Introduction

Linear regression is a very well studied subject in machine learning. Closed-form solutions obtained from various analytical approaches provide tremendous insights into many fundamental issues such as how bias and variance are originated, and how to suppress them. Analytical approaches also unveil an important point in solving machine learning problems, which is either undoing a linear transformation or doing a basis change. Many learning algorithms such as principal component analysis (PCA), independent component analysis (ICA), linear discriminant analysis (LDA) etc., can be understood as working in either a convenient, or more suitable basis. This can also be understood as undoing a linear transformation, as shown in the singular value decomposition (SVD) approach to solve linear regression problem.

Although we cannot solve many machine learning problems with analytical approaches, in many cases with numerical approaches we are actually still doing equivalent things. For example, introducing a l_2 regularization term in the cost function of a linear regression problem is equivalent to adding an regularization term to the diagonal terms of the corresponding covariance matrix. Therefore, analytical solutions can always be helpful for us to have in-depth understandings from the black-box numerical calculations.

Description of the problem

Linear regression is usually an over-determined problem where we have $m >> n$ in the design matrix A_{mn}. For simplicity in description, here we assume there are three points $(x_1^{(i)}, x_2^{(i)}), i = 0, 1, 2$ in the 2D plane (x_1, x_2). Two parameters are assumed to be β_0, β_1. To solve for the parameter, we need solve the following system of equations

$$\begin{cases} \beta_0 & + & \beta_1 x_1^{(0)} & = x_2^{(0)} \\ \beta_0 & + & \beta_1 x_1^{(1)} & = x_2^{(1)} \\ \beta_0 & + & \beta_1 x_1^{(2)} & = x_2^{(2)} \end{cases}$$

Writing as matrix form gives

$$\begin{pmatrix} 1 & x_1^{(0)} \\ 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} x_2^{(0)} \\ x_2^{(1)} \\ x_2^{(2)} \end{pmatrix}$$

For convenient notation, we write the above as $Ax = b$. Note b describes x_2 but not the parameter β. x here describes the parameters β but not the data points (x_1, x_2).

Here the column space of A, denoted as $C(A)$, is at most two dimensional (when two columns are linearly independent). Ax can be understood as the linear combination of two column vectors in $C(A)$ with the coefficients in x. The resulting vector is thus always lying in the 2D plane of $C(A)$. However, due to the error in the data, the vector $b \in \mathbb{R}^3$ is usually in a 3D space. This indicates that vector Ax cannot be equal to vector b for any expansion coefficients x. We can only find x that best approximates Ax and b. Or, we may find x satisfying $\min_x ||Ax - b||_2^2$.

Minimization by projection to column space

One way to find the best x is to project vector b to the column space $C(A)$. Assuming the projected vector on the $C(A)$ is p_A, then if we find $Ax = p_A$, then the p_A in 2D $C(A)$ space will best approximate b in 3D space. This is because the error of two vectors is automatically minimized due to the 'vertical' projection from b to p_A. So the key is to find the p_A first. To find p_A, we then need a proper projection operator which achieves this column space projection.

Projection operator from singular value decomposition (SVD)

For the most general case where A can be singular and any shape, the projection operator to $C(A)$ has the form $P = AA^\dagger$, where $A^\dagger$ is the pseudo-inverse of A. P can collapse to normal projector to column space when A is with full-column rank. In the most general case where A is singular, we have from SVD, $A = U\Sigma V^T$. Thus we have $A^\dagger = V\Sigma^{-1}U^T$, where in Σ^{-1} only non-zero singular values are inversed.

We thus obtain $p_A = AA^\dagger b = AV\Sigma^{-1}U^T b$. Moreover,

$$Ax = p_A = AV\Sigma^{-1}U^T b \Rightarrow x = V\Sigma^{-1}U^T b = \sum_{i=1}^{r} \frac{v_i u_i^T b}{\sigma_i}$$

where r is the rank of A.

One source of variance

If we assume there is an error δb in b, then the solution becomes

$$x = \sum_{i=1}^{r} v_i \left(\frac{u_i^T b}{\sigma_i} + \frac{u_i^T \delta b}{\sigma_i} \right)$$

If the singular values σ_i is very tiny due to the deficiency of a matrix, then small error on the data will give big error as σ_i appears in denominator.

Solving normal equation by inverting covariance matrix

For full column rank matrix A, $A^T A$ is a full-rank $n \times n$ matrix and thus invertible. The projection operator to $C(A)$ is defined as $P = AA_{left}^{-1} = A(A^T A)^{-1} A^T$, where the pseudo-inverse $A^\dagger$ is replaced by A_{left}^{-1} (details in linear algebra notes).

Thus we have

$$\begin{aligned} Ax = p_A &= A(A^T A)^{-1} A^T b \\ \Rightarrow x &= (A^T A)^{-1} A^T b \end{aligned}$$

Regularization

If $A^T A$ is not full rank, then it is singular and not invertible. A way to handle this is to use regularization. Using linear regression for example, we introduce the regularizing term from the cost.

$$C(x) = \frac{1}{2} \sum_i \left(b_i - a_i^T x \right)^2 + \frac{1}{2} \lambda \|x\|^2$$

where a_i^T is the row vector of matrix A, and thus a_i is a column vector obtained from $(a_i^T)^T = a_i$. This indicates different data samples are given in terms of different rows. Taking derivative w.r.t. x and setting it to zero gives,

$$\sum_i \left(b_i - a_i^T x \right) a_i = \lambda x \Rightarrow x = \left(\lambda I + \sum_i a_i a_i^T \right)^{-1} \left(\sum_j b_j a_j \right)$$

In the matrix form, we have

$$x = (A^T A + \lambda I)^{-1} A^T b$$

where I is identity matrix and λ is the regularization parameter. $A^T A$ is square symmetric matrix and is assumed to be singular here. After adding λI to $A^T A$, the singular and symmetric matrix $A^T A$ becomes full rank (for $\lambda > 0$) and invertible.

What regularization does in machine learning?

- The regularization will drive the the denominators σ_i, i.e., the singular values, far away from zero. From the SVD solution earlier, the regularization will reduce the variance arising from the tiny singular values
- From $x = (A^T A)^{-1} A^T b$, we know that x, i.e. the weights, will generally becomes smaller because the addition of λ to $A^T A$. In other words, regularization generally reduce the numerical value of weights. This will reduce the complexity of the models.
- Singular values are not just with clear-cut zero or not-zero values in numerical calculations. In other words, a matrix is not just non-invertible (in the sense of pseudo-inverse) and have zero singular values or infinity condition number. In practice, we may encounter many very tiny singular values which correspond to ill-conditioned problems (very big but not infinity condition number). In this case, regularization will generally suppress the big variance caused by these tiny singular values.
- In machine learning, if the number of samples are huge, then the problem of variance might not be an big issue. This might be understood by the following two ways:
    - If the number of samples are huge, then the problems is very over determined. In this linear regression case, this means that the matrix A might have good change to be at least with full-column rank, and thus $A^T A$ is invertible. Of course, the huge number of samples should be as much independent as possible. Otherwise there will still be big variance.
    - From the SVD solution, we may assume the big variance caused by small denominators σ_i can be cancel each other if we have a huge number of data, as the noise might have a symmetric (e.g. normal) distribution.

Geometric interpretation

Once we projecting vector b to a low-dimensional column space $C(A)$, the linear regression problem has a form $Ax = p_A$, where x and p_A are both confined in $C(A)$. This is then a typical linear algebra problem. Now we have two pictures to understand the solving process of this linear algebra problem:

- The first picture is that the linear transformation denoted by A transforms vector x to p_A. For example, using SVD, $A = U\Sigma V^T$, then $Ax = U\Sigma V^T x$ is to rotate, stretch, and rotate again the vector x by V^T, Σ and U respectively. Thus finding x is just to undo these actions. Because U and V^T are unitary operators, the undoing actions are their transposes. Σ is diagonal and its undoing action is obtained by reversing its non-zero elements. With these undoing actions, we transform p_A back to x, the solution.
- The second picture for solving $Ax = p_A$ is to obtain x by a basis change $x = A^\dagger p_A$. Many algorithms such as PCA, ICA can be intuitively understood with a basis change or linear transformation. Choosing different bases can help resolve different problems.

Stochastic gradient descend (SGD)

In the case of linear regression, the numerical approach with SGD updating rule can be obtained by minimizing the relevant cost function. The hypothesis function for linear regression is

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

where $x, \theta \in \mathbb{R}^{n+1}$. The cost function is

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Note the conventional notations of $J(\theta), L(\theta), l(\theta)$, which describe cost, likelihood, log-likelihood functions respectively.
Minimizing the cost function (note the minus sign before α for minimization) gives the gradient descent update rule for linear regression

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} = \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)})x_j^{(i)}$$

Maximum likelihood estimation (MLE)

Alternative way of deriving the same numerical updating rule is employing the MLE on a probabilistic model. In the linear regression case, we assume the probability of $y^{(i)}$ given $x^{(i)}$ is Gaussian and then writing the likelihood as

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

By maximizing the log likelihood

$$l(\theta) = \log L(\theta)$$

we obtain the same updating rule obtained earlier by minimizing cost function (see details in cs229 notes).

Regularization and implementation details for linear regression, such as the treatment of constant terms, see the course notes on Machine learning (Coursera).

Most machine learning problems, particularly for huge data set, are handled by numerical approaches. Linear regression is one of a few problems that have closed-form solutions. However, these closed-form solutions provides deep insights to the general understanding of machine learning.

Vectorization in numerical method

- It will be much more efficient to solve numerically if we avoid loops as much as possible, but resort to vectorization.
- The efficiency can be improved by even several hundreds of times for certain types of CPUs. This improvement is usually much bigger than the normal multi-threading/multi-processing, the improvement of which is usually limited by the number of physical cores of CPU.
- See vectorization example in the write up for logistic regression.

Model assumptions

https://stats.stackexchange.com/questions/230044/what-are-real-life-examples-of-non-parametric-statistical-models (https://stats.stackexchange.com/questions/230044/what-are-real-life-examples-of-non-parametric-statistical-models)

- Variables should be measured at the continuous level.
- There needs to be a linear relationship between the two variables.
- There should be no significant outliers. In almost all parametric models we need handle outliers first. In these models such as linear regression, because it basically 'regress/go to mean' and outliers will definitely affect the mean calculations.
- Independence of observations. This is required, at least preferred, by almost all statistical machine learning models.
- Your data needs to show homoscedasticity, which is where the variances along the line of best fit remain similar as you move along the line.
- Check that the residuals (errors) of the regression line are approximately normally distributed. This is the assumption for population distribution for linear regression (see this requirement for all parametric machine learning models). However, some people says that a normal distribution is always required to derive a linear regression stochastic gradient decent rule (see cs229 notes).