CSE/ECE-471: Statistical Methods in AI **Spring 2020**

Linear Regression

Prepared by: K L Bhanu, Saptarshi Manna, Aklesh Mishra

1 Predictive Modeling and Linear Regression

In predictive modeling, the task is to learn a model using given observations and predict the output of a new sample accordingly.

1. **Sample data** : the data that we collect that describes our problem with known relationships between inputs and outputs.
2. **Learn a Model** : the algorithm that we use on the sample data to create a model that we can later use over and over again.
3. **Making Predictions** : the use of our learned model on new data for which we don't know the output.

Regression predictive modeling is the task of approximating a mapping function (f) from input variables (X) to a continuous output variable (Y).

A continuous output variable is a real-value, such as an integer or floating point value. These are often quantities, such as amounts and sizes.

To perform a linear regression on a data set, we assume that the data is linearly related to the output. Here in this document following will be discussed :

1. **Linear Least Square Regression** : Formulating cost function for regression
2. **Methods to achieve** : Different methods will be discussed to minimize the cost
3. **Gradient Descent** : Finding minima of the cost function and making small changes along the way to get to a point where cost is minimized
4. **Normal Form** : Finding minima of cost function using matrix calculation
5. **Regularization** : Adding a regularization term to the cost function to avoid overfitting to the training data
6. **Linear Regression when relationship involves polynomial function** : Linear regression can be used even when output is related to input in a non linear way

Linear regression was developed in the field of statistics and is studied as a model for understanding the relationship between input and output numerical variables. Linear regression is a simple approach to supervised learning. It assumes that the dependence of Y on $X_1, X_2, ...X_d$ is linear.

To perform supervised learning, we must decide how we are going to represent functions. As an initial choice, let's say we decide to approximate Y as a linear function of X.

$$Y_i = w_0 + w_1X_1 + w_2X_2 + w_3X_3 + ... + w_dX_d$$

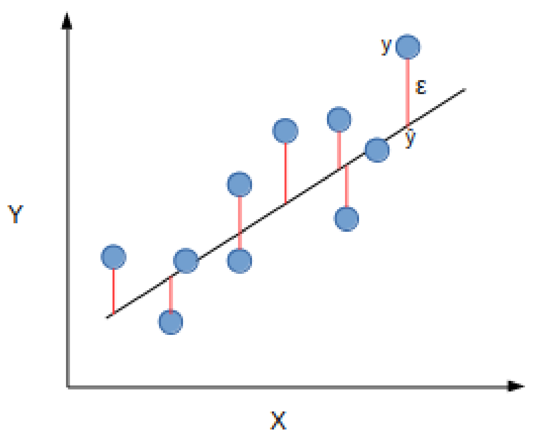

Figure 1: The distance between the predicted and actual value is the error

The above equation with an error term(as we are approximating) can be written as :

$$Y_i = W^T X_i + \epsilon$$

For 2 dimensional space the equation becomes

$$Y = w_0 + w_1 X$$

where w_0 is the intercept on the y axis and w_1 is the slope of the line.

This is an equation of a line. In higher dimensions its referred to as a hyperplane.

1.1 Finding Error : Linear least square formulation

Now, given a training set, how do we pick, or learn, the parameters of W? One reasonable method seems to be to make Y_i close to actual Y, at least for the training examples that we have. In other words we are trying to minimize ϵ (fig 1) as much as we can.

Let $\hat{y}_i = w_0 + w_1 x_i$ be the prediction for Y based on the i^{th} value of X. Then $\epsilon_i = y_i - \hat{y}_i$ represents the i^{th} residual. Total error can be calculated taking a sum of all ϵ_i. And we are interested to minimize the total error.

Before we proceed further, we should notice that the error comes with a sign each time an i^{th} error is calculated, depending on whether the predicted value falls on the upper or lower side of the actual value. Thus, the summation of errors will reduce actual total error [Ng16];. This can be fixed by taking square of error term and then summing it up.

The error equation can be written as

$$J(W) = \sum_{i=1}^{n} \epsilon_i = \sum_{i=1}^{n} (W^T X_i - Y_i)^2$$

$J(W)$ is called cost function.

From the above equation we can see that it is a quadratic equation (2). As the function is a convex function, we should be able to find a minima. That means for some value of W, $J(W)$ will be minimized.

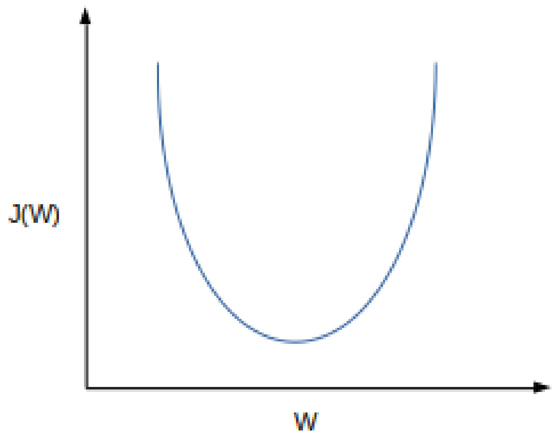

Figure 2: For least squares, J(W) is a quadratic convex function and as a result has only one minima. The W for which J(W) is minimum is the fit with least error.

2 Estimation of Model Parameter

A linear fit to the data points can be represented by the equation $Y = W^T X$ where W^T=[w_0,w_1,......,w_d] and X=[1,x_1,x_2,......,x_d] for d dimensional points. The equation can be modified to be written as equation 1 which displays the cost function.

$$J(W) = \frac{1}{2}\sum_{i=1}^{n}(W^T X_i - Y_i)^2 \tag{1}$$

The equation 2 displays optimization function.

$$\underset{W}{\operatorname{argmin}}\ J(W) \tag{2}$$

Out of the various methods to find the weights which minimise the objective function, two of the common ones, stochastic gradient descent and normal form methods are discussed below.

3 Stochastic Gradient Descent

As we discussed in the previous section, minimizing the total error between predicted and actual values would provide us with the best fit, and therein lies the issue. A closed form solution to finding the global minima for any general function doesn't exist yet. From basic calculus, you may remember that at minima and maxima, the slope of the function becomes zero, and minima would be the points with negative second derivative. Finding such points, ie finding the roots of the first derivative function may not always be possible. Moreover, the minima found may not be the global minima. This issue still persists but in 1951, Robbins and Monro came up with an aproximation method [RM51]; an adaptation of which we know today as stochastic gradient descent (SGD) method; to estimate the minima of a function. The idea behind SGD is quite a simple and an elegant one. Given the objective function, starting from any random point, we move along the function in such a way that with each step taken, the function value decreases, and therefore after a limited number of steps the functional value would converge towards the minima. This method (with some variations) is still one of the most popular (if not, the most popular) methods to estimate minima for any function.

Intuitively, given the objective function, we need to find the step size and direction to move. As we know that function value decreases when the gradient is negative, and increases when the gradient is positive, a logical direction to move could be in the direction of negative gradient. This is what SGD does. Figure

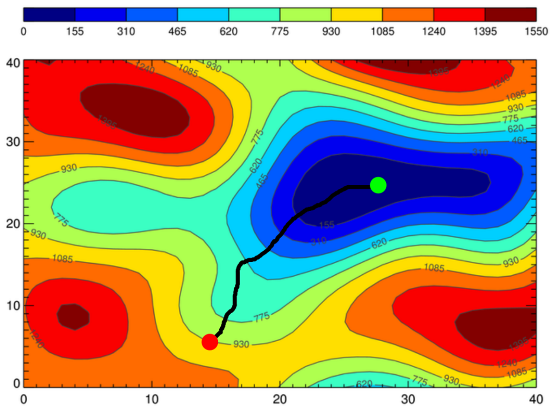

Figure 3: Stochastic gradient descent is used to find the minima of a function (in our case, the function is objective function J(W). The contours indicate the the region where the function value is same and the colour indicates the range of function value. In the figure the red dot is the initial point (chosen randomly) and the green dot is the found minima. The path traced between the starting and ending point is the step by step iteration taken by SGD to find the minima.

4 illustrates this strategy. With the example of previously discussed objective function, mathematically we would want to update w every iteration such that $J(w)$ decreases.

$$w_{t+1} \leftarrow w_t + h \qquad \text{such that J(w) decreases} \tag{3}$$

where h is the vector step to be taken. This step is also known as the update step. Now according to this strategy, we would like to move in the direction of negative gradient, and the magnitude (or size) of the step to be taken can be passed as a positive parameter (δ) also known as step size or learning rate. With that, our update step can be written as

$$w_{t+1} \leftarrow w_t + (-\delta J'(w_t)) \tag{4}$$

We may have shown intuitively that using this strategy to update the weights (w) would decrease the total error, but this is true mathematically too. The value of a function $f(x)$ in the positive neighbourhood of $x = a$ can be estimated with Taylor series expansion as

$$f(a+h) \approx f(a) + f'(a)h + \frac{f''(a)}{2!}h^2 + \frac{f'''(a)}{3!}h^3 + \cdots \tag{5}$$

where h is a small positive value. All the terms after the first derivative term here can be ignored as h is very small, and exponentiation will only make it smaller. Now our estimate becomes

$$f(a+h) \approx f(a) + f'(a)h$$

Substituting $f(x)$ with our objective function $J(w)$ and a with w_t we have

$$J(w_t + h) \approx J(w_t) + J'(w_t)h$$

From (3) we have $w_{t+1} = w_t + h$, and from (4) we have $h = -\delta J'(w_t)$. Substituting, we get

$$J(w_{t+1}) \approx J(w_t) + J'(w_t)(-\delta J'(w_t))$$

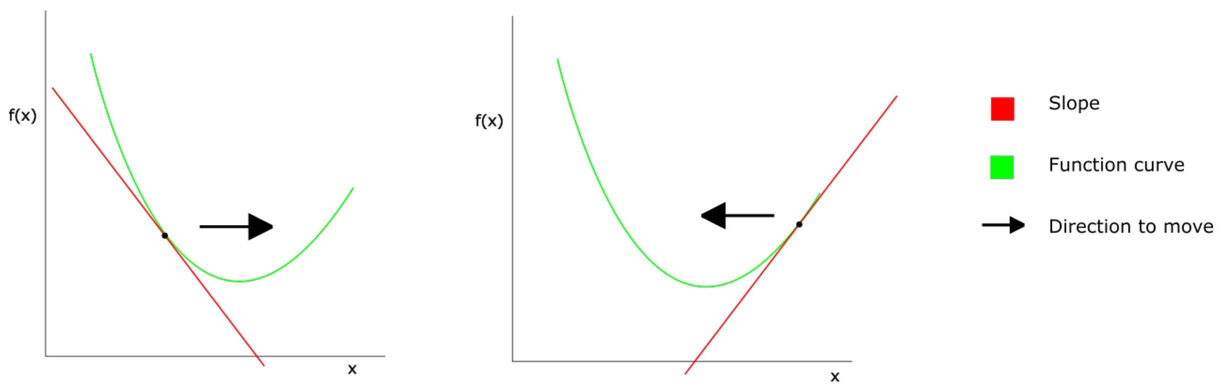

Figure 4: Moving in the direction of negative gradient brings us closer to the minima, from the diagram, when the gradient is negative (left), direction to move is in positive direction and vice versa. With small step size, the descent will converge towards minima.

This is basically the value of the objective function after one update step. Further simplifying, we have

$$J(w_{t+1}) \approx J(w_t) - \delta J'(w_t)^2$$

The term $\delta J'(w_t)^2$ is a positive quantity. Removing a postive quantity from anything will reduce it so therefore

$$J(w_{t+1}) \leq J(w_t)$$

This small proof basically states that, gradient descent algorithm, at every step reduces the objective function value, and as a result after some iterations, $J(w)$ will converge towards the minima.

Finally, in order to find the weights (W) for a linear regression problem using least squares method, the update step (from (4)) would be

$$J(W) = \frac{1}{2}\sum_{i=1}^{n}(W^T X_i - Y_i)^2$$

,

$$J'(W) = \frac{1}{2} * 2 * \sum_{i=1}^{n}(W^T X_i - Y_i)X_i$$

,

$$W_{t+1} = W_t - \delta(\sum_{i=1}^{n}(W^T X_i - Y_i)X_i) \tag{6}$$

4 More about Gradient Descent

We now know: (i) How to define a loss function (ii) how to optimize it with gradient descent update equations. Stochastic gradient descent finds the minima for a function by iteratively travelling along the direction when function value decreases. The preference for the step size is a small one (10^-3), but there is no predefined rule to select it, other than from empirical observations. The number of iteration for SGD to run could be when the change in objective decrease after a step is less than a threshold, that is objective function has converged. Apart from that, frquency of updating the weights has some significance in SGD's convergence. From our update step 6 we see that that the weights (w) are updated after computing error for all the samples present. This is called the *batch version of SGD* and may not be convenient always, especially in cases of large datasets as it is very slow to output any result.

Updating the weights immediately after every sample is called the *online version of SGD*. It is fast to process, but is highly sensitive to outliers. An optimum trade off between time and accuracy would be the mini-batch SGD, which updates the weights after a chunk of samples. This is usually more robust, quick and is the most used version of stochastic gradient descent.

Stochastic gradient descent performs well to find the minima of a given function. But it is not without it's limitations. The algorithm often fails to converge in cases where a large step size is taken. It is also dependant on initialisation of the first weight, and is slow to converge due to small step sizes. People have come up with smarter strategies built upon the idea of SGD to find the minima faster. Three such methods that shine out are the Newton-Rhapson method, the co-ordinate descent and steepest descent. Briefly, Newton-Rhapson methods tries to find the best direction to descend by using the hessian matrix (second derivative) of the data and finds the minima in one shot, but is numerically hard to compute. Steepest descent uses a second optimisation to find the best direction and step size to descend with, and again is computationally intensive. Co-ordinate descent optimises for the best direction and step size in every dimension for the regressor, and converges fast (as the dimensionality of the regressor is fixed). More on these can be read in chapter 9 of the book [BV04]

5 Normal Equation Method

In this method, we minimize the cost function by finding the derivative of $J(W)$ wrt W and equating it to zero rather than updating weights to find the perfect fit. Our objective function is

$$J(W) = \frac{1}{2}\sum_{i=1}^{n}(W^TX_i - Y_i)^2$$

The matrix representation of cost function is by

$$J(W) = \frac{1}{2}(AW-Y)^T(AW-Y)$$

Where A=$[X_1^T, X_2^T.....X_n^T]^T$, Y =$[y_0, y_1, ...y_{n-1}]^T$ and W=$[w_0, w_1, ...w_d]^T$

Derivation to find optimal value of parameter vector is given below:

$$\frac{\partial J(W)}{\partial W} = \frac{\partial}{\partial W}[(AW-Y)^T(AW-Y)]$$
$$= \frac{\partial}{\partial W}[(AW)^TAW - (AW)^TY - Y^TAW + Y^TY)]$$

Let P=AW and Q=Y are both vector.Using symmetry property, P^TQ=PQ^T,the equation can be deduce like this,

$$\frac{\partial J(W)}{\partial W} = \frac{\partial}{\partial W}[(AW)^TAW - 2(AW)^TY + Y^TY]$$
$$= \frac{\partial}{\partial W}[W^TA^TAW - 2(AW)^TY + Y^TY)]$$
$$= \frac{\partial}{\partial W}[2A^TAW - 2A^TY)]$$

Now equate it to zero to get the optimal W.

$$A^T(AW-Y) = 0$$
$$\Rightarrow A^TAW = A^TY$$
$$\Rightarrow W = (A^TA)^{-1}A^TY$$

The term $(A^TA)^{-1}A^TY$ is also known as the pseudo-inverse of A (think of it as writing $\frac{1}{x}$ as $\frac{x}{x^2}$). In this method, there is no need to choose any learning rate δ which is in gradient descent, since δ in gradient descent determine the convergence of cost function at minima. But normal equation method is computationally inefficient as it involves computing inverse of a matrix ($O(n^3)$).

6 Linear Regression for non linear relations

We have already seen if the output is linearly related to input, then we can find the relationship with least error. But can the model predict the output if the relation is not linear? Let us explore the idea.

W and X is defined as follows:

$$W_i = [w_0, w_1, w_2]$$

$$X_i = [1, x_i, x_i^2]$$

$$Y_i = W^T X = w_0 + w_1 x_i + w_2 x_i^2$$

Will we be able to fit a linear model with the above parameters?
Yes. Let us take another variable $z_i = x_i^2$

Then $X_i = [1, x_i, z_i]$. And the relationship becomes linear. We will still be finding $Y_i = W^T X_i$ and will try to fit the data linearly. In a similar fashion, this method can be used to fit any higher degree polynomial, but the choice of the correct degree polynomial is up for discussion. High degree polynomials may over-fit the data which leads to the next topic in our discussion.

7 Regularization

Regularization of a model is introducing penalty on the weights in the objective/loss function of the model to avoid over-fitting or highly complex models. The need to regularize a model decreases as we increase the number of samples to train with. Since in a lot of problems, data points may not be sufficient, or the model grows to be too complex that it over-fits, regularisation is a good solution.

$$J_R(W) = J(W) + \lambda \sum_{i=1}^{d} (||w_i||)^2 \tag{7}$$

The regularized objective function J_R has a penalty on the L2 norm of the weights. λ is a parameter to limit the norm of the weights. This is also known as L2-norm regularization or lasso regression. Another common regularisation method is the L1-norm regularisation, also knows as ridge regression.

References

[RM51] H. Robbins and S. Monro. “A stochastic approximation method”. In: *Annals of Mathematical Statistics* 22 (1951), pp. 400–407.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* USA: Cambridge University Press, 2004. ISBN: 0521833787.

[Ng16] Andrew Ng. “CS229 lecture notes on Supervised Learning”. In: *Stanford Machine Learning Lecture* (2016), pp. 3–4.