Matrix learning

James Newman

April 28, 2010

1 Solving systems of linear equations using matrix inverses

If we have a system of linear equations such as

$$\begin{array}{ccccccc} 3x & + & 2y & - & z & = & 1 \\ 2x & - & 2y & + & 4x & = & -2 \\ -x & + & \frac{1}{2}y & - & z & = & 0 \end{array}$$

We can create a matrix equation that representes the system using 3 matrices:

$$Ax = b$$

x is vector matrix holding the variables

$$x = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

A holds the coefficients of the variables for each equation in the rows.

$$A = \begin{bmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & \frac{1}{2} & -2 \end{bmatrix}$$

When we calculate Ax we get

$$Ax = \begin{bmatrix} 3x + 2y - 1z \\ 2x - 2y + 4z \\ -1z + \frac{1}{2}y - 2z \end{bmatrix}$$

The final matrix, b, contains the constants.

$$b = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$$

Now we solve for the unknowns, x. Since $Ax = b$, if we multiply the inverse of A, namely A^{-1}, by A itself we should get the identity matrix I.

$$A^{-1}Ax = A^{-1}b$$

$$Ix = A^{-1}b$$

$$x = A^{-1}b$$

Matrix inversion is a tricky but algorithmically solvable problem.

To see this is method works, here is the solution to the above system.

$$A^{-1} = \begin{bmatrix} \frac{2}{7} & \frac{1}{2} & \frac{6}{7} \\ 0 & -1 & -2 \\ -\frac{1}{7} & -\frac{1}{2} & -\frac{10}{7} \end{bmatrix}$$

By multiplying A^{-1} by the matrix b

$$x = A^{-1}b = \begin{bmatrix} -\frac{5}{7} \\ 2 \\ \frac{6}{7} \end{bmatrix}$$

Now we can verify the answer by multiplying A by x and seeing that we get b

$$Ax = b = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$$

2 Least squares linear regression

2.1 Problem and background

If we have data, and a model that we think the data follows, how do we find the optimal parameters to fit the data to the model? If we have a function $f(x_1, x_2, \dots)$ that has parameters $a, b, \dots$, and a bunch of data points for the function, how do we find the "best" parameters $a, b, \dots$.

We need a way to define best or optimal. One way to do that is to create an error function and find a minima of it.

is the least squares. for example if we have a model $y = ax + b$ and data point $(x = 2, y = 10)$, we could create a function called the residual which is the actual value subtracted from the value predicted by the parameters a and b

$$residual(a, b) = 10 - (2a + b)$$

The residual alone is not enough to create an error function because it doesn't always produce a positive value and it gives too much weight to outliers in the data. Squaring the residual solves these issues.

We of course don't have just one data point, we have many points that add additional constraints to the function. To create a valid error function, we sum the square of the residuals for all n data points. In this example it would be:

$$error(a, b) = \Sigma_i^n (y_i - (x_i a + b))^2$$

Of course, it is more general to think of this in terms of an $m \times n$ dimensional matrix, where n is the number of samples an m is the number of variables.

2.2 Minimizing the error function

If we want to minimize the error function, we are really looking for when the slope of $error(a, b) = 0$. This would be a local maxima/minima.

For example, using a linear equation with 2 variables x and y,

$$error(a,b) = \Sigma(-b\,z + y - a\,x)^2$$
$$error(a,b) = \Sigma\left(b^2\,z^2 - 2\,b\,y\,z + 2\,a\,b\,x\,z + y^2 - 2\,a\,x\,y + a^2\,x^2\right)$$

We are finding the partial derivative of the function over each parameter, both a and b. The derivative distributes accross addition and thus to inside of the summation.

$$\frac{d\ error(a,b)}{d\ a} \equiv \Sigma\left(2ax^2 + 2bxz - 2xy\right)$$
$$\frac{d\ error(a,b)}{d\ b} \equiv \Sigma\left(2axz + 2bz^2 - 2yz\right)$$

We can distribute the summation accross the terms, and we set the error function to be zero.

$$\begin{array}{ccccccc} 0 & = & \Sigma 2ax_i^2 & + & \Sigma 2bx_iz_i & + & \Sigma - 2x_iy_i \\ 0 & = & \Sigma 2ax_iz_i & + & \Sigma 2bz_i^2 & + & \Sigma - 2z_iy_i \end{array}$$

we can also factor the constants and the parameters that aren't indexed by i out of the summations.

$$\begin{array}{ccccccc} 0 & = & 2a\Sigma x_i^2 & + & 2b\Sigma x_iz_i & + & -2\Sigma x_iy_i \\ 0 & = & 2a\Sigma x_iz_i & + & 2b\Sigma z_i^2 & + & -2\Sigma z_iy_i \end{array}$$

Next we factor the constant 2 out, as well as rearrange the equation to get the parameters on one side.

$$\begin{array}{ccccc} a\Sigma x_i^2 & + & b\Sigma x_iz_i & = & \Sigma x_iy_i \\ a\Sigma x_iz_i & + & b\Sigma z_i^2 & = & \Sigma z_iy_i \end{array}$$

We can rewrite this as a matrix equation.

$$\begin{bmatrix} \Sigma x_i^2 & \Sigma x_iz_i \\ \Sigma x_iz_i & \Sigma z_i^2 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \Sigma x_iy_i \\ \Sigma z_iy_i \end{bmatrix}$$

This equation is tidy. But it can be even more tidy by utilising that it is equivalent to $X^TX\theta = X^T\vec{y}$. Where X is the training matrix containing all of the data. In this example

$$X = \begin{bmatrix} x_1 & z_1 \\ \vdots & \vdots \\ x_m & z_m \end{bmatrix}$$

To see how they are equivalent, see this,

$$\begin{bmatrix} \Sigma x_i^2 & \Sigma x_iz_i \\ \Sigma x_iz_i & \Sigma z_i^2 \end{bmatrix} \equiv \begin{bmatrix} x_1 & \dots & x_m \\ z_1 & \dots & z_m \end{bmatrix} \cdot \begin{bmatrix} x_1 & z_1 \\ \vdots & \vdots \\ x_m & z_m \end{bmatrix}$$

θ is the vector of the unknown parameters in this case

$$\theta = \begin{bmatrix} a \\ b \end{bmatrix}$$

and $\vec{y}$ is the 'result' or the target of the training data. In this case

$$\vec{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

$$\begin{bmatrix} \Sigma x_i y_i \\ \Sigma z_i y_i \end{bmatrix} \equiv \begin{bmatrix} x_1 & \dots & x_m \\ z_1 & \dots & z_m \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

We can now see that

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

2.3 Solution/Exampe

let us consider an example of of a linear function $f(x) = ax + b$, and a set of 4 data samples that map x to $f(x)$

x	1	2	3	4
$f(x)$	3.1	4.88	7.3	8.6

Looking at this data we can estimate that $a = 2$ and $b = 1$. But let's solve it a super cool way instead!

$$\begin{array}{rcrcr} 1a & + & 1b & \approx & 3.1 \\ 2a & + & 1b & \approx & 4.88 \\ 3a & + & 1b & \approx & 7.3 \\ 4a & + & 1b & \approx & 8.6 \end{array}$$

We can now create a matrix called the training matrix X

$$X = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}$$

And the target labels $\vec{y}$

$$\vec{y} = \begin{bmatrix} 3.1, 4.88, 7.3, 8.6 \end{bmatrix}$$

And our variables,

$$\theta = \begin{bmatrix} a \\ b \end{bmatrix}$$

By solving the equation we found in the above section, we get

$$\theta^* = A^{-1}\vec{b}$$

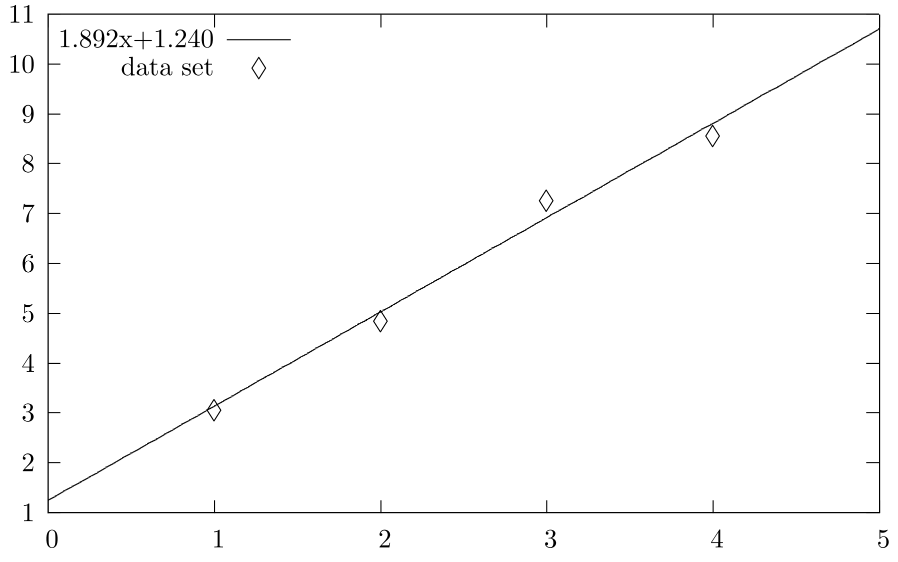
Where $A = X^T X$ and $\vec{b} = X^T \vec{y}$. Now simply perform the calculations

$$A = X^T X = \begin{bmatrix} 30 & 10 \\ 10 & 4 \end{bmatrix}$$

$$\vec{b} = X^T \vec{y} = \begin{bmatrix} 69.16 \\ 23.88 \end{bmatrix}$$

$$\theta^* = A^{-1} \vec{b} = \begin{bmatrix} 1.892 \\ 1.240 \end{bmatrix}$$

This means $a = 1.892$ and $b = 1.24$, so our model becomes $f(x) = 1.892x + 1.24$.



2.4 Parallelizing least squares regression

In the preceding sections, we learned that

$$\theta^* = A^{-1} \vec{b}$$

Where $A = X^T X$ and $\vec{b} = X^T \vec{y}$. The calculations of A and $\vec{b}$ can easily be parallelized. After A and $\vec{b}$ have been calculated, The inverse of A times $\vec{b}$, or $A^{-1}\vec{b}$ can be calculated. This last calculation doesn't necesarily have to be parallelized because A is at most going to be a $n \times n$ matrix and $\vec{b}$ also of length n. n is the number of dimensions of the equation and m is the number of samples or datapoint. Since $n << m$, the benefits of doing so are not that great.

To put this in map-reduce form, you need two map-reduce tasks, one to compute A and one to computer b. For A, the map function takes a vector that is a line of the matrix X and computes the square matrix that is $n \times n$ which is x_i times the transpose of itself, or $x_i x_i^T$. The reduce function takes the matrix sum of all of these matrices.

$$\begin{array}{ccc} X^T & \cdot & X = \\ \begin{bmatrix} x_{11} & \dots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & z_{nm} \end{bmatrix} & \cdot & \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \dots & x_{mn} \end{bmatrix} \end{array}$$

This is equivalent to the taking a row of X, treating it as a vector, or a $n \times 1$ matrix, and multiplying it by the transpose of itself, a $1 \times n$ matrix, then summing all of the rows that this is mapped to.

$$\sum_{i=1}^{m} x_i \cdot x_i^T$$

The map-reduce task that computes $b = X^T \cdot y$

Not all numerical algorithms are created equal, some are more numerically stable than others. While two algorithms could be algebraically identicle, one might have less truncation or rouding problems.