ML course notes and derivations

Erik-Cristian Seulean

January 27, 2022

1 Linear models

Linear models are basically concerned with finding the relationship between a target variable (predicted) and a set of features. The relationship is linear in the coefficients not in the features. The features can be polynomials or combinations of them.

Generally we have:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n + \epsilon$$

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 ... + \beta_n x^n + \epsilon$$

These can be combined, or we can even have interactions between features in the model.

In statistics, where we are interested in inference, and what exactly the coefficients tell us. In that case, we would make sure that the features - $x$es are independent, and we would do checks such as variance inflation factors to make sure this is respected. However, ML is mostly interested in predictions, so this is not that much of a concern. Additionally, statistics would be also concerned about the assumptions of the model such as independence between

different observation, constant variance and the sampling distribution of the data. ML abstracts most of these things out and ignores completely the assumptions. A concern that is present in ML is overfitting and underfitting the data. Models that fit the training that very well might not be able to perform well on newly seen observations, so there usually are procedures in place to verify the capabilities of a model on unseen data. This usually comes as cross-validation where a subset of the data is removed from the training set (usually 20%) and it is used to assess the model performance on it. The error between the predicted and actual values is called generalization error. The generalization error is then used as a way to select between multiple models. Models with lower generalization error (without other considerations) are considered better as they perform better on the testing subset of data.

1.1 How to find the coefficients for the model ? The analytical way.

There are at least 3 ways of finding the coefficients. This is an optimization problem, which requires a loss function. There are multiple loss function that can be considered, but the most common ones are the OLS (Ordinary least squares), MSE (Mean squared error) or the mean absolute error.

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$\hat{y}_i = f(X, \theta) = \beta_0 + \beta_1 x$$

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \beta_0 - \beta_1 x)^2$$

Here we consider the prediction y_i as being a function of only one feature, but this can be generalized to multiple features.

There are a few analytical methods to finding the coefficients. One is to consider y_i and x as random variables and use properties of random variables:

$$Y = \beta_0 + \beta_1 X$$

$$E(Y) = \beta_0 + \beta_1 E(X)$$

$$Cov(Y, X) = Cov(\beta_0 + \beta_1 X, X) = \beta_1 Var(X)$$

$$\beta_0 = E(Y) - \beta_1 E(X)$$

$$\beta_1 = \frac{Cov(X, Y)}{Var(X)}$$

This however does not generalize well, so it's mostly theoretical. I do like it however as it's simple and easy to get (perhaps might come useful in an interview).

An alternative is to consider this as a minimization problem and find the places where the partial derivatives with respect to the coefficients are 0.

$$\frac{\partial MSE}{\partial \beta_0} = \frac{1}{n} \sum_1^n 2(y_i - \beta_0 - \beta_1 x_i)(-1) = 0$$

$$\sum_1^n (y_i - \beta_0 - \beta_1 x_i) = 0$$

$$n\bar{y} - n\beta_0 - \beta_1 n\bar{x} = 0$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

The equation is the same as in the random variable case, we just used the sample to get there.

$$\frac{\partial MSE}{\partial \beta_1} = \frac{1}{n} \sum_1^n 2(y_i - \beta_0 - \beta_1 x)(-x_i) = 0$$

$$\sum_1^n (y_i x_i - \beta_0 x_i - \beta_1 x_i^2) = 0$$

Substituting β_0 found above we get:

$$\sum_1^n (y_i x_i - (\bar{y} - \beta_1 \bar{x}) x_i - \beta_1 x_i^2) = 0$$

$$\sum_1^n (y_i x_i - \bar{y} x_i - \beta_1 \bar{x} x_i - \beta_1 x_i^2) = 0$$

$$\sum_1^n (y_i x_i - \bar{y} x_i) - \beta_i \sum_1^n (\bar{x} x_i - x_i^2) = 0$$

$$\beta_1 = \frac{\sum_1^n (y_i x_i - \bar{y} x_i)}{\sum_1^n (\bar{x} x_i - x_i^2)}$$

It turns out that this generalizes, and you can actually use this to get the coefficients for multiple linear regressions following the same patter. It gets messy in higher dimensions, but the matrix notation can actually help with this.

$$\underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}}_{\beta}$$

$$Y = X\beta$$

Now we can rewrite the MSE equation in matrix notation, and we can apply the same principle, but given that we have now all the coefficients "clustered" together into a matrix, we will only have one differential equation that we need to check.

$$RSS = (Y - X\beta)^T (Y - X\beta)$$

$$\frac{\partial RSS}{\partial \beta} = -X^T (Y - X\beta) + -X(Y - X\beta)^T = 0$$

$$\frac{\partial RSS}{\partial \beta} = -2X^T(Y - X\beta) = 0$$

$$X^TY - X^TX\beta = 0$$

$$X^TX\beta = X^TY$$

$$\beta = (X^TX)^{-1}X^TY$$

It's worth mentioning that this analytical solution is much faster (due to the nature of the way matrices are represented in memory and the way operations are resolved) but it's still analytical. The inverse might not exist, so we might need to use a pseudo-inverse (if the matrix is singular).

1.2 How to find the coefficients for the model ? The iterative way

We can reuse what we have above, but instead we're interested in the slope only $\frac{\partial MSE}{\partial \beta_0}$, $\frac{\partial MSE}{\partial \beta_0}$. There are two cases that need to be considered: first is when the slope is positive, in that case we need to decrease the value of the coefficients (as we move from right to left) and when the slope is negative, in that case we need to increase the value of the coefficients (as we move from left to right).

In this case we have:

$$\beta_0 = \beta_0 - \alpha * \frac{\partial MSE}{\partial \beta_0}$$

$$\beta_1 = \beta_1 - \alpha * \frac{\partial MSE}{\partial \beta_1}$$

$$\beta_0 = \beta_0 - \alpha * \frac{2}{n}\sum_{1}^{n}(\beta_1 x_i + \beta_0 - y_i)$$

$$\beta_1 = \beta_1 - \alpha * \frac{2}{n}\sum_{1}^{n}2x_i(\beta_1 x_i + \beta_0 - y_i)$$

Alpha is the learning rate, that is the step, which controls how big are the steps in the direction of the global (or local) minima. The function itself is not always convex, but this shouldn't affect the algorithm.

The algorithm starts by setting random starting points for the coefficients and update the values of the coefficients iteratively until the change in the values are very small. At this point we can say that the algorithm converged towards a minimum (which could be the local minima) and we can stop.

The downside of the algorithm above is that we need to use all the points in the dataset for every step of the optimization, which is extremely expensive (especially for large datasets).

Additional optimization exists that use a bootstrapping mechanism to select a subset of the observations to compute the slopes or just one single observation at a time. These are theoretically worse in terms of convergence however practically there's little downsides to them.

The matrix notation for the above is:

$$\beta = \beta - \frac{2}{n} X^T (X\beta - Y)$$

2 A note on complexity

Basically if matrix X is $n \times n$ then we have n linear equations. That means that finding the inverse is an $O(n^3)$ (By the LU decomposition following the Gauss-Jordan echelon form [I need to learn more about this :)]) so if we double the number of features (instead of n to 2n) the complexity is 8 times higher. That's a lot! This means that the matrix operations are not as efficient as I expected.

This is the reason why we use iterative solutions that don't use the matrix

form (and use stochastic solutions) to avoid computing the inverse of the matrix.

3 Gradient Descent variations

There are 3 variations that I read about: the batch, stochastic and minibatch. The stochastic gradient descent uses only 1 observation from the dataset to compute the gradient, which means that it is faster as it doesn't have to compute the pseudoinverse for a very large matrix. This also helps to get out of local minimums and eventually converge to the global minimum. Minibatch is a combination of the two, where instead of using only one observation or the entire training set, it uses a subset of the training set. This solution is apparently faster on GPUs.