Linear models for regression

Introduction

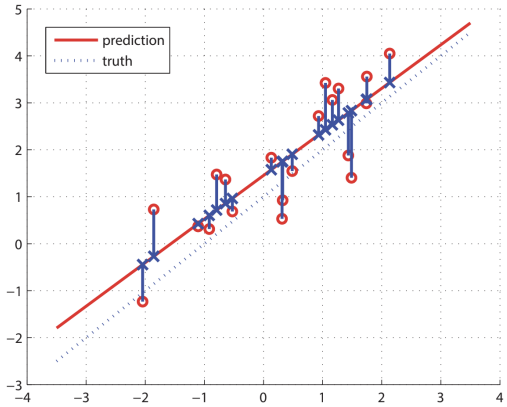
Linear models for regression aim to finding models able to predict real values. Formally a linear model is a function in the form $f : \mathbb{R}^d \rightarrow \mathbb{R}$.

Given a dataset $D = \{(x_n, t_n)_{n=1}^N\}$ $t_n \in \mathbb{R}$, we define a linear model as:

$$y = w_0 + w_1 x_1 + ... + w_d x_d = \begin{bmatrix} w_0 & w_1 & ... & w_d \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ ... \\ x_d \end{bmatrix} = W^T X$$

We can write an explicit formula for the model as $y(x; w) = W^T X$.

When d=1, we will have a one dimensional input space, and a two dimensional representation of the dataset. In this case, the linear model will be a line represented as $y(x; w) = w_0 + w_1 x_1$. The goal of regression is to compute an approximation of the real function that will maximize the prediction (hence, minimize the error).



The following representation shows the data points and the prediction model (the red line). The error is expressed as the distance of each point from the prediction model.

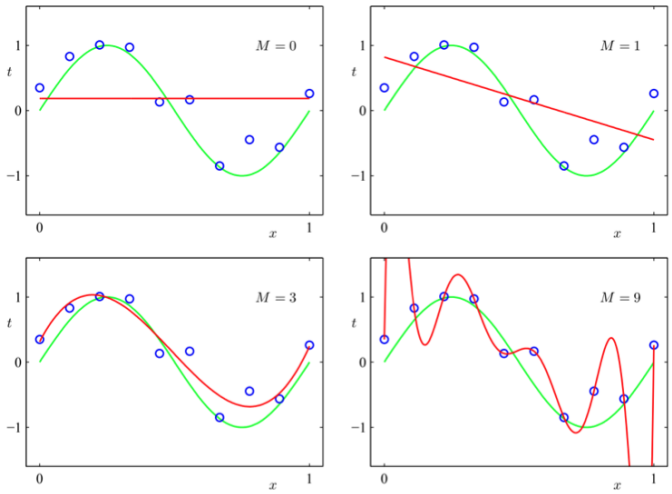
When considering this problem, transformation of the input space are allowed, since we will still keep the model linear with respect to the parameters. Therefore, we can defines a set

of nonlinear transofrmations as:

$$\phi(x) = \begin{bmatrix} \phi_0(x) \\ ... \\ \phi_M(x) \end{bmatrix} \rightarrow y(x;w) = W^T \phi(x)$$

For example, let's consider the transformation $\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ ... \\ x^M \end{bmatrix}$.

Our model will be an M-degree polinomial $y(x;w) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M$ that we can try tuning in order to get as close as possible to the real function. The following image show an important characteristic of this approach:



As we can see, the more parameter we use, the more we will risk the phenomena of overfitting. In fact we will progressively reduce the error on training dataset, but we will have poor generalization for new predictions.

Computing the model

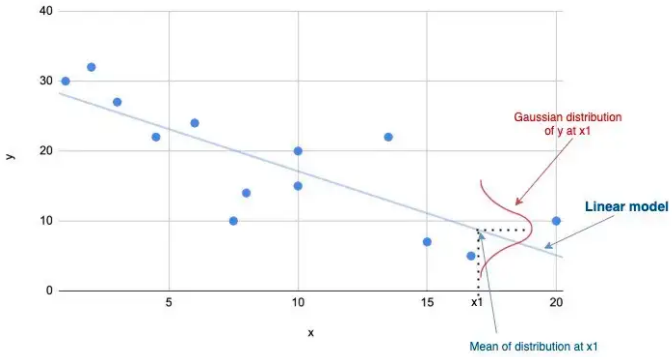
The probabilistic approach

We start by assuming that points of the training dataset come from the true function plus some noise:

$$t = y(x;w) + \epsilon$$

The logic is that we assume that there is a true function, and each point of the dataset correspond to the points of that function plus some noise error generated by a certain

distribution:



Noise will be generated by a zero-mean Gaussian noise distribution:

$$P(\epsilon|\beta) = \mathcal{N}(\epsilon|0, \beta^{-1})$$

Where the term β is called precision (the inverse of the variance). Given our model, we ask ourselves: what is the probability of having that points in the dataset (likelyhood)? This probability is defined as:

$$P(t|x, w, \beta) = \mathcal{N}(t|y(x; w), \beta^{-1})$$

Where $y(x; w)$ will be the mean of the Gaussian. What we want is maximizig the likelihood, aka minimizing the negative log-likelihood. The following is the formulation for this likelihood, obtained assuming that each input is indipendent and identically distributed:

$$P(\{t_1, ..., t_N\}|x_1, ..., x_N, w, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|w^T\phi(x_n); \beta^{-1})$$

The negative log-likelihood will be expressed as

$$lnP(\{t_1, ..., t_N\}|x_1, ..., x_N, w, \beta) = .. = -\beta\frac{1}{2}\sum_{n=1}^{N}[t_n - w^T\phi(x_n)]^2 - \frac{N}{2}ln(2\pi\beta^{-1})$$

Notice that the only term that we care about (the one that contains the parameters) is $\sum_{n=1}^{N}[t_n - w^T\phi(x_n)]^2$, so we can reduce the problem as finding

$$argmaxP(\{t_1, ..., t_N\}|x_1, ..., x_N, w, \beta) = argmin\frac{1}{2}\sum_{n=1}^{N}[t_n - w^T\phi(x_n)]^2$$

That corresponds to minimizing the squared error between the actual data and our model.

Computing the solution: the closed-form approach

The closed-form solutions aims at solving the error by directly computing the solution for the parameters matrix. We start defining the error:

$$E_D(w) = \frac{1}{2}(t - \phi_w)^T(t - \phi_w)$$

We will solve the problem by computing the gradient of the error and putting it to zero:

$$\Delta E_D(w) = 0 \iff \phi^T \phi W = \phi^T t$$

Hence

$$W_{ML} = (\phi^T \phi)^{-1} \phi^T t$$

Computing the solution: the iterative approach

Ideally the closed-form approach already computes the correct solution. The problem with it is that we are trying to solve the problem for the whole ϕ matrix, which means risking to have really bas performances in term of computational time when dealing with huge dimensions.

The iterative approach addresses this problem by computing a step by step solution, by finding each time the negative direction for the gradient and updating the weights. This method is called stochastig gradient descent:

$$\hat{w} \leftarrow \hat{w} + \eta \Delta E_w$$

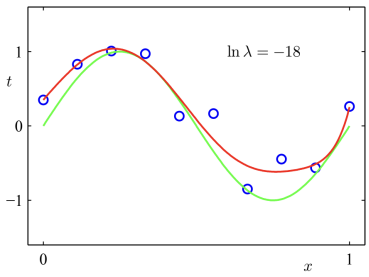
Where η is the learning rate (the algoritm converges when the learning rate is small enough). Therefore

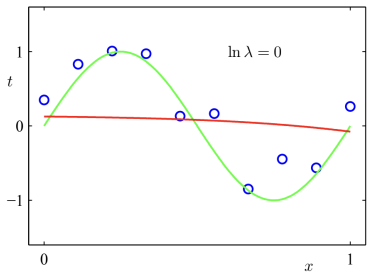
$$\hat{w} \leftarrow \bar{w} + \eta[t_n - \hat{w}^T \phi x_n]\phi(x_n)$$

We still have to address the problem of overfitting. This is why we introduce the regularization term λ. Its function will be to penalize the coefficients that are too high independently from the samples in the dataset. Hence, we are interested in:

$$argmin[E_D(w) + \lambda E_w(w)]$$

A common choice is $E_w = \frac{1}{2} W^T W$. Notice that if the parameter is not well tunes, we risk to reach an underfitting region.





Regression with multiple outputs

Let's assume the case where we have a d-dimensional input and a k-dimensional output $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$. The output will be a vector of size k: $y = [y_1 \ ... \ y_k]^T$ made by k linear models.

Since we are dealing with a k-dimentional output, also the target attributes will be expressed as an NxK matrix in the form:

$$T = \begin{bmatrix} ... \ t_n \ ... \\ ... \end{bmatrix}$$

Similarly to the 1D case, we will obtain the maximum likelihood for:

$$W_{ML} = (\phi^T \phi) \phi^T T$$