Linear in The Parameters Regression

Intelligent Systems and Control

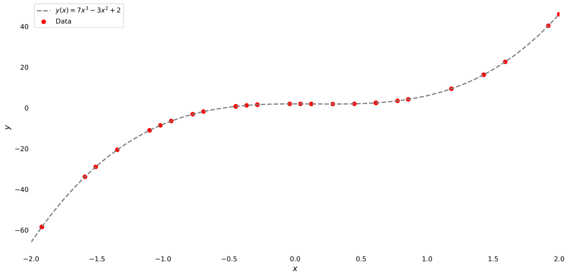
2019

Sepehr Maleki

University of Lincoln
School of Engineering

Introduction

One of the most widely used models for regression in machine learning.



Given a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$ of n pairs of inputs $\mathbf{x}_i$ and targets y_i, the goal is to predict the target $y^\star$ for any arbitrary input $\mathbf{x}^\star$.

(S. Maleki 2019)

Model of The Data

In order to predict at a new $\mathbf{x}^\star$ we need to postulate a model of the data. We will estimate $y^\star$ with $f(\mathbf{x}^\star)$. In general:

$$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \ .$$

Therefore:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + \epsilon = \sum_{j=1}^{d} w_j x^{(j)} + \epsilon \ .$$

where $\mathbf{w}^\top \mathbf{x}$ represents the inner or scalar product between the input vector $\mathbf{x}$ and the model's weight vector $\mathbf{w}$, and ϵ is the residual error between our linear predictions and the true response.

Matrix Representation of Data

$$X = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & ... & x_1^{(d)} \\ x_2^{(1)} & x_2^{(2)} & ... & x_2^{(d)} \\ \vdots & \vdots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & ... & x_n^{(d)} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

$$\mathbf{f} = X \,.\, \mathbf{w} + \epsilon$$

$$X = \begin{bmatrix} 1 & \mathbf{x}_1 \\ 1 & \mathbf{x}_2 \\ 1 & \vdots \\ 1 & \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_1^{(2)} & ... & x_1^{(d)} \\ 1 & x_2^{(1)} & x_2^{(2)} & ... & x_2^{(d)} \\ 1 & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n^{(1)} & x_n^{(2)} & ... & x_n^{(d)} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} \epsilon \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

$$\mathbf{f} = X \,.\, \mathbf{w}$$

(S. Maleki 2019)

Model Specification

Linear regression can be made to model non-linear relationships by replacing $\mathbf{x}$ with some non-linear function of the inputs, $\phi(\mathbf{x})$. That is, we use:

$$f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) \ .$$

This is known as *basis function expansion* (note that the model is still linear in the parameters $\mathbf{w}$).

A simple example are polynomial basis functions, where the model has the form:

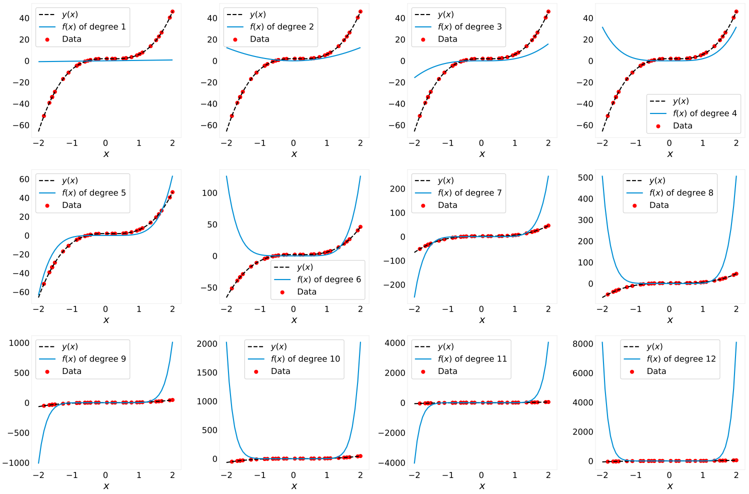
$$\phi(\mathbf{x}) = [1, x, x^2, \ldots, x^d] \ , \qquad \phi_j(x) = x^j$$

Then:

$$f(\mathbf{x}) = w_0 1 + w_1 x + w_2 x^2 + \ldots + w_d x^d = \sum_{j=0}^{d} w_j \phi_j(x) \ .$$

(S. Maleki 2019)

Model of The Data – Example

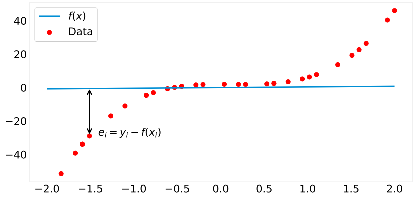


(S. Maleki 2019)

Model of The Data – Questions?

- Should we choose a polynomial? (model structure)
- What degree should we choose for the polynomial? (model structure)
- For a given degree, how do we choose the weights? (model parameters)
- For now, let's find the single "best" polynomial: degree and weights.

The Least Squares Approach



- Idea: measure the quality of the fit to the training data.
- For each training vector, measure the squared error:

$$e_i^2 = \Big(f(\mathbf{x}_i) - y_i\Big)^2 .$$

- Find the parameters ($\mathbf{w}$) that minimise the sum of squared errors, defined by the following energy (objective) function:

$$\mathcal{E}(\mathbf{w}) = \sum_{i=1}^{n} e_i^2 .$$

(S. Maleki 2019)

The Least Squares Approach

Now define:

$$\begin{aligned}\mathbf{y} &:= [y_1, y_2, \ldots, y_n]^\top \ , \\ \mathbf{f} &:= [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)]^\top \ , \\ \mathbf{e} &:= \mathbf{f} - \mathbf{y} \ ,\end{aligned}$$

then, the sum of squared errors is given by:

$$\mathcal{E}(\mathbf{w}) = ||\mathbf{e}||^2 = \mathbf{e}^\top \mathbf{e} = (\mathbf{f} - \mathbf{y})^\top (\mathbf{f} - \mathbf{y}) \ .$$

Remember that: $||\mathbf{x}||^2 = x_1^2 + x_2^2 + ... + x_n^2$, for $\mathbf{x} \in \mathbb{R}^n$.

(S. Maleki 2019)

A Gradient View to The Least Squares Approach

The sum of squared errors is a (convex) function of $\mathbf{w}$:

$$\mathcal{E}(\mathbf{w}) = (\mathbf{f} - \mathbf{y})^\top(\mathbf{f} - \mathbf{y}) = (\mathbf{\Phi}\mathbf{w} - \mathbf{y})^\top(\mathbf{\Phi}\mathbf{w} - \mathbf{y}) \ ,$$

where $\quad \mathbf{\Phi} = \begin{bmatrix} \phi_(\mathbf{x}_1)^\top \\ \phi_(\mathbf{x}_2)^\top \\ \vdots \\ \phi_(\mathbf{x}_n)^\top \end{bmatrix}$

Then, the gradient with respect to the weights is:

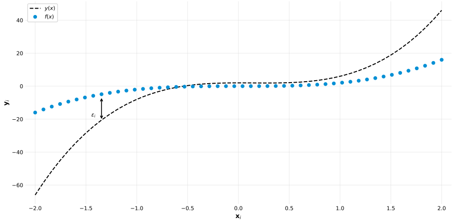
$$\frac{\partial \mathcal{E}(\mathbf{w})}{\partial \mathbf{w}} = 2\mathbf{\Phi}^\top(\mathbf{\Phi}\mathbf{w} - \mathbf{y}) = 2\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{w} - 2\mathbf{\Phi}^\top\mathbf{y} \ .$$

The weight vector $\hat{\mathbf{w}}$ that sets the gradient to zero (minimises $\mathcal{E}(\mathbf{w})$) is:

$$\hat{\mathbf{w}} = (\mathbf{\Phi}^\top \ \mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{y} \ .$$

(S. Maleki 2019)

Observation Noise



- Imagine the data was in reality generated by the dashed black function.
- But each $f(\mathbf{x}_i)$ was independently contaminated by a noise term ϵ_i.
- The observations are noisy: $y_i = f(\mathbf{x}_i) + \epsilon_i$.
- We can characterise the noise with a probability density function (often $\epsilon_i \sim \mathcal{N}(0, \sigma^2_{noise})$).

(S. Maleki 2019)

Probability of The Observed Data Given The Model

If we stack up independent noise terms we get:

$$\epsilon = [\epsilon_1, \epsilon_2, \ldots, \epsilon_n]^\top .$$

Since $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma_{noise}^2 \mathbf{I})$:

$$p(\epsilon) = \prod_{i=1}^{n} p(\epsilon_i) = \left(\frac{1}{\sqrt{2\pi\sigma_{noise}^2}} \right)^n e^{-\frac{\epsilon^\top \epsilon}{2\sigma_{noise}^2}} .$$

(S. Maleki 2019)

Probability of The Observed Data Given The Model

Given $\mathbf{f} = \mathbf{y} + \epsilon$, we can write the probability of $\mathbf{y}$ given $\mathbf{f}$:

$$p(\mathbf{y}|\mathbf{f}, \sigma^2_{noise}) = \mathcal{N}(\mathbf{y}; \mathbf{f}, \sigma^2_{noise}) = \left(\frac{1}{\sqrt{2\pi\sigma^2_{noise}}}\right)^n e^{-\frac{||\mathbf{f}-\mathbf{y}||^2}{2\sigma^2_{noise}}}$$
$$= \left(\frac{1}{\sqrt{2\pi\sigma^2_{noise}}}\right)^n e^{-\frac{\mathcal{E}(\mathbf{w})}{2\sigma^2_{noise}}}.$$

- $\mathcal{E}(\mathbf{w}) = \sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2 = ||\mathbf{\Phi}\mathbf{w} - \mathbf{y}||^2 = \epsilon^\top \epsilon$ is the sum of squared errors.
- Since $\mathbf{f} = \mathbf{\Phi}\mathbf{w}$, we can write $p(\mathbf{y}|\mathbf{w}, \sigma^2_{noise}) = p(\mathbf{y}|\mathbf{f}, \sigma^2_{noise})$, for a given $\mathbf{\Phi}$.

(S. Maleki 2019)

Likelihood Function

The *likelihood* of the parameters expressed how likely it is to observe the data, given the parameters.

- $p(\mathbf{y}|\mathbf{w}, \sigma^2_{noise})$ is the probability of the observed data given the weights.

- $\mathcal{L}(\mathbf{w}) \propto p(\mathbf{y}|\mathbf{w}, \sigma^2_{noise})$ is the likelihood of the weights.

(S. Maleki 2019)

Maximum Likelihood

We can fit the model weights to the data by maximising the likelihood:

$$\hat{\mathbf{w}} = \operatorname{argmax} \mathcal{L}(\mathbf{w}) = \operatorname{argmax} e^{-\frac{\mathcal{E}(\mathbf{w})}{2\sigma^2_{noise}}} = \operatorname{argmin} \mathcal{E}(\mathbf{w}) \; .$$

With an additive Gaussian independent noise model, the maximum likelihood and the least squares solutions are the same.

(S. Maleki 2019)

Gradient Descent

Define the cost function:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\left(f(\mathbf{x}_i) - y_i\right)^2 .$$

We want to choose $\mathbf{w}$ so as to minimise $J(\mathbf{w})$.

To do so, let's use a search algorithm that starts with some "initial guess" for $\mathbf{w}$, and that repeatedly changes $\mathbf{w}$ to make $J(\mathbf{w})$ smaller, until hopefully we converge to a value of $\mathbf{w}$ that minimises $J(\mathbf{w})$.

(S. Maleki 2019)

Gradient Descent

Specifically, we consider the gradient descent algorithm, which starts with some initial $\mathbf{w}$, and repeatedly performs the update:

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w}) \; .$$

Note that this update is simultaneously performed for all values of $j = 0, ..., d$.

Here, α is called the learning rate. This is a very natural algorithm that repeatedly takes a step in the direction of steepest decrease of J.

Gradient Descent

For simplicity, assume we only have one training example $(\mathbf{x} \in \mathbb{R}^m, y)$. Then:

$$\begin{aligned}
\frac{\partial}{\partial w_j} J(\mathbf{w}) &= \frac{\partial}{\partial w_j} \frac{1}{2}\Big(f(\mathbf{x}) - y\Big)^2 \\
&= 2.\frac{1}{2}\Big(f(\mathbf{x}) - y\Big).\frac{\partial}{\partial w_j}\Big(f(\mathbf{x}) - y\Big) \\
&= \Big(f(\mathbf{x}) - y\Big).\frac{\partial}{\partial w_j}\Big(\sum_{j=0}^{d} w_j x^{(j)} - y\Big) \\
&= \Big(f(\mathbf{x}) - y\Big)x^{(j)} \ .
\end{aligned}$$

For a single training example, this gives the update rule:

$$w_j := w_j - \alpha\Big(y - f(\mathbf{x})\Big)x^{(j)} \ .$$

(S. Maleki 2019)

Gradient Descent

In the case we have n examples, the update rule becomes:

$$w_j := w_j - \alpha \sum_{i=1}^{n} \Big(f(\mathbf{x}_i) - y_i \Big) x_i^{(j)} \ ,$$

where $x_i^{(j)}$ is the j-th element of i-th training example.

The update rule should be repeated until convergence.

$\Big(f(\mathbf{x}_i) - y_i \Big)$ is called the error term. If we encounter a small error, the update to the parameters will be also proportionally small. In contrast, if the error term is large, a larger change to the parameters will be made.

(S. Maleki 2019)

Gradient Descent - Matrix Form

The cost function is given by:

$$J(\mathbf{w}) = \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \ ,$$

and its gradient is:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \ .$$

Then the updating rules for the weights are given by:

$$\mathbf{w} := \mathbf{w} - \alpha \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \ ,$$

where α is the learning rate.

(S. Maleki 2019)