Linear Least Squares Algorithm

Least Squares is an algorithm most commonly used to fit a model to an over-defined data set in the most optimal way.

A good example of this is finding a curve of best fit through a collection of points. Developing a model to fit such a data set is a very standard optimization problem that could be approached with a variety of methods, however if Least Squares can be used it is often a very fast and efficient option.

Example:

Consider we're trying to fit a plane to three points. Given points P_1, P_2, P_3, such that $P_i = (x_i, y_i, z_i)$, the equation of the plane that fits all three points is:

$$z = Ax + By + C$$

with A, B, C, D which satisfy the equations:

$$\left\{ \begin{matrix} z_1 = Ax_1 + By_1 + C \\ z_2 = Ax_2 + By_2 + C \\ z_3 = Ax_3 + By_3 + C \end{matrix} \right\}$$

this can be rearranged as:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \end{bmatrix},$$

$$\boldsymbol{z = M \cdot p}$$

the parameters A, B, and C can then simply be found by computing the inverse of M and multiplying it by z:

$$\boldsymbol{p = M^{-1} \cdot z}$$

Now this example was quite simple, and could have been solved using more basic methods, but the procedure here is essential to understand before performing Linear Least Squares.

The Procedure:

The procedure here is finding a way to reduce the equation at hand (the model) into a simple equation that equates the dependant variable of an equation to a matrix product between the independent variables and the model parameters.

1. Develop model: $z = Ax + By + C$
2. Convert into matrix product: $z = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \end{bmatrix}$
3. Replace the variables with vectors containing all of the data points:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

4. Solve for the parameters by matrix inverse:

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \left(\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \right)^{-1} \cdot \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

Now the problem that we are looking at is not to solve for the parameters of a plane, we are rather looking to find the plane of best fit. This problem is traditionally orders of magnitude harder than simply solving for a plane using three points.

What I would like to show is that, using Linear Least Squares, this problem of finding a model of best fit is no harder than the procedure detailed above.

The Algorithm:

In the case of the over-defined data set, there are more points than parameters to find, which means that the matrix involved is non-square. For example, if there were four points, the equation would look more like this:

$$\left\{ \begin{array}{l} z_1 = Ax_1 + By_1 + C \\ z_2 = Ax_2 + By_2 + C \\ z_3 = Ax_3 + By_3 + C \\ z_4 = Ax_4 + By_4 + C \end{array} \right\},$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

The above procedure would be impossible to apply to this not only because there is no inverse to a 4 by 3 matrix, but it is also because (generally) the system would have no concrete solution {A, B, C} that satisfies all of the equations. The best that we can do is find a solution that has the minimum error.

Traditionally we would use a complicated algorithm like gradient descent, however if we can reduce the problem to a matrix multiplication, there is a much simpler solution.

The error of any equation in the system is the difference between z value of any data point and the product of the dependant values and the parameters. In other words, if the equation of the model is defined as:

$$z = Ax + By + C$$
$$(z = M \cdot p)$$

the error of any given data point is given by:

$$e_i = z_i - (Ax_i + By_i + C)$$

therefore the total system is defined as:

$$\boldsymbol{e = z - M \cdot p}$$

Now this is where it gets interesting! It has been proven that this error metric can be minimized in the general case, by calculating the derivative of the error and setting it to zero.

$$0 = -2M^T \cdot z + 2M^T \cdot M \cdot p$$
$$p = (M^T M)^{-1} M^T \cdot z$$

WHAT! WHAT?!? This is a concrete equation that gives the optimal parameters for them model of best fit, using just a couple of matrix transposes and multiplication. Better yet, the operation shown in the equation above is known as the Moore Penrose pseudo inverse function ($M^\dagger$), and it is already coded into many matrix libraries such as Numpy and Matlab.

$$M^\dagger = (M^T M)^{-1} M^T$$
$$\therefore p = M^\dagger \cdot z$$
OR
$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} x_1 + y_1 + 1 \\ x_2 + y_2 + 1 \\ x_3 + y_3 + 1 \\ x_4 + y_4 + 1 \\ \vdots \end{bmatrix}^\dagger \cdot \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ \vdots \end{bmatrix}$$

Let's let that sink in.

Not only does this works for any number of points, but we can even replace the model entirely. We can use it to fit any line, plane, or hyper-plane, we can use it to fit curves such as circles, it's literally beautiful. I'll show the circle example and then we can talk about implementing it in code.

In Action

So let's say we have an arbitrary number of 2D points, and we want to find the circle that best fits them. Well we start by defining the model mathematically:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

we then rearrange it to separate model parameters (r, x_0, y_0) from data variables:

$$-2xx_0 + {x_0}^2 - 2yy_0 + {y_0}^2 - r^2 = -x^2 - y^2$$

then rearrange the equation into a single matrix multiplication:

$$\begin{bmatrix} -2x & -2y & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ y_0 \\ {x_0}^2 + {y_0}^2 - r^2 \end{bmatrix} = -x^2 - y^2$$

Now that we have this equation in the form:

$$\boldsymbol{M} \cdot \boldsymbol{p} = \boldsymbol{z}$$

we can substitute in each value (x_i, y_i) for x and y, and we can simply find optimal parameters x_0, y_0 and r using our handy dandy equation from earlier:

$$\begin{bmatrix} x_0 \\ y_0 \\ {x_0}^2 + {y_0}^2 - r^2 \end{bmatrix} = \begin{bmatrix} -2x_1 & -2y_1 & 1 \\ -2x_2 & -2y_2 & 1 \\ & \vdots & \end{bmatrix}^{\dagger} \cdot \begin{bmatrix} -{x_1}^2 - {y_1}^2 \\ -{x_1}^2 - {y_1}^2 \\ \vdots \end{bmatrix}$$

as you may be able to see, this method requires a bit of hand calculation, however once that is done, the implementation is very consistent and simple.

in python (using numpy), the function used to fit a circle to a set of points could look as follows. I literally can't stress enough how simply and fast this implementation is. If you can ever find a to make this method work in your regression algorithms, I would highly recommend using it.

```
def fit_circle(point_set):
       M = [[-2*point[0], -2*point[1], 1] for point in point_set]
       z = [[-1*point[0]**2 - point[1]**2] for point in point_set]
       params = numpy.dot( numpy.linalg.pinv(M), z )
       x0 = params[0]
       y0 = params[1]
       r = math.sqrt( x0**2 + y0**2 - params[2] )
       print "The circle has center: (", x0, ",", y0, ") and radius", r
       return x0, y0, r
```

(Note that the Moore Penrose pseudo inverse function is implemented in numpy as `numpy.linalg.pinv()`)

Conclusion:

I hope that this explanation has done a good job explaining what the Linear Least Squares algorithm is and one way that you can use it to make optimization problems very easy to solve in certain cases.

If you have any questions, feel free to check out the following awesome links (in order of awesomeness):

https://isites.harvard.edu/fs/docs/icb.topic515975.files/OLSDerivation.pdf
http://mathworld.wolfram.com/LeastSquaresFitting.html
https://web.williams.edu/Mathematics/sjmiller/public_html/BrownClasses/54/handouts/MethodLeastSquares.pdf
https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_pseudoinverse
http://mathworld.wolfram.com/Moore-PenroseMatrixInverse.html

Or feel free to email me at any time at rowan.ferrabee@gmail.com. I would love to talk to anybody about this, especially to try to understand and explain this whole thing better.

Bonus sections can be found below as usual.

The pseudo inverse function (skippable):

Before explaining the algorithm, it is important to understand a few minor linear algebra operations. Namely, the pseudo inverse function (also known as the Moore Penrose Matrix Inverse) is core to understanding Least Squares. The pseudo inverse function is a method for finding the "inverse" of a matrix that is non-square. Rather that the inverse matrix, which satisfies the property:

$$A \cdot A^{-1} = I$$

the pseudo inverse function satisfies the property:

$$B \cdot B^{\dagger} \cdot B = B$$

which is similar to the inverse function, and will actually be equivalent to it for all square linearly independent matrices.

The pseudo inverse function can be calculated using the formula:

$B^{\dagger} = (B^H \cdot B)^{-1} \cdot B^H$, where B^H is the conjugate transpose function

TL;DR: there exists a pseudo inverse function we can use for non square matrices.

The Proof (also skippable):

Just look at https://isites.harvard.edu/fs/docs/icb.topic515975.files/OLSDerivation.pdf tbh they do a way better job of the proof than I ever could.