Statistics

Jacob Denson

October 29, 2015

Table Of Contents

Chapter 1

Regression

The game of regression is simple. Let $(X_1, Y_1), \ldots, (X_n, Y_n) \sim (X, Y)$ be some observed data. Our job is to statistically approximate the regression function $f(x) = \mathbf{E}(Y|X)$ as a function of the data $(X_1, Y_1), \ldots, (X_n, Y_n) = D$. We will denote an arbitrary estimate by $\hat{f}_D$. Then $\hat{f}_D$ is a random variable, taking values in the space of *measurable functions*. This is an infinite dimensional vector space, which probably explains why the task is difficult in general. In order to obtain results, we often restrict our regression functions to a specific subspace of functions.

1.1 Linear Regression

In some case, we assume our regression functions take the form $\beta^* X$, where $\beta^* \in (\mathbf{R}^n)^*$. Given the data $(X_1, Y_1), \ldots, (X_k, Y_k)$, we determine the best estimate $\hat{\beta}$ of β^* by evaluating it against the loss function $\mathcal{L}(\beta) = \mathbf{E}[(Y - \beta X)^2]$. Of course, we cannot calculate $\mathcal{L}$ directly, but we may estimate it with our samples. Because the loss function is a differentiable function of β, we may take derivatives to determine β:

$$\nabla \mathcal{L}(\beta) = 2\mathbf{E}[(Y - \beta X)X^T] = 2\mathbf{E}(YX^T) - 2\beta\mathbf{E}(XX^T)$$

This is optimized when the derivative of this function is zero. i.e., when

$$\beta\mathbf{E}(XX^T) = \mathbf{E}(YX^T)$$

Assuming $\mathbf{E}(XX^T)$ is invertible, we may invert, and determine that the optimal value β^* can be calculated as

$$\beta^* = \mathbf{E}(YX^T)\mathbf{E}(XX^T)^{-1}$$

Now if we only have the samples (X_i, Y_i), we may approximate this value by forming the conglomerate matrices $\mathbf{X} = (X_1|X_2|\dots|X_n)$ and $\mathbf{Y} = (Y_1, \dots, Y_n)$, and calculating $\hat{\beta} = \mathbf{Y}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}$. This minimizes the error over the training data set $\sum(Y_i - \beta X_i)^2 = \|\mathbf{y} - \beta\mathbf{X}\|^2$.

How do we estimate how accurate our prediction is. First, assume that each Y_i is independant, with the same variance σ^2. Then

$$\mathbf{V}(\hat{\beta}) = \mathbf{V}(\mathbf{Y}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1})$$

INSERT THEORETICAL ESTIMATES, Gauss Markov theorem, etc.

1.2 Additive Models

A generalized additive model has a regression function of the form

$$\mathbf{E}(Y|X) = \alpha + f_1(X^1) + \cdots + f_n(X^n)$$

where the f_i are unspecified smooth (C^∞) functions, and $X = (X^1, \dots, X^n)$ is a random vector. To fit an additive model, given a sample $(X_1, Y_1), \dots, (X_m, Y_m)$, we take as a cost function the penalized sum of squares to find the constant α and functions f_i,

$$\sum_{i=1}^{m}\left(Y_i - \alpha - \sum_{j=1}^{n} f_j(X_i^j)\right)^2 + \sum_{j=1}^{m}\lambda_j \int (f_j'')^2$$

Where the $\lambda_j \geq 0$ are arbitrary parameters. The minimizer of this cost function is not unique – it is standard convention to require that $\sum_{i,j} f_i(X_i^j) = 0$. One can apply an iterative cubic smoothing spline solution to find this minimum.

1.3 Tree-Based Models

Tree based methods partition the feature space , and then fit a simple model (normally a constant) into each one. If $\mathcal{S}$ is such a space, and we partition it into $S_1, \ldots, S_n$, each with a model $f_1, \ldots, f_n$, then are model is

$$\mathbf{E}(Y|X) = \sum_{i=1}^{n} f_i(X)[X \in S_i]$$

If S_i has already been decided, and we are using constants for the f_i, then the best choice of constants (according to the least squares cost function) given a sample $(X_1, Y_1), \ldots, (X_n, Y_n)$ are just the mean values of the Y_j with $X_i \in S_i$. The questions remains, however, on how to choose our partitions.

To find optimal partitions, we assume our feature space is $\mathbf{R}^n$, and our regions formed by 'binary splits'. We start with the whole space, pick a 'splitting coordinate' i and 'splitting point' $t \in \mathbf{R}$, and partition our region into two sets $A = \{x \in \mathbf{R}^n : x_i < t\}$ and $B = \{x \in \mathbf{R}^n : x_i \geq t\}$. We then recursively partition A and B up in this manner, until we are satisfied with our splits.

Finding the best choice of partition using the method above is generally computationally infeasible. We shall proceed with a greedy approximation. Given a region A containing features $X_1, \ldots, X_n$, we seek to find a splitting variable i and split point t which minimize the cost function

$$\arg\min_{i,t} \min_{a} \sum_{X_j^i \leq t} (Y_j - a)^2 + \min_{b} \sum_{X_j^i > t} (Y_j - b)^2$$

Given i and t, the minimum values of a and b are just obtained by taking the mean of the results Y_j. By doing a linear scan on each coordinate, it is fairly simple to find i and t. Then we recursively perform this greedy approach on the subpartitions.

Now when do we stop splitting? If we split far enough, then we will only have very few examples in each subregion, and we will have overfitted our training data! Furthermore, it will be very difficult to interpret the model we have created.

Chapter 2

Neural Networks

Neural Networks arise from the solution of a certain model, known as the Projection Pursuit Regression model. Assume we have an input vector $X \in \mathbf{R}^n$, with target Y. The projection pursuit regression model has the form

$$f(X) = \sum_{i=1}^{M} g_i(\beta_i X)$$

Where the g_i are unspecified, and $\beta_i \in (\mathbf{R}^n)^*$. This is an additive model, but in the features $V_i = \beta_i X$. Each $g_i(\beta_i X)$ is called a ridge function in $\mathbf{R}^n$.

This model is really general. For instance, the product of the coordinates can be written

$$X_1 X_2 = \frac{(X_1 + X_2)^2 - (X_1 - X_2)^2}{4}$$

In fact, if we let M be large enough, for appropriate choices of g_i can approximate arbitrary continuous functions on $\mathbf{R}^n$ (this model is a universal approximator). Unfortunately, this means this model will be hard to fit exactly, and thus the model is better for estimating data rather than obtaining an understandable model.

Given some data $(X_1, Y_1), \ldots, (X_n, Y_n)$ from the regression model, we thus seek the minimize the error

$$\sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{M} g_j(\beta_j x_i) \right)$$

as a choice of g_j and β_j. We need to impose constraints on g_j to prevent overfitting.

Suppose we have $M = 1$, and that $g = g_1$ is differentiable, and $\beta = \beta_i$ is the linear functional. Then, taking the initial terms around the taylor series,

$$g(\beta x_i) \approx g(\alpha x_i) + g'(\alpha x_i)(\alpha - \beta)x_i$$

$$\begin{aligned}\sum_{i=1}^{n}[y_i - g(\beta x_i)]^2 &\approx \sum_{i=1}^{n}[y_i - g(\alpha x_i) - g'(\alpha x_i)(\alpha - \beta)x_i]^2 \\ &= \sum_{i=1}^{n} g'(\alpha x_i)^2 \left[\alpha x_i + \frac{y_i - g(\alpha x_i)}{g'(\alpha x_i)} - \beta x_i\right]^2\end{aligned}$$

We can minimize the right-hand side by carrying out a least squares regression with target

$$\alpha x_i + \frac{y_i - g(\alpha x_i)}{g'(\alpha x_i)}$$

We can then iterate this regression until convergence. With more than one term in the model, we just perform forward stage-wise reggresion.

The projection pursuit regression model has not been widely used in the field of statistics, possibly due to the lack of computational resources when it was created. Nonetheless, it leads to the field of neural networks, which are much more useful.

2.1 Nets

There is a lot of mysticism surrounding neural networks (perhaps for the same reason 'the god particle' is so controversial) but they are really just non-linear statistical models. Here we will discuss the most basic kind of neural nets, the single hidden layer back-propogation network.

Suppose we are given a set of inputs $X = (X_1, \ldots, X_n)$. A neural net creates layers of derived features $Z = (Z_1, \ldots, Z_m)$ as affine transformations of the X_i, 'flattened' by some activation function σ. In the single layer approach, we have one layer of these derived features, and then these derived features are used to generate the target $Y = (Y_1, \ldots, Y_k)$ as a function

of the Z_i, again modified by an output function. In terms of formulas, our mathematical model is

$$Z = \sigma(\Lambda X + v) \qquad W = \Delta Z + w \qquad f(X) = g(W)$$

where Λ and Δ are linear transformations, and our regression function is f.

For regression, we normally choose not to modify our outputs via an output function (that is, we let $g = \mathbf{1}$). For classification, we need to choose an output function which results in reasonable results. The sigmoid function is often chosen as the activation, $\sigma(t) = (1+e^{-t})^{-1}$. Sometime Gaussian radial basis functions are used, producing a radial basis function network. Note that if we let σ and the output regularization function be the identity, we obtain a linear model. Thus in this way, a neural network is a generalization of the linear model.