)}$$

This w should be a better guess than w_n. So we can set

$$w_{n+1} = w_n - \frac{f'(w_n)}{f''(w_n)}$$

The above formula is the update rule of Newton's method. Or, we can write it this way:

$$w := w - \frac{f'(w)}{f''(w)}$$

We can repeat this procedure many times until the guess converges.

Now suppose the function to be minimized f has multiple parameters $w^1, w^2, ..., w^d$. Following the same derivation, we can get the following update rule:

$$\mathbf{w} := \mathbf{w} - \mathbf{H^{-1}} \nabla f(\mathbf{w}),$$

where $\nabla f(\mathbf{w})$ is the vector of partial derivative of f with respect to $\mathbf{w}$; $\mathbf{H}$ is the Hessian matrix defined by

$$\mathbf{H}_{ij} = \frac{\partial^2 f}{\partial w^i \partial w^j},$$

and $\mathbf{H^{-1}}$ is the inverse of the Hessian matrix.

Newton's method often requires fewer iterations to converge compared with batch gradient descent method. However, in each iteration, computing the inverse of the full Hessian matrix can take a lot of space and time, especially when the dimension d is large. In practice, quasi-Newton algorithms such as BFGS algorithm have been developed that use approximations to the Hessian.