

Machine Learning

FIB, Master in Innovation and Research in Informatics

Marta Arias, Computer Science @ UPC

Lecture 2: Linear regression

Outline

1. Solving least squares linear regression problem by optimization
2. Probabilistic perspective:
 - ▶ maximum likelihood
 - ▶ bias/variance decomposition of MSE
 - ▶ maximum a posteriori, regression
3. Bayesian linear regression

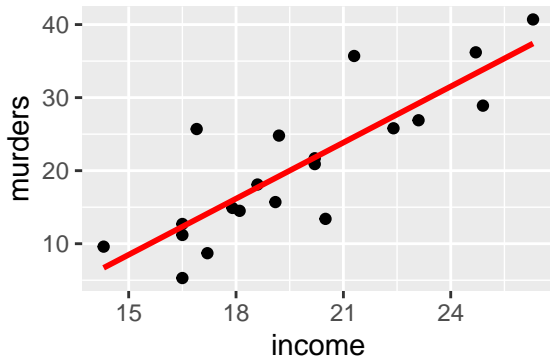
Regression

We have a dataset with data for 20 cities; for each city we have information on:

1. Nr. of inhabitants (in 10^3)
2. Percentage of families' incomes below 5000 USD
3. Percentage of unemployed
4. Number of murders per 10^6 inhabitants per annum

<i>inhabitants</i>	<i>income</i>	<i>unemployed</i>	<i>murders</i>
587	16.50	6.20	11.20
643	20.50	6.40	13.40
635	26.30	9.30	40.70
692	16.50	5.30	5.30
\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots
3353	16.90	6.70	25.70

Let us focus on a *single* variable (2D example)



Each point in the plot corresponds to a **row** of our data, and its coordinates are the (income, murders) values of that row.

The **red line** is “*the best*” linear model for this univariate regression model.

The optimization way

2D example, cont.

For $i = 1, \dots, 20$, we have plotted (x_i, y_i) , where x_i is the income and y_i is the murders rate for city i .

Want a line that approximates murders as a function of income. The slope θ_1 and intercept θ_0 define the shape of the line:

$$\hat{y}(x_i) = \hat{y}_i = \theta_0 + x_i\theta_1$$

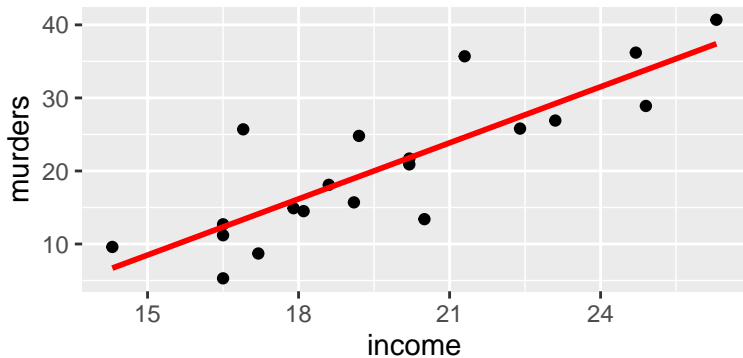
The **least squares** linear regression method tells us to choose the line that minimizes the following **error function** (a.k.a. objective function, loss function, cost function, ..):

$$J(\theta_0, \theta_1) = \sum_{i=1}^{20} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{20} (y_i - \theta_0 - x_i\theta_1)^2$$

This function depends on the **parameters** θ_0, θ_1 , since data is assumed fixed (whatever was observed)

2D example, cont.

$$J(\theta_0, \theta_1) = \sum_{i=1}^{20} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{20} (y_i - \theta_0 - x_i \theta_1)^2$$



2D example, cont.

To find θ_0, θ_1 s.t. $J(\theta_0, \theta_1)$ is minimized we will compute partial derivatives, set them to 0 and solve for θ_0, θ_1 .

Least squares regression: multi-variate case

But let us solve the general case, using all three features available. Now we have that the least squares solution is no longer a line, but a **hyperplane in 4D** given by the equation:

$$\hat{y}(\mathbf{x}_i) = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \theta_3 x_{i3} = \theta_0 + \sum_{j=1}^3 x_{ij} \theta_j = \sum_{j=0}^3 x_{ij} \theta_j$$

where we introduce $x_{i0} = 1$ for all i

Least squares regression: multi-variate case

For ease of notation, we will use vector and matrix operations for the multi-variate case

	<i>inhabitants</i>	<i>income</i>	<i>unemployed</i>	<i>murders</i>
1	587	16.50	6.20	11.20
2	643	20.50	6.40	13.40
3	635	26.30	9.30	40.70
4	692	16.50	5.30	5.30
\vdots	\vdots	\vdots	\vdots	\vdots
20	3353	16.90	6.70	25.70

is represented as

$$\mathbf{X} = \begin{bmatrix} 1 & 587 & 16.50 & 6.20 \\ 1 & 643 & 20.50 & 6.40 \\ 1 & 635 & 26.30 & 9.30 \\ 1 & 692 & 16.50 & 5.30 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 3353 & 16.90 & 6.70 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 11.20 \\ 13.40 \\ 40.70 \\ 5.30 \\ \vdots \\ 25.70 \end{bmatrix}$$

Least squares regression: multi-variate case, cont.

$$\mathbf{X} = \begin{bmatrix} 1 & 587 & 16.50 & 6.20 \\ 1 & 643 & 20.50 & 6.40 \\ 1 & 635 & 26.30 & 9.30 \\ 1 & 692 & 16.50 & 5.30 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 3353 & 16.90 & 6.70 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 11.20 \\ 13.40 \\ 40.70 \\ 5.30 \\ \vdots \\ 25.70 \end{bmatrix}$$

placing all coefficients θ_j into a column vector $\theta = [\theta_0 \ \theta_1 \ \dots \theta_d]^T$:

$$\hat{\mathbf{y}} = \mathbf{X}\theta$$

Spelling it out this is:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}$$

Least squares regression: multi-variate case, cont.

$$J(\theta) = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = (\mathbf{y} - \hat{\mathbf{y}})^T (\mathbf{y} - \hat{\mathbf{y}}) = (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta)$$

And so the least squares linear regression problem is solved by setting $\frac{\partial J(\theta)}{\partial \theta} = 0$ and solving for θ . Here, we will use the following facts: $\frac{\partial \mathbf{A}\theta}{\partial \theta} = \mathbf{A}^T$ and $\frac{\partial \theta^T \mathbf{B}\theta}{\partial \theta} = 2\mathbf{B}\theta$ if \mathbf{B} symmetric:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta} &= \frac{\partial (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta)}{\partial \theta} \\ &= \frac{\partial}{\partial \theta} [\mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\theta - \theta^T \mathbf{X}^T \mathbf{y} + \theta^T \mathbf{X}^T \mathbf{X}\theta] \\ &= \frac{\partial}{\partial \theta} [\mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T \mathbf{X}\theta + \theta^T \mathbf{X}^T \mathbf{X}\theta] \\ &= 0 - 2\mathbf{X}^T \mathbf{y} + 2\mathbf{X}^T \mathbf{X}\theta \end{aligned}$$

So $2\mathbf{X}^T \mathbf{X}\theta = 2\mathbf{X}^T \mathbf{y}$ implies

$$\theta_{lse} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Least squares regression: multi-variate case, cont.

The “best” linear model (defined by the one that minimizes least squares error) is given by

$$\theta_{lse} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Now, in order to make predictions on unseen test data $x' = [x'_1 \ x'_2 \ \dots \ x'_d]^T$ all we need to do is compute

$$y' = \begin{bmatrix} 1 & x'_1 & x'_2 & \dots & x'_d \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} = \theta_0 + \sum_{j=1}^d x'_j \theta_j$$

*This is the **optimization** view of learning; (1) set up error function as a fn. of parameters, (2) optimize it to find suitable values for the parameters, (3) use values to make predictions.*

Computation of least squares solution via the SVD

Let us look closely at the solution for coefficients θ via minimization of the squared error:

$$\theta_{lse} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- ▶ $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$
- ▶ $\mathbf{X}^T \mathbf{X} \in \mathbb{R}^{(d+1) \times (d+1)}$
- ▶ If \mathbf{X} has independent columns, then $\mathbf{X}^T \mathbf{X}$ is invertible
- ▶ Inverting this matrix can have numerical problems and so the SVD is used instead

Singular Value Decomposition of a rectangular matrix $A \in \mathbb{R}^{m \times n}$

Any matrix $A \in \mathbb{R}^{m \times n}$ with $m > n$ can be expressed as

$$A = U\Sigma V^T$$

where:

- ▶ $U \in \mathbb{R}^{m \times n}$ has orthonormal columns (so $U^T U = I$)
- ▶ $\Sigma \in \mathbb{R}^{n \times n}$ is diagonal and contains the singular values in its diagonal
- ▶ $V \in \mathbb{R}^{n \times n}$ has orthonormal rows and columns (so $V^T V = I$, $V V^T = I$ and $V^{-1} = V^T$)

Visually:

$$A = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix}$$

$$= \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \dots + \sigma_n \mathbf{u}_n \mathbf{v}_n^T$$

Computing least squares solution via the SVD

Let $\mathbf{X} = U\Sigma V^T$ be the SVD decomposition of data matrix \mathbf{X}

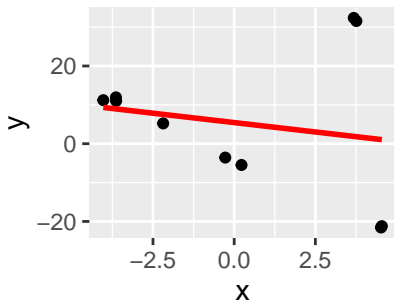
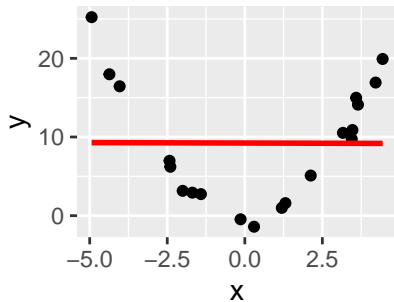
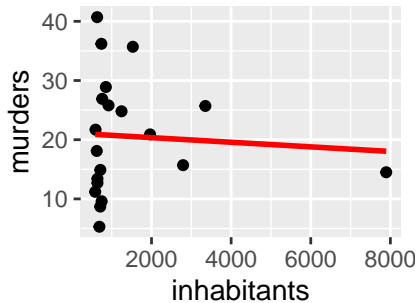
Then:

$$\begin{aligned}\theta_{lse} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \\ &= ((U\Sigma V^T)^T U\Sigma V^T)^{-1} (U\Sigma V^T)^T \mathbf{y} \\ &= (V\Sigma U^T U\Sigma V^T)^{-1} V\Sigma U^T \mathbf{y} \\ &= (V\Sigma^2 V^T)^{-1} V\Sigma U^T \mathbf{y} \\ &= (V^T)^{-1} \Sigma^{-2} V^{-1} V\Sigma U^T \mathbf{y} \\ &= V\Sigma^{-1} U^T \mathbf{y}\end{aligned}$$

```
import numpy as np

U, d, Vt = np.linalg.svd(X, full_matrices=False)
D = np.diag(1/d)
theta = Vt.T @ D @ U.T @ y
```


Things that could go wrong



Fixing the second problem.. use basis functions!

Linear regression with non-linear input features

What if we could do linear regression on an *expanded input*? In particular, if the new inputs are non-linear, we increment the expressive power of our prediction function. The predictive function is still **linear** because linearity is defined with respect to the *parameters* of the functions.

In general, the feature mapping is a non-linear transformation of the inputs $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^k$. The resulting predictive function is $y = \phi(x)\theta$.

For example, we could use a **polynomial expansion of degree k** so that:

- ▶ $\phi(x) = (1 \ x \ x^2 \ \dots \ x^k)$
- ▶ $y = \phi(x)\theta = \theta_0 + x\theta_1 + x^2\theta_2 + \dots + x^k\theta_k$

Note that functions start to have more complexity (if k is very high for example), so **complexity control** is going to be crucial to **avoid overfitting**.

Fixing the second problem.. use basis functions!

Linear regression with non-linear input features

The *new input data matrix* becomes

$$\Phi = \begin{bmatrix} \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_n) \end{bmatrix} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_k(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_k(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_n) & \phi_2(x_n) & \dots & \phi_k(x_n) \end{bmatrix}$$

And the *optimal* solution is:

$$\begin{aligned} \theta_{min} &= \arg \min_{\theta} (\mathbf{y} - \Phi\theta)^T (\mathbf{y} - \Phi\theta) \\ &= (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y} \end{aligned}$$

The probabilistic perspective

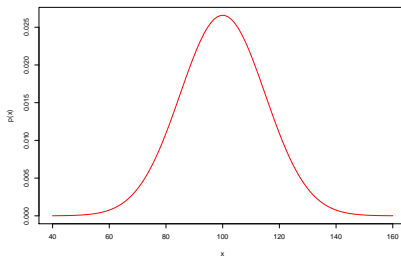
using probabilities to quantify uncertainty

Least squares regression, from a probabilistic perspective

Now we cast the problem in a probabilistic setting, and use the principle of **maximum likelihood** to derive the same linear regression estimates.

A key player is the **univariate Gaussian distribution** – the **normal** distribution – with *probability density function*:

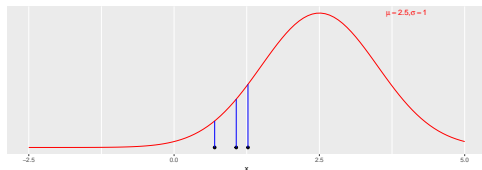
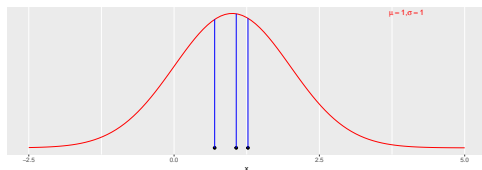
$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \quad \text{when } x \sim \mathcal{N}(\mu, \sigma^2)$$



Maximum likelihood principle

Out of these two normal densities, we will prefer the one that **maximizes the likelihood**

$$\mathcal{L}(\mu, \sigma; \{x_1, x_2, x_3\}) := P(x_1, x_2, x_3; \mu, \sigma) = \prod_i p(x_i; \mu, \sigma)$$

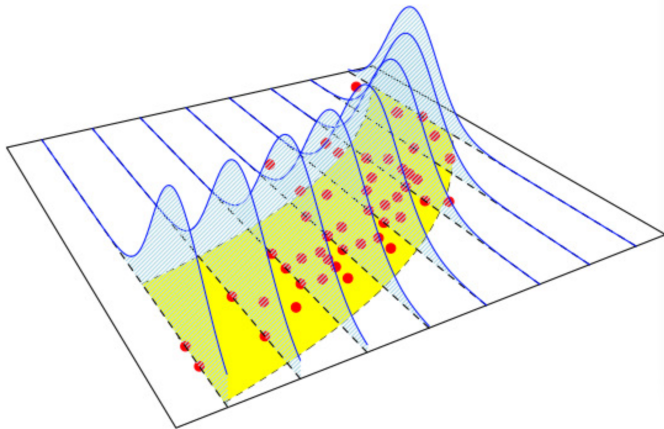


Note that the likelihood is a **function of the parameters**, assuming the data is fixed. Additionally, we assume that all x_i are *independent and identically distributed* according to $\mathcal{N}(\mu, \sigma^2)$.

Back to regression

In the probabilistic setting of linear regression, we assume that each label y_i we observe is normally distributed with mean $\mu = \mathbf{x}_i\theta$ and variance σ^2 :

$$y_i = \mathbf{x}_i\theta + \epsilon_i, \text{ where } \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$



Maximum likelihood for linear regression

Our data is given by a set of n labelled examples (\mathbf{x}_i, y_i) which are assumed to be iid according to $y_i \sim \mathcal{N}(\mathbf{x}_i\theta, \sigma^2)$ for unknown θ and σ .

As usual, it is convenient to place the data into a column vector \mathbf{y} of labels and a data matrix \mathbf{X} :

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

The likelihood of parameter vector θ is given by

$$\mathcal{L}(\theta, \sigma) := P(\mathbf{y}|\mathbf{X}; \theta, \sigma) = \prod_{i=1}^n p(y_i|\mathbf{x}_i; \theta, \sigma)$$

Maximum likelihood for linear regression, cont.

For numerical reasons we will maximize the *log-likelihood* instead:

$$\begin{aligned}l(\theta, \sigma) &:= \ln \mathcal{L}(\theta, \sigma) = \ln \prod_i p(y_i | \mathbf{x}_i; \theta, \sigma) \\&= \sum_i \ln p(y_i | \mathbf{x}_i; \theta, \sigma) \\&= \sum_i \ln \left[\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2} (y_i - \mathbf{x}_i \theta)^2} \right] \\&= \sum_{i=1}^n \left[\ln \frac{1}{\sqrt{2\pi\sigma^2}} + \ln \left(e^{-\frac{1}{2\sigma^2} (y_i - \mathbf{x}_i \theta)^2} \right) \right] \\&= -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_i (y_i - \mathbf{x}_i \theta)^2 \\&= -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta)\end{aligned}$$

Maximum likelihood for linear regression, cont.

$$l(\theta, \sigma) = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta)$$

Differentiating w.r.t to paramters and setting equal to 0

$$\frac{\partial l(\theta, \sigma)}{\partial \theta} = -\frac{1}{2\sigma^2} (-2\mathbf{X}^T \mathbf{y} + 2\mathbf{X}^T \mathbf{X}\theta) = 0$$

$$\frac{\partial l(\theta, \sigma)}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4} (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) = 0$$

leads to:

$$\theta_{ML} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

$$\sigma_{ML}^2 = \frac{1}{n} (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) = \frac{1}{n} \sum_i (y_i - \mathbf{x}_i \theta_{ML})^2 = MSE$$

Maximum likelihood for linear regression, cont.

Notice that the maximum likelihood solution **coincides** with the one we found minimizing **squared error**. The squared loss is a consequence of assuming gaussian noise. Other types of distributions are of course possible, and they correspond to minimizing **other** error functions.

So:

least squares linear regression == linear regression with Gaussian noise
--

and as a bonus, we get an estimate of how confident we can be on our predictions (given by the MSE)

Bias-Variance decomposition

Assume the following:

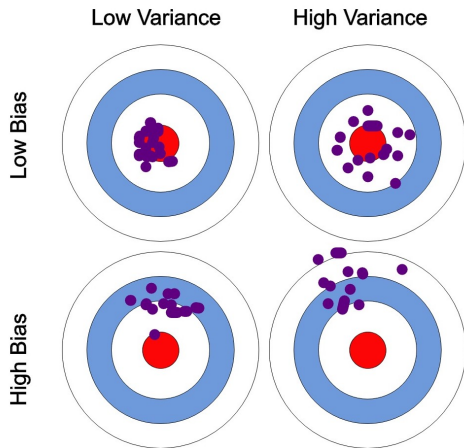
- ▶ let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be the **true function** that we are trying to approximate
- ▶ let D be a finite dataset for training $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, where $y_i = f(\mathbf{x}_i) + \epsilon_i$, and all ϵ_i are iid according to $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$
- ▶ let $\mathbf{x} \in \mathbb{R}^d$ be a test data point
- ▶ using D , we train a model \hat{f} ; the prediction according to \hat{f} is $\hat{y}_D = \hat{f}(\mathbf{x})$; here we add the subscript to emphasize that the prediction depends on the training dataset D

In this derivation we are going to see that its expected¹ squared error $(y - \hat{y}_D)^2$ can be decomposed as a sum of the following:

- ▶ the **irreducible error** given by σ^2
- ▶ the (squared) **bias** of the learning method; this is the systematic limitation that the modelling assumptions impose (e.g. using *linear functions* will never have low error on complex “non-linear” data)
- ▶ the **variance** of the learning method; this is how sensitive the modelling is to small variations of D

¹this **expectation** is taken over the possible choices of D of size n

Bias-Variance decomposition, dartboard intuition



Bias-Variance decomposition, derivation

In the following **derivation**, we are using basic facts about expectations; for brevity, we use f for the true value of $f(\mathbf{x})$

$$\begin{aligned}\mathbb{E}_D[(y - \hat{y}_D)^2] &= \mathbb{E}_D[(f + \epsilon - \hat{y}_D)^2] \\&= \mathbb{E}_D[(f + \epsilon - \hat{y}_D + \mathbb{E}_D[\hat{y}_D] - \mathbb{E}_D[\hat{y}_D])^2] \\&= \mathbb{E}_D[\underbrace{((f - \mathbb{E}_D[\hat{y}_D]))}_A + \underbrace{\epsilon}_B + \underbrace{(\mathbb{E}_D[\hat{y}_D] - \hat{y}_D)}_C]^2] \\&= \mathbb{E}_D[\underbrace{(f - \mathbb{E}_D[\hat{y}_D])^2}_{A^2} + \underbrace{\mathbb{E}_D[\epsilon^2]}_{B^2} + \underbrace{(\mathbb{E}_D[\hat{y}_D] - \hat{y}_D)^2}_{C^2}] \\&\quad + 2 \underbrace{\mathbb{E}_D[(f - \mathbb{E}_D[\hat{y}_D])\epsilon]}_{AB} \\&\quad + 2 \underbrace{\mathbb{E}_D[(f - \mathbb{E}_D[\hat{y}_D])(\mathbb{E}_D[\hat{y}_D] - \hat{y}_D)]}_{AC} \\&\quad + 2 \underbrace{\mathbb{E}_D[\epsilon(\mathbb{E}_D[\hat{y}_D] - \hat{y}_D)]}_{BC} \\&= \dots \\&= \mathbb{E}_D[\underbrace{(f - \mathbb{E}_D[\hat{y}_D])^2}_{A^2} + \underbrace{\mathbb{E}_D[\epsilon^2]}_{B^2} + \underbrace{(\mathbb{E}_D[\hat{y}_D] - \hat{y}_D)^2}_{C^2}] \\&= \text{Bias}[\hat{y}_D]^2 + \sigma^2 + \text{Var}[\hat{y}_D]\end{aligned}$$

Bias-Variance decomposition, conclusion

For one test example we had that:

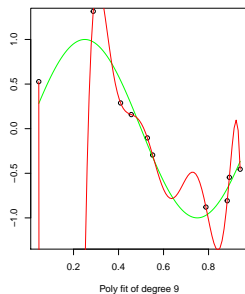
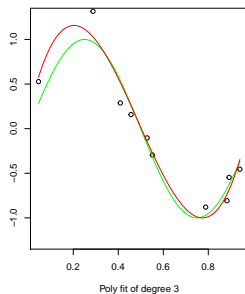
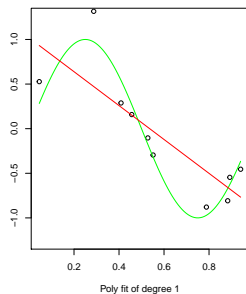
$$\mathbb{E}_D[(f(\mathbf{x}) - \hat{f}(\mathbf{x}; D))^2] = Bias_D[\hat{f}(\mathbf{x}; D)]^2 + Var_D[\hat{f}(\mathbf{x}; D)] + \sigma^2$$

So, to take into account the whole space (true error) we integrate over all possible values of \mathbf{x} :

$$\begin{aligned} MSE_{true} &= \int_{\mathbf{x}} [Bias_D[\hat{f}(\mathbf{x}; D)]^2 + Var_D[\hat{f}(\mathbf{x}; D)] + \sigma^2] p(\mathbf{x}) d\mathbf{x} \\ &= \mathbb{E}_x [Bias_D[\hat{f}(\mathbf{x}; D)]^2 + Var_D[\hat{f}(\mathbf{x}; D)]] + \sigma^2 \end{aligned}$$

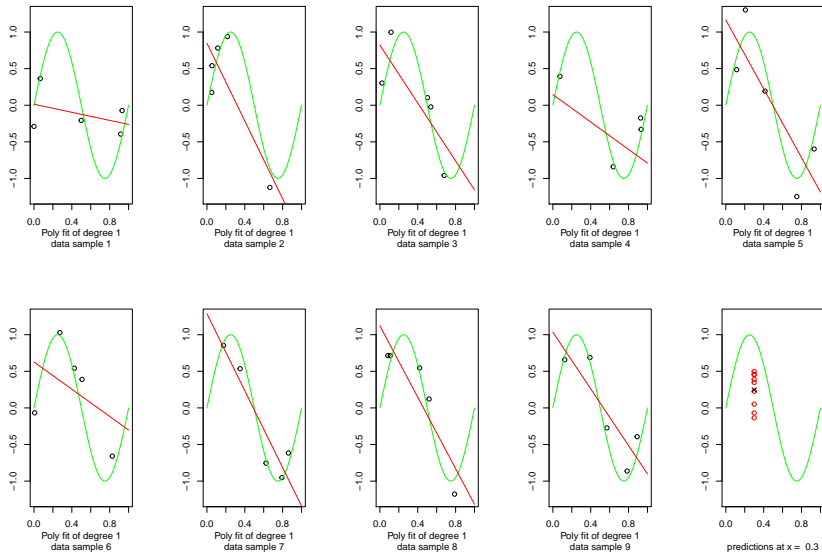
which is the **expected true error** or **expected generalization error**

Bias-Variance decomposition, relation to over/underfitting



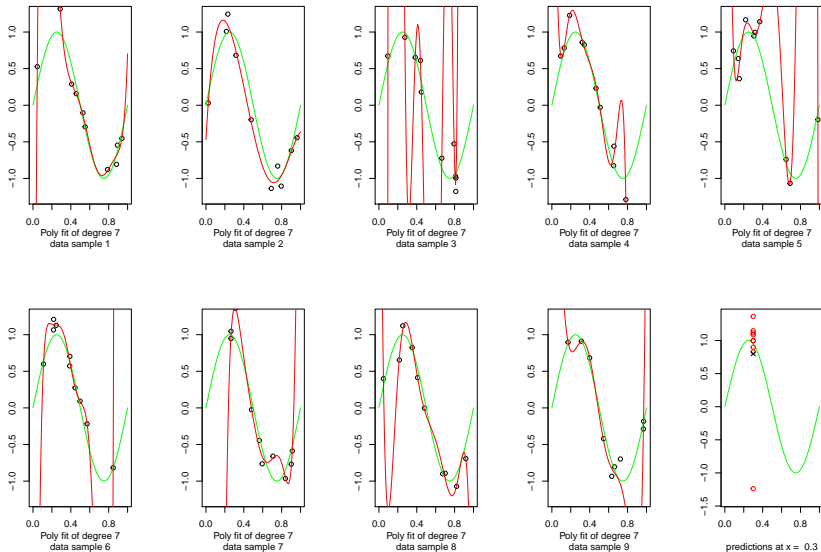
- ▶ high bias models will **underfit**
- ▶ high variance models will **overfit** especially if data is scarce (or: high variance model need tons of data to be fit properly)

Bias, variance of low complexity models (degree 1)



1. the model is quite stable (therefore it has **low Variance**)
2. the model is quite bad on average (therefore it has **high Bias**)

Bias, variance of high complexity models (degree 7)



1. the model is quite unstable (therefore it has **high Variance**)
2. the model is quite good *on average* (therefore it has **low Bias**)

Common error functions used in regression

<i>name</i>	<i>abbrev.</i>	<i>formula</i>
mean squared error	MSE	$\frac{1}{n} \sum_{i=1}^n (y_i - \mathbf{x}_i \theta)^2$
root mean squared error	$RMSE$	$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \mathbf{x}_i \theta)^2}$
normalized root mean squared error	$NRMSE$	$\sqrt{\frac{MSE}{Var(y)}}$
coefficient of determination	R^2	$1 - \frac{MSE}{Var(y)} = 1 - NRMSE^2$
mean absolute error	MAE	$\frac{1}{n} \sum_{i=1}^n y_i - \mathbf{x}_i \theta $

Regularization by Maximum a Posteriori (MAP)

Looking at Bayes for learning

Before proceeding, let us refresh Bayes rule in the context of learning (here, θ are the *parameters* and D is the given *data*):

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} = \frac{\textit{likelihood} \times \textit{prior}}{\textit{evidence}} = \textit{posterior}$$

And so, maximum likelihood ignores prior information on θ (or equivalently assumes all θ are equally likely before doing any modelling):

$$\theta_{ML} \stackrel{\text{def}}{=} \arg \max_{\theta} \log P(X|\theta)$$

If we do assume some prior distribution over the parameters θ that is we have a notion of what types of solution we prefer, then we can use **maximum a posteriori** estimate for θ :

$$\begin{aligned} \theta_{MAP} &\stackrel{\text{def}}{=} \arg \max_{\theta} \log [P(X|\theta)P(\theta)] \\ &= \arg \max_{\theta} [\log P(X|\theta) + \log P(\theta)] \end{aligned}$$

Ridge regression from Gaussian prior on θ

Let us turn to an example; assume Gaussian prior on d -dimensional θ . Here, we use the special case of an *isotropic Gaussian* (i.e. $\Sigma = \tau^2 I$) and so:

- ▶ $\Sigma^{-1} = \frac{1}{\tau^2} I$
- ▶ $\det \Sigma = \tau^{2d}$

$$\theta \sim \mathcal{N}(\mu = 0, \Sigma = \tau^2 I)$$

And so:

$$\begin{aligned} P(\theta; \mu = 0, \Sigma = \tau^2 I) &= \frac{1}{|\Sigma|^{\frac{1}{2}} (2\pi)^{\frac{d}{2}}} \exp \left\{ -\frac{1}{2} (\mathbf{y} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{y} - \boldsymbol{\mu}) \right\} \\ &= \frac{1}{(2\pi\tau^2)^{\frac{d}{2}}} \exp \left\{ -\frac{1}{2\tau^2} \theta^T \theta \right\} \\ &= \frac{1}{(2\pi\tau^2)^{\frac{d}{2}}} \exp \left\{ -\frac{\|\theta\|^2}{2\tau^2} \right\} \end{aligned}$$

Ridge regression from Gaussian prior on θ , cont.

$$\begin{aligned}P(\theta|\mathbf{y}, \mathbf{X}) &\propto P(\mathbf{y}|\mathbf{X}, \theta) P(\theta) \\&\propto \exp \left\{ -\frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) \right\} \exp \left\{ -\frac{\|\theta\|^2}{2\tau^2} \right\} \\&= \exp \left\{ -\frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) - \frac{\|\theta\|^2}{2\tau^2} \right\}\end{aligned}$$

And so the **maximum a posteriori** estimate becomes:

$$\begin{aligned}\theta_{MAP} &= \arg \max_{\theta} \log [P(\mathbf{y}|\mathbf{X}, \theta) P(\theta)] \\&= \arg \max_{\theta} \left[-\frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) - \frac{\|\theta\|^2}{2\tau^2} \right] \\&= \arg \min_{\theta} \left[(\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) + \frac{\sigma^2}{\tau^2} \|\theta\|^2 \right] \\&= \arg \min_{\theta} \left[(\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) + \lambda \|\theta\|^2 \right]\end{aligned}$$

which is the **ridge regression** estimate when $\lambda = \frac{\sigma^2}{\tau^2}$

Ridge regression estimate

As we did with ordinary least squares regression, we differentiate and set to 0 in order to find the minimum:

$$\frac{\partial}{\partial \theta} \left\{ \|\mathbf{y} - \mathbf{X}\theta\|^2 + \lambda \|\theta\|^2 \right\} = (-2\mathbf{X}^T \mathbf{y} + 2\mathbf{X}^T \mathbf{X} \theta) + 2\lambda \theta = 0$$

leading to

$$\theta_{MAP} = \theta_{ridge} = (\mathbf{X}^T \mathbf{X} + \lambda I)^{-1} \mathbf{X}^T \mathbf{y}$$

Observations:

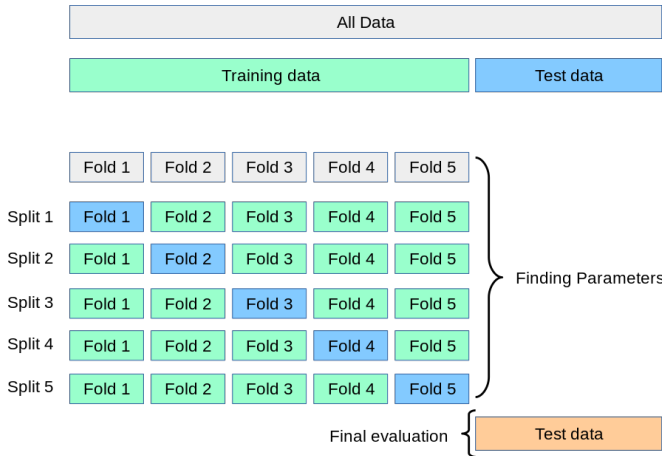
- ▶ λ controls the complexity of the solution θ (smaller “length” == smaller coefficients == simpler solution)
- ▶ $\mathbf{X}^T \mathbf{X} + \lambda I$ is guaranteed to be non-singular and behaves better numerically than $\mathbf{X}^T \mathbf{X}$, especially if the columns of \mathbf{X} are highly correlated, or if there are few observations (rows of \mathbf{X}) relative to number of predictors (columns of \mathbf{X}).
- ▶ as a general recipe when we have a regularized objective function, is to use *potentially-more-complex-than-needed functions* and then adjust λ

Tuning λ

We have a hyper-parameter λ . How do we set it?

1. training/validation split
2. cross-validation
3. leave-one-out-cross-validation (loocv)

Cross-validation



Cross-validation, cont.

1. Decide on a set of values for $\lambda \in \{\lambda_1, \dots, \lambda_l\} = \Lambda$
2. Partition training data into $K > 1$ folds
3. Repeat for $k = 1, \dots, K$:
 - ▶ use k -th fold for *validation*
 - ▶ use the remaining $K - 1$ for *training*; train with all λ values
 - ▶ estimate generalization on the one validation fold for all λ values
4. Average K generalization estimates, select λ that gives best cross-val estimation

Leave-one-out-cross-val is cross-validation for $K = n$ (one fold per example)

Notice that, in general, we have to train $K \times |\Lambda|$ times, so this may be costly ..

Loocv for ridge

As a particularity of linear and ridge regression, for a given λ value, only **one training** is necessary for *loocv*; so:

1. For each $\lambda \in \Lambda$:

- ▶ compute optimal solution $\hat{\theta}_\lambda = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T \mathbf{y}$
- ▶ compute “hat” matrix $H_\lambda = \Phi(\Phi^T \Phi + \lambda I)^{-1} \Phi^T$
- ▶ compute loocv directly for each λ (no need to use folds etc.):

$$loocv(\lambda) = \frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \phi(\mathbf{x}_i) \theta_\lambda}{1 - h_{ii}} \right)^2$$

2. Return λ with minimum *loocv*

GCV for ridge

The *loocv* for each λ can be approximated by the (numerically more stable) GCV; here we have substituted each h_{ii} by the *average value of the elements in the diagonal*:

$$GCV(\lambda) = \frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \phi(\mathbf{x}_i)\theta_\lambda}{1 - \frac{Tr(H_\lambda)}{n}} \right)^2 = \frac{MSE_\lambda}{\left(1 - \frac{Tr(H_\lambda)}{n}\right)^2}$$

where $Tr(H_\lambda)$ is the trace of H_λ (sum of diagonal elements).

LASSO regression

The p -norm of vector θ is:

$$\|\theta\|_p \stackrel{\text{def}}{=} \left(\sum_d |\theta_d|^p \right)^{\frac{1}{p}}$$

As we have seen, assuming an isotropic Gaussian prior on the parameters leads to **ridge** regression, which jointly minimizes the (squared) of the L2-norm of θ (so, $p = 2$) and the squared error. Another very common choice is $p = 1$, which leads to **lasso** regression.

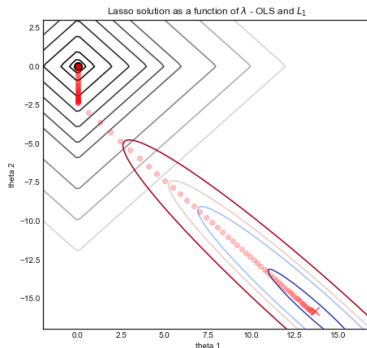
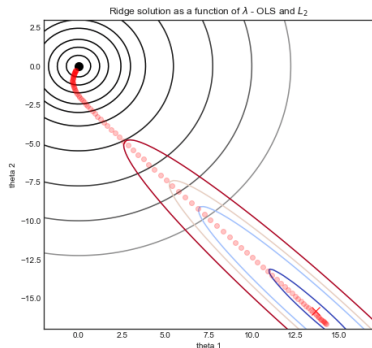
So **lasso** regression minimizes L1-norm of parameters and squared error:

$$\theta_{lasso} = \arg \min_{\theta} \left[\|\mathbf{y} - \mathbf{X}\theta\|_2^2 + \lambda \|\theta\|_1 \right]$$

Actually, **lasso** regression arises assuming a **Laplace distribution** prior over the parameters.

LASSO regression, cont.

Lasso regression gives **sparse** solutions in the sense that many of θ coefficients might be 0. So, it performs **feature selection**.



Lasso regularized cost function is no longer quadratic and has **no closed solution**, so an approximation procedure is used to optimize it called the **least angle regression**. This procedure exploits the special structure of the problem and provides an efficient way to compute the solutions for a list of possible values for $\lambda > 0$, giving the **regularization path**.

The full-Bayesian perspective

Bayesian linear regression