Chapter 2

Regression

2.1 Multivariate Linear Regression

2.1.1 Statement of the problem

We start by bridging the field of statistics into machine learning.

Statistics	Machine Learning	Notation	Remarks
independent variable	feature	$x_j^{(i)}$	$j = 1, \cdots, N$
dependent variable	outcome	$y^{(i)}$	
sample	example	$(x^{(i)}, y^{(i)})$	$i = 1, \cdots, M$
model	hypothesis	$h_\theta(x)$	
parameter	parameter	θ_j	
intercept	bias	θ_0	

In particular, we denote M as the total number of examples, and N the total number of features. $x_j^{(i)}$ shall represent the j^{th}-feature of the i^{th} example, and $y^{(i)}$ the outcome of the i^{th}-example.

Hypothesis

For multivariate linear regression, the mainstream hypothesis is

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_N x_N = \theta^T x \tag{2.1}$$

where $x_0 = 1$.[1] Here, $\theta \in \mathbb{R}^{N+1}$, $x^{(i)} \in \mathbb{R}^{N+1}$, $y^{(i)} \in \mathbb{R}$. Note that $\theta^T x$ is commonly known as the logit.

[1] $x_0 = 1$ is deliberately chosen so that the hypothesis can be compactly expressed as $h_\theta(x) = \sum_{j=0}^{N} \theta_j x_j = \theta^T x$.

The particular case of $x_1 = x$, $x_2 = x^2$, $\cdots$ is known as polynomial regression. In general, x_{j_1} and x_{j_2} need not be independent from each other.

Maximum log-likelihood estimation / Cost function

For multivariate linear regression, we have

$$y^{(i)} = \theta^T x^{(i)} + \epsilon \tag{2.2}$$

with error $\epsilon \sim \mathcal{N}(0, \sigma^2)$, for some σ^2, in the limit of large M. Assuming[2] that $x^{(i)}$ ($y^{(i)}$) are samples drawn from i.i.d. random variables $X^{(i)}$ ($Y^{(i)}$), we have

$$Y^{(i)} \sim \mathcal{N}(\theta^T X^{(i)}, \sigma^2) \tag{2.3}$$

thus the probability density function

$$f(y^{(i)} \,|\, x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) . \tag{2.4}$$

The likelihood function

$$\mathbb{L}(\theta \,|\, (x^{(1)}, y^{(1)}) \cdots (x^{(M)}, y^{(M)})) = f(y^{(1)} \cdots y^{(M)} \,|\, x^{(1)} \cdots x^{(M)}; \theta) \tag{2.5}$$

measures how likely the training examples $\{(x^{(i)}, y^{(i)})\}$ can be explained with our hypothesis characterized by θ in this case. Under i.i.d. assumptions, we can express the joint probability density function as

$$\mathbb{L}(\theta) = \prod_{i=1}^{M} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) , \tag{2.6}$$

and thus

$$\hat{\theta} = \arg\max_{\theta} \mathbb{L}(\theta) = \arg\max_{\theta} \log \mathbb{L}(\theta) \tag{2.7}$$

$$\hat{\theta} = \arg\max_{\theta} \log \mathbb{L}(\theta) = \arg\max_{\theta} \left\{ -\sum_{i=1}^{M} \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2} \right\} \tag{2.8}$$

$$\hat{\theta} = \arg\min_{\theta} \left\{ \sum_{i=1}^{M} \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2} \right\} = \arg\min_{\theta} \left\{ \sum_{i=1}^{M} \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2M} \right\} \tag{2.9}$$

[2] The fundamental assumptions underlying machine learning are that the examples are independent and identically distributed (i.i.d.) according to some probability distribution.

Mathematically, optimizing $\hat{\theta}$ involves a minimization of the cost function

$$J(\theta) = \frac{1}{2M}\sum_{i=1}^{M}(h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{M}\sum_{i=1}^{M} L(y^{(i)}, h_\theta(x^{(i)})) \tag{2.10}$$

w.r.t. θ. We denote $L(y^{(i)}, h_\theta(x^{(i)}))$ as the loss function.

For batch learning where all examples are known at compute time, the cost function J can be efficiently calculated at one go, and thus is a suitable candidate as our objective function. On the other hand, for online learning where examples arrive sequentially over time, we have to resort to the loss function L as our objective function. Somewhere in between batch learning and online learning lies the concept of mini-batch learning.

Convex optimization

We evaluate the Laplacian of $J(\theta)$ w.r.t. θ

$$\begin{aligned}
\Delta_\theta J(\theta) &= \nabla_\theta \cdot \nabla_\theta J(\theta) \\
&= \nabla_\theta \cdot \nabla_\theta \left[\frac{1}{2M}\sum_{i=1}^{M}(\theta^T x^{(i)} - y^{(i)})^2\right] \\
&= \nabla_\theta \cdot \left[\frac{1}{M}\sum_{i=1}^{M}(\theta^T x^{(i)} - y^{(i)})\, x^{(i)}\right] \\
&= \frac{1}{M}\sum_{i=1}^{M}|x^{(i)}|^2 \;\; \geq 0
\end{aligned} \tag{2.11}$$

Therefore, $J(\theta)$ is convex, which means that there exists no local minima in the entire landscape of $J(\theta)$. The existence of only one global minimum lies the foundation for stochastic optimization in multivariate linear regression.

Feature scaling

In order for different features to have similar scale in their measures, we need to perform feature scaling. For multivariate linear regression, in particular, feature scaling ensures all θ_j lie in the same range, giving numerical advantages. First, regularization, as we shall see later as an effective tool against overfitting, via $\sum_{j=1}^{N}\theta_j^2$, shall now make better sense when we even out all our features. Second, stochastic algorithms, such as gradient descent, will have enhanced numerical stability in their search towards the global minimum, thus having faster convergence.

We adopt normalization as our choice of feature scaling, i.e.

$$x_j^{(i)} := \frac{x_j^{(i)} - \mu_j^x}{s_j^x} \tag{2.12}$$

where μ_j^x (s_j^x) is the mean (s.d.) of the j^{th}-feature of our M examples.

2.1.2 Deterministic method

Our objective is to minimize the cost function (Eq. 2.10) via deterministic method.

Design matrix

Minimizing (Eq. 2.10), i.e.

$$J(\theta) = \frac{1}{2M}\sum_{i=1}^{M}(\theta^T x^{(i)} - y^{(i)})^2 = \frac{1}{2M}\left|\begin{array}{c} \theta^T x^{(1)} - y^{(1)} \\ \vdots \\ \theta^T x^{(M)} - y^{(M)} \end{array}\right|^2 \tag{2.13}$$

or

$$J(\Theta) = \frac{1}{2M}\left|X\Theta - Y\right|^2 \tag{2.14}$$

with **design matrix** (also known as the **feature matrix**):

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_N^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_N^{(2)} \\ & \vdots & & \vdots & \\ 1 & x_1^{(M)} & x_2^{(M)} & \cdots & x_N^{(M)} \end{bmatrix} = \begin{bmatrix} -\left(x^{(1)}\right)^T - \\ -\left(x^{(2)}\right)^T - \\ \vdots \\ -\left(x^{(M)}\right)^T - \end{bmatrix} \in \mathbb{R}^{M\times(N+1)} , \tag{2.15}$$

parameter matrix:

$$\Theta = [\theta_0\ \theta_1\ \theta_2\ \cdots\ \theta_N]^T \in \mathbb{R}^{(N+1)\times 1} , \tag{2.16}$$

outcome matrix:

$$Y = [y^{(1)}\ y^{(2)}\ \cdots\ y^{(M)}]^T \in \mathbb{R}^{M\times 1} . \tag{2.17}$$

Warning: Do NOT attempt to construct the design matrix in reality. In practice, M is usually very large, thus constructing such a huge matrix in memory is computationally inefficient nor unrealistic. As we shall see in the following section, X and Y are intermediate constructs which could be avoided in reality.

Normal equation

From (2.14)

$$\begin{aligned} J(\Theta) &= \frac{1}{2M}\left|X\Theta - Y\right|^2 = \frac{1}{2M}\left(\Theta^T X^T - Y^T\right)(X\Theta - Y) \\ J(\theta) &= \frac{1}{2M}\left(\sum_{\alpha\beta}\theta_\alpha\,(X^TX)_{\alpha\beta}\,\theta_\beta - \sum_\alpha \theta_\alpha\,(X^TY)_{\alpha 0} - \sum_\alpha (Y^TX)_{0\alpha}\,\theta_\alpha + (Y^TY)_{00}\right) \\ \partial_{\theta_\delta} J(\theta) &= \frac{1}{M}\left(\sum_\beta (X^TX)_{\delta\beta}\,\theta_\beta - (X^TY)_{0\delta}\right)\ , \end{aligned}$$

we have

$$\nabla_\theta J(\theta) = \frac{1}{M}\left((X^TX)\theta - (X^TY)\right) \tag{2.18}$$

which should equate to zero at minimum cost function.

Therefore, we arrive at the **normal equation**:

$$\boxed{Q\theta \equiv \frac{1}{M}(X^TX)\,\theta = \frac{1}{M}X^TY} \tag{2.19}$$

where Q is the Hessian (matrix) of $J(\theta)$. Note that Q must be positive-semidefinite, i.e. $Q \geq 0$, due to the convexity of $J(\theta)$.

Multivariate linear regression:

$$\begin{pmatrix} \langle x_0^2\rangle & \langle x_0x_1\rangle & \langle x_0x_2\rangle & \cdots & \langle x_0x_N\rangle \\ \langle x_1x_0\rangle & \langle x_1^2\rangle & \langle x_1x_2\rangle & \cdots & \langle x_1x_N\rangle \\ \langle x_2x_1\rangle & \langle x_2x_1\rangle & \langle x_2^2\rangle & \cdots & \langle x_2x_N\rangle \\ & \vdots & & \vdots & \\ \langle x_Nx_0\rangle & \langle x_Nx_1\rangle & \langle x_Nx_2\rangle & \cdots & \langle x_N^2\rangle \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_N \end{pmatrix} = \begin{pmatrix} \langle x_0y\rangle \\ \langle x_1y\rangle \\ \langle x_2y\rangle \\ \vdots \\ \langle x_Ny\rangle \end{pmatrix} \tag{2.20}$$

where $Q_{\alpha\beta} \equiv \langle x_\alpha x_\beta\rangle = \sum_{i=1}^M x_\alpha^{(i)} x_\beta^{(i)}$, and $b_\alpha \equiv \langle x_\alpha y\rangle = \sum_{i=1}^M x_\alpha^{(i)} y^{(i)}$.

Note: $x_0 = 1$.

Polynomial regression:

$$\begin{pmatrix} \langle 1\rangle & \langle x\rangle & \langle x^2\rangle & \cdots & \langle x^N\rangle \\ \langle x\rangle & \langle x^2\rangle & \langle x^3\rangle & \cdots & \langle x^{N+1}\rangle \\ \langle x^2\rangle & \langle x^3\rangle & \langle x^4\rangle & \cdots & \langle x^{N+2}\rangle \\ & \vdots & & \vdots & \\ \langle x^N\rangle & \langle x^{N+1}\rangle & \langle x^{N+2}\rangle & \cdots & \langle x^{2N}\rangle \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_N \end{pmatrix} = \begin{pmatrix} \langle y\rangle \\ \langle xy\rangle \\ \langle x^2y\rangle \\ \vdots \\ \langle x^Ny\rangle \end{pmatrix} \tag{2.21}$$

Invertibility of normal equation

Most of the time when X^TX is invertible, θ can be deterministically obtained:

$$\theta = \left(X^TX\right)^{-1} X^TY = Q^{-1}b \tag{2.22}$$

Nonetheless, X^TX is not guaranteed to be invertible at all times. This creates problems and inconveniences to numerical recipes that attempt to use (2.22) as their underlying blackbox engine.

To resolve this, we resort to **regularization** – a useful technique, we shall see later, to systematically overcome overfitting. Here,

$$J(\theta) = \frac{1}{2M}\left(\sum_{i=1}^{M}\left(\theta^Tx^{(i)} - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{N}\theta_j^2\right) \tag{2.23}$$

with regularization parameter $\lambda \geq 0$. Note that the regularization term $\lambda\sum_{j=1}^{N}\theta_j^2 = \lambda(\theta_1^2 + \theta_2^2 + \cdots\theta_N^2)$ explicitly excludes θ_0^2, otherwise, in the limit of infinite λ, the optimization of $J(\theta)$ would be ill-defined[3]. In practice, increasing λ is an efficient method to reduce overfitting.

By setting $\nabla_\theta J(\theta) = 0$, we arrive at the normal equation with regularization:

$$\boxed{Q_\lambda\theta \equiv \frac{1}{M}\left(X^TX + \lambda\begin{bmatrix}0 & 0\\ 0 & 1_N\end{bmatrix}\right)\theta = \frac{1}{M}X^TY} \tag{2.24}$$

where 1_N is a N-dimensional identity matrix. With Q_λ being always invertible for $\lambda > 0$[4], the regularized normal equation (2.24) is numerically stable

[3] When $\lambda \to \infty$, the regularization term $\lambda\sum_{j=1}^{N}\theta_j^2$ dominates, and the optimization of $J(\theta)$ shall result in $\theta_1 = \theta_2 = \cdots = \theta_N = 0$, thus only the bias term θ_0 survives. Hence, increasing regularization parameter λ would suppress all non-bias parameters $\theta_1 \cdots \theta_N$.

[4] Without loss of generality, we shall assume that feature scaling has been performed. As a result, $\langle x_1\rangle = \langle x_2\rangle = \cdots = \langle x_N\rangle = 0$. Here,

$$Q_\lambda = \begin{pmatrix}1 & 0_{1\times N}\\ 0_{N\times 1} & Q'_\lambda\end{pmatrix} \text{ , s.t. } Q'_\lambda = \frac{1}{M}X'^TX' + \lambda 1_N$$

with X' being the reduced design matrix with the bias column removed. Since X'^TX' is positive-semidefinite, its eigenvalues must be non-negative. As $\lambda 1_N$ would only shift up the eigen-spectrum of X'^TX', the eigenvalues of Q'_λ must be strictly positive. As the determinant of Q'_λ is mathematically equivalent to the product of its eigenvalues, we have

$$\det Q = \det Q' > 0 \ .$$

Therefore, Q_λ must be invertible for $\lambda > 0$.

and suitable to be used as an algorithm blackbox.

The only remaining question now is the optimal choice for λ, something we shall investigate later in this chapter.

Some practical tips

As the deterministic method involves matrix inversion, its numerical complexity scales as $\mathcal{O}(N^3)$, with N to the total number of features.

In practice, with current computational architecture, should N fall within $10,000$, we shall always use deterministic method without any difficulty. When $N > 10,000$, we shall use stochastic method instead.

2.1.3 Stochastic method

Convex optimization

As we have seen in section (2.1.1), the cost function $J(\theta)$ is convex in the parameter space. In the absence of local minimum, the stochastic search will always converge to the same optimal θ. Note that feature scaling is always in-place to speed up numerical efficiency.

For multivariate linear regression, $J(\theta)$ takes the quadratic form

$$J(\theta) = \frac{1}{2}\,\theta^T Q\theta - b^T\theta + c \tag{2.25}$$

for some $Q \in \mathbb{R}^{(N+1)\times(N+1)}$, $b \in \mathbb{R}^{N+1}$, and $c \in \mathbb{R}$, such that the Hessian must be positive-semidefinite, i.e. $Q \geq 0$. Here,

$$Q_{\alpha\beta} = \langle x_\alpha x_\beta\rangle \ ,\ b_\alpha = \langle x_\alpha y\rangle \ ,\ c = \frac{1}{2}\langle y^2\rangle \ . \tag{2.26}$$

Last but not least, note that

$$\nabla^2_\theta J(\theta) = Q \tag{2.27}$$
$$\nabla_\theta J(\theta) = Q\theta - b \tag{2.28}$$

Mathematically, we say that $u^{[k]}$ is Q-orthogonal to $u^{[k']}$ if and only if

$$u^{[k]\,T}\,Q\,u^{[k']} = 0 \ .$$

Alternatively, we say that $u^{[k]}$ is conjugate to $u^{[k']}$ w.r.t. Q.

Gradient descent

Geometrically, $\nabla_\theta J(\theta)$ is a vector in the parameter space that points in the direction of maximal change in J at any arbitrary θ.

Intuitively, the gradient descent algorithm

$$\theta := \theta - \alpha \nabla_\theta J(\theta) \tag{2.29}$$

would therefore bring us to the optimal θ that minimizes $J(\theta)$, conditioned on the fact that the **training rate** α must be kept small enough for the underlying linear approximation to remain valid.

In practice, if α is chosen too small, the convergence would be highly inefficient. Otherwise, if α is chosen too large, the gradient descent algorithm would not be numerically stable, and thus would cause $J(\theta)$ to diverge. To make matters worse, the choice for optimal α differs for every single problem, which always involves manual intervention.

Hence, gradient descent is unsuitable for being used as a blackbox algorithm.

Conjugate gradient – brute force

In some Q-orthogonal basis $\{u^{[k]}\}$, we can express the optimal parameter

$$\hat{\theta} = \alpha_0 u^{[0]} + \alpha_1 u^{[1]} + \cdots + \alpha_N u^{[N]} \tag{2.30}$$

with $(N+1)$ scalar coefficients α_k, due to completeness relation.

With $\nabla_\theta J(\hat{\theta}) = 0$, or

$$Q\,\hat{\theta} = Q\left(\alpha_0 u^{[0]} + \alpha_1 u^{[1]} + \cdots + \alpha_N u^{[N]}\right) = b\ , \tag{2.31}$$

we have:

$$\alpha_k = \frac{u^{[k]\,T}\, b}{u^{[k]\,T}\, Q\, u^{[k]}}\ . \tag{2.32}$$

Hence, starting from the origin of the parameter space, we can locate $\hat{\theta}$ in exactly $(N+1)$ steps, given the Q-orthogonal basis $\{u^{[k]}\}$.

In reality, we can construct the Q-orthogonal basis from the negative gradients, i.e. $-g^{[k]} = -\nabla_\theta J(\theta^{[k]})$, at $\theta^{[0]}$, $\theta^{[1]}$, $\cdots$, $\theta^{[N]}$, via Gram-Schmidt

process:

$$\begin{aligned} u^{[0]} &= -g^{[0]} && (2.33)\\ u^{[1]} &= -g^{[1]} + {}_0\beta_1 u^{[0]} && (2.34)\\ u^{[2]} &= -g^{[2]} + {}_0\beta_2 u^{[0]} + {}_1\beta_2 u^{[1]} && (2.35)\\ &\vdots && (2.36)\\ u^{[N]} &= -g^{[N]} + {}_0\beta_N u^{[0]} + {}_1\beta_N u^{[1]} + \cdots + {}_{N-1}\beta_N u^{[N-1]} && (2.37) \end{aligned}$$

where the projection is given by

$$ {}_k\beta_{k'} = \frac{u^{[k]\,T}\, Q\, g^{[k']}}{u^{[k]\,T}\, Q\, u^{[k]}} \ . \tag{2.38}$$

Here, the (N+1) points, i.e. $\theta^{[0]} \cdots \theta^{[N]}$, can be randomly chosen to construct the linearly-independent Q-orthogonal basis $\{u^{[k]}\}$. Together with equation (2.30), we can obtain the optimal parameter $\hat{\theta}$ in exactly $(N+1)$ steps.

Conjugate gradient – iteration

Alternatively, we have an iterative scheme for conjugate gradient:

1. Randomly initialize $\theta^{[0]}$, and evaluate $u^{[0]} = -g^{[0]} = -\nabla_\theta J(\theta^{[0]})$.
2. Evaluate α_0 via equation (2.32).
3. Evaluate $\theta^{[1]} = \theta^{[0]} + \alpha_0 u^{[0]}$, and thus $g^{[1]} = \nabla_\theta J(\theta^{[1]})$.
4. Evaluate ${}_0\beta_1$ from equation (2.38), and thus $u^{[1]}$ via equation (2.34).
5. Reassign $u^{[0]} := u^{[1]}$, and repeat (2)-(5) until $u^{[1]}$ converges to below some tolerance ϵ.

Unlike gradient descent, the absence of a learning rate α makes conjugate gradient the default choice for blackbox algorithms in the topic of multivariate linear regression. In practice, conjugate gradient is not only much faster than gradient descent, but also much neater as well. Regarding computational complexity, as conjugate gradient involves only matrix-vector multiplication, it scales with $\mathcal{O}(N^2)$. It is relatively efficient.

It is worthwhile to note that there exist alternative algorithms, such as 1) BFGS[5], and 2) L-BFGS[6].

[5]Broyden-Fletcher-Goldfarb-Shanno

[6]Limited memory - BFGS