Foundations of Machine Learning

Mohammad Emtiyaz Khan
RIKEN-AIP, Tokyo
OIST, Okinawa

http://emtiyaz.github.io

emtiyaz.khan@riken.jp

May 12, 2022

Hello!

Goals

Understand (some) fundamentals of Machine learning[1].

Part I : Understand the basic set-up to analyze data under a machine-learning framework.

1. Before Machine Learning. Problem
2. ML Problem: Regression. Data
3. Model: Linear Regression. Model
4. Cost Function: MSE.
5. Algorithm 1: Gradient Descent.
6. Algorithm 2: Least Squares.

algorithm

"Foundation"

Part II : Understand what can go wrong when learning from data and how to correct it.

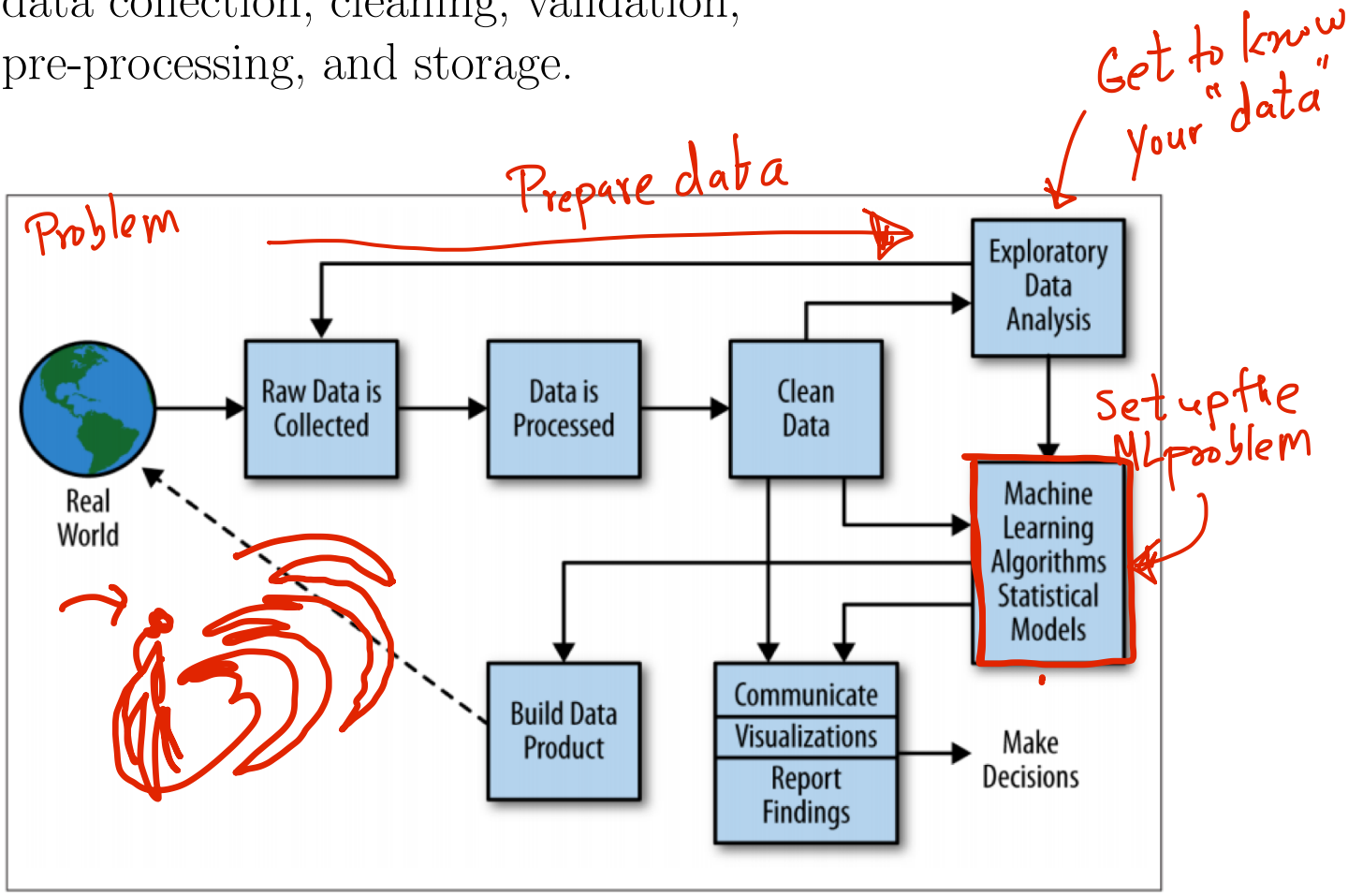
6. Challenge: Overfitting.
7. Solutions: Regularization.
8. Bias-Variance Decomposition. "Trade-off"
9. Recent Advances.

[1]Some figures are taken from Hastie, Tibshirani, and Friedman's book on statistical learning and also from Chris Bishop's Machine learning book

1 Before Machine Learning

Acquiring Data

Data is the most important component of modern Machine Learning. There are many important steps that can have a huge impact on the performance of a machine-learning system. To name a few: data collection, cleaning, validation, pre-processing, and storage.



Picture taken from "Doing data science".

Defining an ML problem

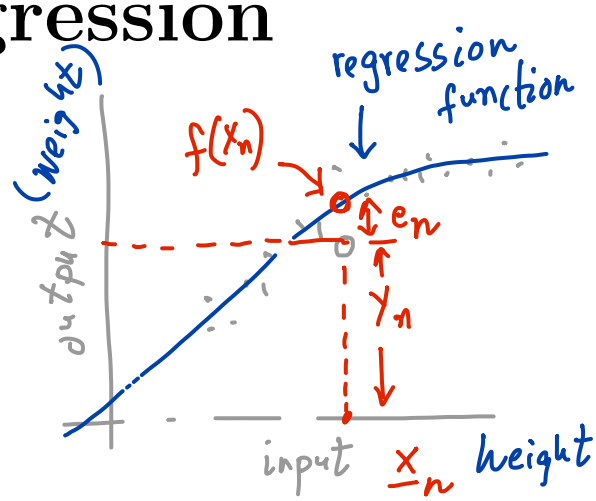
Once we have some data, the next step is to re-define the real-world problem in the context of data, and then to convert it to a machine-learning problem.

ML problems can be categorized into 3 main types: 1 supervised, 2 unsupervised, and 3 reinforcement learning. In practice, a successful end-to-end system might require a combination of these problems.

2 ML Problem: Regression

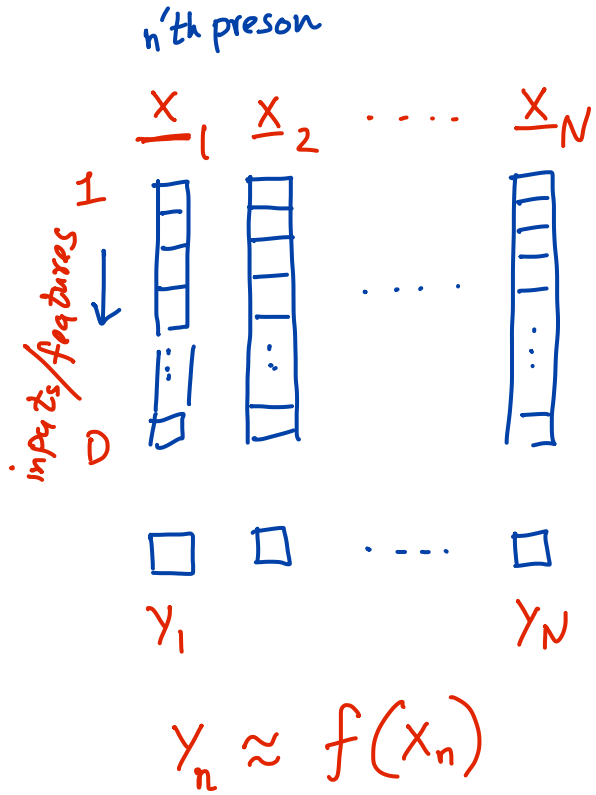
What is regression?

Regression is to relate input variables to the output variable, to either predict outputs for new inputs and/or to understand the effect of the input on the output.

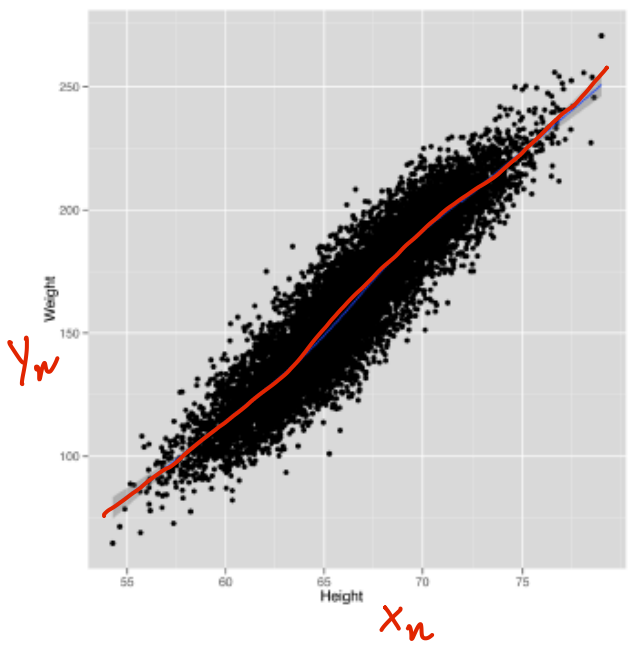


Dataset for regression

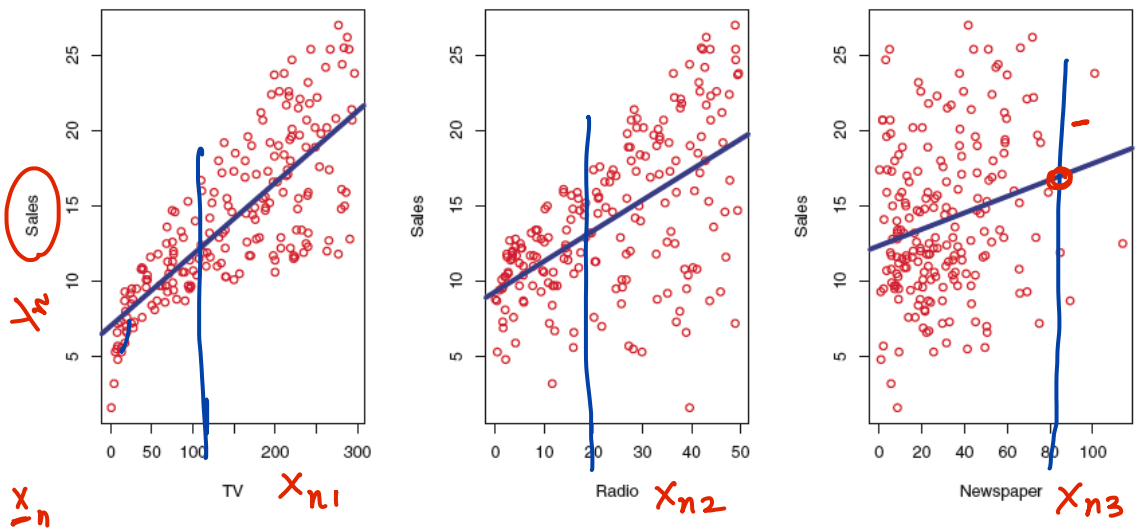
In regression, data consists of pairs $(y_n, \mathbf{x}_n)$, where y_n is the n'th output and $\mathbf{x}_n$ is a vector of D inputs. Number of pairs N is the data-size and D is the dimensionality.



Examples of regression



(a) Height is correlated with weight. Taken from "Machine Learning for Hackers"



$\hat{y}_* = f(\underline{x}_*)$

sales

TV
Radio
News paper

$\underline{x}_n$

$\hat{y}_*$ sales

$\underline{x}_*$

(b) How does advertisement in TV, radio, and newspaper affect sales? Taken from the book "An Introduction to statistical learning"

Two goals of regression

In prediction, we wish to predict the output for a new input vector, e.g. what is the weight of a person who is 170 cm tall?

In interpretation, we wish to understand the effect of inputs on output, e.g. are taller people heavier too?

The regression function

For both the goals, we need to find a function that approximates the output "well enough" given inputs.

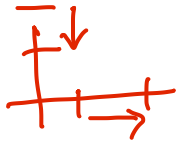
$$y_n \approx f(\mathbf{x}_n), \text{ for all } n$$

Additional Notes

Prediction vs Interpretation

Some questions to think about: are these prediction tasks or interpretation task?

Prediction 1. What is the life-expectancy of a person who has been smoking for 10 years?



interpretation 2. Does smoking cause cancer?

Prediction interp... 3. When the number of packs a smoker smokes per day doubles, their predicted life span gets cut in half? ??

I, P ? 4. A massive ? scale earthquake will occur in California within next 30 years.

5. More than 300 bird species in north America could reduce their habitat by half or more by 2080.

Qualitative vs Quantitative

Predictive models > Interpretive models ?

Nate Silver "the Signal & noise ..." ??

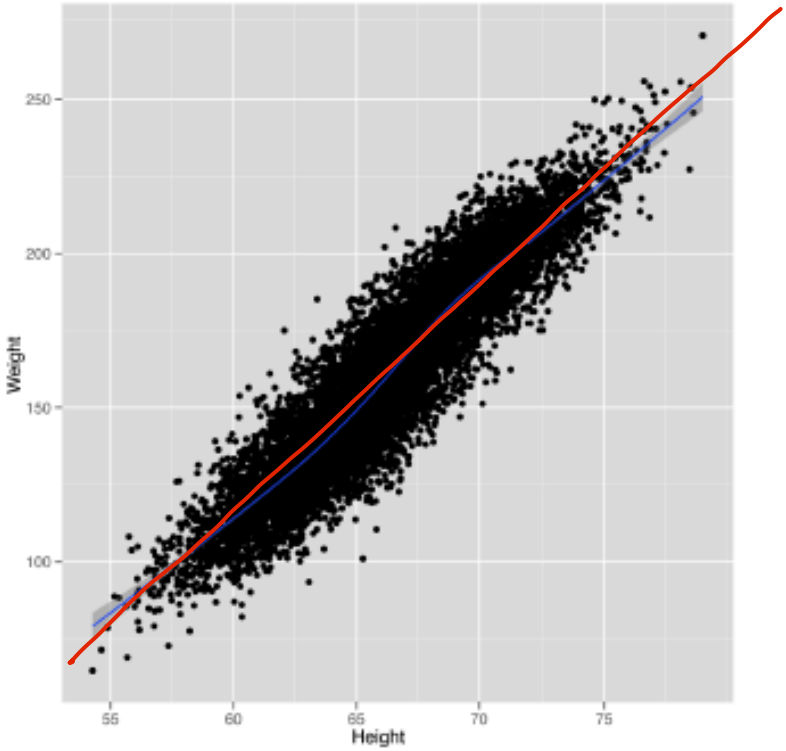
3 Model: Linear Regression

What is it?

Linear regression is a $\boxed{\text{model}}$ that assumes a linear relationship between inputs and the output.

$Y_n \approx f(x_n)$

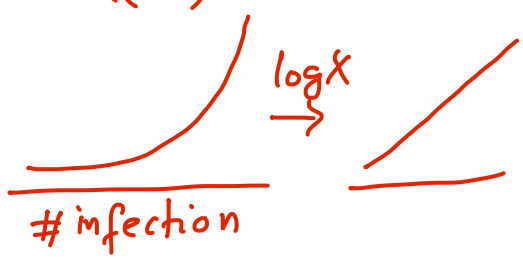
regression function



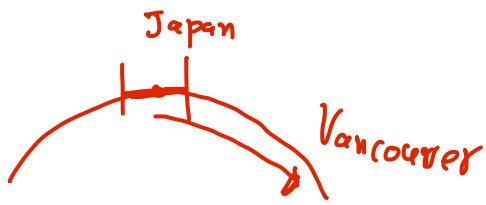
ax is linear in x

ax^2 is quadratic in x but linear in x^2

$a(x')$ with $x' = x^2$



infection



Why learn about *linear* regression?

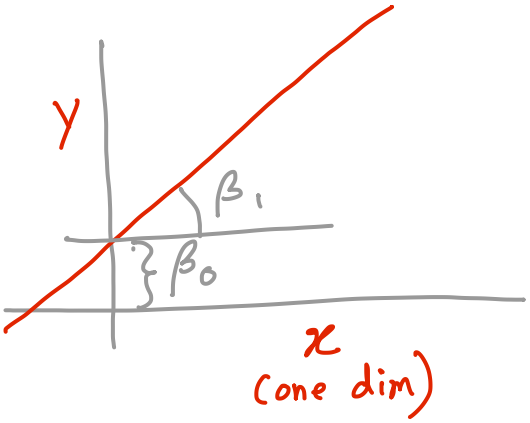
Plenty of reasons: simple, easy to understand, most widely used, easily generalized to non-linear models. Most importantly, you can learn almost all fundamental concepts of ML with regression alone.

Simple linear regression

With only one input dimension, it is simple linear regression.

$$y_n \approx f(\mathbf{x}_n) := \underbrace{\beta_0}_{\text{"bias"}} + \underbrace{\beta_1}_{\text{slope}} x_{n1}$$

Here, β_0 and β_1 are parameters of the model.



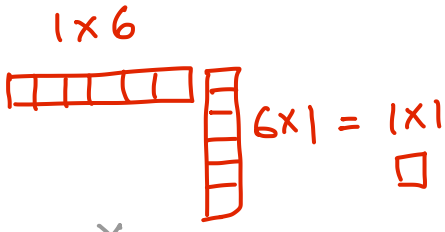
Data → Model (parameters)

Multiple linear regression

With multiple input dimension, it is multiple linear regression.

$$\begin{aligned} y_n &\approx f(\mathbf{x}_n) \\ &:= \beta_0 + \beta_1 x_{n1} + \ldots + \beta_D x_{nD} \\ &= \tilde{\mathbf{x}}_n^T \boldsymbol{\beta} \end{aligned} \tag{1}$$

bias



Dimension

$$\underbrace{\overbrace{[1 \; x_{n1} \; x_{n2} \cdots x_{nD}]}^{\underline{x}_n}}_{\tilde{\underline{x}}_n} \underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_D \end{bmatrix}}_{\underline{\beta}} = \tilde{\underline{x}}_n^T \underline{\beta}$$

$$\underset{N\times D}{A} \; \underset{D\times M}{B} = \underset{N\times M}{C}$$

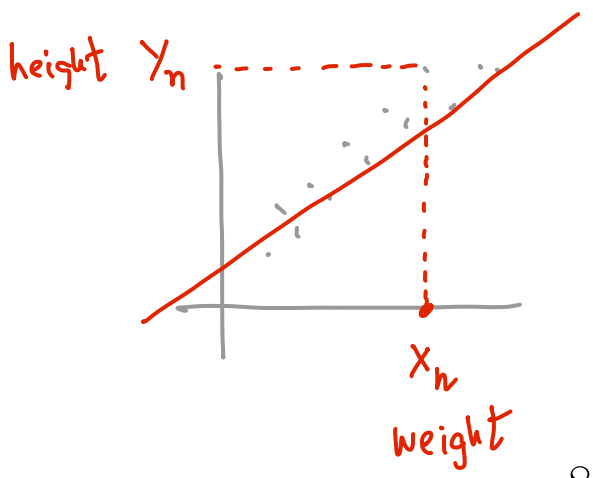
$$\underset{1\times 6}{\beta_0 \; \beta_1} \; \underset{6\times 1}{x_{n,1} \quad x_{n,2}} \; \underset{1\times 1}{y_n}$$

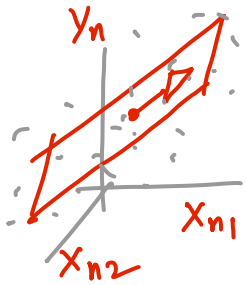
y_1 180 $\approx 0 + 3\times 60 + 1\times 2K$

y_2 185 $\approx 0 + 3\times 70 + 1\times 3K$

Learning/estimation/fitting

Given data, we would like to find $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_D]$. This is called learning or estimating the parameters or fitting the model.





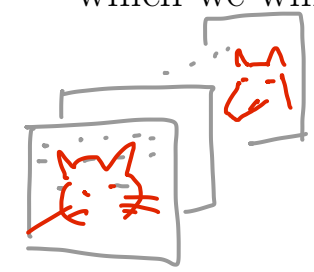
Additional Notes

$p > n$ Problem

Consider the following simple situation: You have $N = 1$ and you want to fit $y_1 \approx \beta_0 + \beta_1 x_{11}$, i.e. you want to find β_0 and β_1 given one pair (y_1, x_{11}). Is it possible to find such a line? Any line is good! degrees of freedom, (N, D)

This problem is related to something called $p > n$ problem. In our notation, this will be called $D > N$ problem, i.e. the number of parameters exceeds number of data examples.

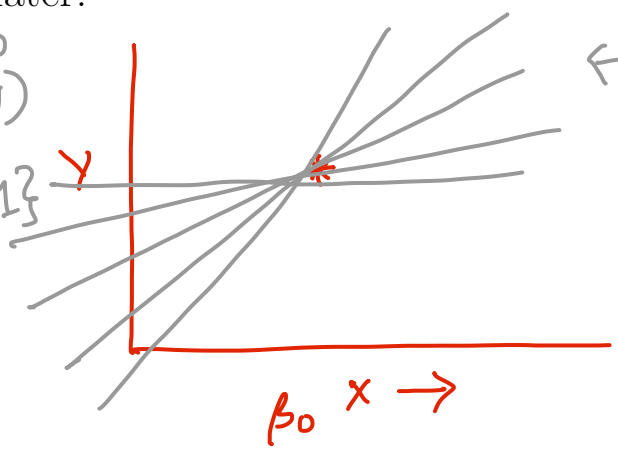
Similar issues will arise when we use gradient descent or least-squares to fit a linear model. These problems are all solved by using regularization, which we will learn later.



10,000 (N)

$Y \in \{0,1\}$

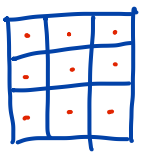
32 x 32 = 1024 (D)



← Infinite # sol^n

$\boxed{Y_1 = \beta_0 + \beta x_1}$

$Y_2 = \beta_0 + \beta x_2$

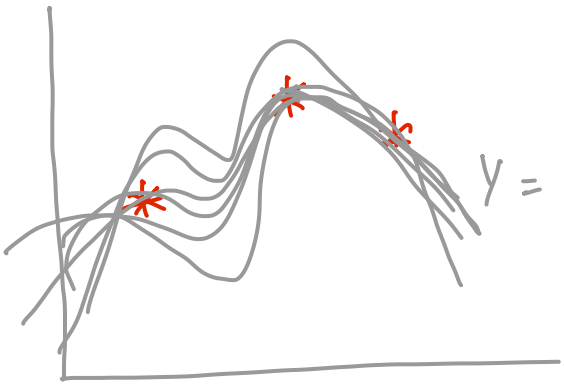


$$\begin{bmatrix} x_{n1} \\ x_{n2} \\ \vdots \\ x_{ng} \end{bmatrix} \begin{matrix} \beta_1 \\ \beta_2 \\ \\ \beta_g \end{matrix} = \tilde{X}_n^T \underline{\beta} \rightarrow Y_n$$

↓

transfor

$$[\underline{\phi}(\tilde{X}_n)]\underline{\beta}$$



$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_1^3$$

4 Cost Function: MSE

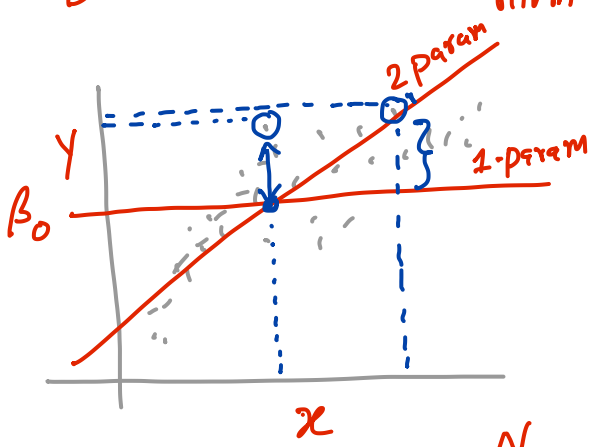
Data → Model → algorithm

Motivation

Consider the following models.

1-parameter model: $y_n \approx \beta_0$
2-parameter model: $y_n \approx \beta_0 + \beta_1 x_{n1}$

How can we estimate (or guess) values of $\boldsymbol{\beta}$ given the data $\mathcal{D}$?



Data: $\mathcal{D} = \{y_n, \underline{x}_n\}_{n=1}^{N}$

Model: $y_n \approx \tilde{\underline{x}}_n^T \underline{\beta}$

Cost func: $C(\mathcal{D}, \underline{\beta})$

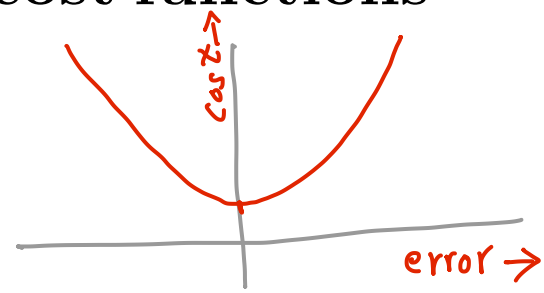
Error: $e_n = y_n - \tilde{\underline{x}}_n^T \underline{\beta}$

What is a cost function?

Cost functions (or utilities or energy) are used to learn parameters that explain the data well. They define how costly our mistakes are.

Two desirable properties of cost functions

When y is real-valued, it is desirable that the cost is symmetric around 0, since both +ve and -ve errors should be penalized equally.



$C(e_1, e_2, \ldots e_N)$

Also, our cost function should penalize "large" mistakes and "very-large" mistakes almost equally.

Mean Square Error (MSE)

$C(e_1, e_2 \dots e_N)$

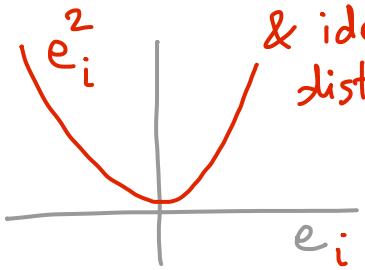
$= \frac{1}{2N}\sum_{i=1}^{N} MSE(e_i) = \frac{1}{2N}\sum_{i=1}^{N} e_i^2$

MSE is one of the most popular cost function.

Assumption: "all data are equal"

More precisely: i.i.d. (independent & identically distributed)

$$MSE(\boldsymbol{\beta}) := \frac{1}{2N}\sum_{n=1}^{N}[y_n - f(\mathbf{x}_n)]^2$$



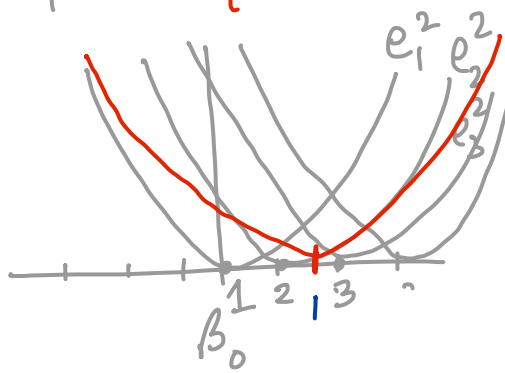
Does it have both the properties?

symmetric, yes!

An exercise for MSE

Compute MSE for 1-param model:

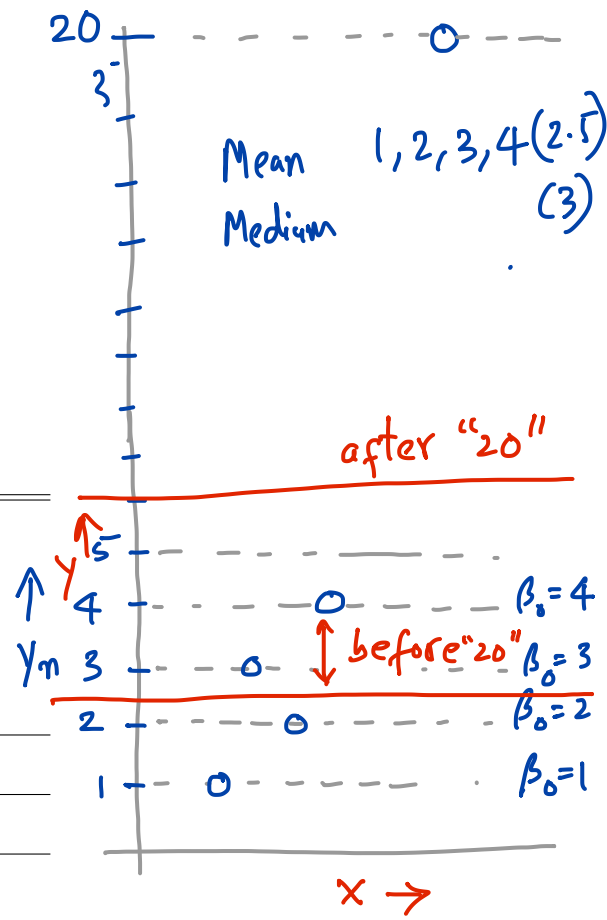
$$\mathcal{L}(\beta_0) := \frac{1}{2N}\sum_{n=1}^{N}[y_n - \beta_0]^2 \qquad (2)$$



Each row contains a y_n and column is β_0. First, compute MSE for for $y_n = \{1, 2, 3, 4\}$ and draw MSE as a function of β_0 (by adding the first four rows). Then add $y_n = 20$ to it, and redraw MSE. What do you observe and why?

$\beta_0 \rightarrow$

$y_n \backslash \beta_0$	1	2	3	4	5	6	7
e_1 1	0	1	2	3	4		
e_2 2	1	0	1	2	3		
e_3 3	2	1	0	1	2		
e_4 4	3	2	1	0	1		
MSE	14	6	6	14	30		
20	19	18	17	16	15		
MSE	.	.	.	.	.		



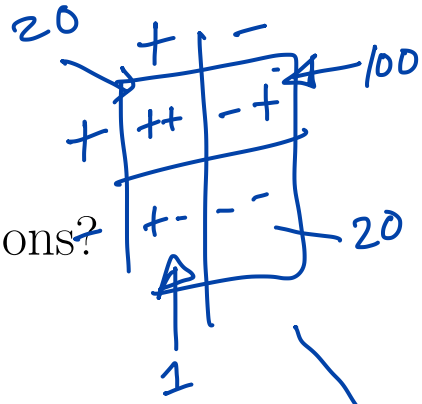
it may not be "bad", scale of data

Tukey

Additional Notes

A question for cost functions

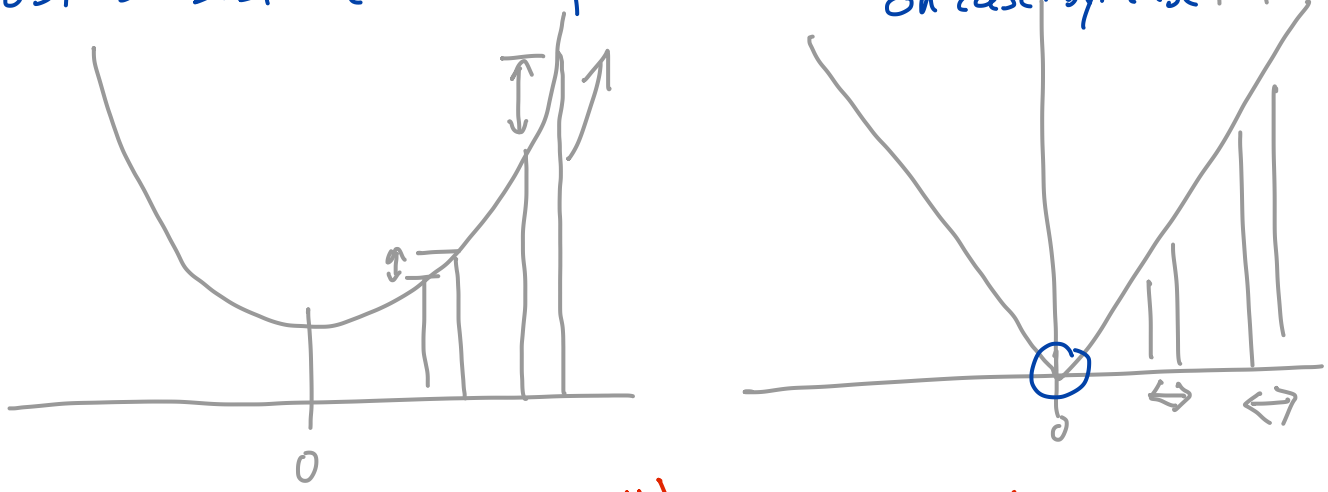
Is there an automatic way to define loss functions?



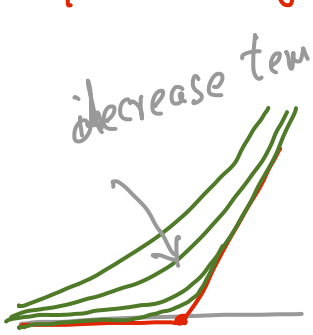
Nasty cost functions: Visualization

See Andrej Karpathy Tumblr post for many cost functions gone "wrong" for neural network. http://lossfunctions.tumblr.com/.

"Cost-sensisitive" loss/Cost functions: Need to design them on case-by-case basis.

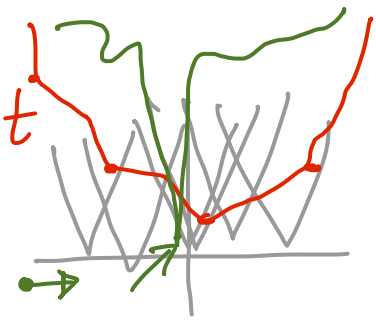


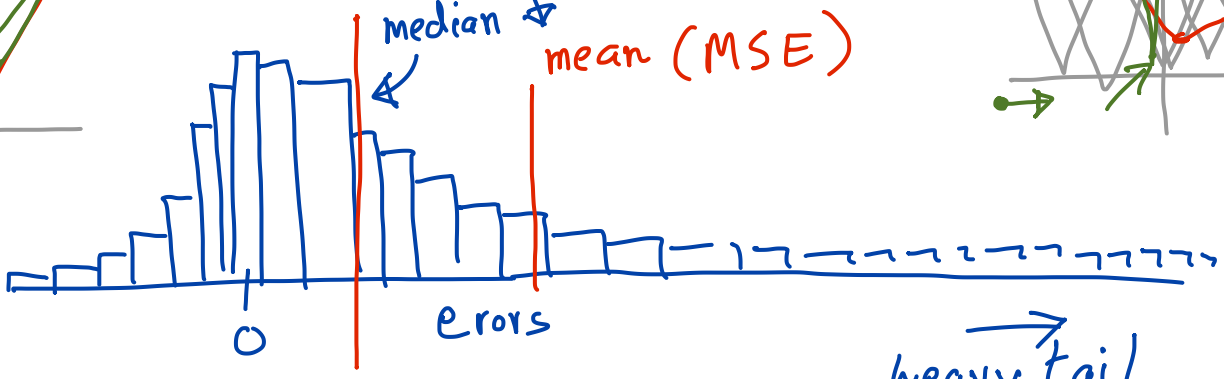
$\beta_0 = \arg\min \sum_{n=1}^{N} e_n^2 = \underbrace{\text{mean}}_{\text{mean or average}}\begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$ $\beta_0 = \arg\min \sum_{n=1}^{N} |e_n| = \text{median}\begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$



Less Robust

More Robust

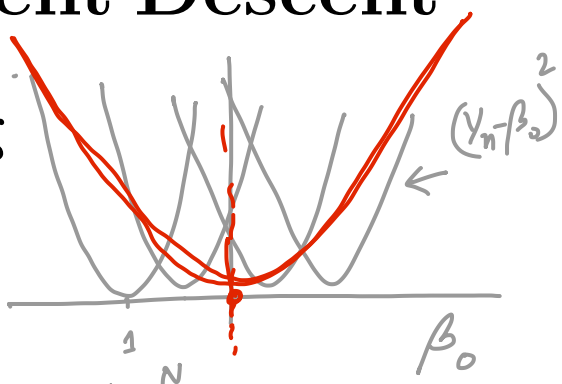




Desirable Properties of cost function: Continuous, differentiable (twice), Convex (strictly, strongly)

$|e|$ → e^2 MSE

5 Algorithm 1: Gradient Descent



Learning/estimation/fitting

Given a cost function $\mathcal{L}(\boldsymbol{\beta})$, we wish to find $\boldsymbol{\beta}^*$ that minimizes the cost:

$$\min_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}), \qquad \text{subject to } \boldsymbol{\beta} \in \mathbb{R}^{D+1}$$

$$\mathcal{L}(\beta_0) = \frac{1}{2N}\sum_{n=1}^{N}(y_n-\beta_0)^2$$

$$\mathcal{L}(\beta_0,\beta_1) = \frac{1}{2N}\sum_{n=1}^{N}(y_n-\beta_0-\beta_1 x_{n1})^2$$

$$\mathcal{L}\left(\begin{pmatrix}\beta_0\\ \beta_1\\ \vdots\\ \beta_D\end{pmatrix}\right) = \frac{1}{2N}\sum_{n=1}^{N}(y_n-\underline{\tilde{x}}_n^T\underbrace{\underline{\beta}}_{D+1})^2$$

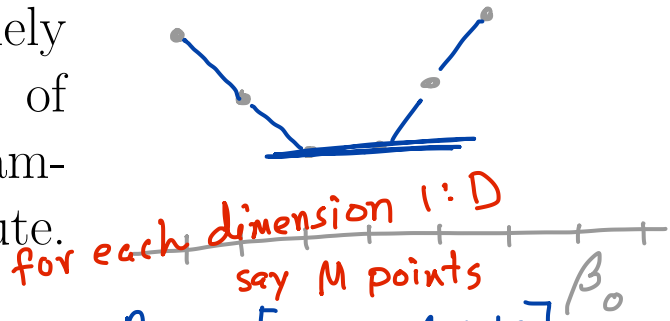
$$\mathcal{L}(\underline{\beta}) : \mathbb{R}^{D+1} \to \mathbb{R}$$

This is learning posed as an optimization problem. We will use an algorithm to solve the problem.

$$\underbrace{\begin{bmatrix}|&|&|&|\end{bmatrix}}_{\tilde{x}_n^T}\underbrace{\begin{bmatrix}\\ \\ \\ \\ \end{bmatrix}}_{\underline{\beta}} =$$

Grid search

Grid search is one of the simplest algorithms where we compute cost over a grid (of say M points) to find the minimum. This is extremely simple and works for any kind of loss when we have very few parameters and the loss is easy to compute.



for each dimension 1:D

say M points

```
β0 ← [-10 ; 0.1 : 10]
                1:M
for i = 1: length(β0)
    MSE[i] ← compute loss(β0, D)
end
i ← argmin(MSE)
```

For a large number of parameters, however, grid search has too many "for-loops" (1), resulting in exponential computational complexity. (2) Choosing a good range of values is another problem.

Q: What is the computational complexity of this algorithm?

$O(\underbrace{ND}_{\text{for each MSE[i]}} M^D)$ ← All the points

```
MSE ← 0
for n = 1:N
    e_n ← y_n - x̃_n^T β
    MSE = MSE + e_n^2
end
```

Are there any other issues?

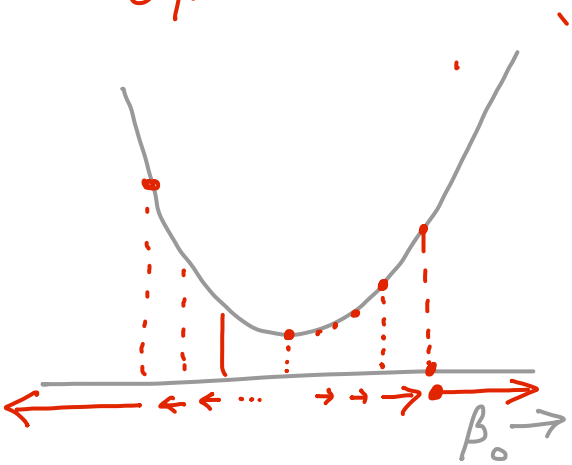
Look into Wikipedia (big-O notation)

$O(ND\ \cancel{M^D})$ Loss function is "smooth". (Lipschitz)

Follow the gradient

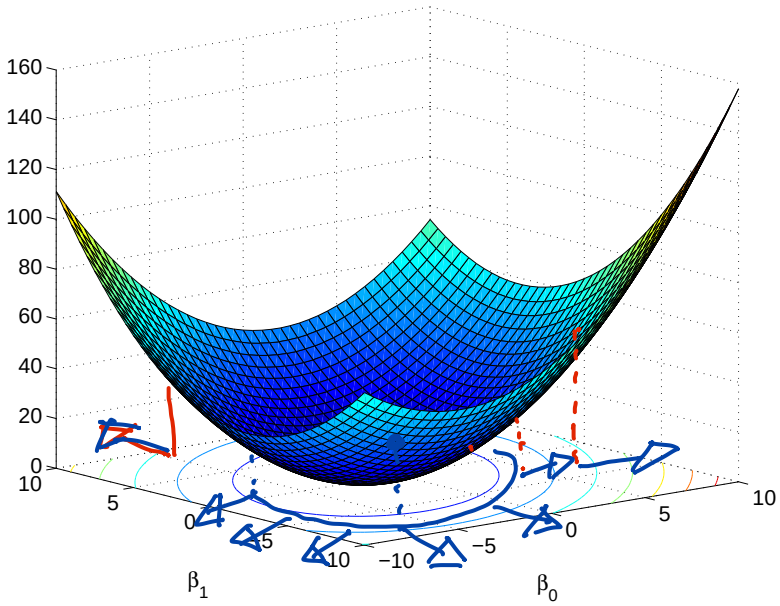
A gradient (at a point) is the slope of the tangent (at that point). It points to the direction of largest increase of the function.

$$\frac{\partial \mathcal{L}(\underline{\beta})}{\partial \underline{\beta}}$$



For 2-parameter model, MSE is shown below.

(I used $\mathbf{y}^T = [2, -1, 1.5]$ and $\mathbf{x}^T = [-1, 1, -1]$).



$$\frac{\partial \mathcal{L}(\beta_0)}{\partial \beta_0} = \frac{1}{2N}\frac{\partial}{\partial \beta_0}\sum_{n=1}^{N}(y_n - \beta_0)^2$$

$$= \frac{1}{2N}\sum_{n=1}^{N} -2(y_n - \beta_0)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\underbrace{(\beta_0 - y_n)}_{\text{error} = e_n}$$

$$\mathbb{R}^2 \ni \frac{\partial \mathcal{L}(\beta_0, \beta_1)}{\partial(\beta_0, \beta_1)} = \frac{\partial}{\partial \underline{\beta}}\frac{1}{2N}\sum_{n=1}^{N}(y_n - \beta_0 - \beta_1 x_{n1})^2$$

$$= \frac{1}{\cancel{2}N}\sum_{n=1}^{N}\begin{bmatrix} -\cancel{2}\,\overbrace{(y_n - \underbrace{\beta_0 - \beta_1 x_{n1}}_{\tilde{x}_n^T \underline{\beta}})}^{\text{error} = e_n} \\ -\cancel{2}\,(y_n - \beta_0 - \beta_1 x_{n1})\, x_{n1} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{n=1}^{N}(\tilde{x}_n^T \underline{\beta} - y_n) \\ \frac{1}{N}\sum_{n=1}^{N}\underbrace{(\tilde{x}_n^T \underline{\beta} - y_n)}_{\text{error (scalar)}}\, x_{n1} \end{bmatrix}$$

$$= \frac{1}{N}\sum_{n=1}^{N} e_n \begin{bmatrix} 1 \\ x_{n1} \end{bmatrix} = \frac{1}{N}\sum_{n=1}^{N} \underset{\text{error}}{e_n}\, \underset{\text{input}}{\tilde{x}_n}$$

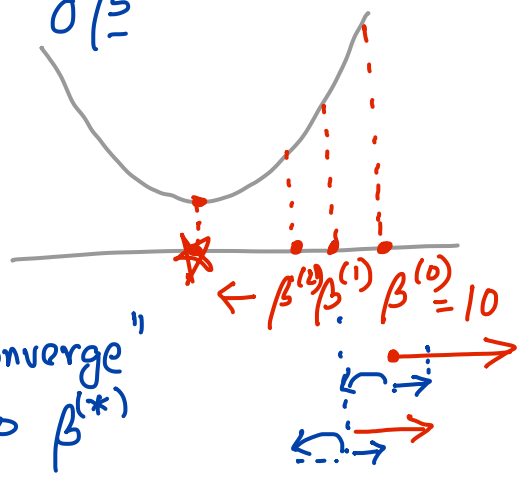
Batch gradient descent

To minimize the function, take a step in the (opposite) direction of the gradient

$$\underbrace{\boldsymbol{\beta}^{(k+1)}}_{\text{New}} \leftarrow \underbrace{\boldsymbol{\beta}^{(k)}}_{\text{current}} - \underline{\underline{\alpha}}\frac{\partial \mathcal{L}(\boldsymbol{\beta}^{(k)})}{\partial \boldsymbol{\beta}}$$

$$\left.\frac{\partial \mathcal{L}(\underline{\beta})}{\partial \underline{\beta}}\right|_{\underline{\beta} = \underline{\beta}^{(k)}} = \frac{\partial \mathcal{L}(\underline{\beta}^{(k)})}{\partial \underline{\beta}}$$

← k'th iteration



where $\alpha > 0$ is the step-size (or learning rate).

We want the sequence $\beta^{(0)}$ $\beta^{(1)}$ $\beta^{(2)}$ .... to "converge" to $\beta^{(*)}$

Gradient descent for 1-parameter model to minimize MSE:

$$\beta_0^{(k+1)} = (1-\alpha)\beta_0^{(k)} + \alpha\bar{y}$$

Q: Where $\bar{y} = \sum_n y_n/N$. When is this sequence guaranteed to converge?

(for what value of α?)

$$\frac{\partial \mathcal{L}(\beta_0)}{\partial \beta_0} = \frac{1}{N}\sum_{n=1}^{N}(\beta_0 - y_n)$$

$$= \beta_0 - \underbrace{\frac{1}{N}\sum_{n=1}^{N} y_n}_{\bar{y}}$$

$$= \beta_0 - \bar{y}$$

$$\beta_0^{(k+1)} \leftarrow \beta_0^{(k)} - \alpha(\beta_0^{(k)} - \bar{y})$$

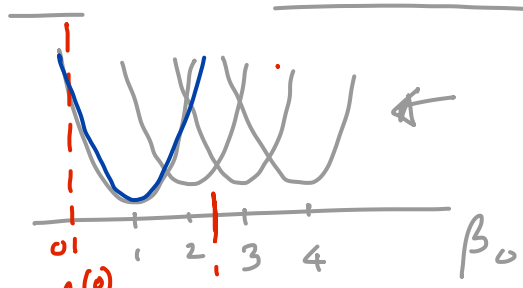
$$= (1-\alpha)\beta_0^{(k)} + \alpha\bar{y}$$

Gradients for MSE

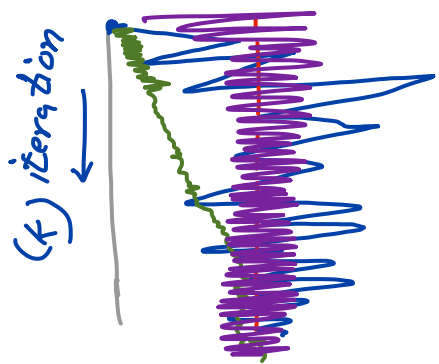
$$\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2N}\sum_{n=1}^{N}(y_n - \widetilde{\mathbf{x}}_n^T\boldsymbol{\beta})^2 \qquad (3)$$

then the gradient is given by,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = -\frac{1}{N}\sum_{n=1}^{N}\overbrace{(y_n - \widetilde{\mathbf{x}}_n^T\boldsymbol{\beta})}^{\text{error} = e_n}\widetilde{\mathbf{x}}_n \qquad (4)$$



$\beta_0^{(0)} = 0$



What is the computational complexity of batch gradient descent?

$O(NDM^D) \to O(ND) \to O(D)$

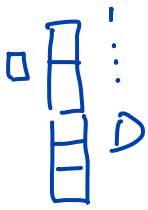
grid search GD SGD

Stochastic gradient descent

$$\frac{\partial \mathcal{L}}{\partial \beta} = \frac{1}{N}\sum_{n=1}^{N} e_n \underline{x}_n$$

$$= \frac{1}{N} N \times e_i \underline{x}_i$$

$$= e_i \underline{x}_i$$



Assumption: i.i.d. over n

When N is large, choose a random pair $(\mathbf{x}_i, y_i)$ in the training set and approximate the gradient:

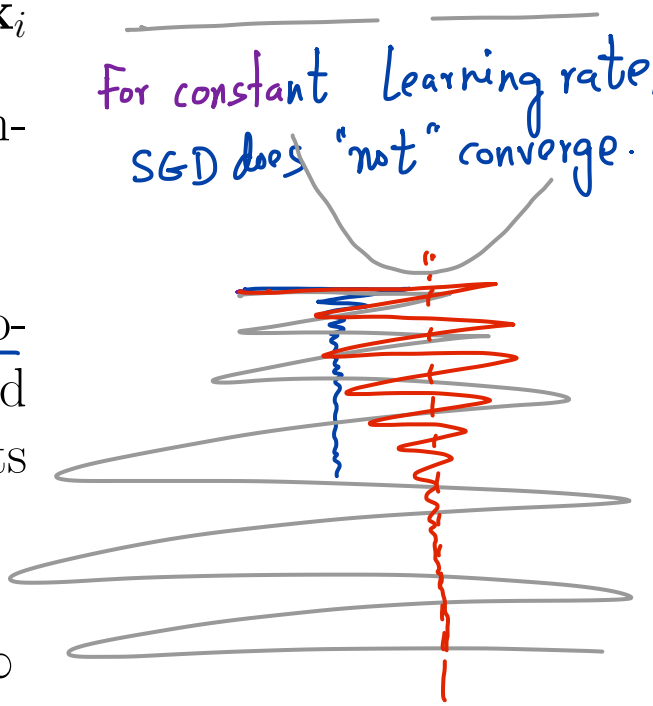
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} \approx -\frac{1}{N}\left[N(y_i - \widetilde{\mathbf{x}}_i^T \boldsymbol{\beta})\widetilde{\mathbf{x}}_i\right] \qquad (5)$$

Using the above "stochastic" gradient, take a step:

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \alpha^{(k)}(y_i - \widetilde{\mathbf{x}}_i^T \boldsymbol{\beta}^{(k)})\widetilde{\mathbf{x}}_i$$

What is the computational complexity?

For constant learning rate, SGD does "not" converge.



For convergence, $\alpha^k \to 0$ "appropriately". One such condition called Robbins-Monroe condition suggests to take α^k such that:

$$\sum_{k=1}^{\infty} \alpha^{(k)} = \infty, \qquad \sum_{k=1}^{\infty} (\alpha^{(k)})^2 < \infty \qquad (6)$$

$\alpha^{(k)} <$ constant.

$\sum \alpha^{(k)} = \infty$

$\sum (\alpha^{(k)})^2 < \infty$

One way to obtain such sequence is $\alpha^{(k)} = \boxed{1/(1+k)^r}$ where $r \in (0.5, 1)$.

$\left\{ \begin{array}{l} r = 2 \\ r = 0.4 \end{array} \right\}$

Data

$$\{\underline{x}_i, y_i\}_{i=1}^{N}$$

Model

$$f_{\underline{\beta}}(y_n) = \tilde{\underline{x}}_n^T \underline{\beta}$$

Algorithm

$$\min_{\underline{\beta}} \frac{1}{2N} \sum_{n=1}^{N} \left(y_n - \tilde{\underline{x}}_n^T \underline{\beta}\right)^2$$

$$\frac{\partial \mathcal{L}(\beta^{(k)})}{\partial \underline{\beta}} = \frac{1}{N} \sum_{n=1}^{N} \left(y_n - \underbrace{f_{\beta^{(k)}}(x_n)}_{\text{forward pass}}\right) \underline{x}_n$$

$$\begin{bmatrix} 1 \\ \underline{x}_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_0 \end{bmatrix}$$

$$\tilde{\underline{x}}_n^T \underline{\beta}^{(k)}$$

$$\downarrow$$

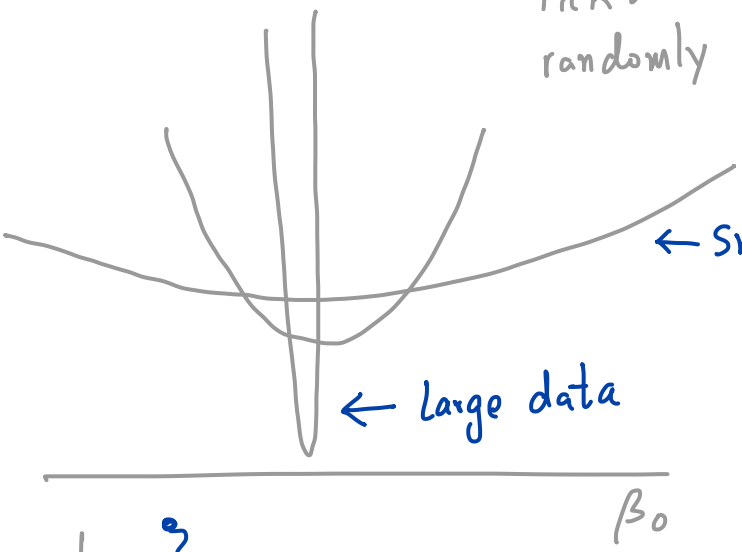
$$\approx e_i x_i$$

Pick i randomly

Compute error $\quad e_n$

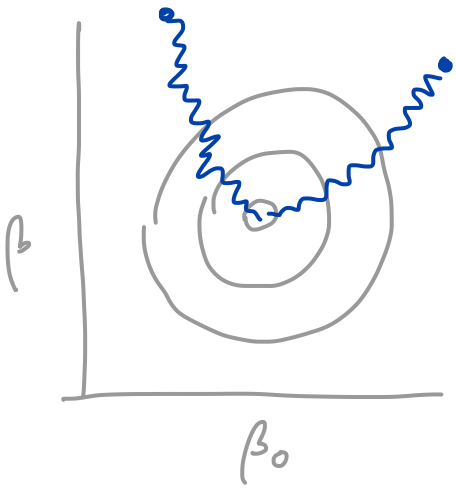
$$\downarrow$$

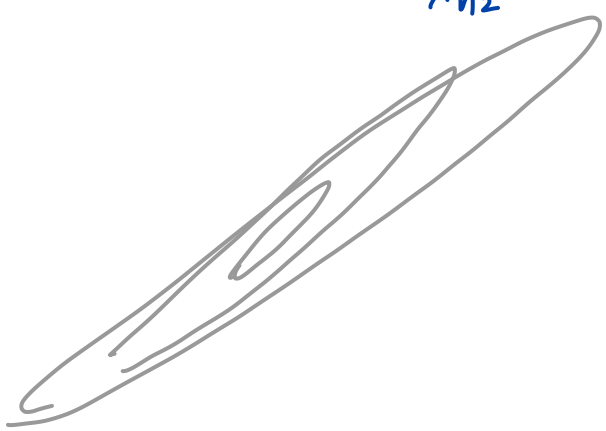
back pass $\quad e_n \underline{x}_i$



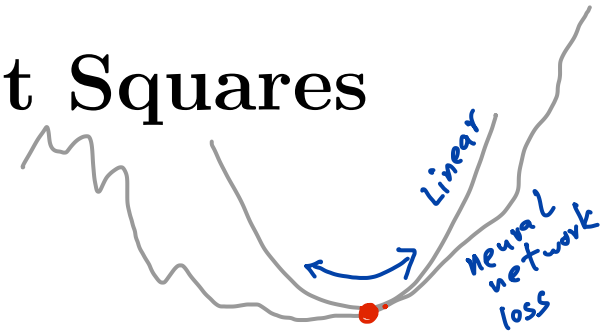
$$\underbrace{(\beta_1 + c\beta_2)}_{\beta_{new}} x_{n1} = \beta_1 x_{n1} + \beta_2 x_{n2}$$

$$x_{n2} = c\, x_{n1}$$





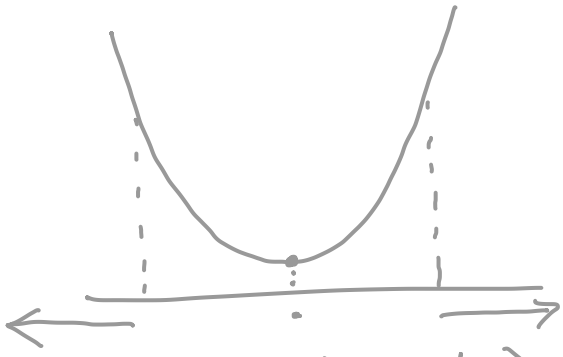
6 Algorithm 2: Least Squares



In rare cases, we can compute the minimum of the cost function analytically. Linear regression using MSE is one such case. The solution is obtained using normal equations. This is called least squares. (Gauss)

$$\frac{1}{N}\sum(Y_n - \underbrace{f_\beta(X_n)}_{\tilde{X}_n^T\beta})^2 \qquad \mathcal{L}(\beta)$$

To derive the equation, we use the optimality (stationary) conditions. See the lecture notes for Gradient Descent.



$$\frac{\partial \mathcal{L}(\boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = 0$$

$\beta^{(0)} \; \beta^{(1)} \; \dots \; \beta^*$

$\xrightarrow{\text{converges}}$

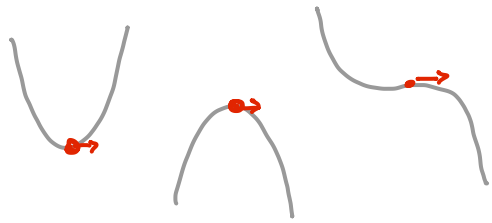
$$\lim_{\delta \to 0} \frac{\mathcal{L}(\beta_0+\delta) - \mathcal{L}(\beta_0)}{\delta}$$

Informal statement

When $\beta = \arg\min_\beta \mathcal{L}(\beta)$

then $\frac{\partial \mathcal{L}(\beta^*)}{\partial \beta} = 0$

(Opposite is not true)

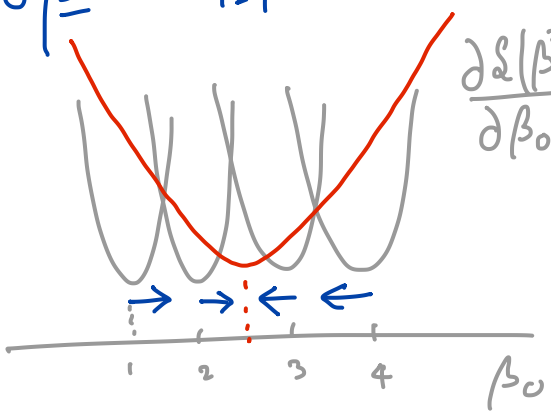


Using this, derive the normal equation for 1-parameter model.

$$\frac{\partial \mathcal{L}(\beta^*)}{\partial \beta} = \frac{1}{2N}\sum_{i=1}^{N} e_n \tilde{X}_n = 0$$

$$\frac{\partial \mathcal{L}(\beta_0^*)}{\partial \beta_0} = -\frac{1}{N}\sum e_n$$

$$e_n = -(Y_n - \beta_0)$$



$$\beta^{(k+1)} \leftarrow \beta^{(k)} - \alpha^{(k)}(Y_i - \beta_0^{(k)})$$

Normal equations

Recall the expression of the gradient for multiple linear regression:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = -\frac{1}{N}\widetilde{\mathbf{X}}^T\mathbf{e} = -\frac{1}{N}\widetilde{\mathbf{X}}^T(\mathbf{y} - \widetilde{\mathbf{X}}\boldsymbol{\beta})$$

Set it to zero to get the normal equations for linear regression.

$$\widetilde{\mathbf{X}}^T\mathbf{e} = \underline{\underline{\widetilde{\mathbf{X}}^T(\mathbf{y} - \widetilde{\mathbf{X}}\boldsymbol{\beta})}} = 0$$

implying that the error is orthogonal to rows of $\widetilde{\mathbf{X}}^T$ and columns of $\widetilde{\mathbf{X}}$.

$$\frac{\partial \mathcal{L}(\underline{\beta}^*)}{\partial \underline{\beta}} = \frac{1}{N}\sum_{n=1}^{N} e_n \underline{\tilde{x}}_n$$

$$= \frac{1}{N}\left(e_1 \underline{\tilde{x}}_1 + \cdots + e_N \underline{\tilde{x}}_N \right)$$

$$= \frac{1}{N}\underbrace{\begin{bmatrix} \underline{\tilde{x}}_1 & \underline{\tilde{x}}_2 & \cdots & \underline{\tilde{x}}_N \end{bmatrix}}_{(D+1)\times N} \underbrace{\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix}}_{N\times 1}$$

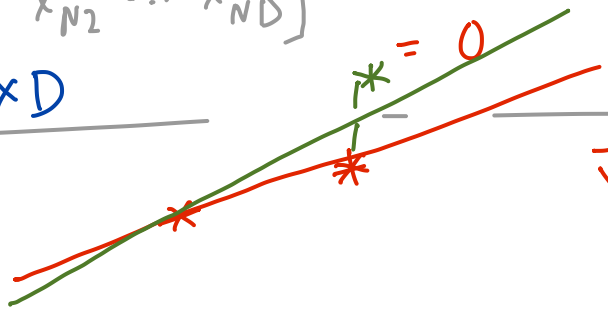
$$= \frac{1}{N}\widetilde{X}^T \underline{e}$$

$$= \frac{1}{N}\widetilde{X}^T(\widetilde{X}\underline{\beta}^* - \underline{y})$$

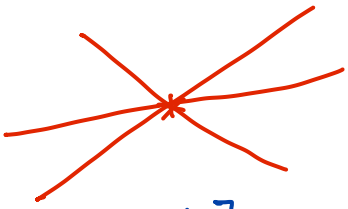
$$= 0$$

$$\widetilde{X} = \begin{bmatrix} \underline{\tilde{x}}_1^T \\ \vdots \\ \underline{\tilde{x}}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1D} \\ 1 & x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

$N \times D$



$$\bar{y} = \widetilde{X}\underline{\beta}$$



$$e_1 \begin{bmatrix} 1 \\ x_{11} \end{bmatrix} = 0$$

$$\underbrace{(\beta_0 + \beta_1 x_{11} - y_1)}_{e_1 = 0} \begin{bmatrix} 1 \\ x_{11} \end{bmatrix} = 0$$

$$e_1 \begin{bmatrix} 1 \\ x_{11} \end{bmatrix} + e_2 \begin{bmatrix} 1 \\ x_{21} \end{bmatrix} = 0$$

$$+ e_3 \begin{bmatrix} 1 \\ x_{31} \end{bmatrix} = 0$$

$$\widetilde{X} = \begin{array}{c} \begin{bmatrix} 1 & x_{11} \\ 1 & x_{21} \\ 1 & x_{31} \end{bmatrix} \end{array} \begin{array}{l} \underline{\tilde{x}}_1 \\ \underline{\tilde{x}}_2 \\ \underline{\tilde{x}}_3 \end{array}$$

$\underline{x}_1$

concept of rank

↓

Maximum "degree of freedom"

$\sum_{n=1}^{N} e_n \tilde{\underline{x}}_n = 0 \Rightarrow \tilde{X}^T(\tilde{X}\beta^* - \underline{y})$ } error

$\tilde{X}\beta^*$ → linear combination of $\underline{x}_d$

$\boxed{\tilde{X}^T \underline{e} = 0}$

Geometric Interpretation

Denote the d'th column of $\widetilde{\mathbf{X}}$ by $\bar{\mathbf{x}}_d$.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \widetilde{\mathbf{X}} = \begin{array}{c} \begin{array}{ccccc} & \underline{x}_1 & \underline{x}_2 & \cdots & \underline{x}_D \end{array} \\ \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix} \end{array} \begin{array}{c} \tilde{\underline{x}}_1^T \\ \tilde{\underline{x}}_2^T \\ \vdots \\ \tilde{\underline{x}}_N^T \end{array}$$

$N \times D$

The normal equations suggest to $\boxed{\text{choose a vector}}$ in the span of $\widetilde{\mathbf{X}}$. The following figure illustrates this (taken from Bishop's book).

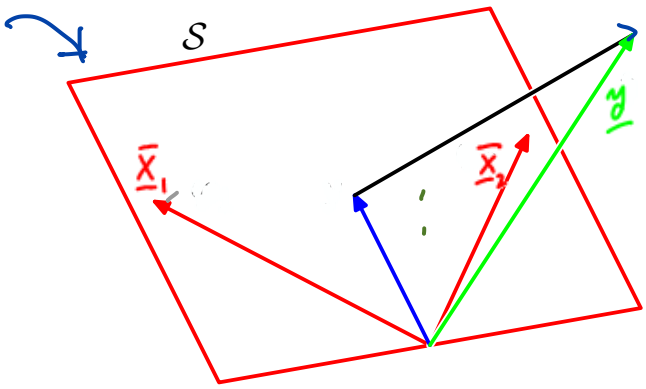
$$\underline{e} = \tilde{X}\underline{\beta} - \underline{y}$$

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} \begin{array}{l} \leftarrow \tilde{\underline{x}}_1^T\underline{\beta} - y_1 \\ \leftarrow \tilde{\underline{x}}_2^T\underline{\beta} - y_2 \\ \\ \leftarrow \tilde{\underline{x}}_N^T\underline{\beta} - y_N \end{array}$$

$N \times 1 \quad \underbrace{N \times D \quad D \times 1}_{\tilde{X} \quad \underline{\beta}} \quad N \times 1$

$N \times 1$

space of all vectors obtained with a linear combination of $\underline{x}_1 \dots \underline{x}_D$



$\underline{y} - \tilde{X}\underline{\beta}$

$$\tilde{X}\underline{\beta} = \begin{bmatrix} \underline{1} & \underline{x}_1 & \dots & \underline{x}_D \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_D \end{bmatrix}$$

$$= \underline{1}\beta_0 + \beta_1 \underline{x}_1 + \cdots + \beta_D \underline{x}_D$$

Least-squares

When $\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}}$ is invertible, we have a closed-form expression for the minimum.

$$\tilde{X}^T(\tilde{X}\underline{\beta}^* - \underline{y}) = 0$$

$$\boldsymbol{\beta}^* = (\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^T\mathbf{y}$$

$$\Rightarrow \underbrace{(\tilde{X}^T\tilde{X})}_{\text{Gram matrix}}\underline{\beta}^* = \tilde{X}^T\underline{y}$$

assume that it's invertible

We can predict values for a new $\mathbf{x}_*$.

$$\hat{y}_* = \underline{\tilde{\mathbf{x}}_*^T\boldsymbol{\beta}^*} = \tilde{\mathbf{x}}_*^T(\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^T\mathbf{y}$$

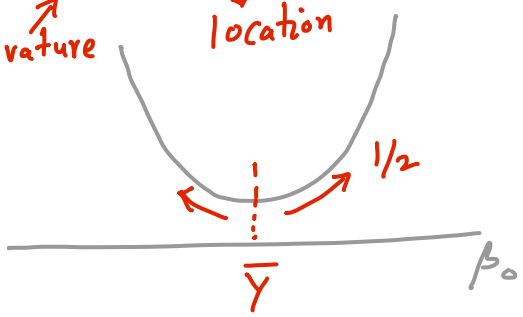
$$\frac{1}{2N}\sum_{i=1}^{N}(y_i-\beta_0)^2 = \frac{1}{2N}\left(\underbrace{\sum_{i=1}^{N}\beta_0^2}-2\sum_{i=1}^{N}y_i\beta_0 \cancel{+\sum_{i=1}^{N}y_i^2}\right)$$

$$= \frac{1}{2}\beta_0^2 - \bar{y}\,\beta_0 + \text{cnst} \qquad \bar{y}=\frac{1}{N}\sum_{i=1}^{N}y_i$$

curvature location

$$\beta_0^* = \underset{\beta_0}{\operatorname{argmin}} \quad ''$$

$$\beta_0^* = \bar{y}$$



$$\frac{1}{2N}\sum_{i=1}^{N}(y_i-\underline{x}_i^T\underline{\beta})^2 = \frac{1}{2}\underline{\beta}^T H\,\underline{\beta} - \underline{b}^T\underline{\beta} \qquad H\underline{\beta}=\underline{b} \qquad \underline{\beta}=H^{-1}\underline{b}$$

$$H=\frac{1}{N}\tilde{X}^T\tilde{X}, \quad \underline{b}=\tilde{X}^T\underline{y}$$

curvature location

$$\frac{1}{N}\sum_{i=1}^{N}\tilde{\underline{x}}_i\tilde{\underline{x}}_i^T \qquad \sum_{i=1}^{N}\tilde{\underline{x}}_i y_i$$

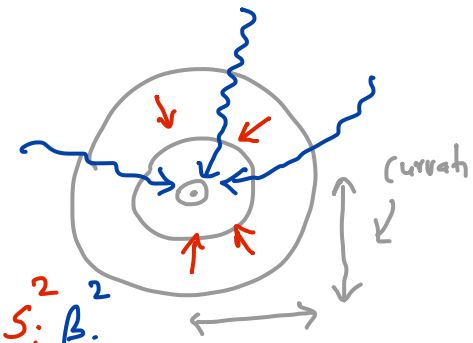
D×D D×1

"well-conditioned"

Isotropic + Diagonal

$$H=I \Rightarrow \underline{\beta}^T H\underline{\beta} = \sum_{j=1}^{D}\beta_j^2 \qquad \beta_1^2+\beta_2^2$$

$$\sum_{i=1}^{10,000}\underset{\tilde{\underline{x}}_i}{\begin{bmatrix}\cdot\\ \cdot\end{bmatrix}}\begin{bmatrix}\cdot & \cdot\end{bmatrix} = \tilde{X}^T\tilde{X} = \begin{bmatrix}1&0\\0&1\end{bmatrix}$$



ill-conditioned

Diagonal

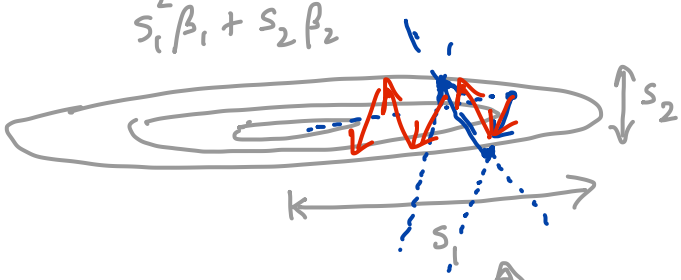
off-diagonal

$$H=\begin{bmatrix}s_1^2 & & 0\\ & \ddots & \\ 0 & & s_D^2\end{bmatrix} \Rightarrow \underline{\beta}^T H\underline{\beta} = \sum_{j=1}^{D} s_j^2\beta_j^2$$

$$\begin{bmatrix}s_1&0\\0&s_2\end{bmatrix}\underline{\beta}^{(k)} = \begin{bmatrix}s_1\beta_1^{(k)}\\ s_2\beta_2^{(k)}\end{bmatrix} \qquad \tilde{X}^T\tilde{X} = \begin{bmatrix}s_1^2&0\\0&s_2^2\end{bmatrix}$$

say $s_p^2 \gg s_2^2$

$$s_1^2\beta_1^2+s_2^2\beta_2^2$$

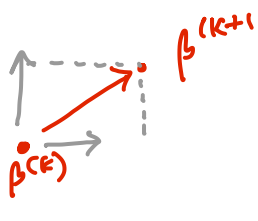


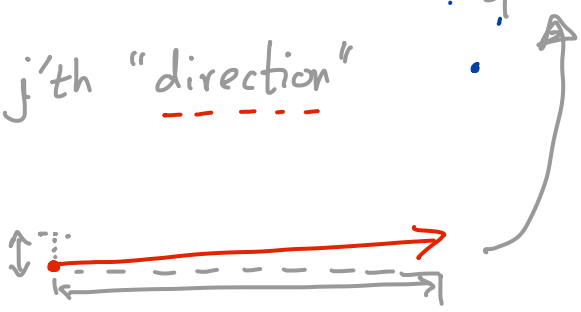
Ratio: $\frac{s_1}{s_2}$ is really large.

s_j : largest "move" one can makes in j'th "direction"

$$\underline{\beta}^{(k+1)} \leftarrow H\underline{\beta}^{(k)}$$

Read "condition-number" wikipedia





General:

$$\underset{D\times D}{\tilde{X}^T\tilde{X}} \xrightarrow{\text{diagonalization}} \underset{D\times K}{\overset{U}{\begin{bmatrix} \uparrow & & \uparrow \\ \underline{u}_1 & \cdots & \underline{u}_K \\ \downarrow & \cdots & \downarrow \end{bmatrix}}} \underset{\text{singular values}}{\overset{S^2}{\begin{bmatrix} s_1^2 & & 0 \\ & s_2^2 & \\ 0 & & \ddots \; s_K^2 \end{bmatrix}}} \underset{K\times D}{\overset{U^T}{\begin{bmatrix} \leftarrow & \underline{u}_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & \underline{u}_K^T & \rightarrow \end{bmatrix}}}$$

singular vector

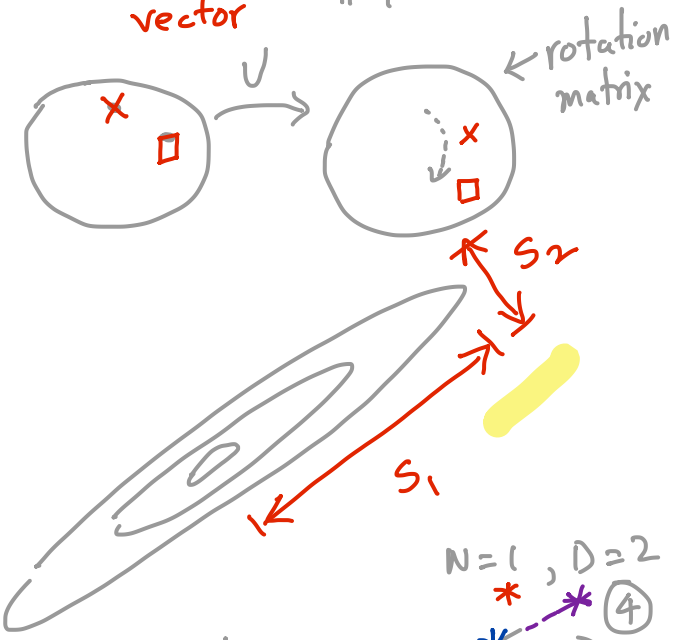
$$\|u_i\|^2 = 1$$

$$\underset{D\times N}{\tilde{X}^T} = \underset{D\times K}{U}\,\underset{K\times K}{S}\,\underset{K\times N}{V^T} \quad (\text{SVD})$$

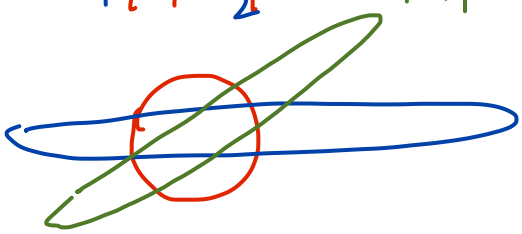
$$V^TV = I_K$$

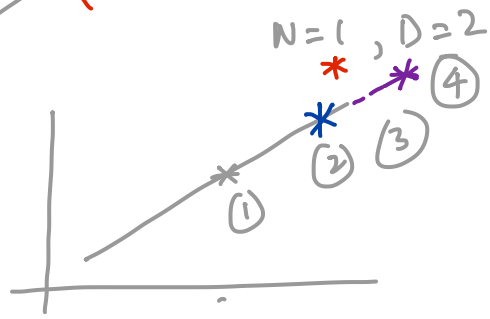
$$\tilde{X}^T\tilde{X} = (USV^T)(VSU^T) = US^2U^T$$

Read Wikipedia



$$s_1^2\beta_1^2 + s_2^2\beta_2^2 + \beta_1\beta_2$$





Condition number

$$\frac{s_{max}}{s_{min}}$$

minimum singular value is 0

① $\underset{2\times 1}{\tilde{X}^T} = \begin{bmatrix} \cdot \\ \cdot \end{bmatrix} = \underset{2\times 1}{\begin{bmatrix} \cdot \\ \cdot \end{bmatrix}}\,\underset{1\times 1}{\begin{bmatrix} \cdot & 0 \end{bmatrix}}\,\underset{1\times 1}{\begin{bmatrix} \cdot \\ \cdot \end{bmatrix}}$

D N

$s_{max} > 0$ $\quad K=1$

$s_{min} = 0$ $\quad D=2$

② $\underset{2\times 2}{\tilde{X}^T} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}\begin{bmatrix} \square & \\ & \square \end{bmatrix}\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$

③ $\underset{2\times 3}{\tilde{X}^T} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}\begin{bmatrix} \square & \\ & \square \end{bmatrix}\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$ 3rd

Learning rate:

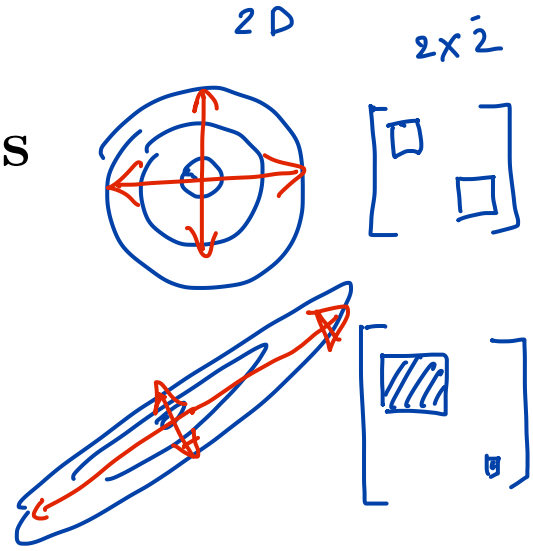
$$\underline{\beta}^{(k+1)} \leftarrow \underline{\beta}^{(k)} - \alpha^{(k)} X^T(X\underline{\beta}^{(k)} - \underline{Y})$$

$$\underbrace{\alpha^{(k)}(X^TX)}_{\text{effective learning rate}}\beta^{(k)}$$

different dimensions "move" differently!

α is constrained by s_{max} & s_{min}

Invertibility and uniqueness



The Gram matrix $\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}}$ is invertible iff $\widetilde{\mathbf{X}}$ has full column rank.

Proof: Assume $N > D$. The fundamental theorem of linear algebra states that the dimensionality of null space is zero for full column rank. This implies that the Gram matrix is positive definite, which implies invertibility.

Rank deficiency and ill-conditioning

Unfortunately, $\widetilde{\mathbf{X}}$ could often be rank deficient in practice, e.g. when $D > N$, or when the columns $\bar{\mathbf{x}}_d$ are (nearly) collinear. In the later case, the matrix is ill-conditioned, leading to numerical issues.

Summary of linear regression

We have studied three methods:

1. Grid search
2. (Stochastic) gradient descent
3. Least squares

$(\underline{x}_1 - \bar{\underline{x}}_1)$ $\quad \underline{x}_1 = \frac{\underline{x}_1}{\|\underline{x}_1\|}$ $\quad$ Normalization: $(X^T X)_{ij} = \underline{x}_i^T \underline{x}_j$

mean

Additional Notes

$D \times 1$ $\quad \begin{bmatrix} x_{1i} \\ \vdots \\ x_{Ni} \end{bmatrix}$ $N \times 1$

$X^T X = \begin{bmatrix} \|\underline{x}_1\|^2 & & (\underline{x}_i^T \underline{x}_j) \\ & \ddots & \\ \ddots & & \|x_D\|^2 \end{bmatrix}$ $\quad \sum_{i=1}^{N} \tilde{\underline{x}}_i \tilde{\underline{x}}_i^T$

$[1, x_{n1}, x_{n2} \ldots x_{nd}]^T$ $D \times 1$

Implementation

There are many ways to implement matrix inversion, but using QR decomposition is one of the most robust ways. Matlab's backslash operator implements this (and much more) in just one line.

```
beta = inv(X'*X) * (X'*y)
beta = pinv(X'*X) * (X'*y)
beta = (X'*X) \ (X'*y)
```

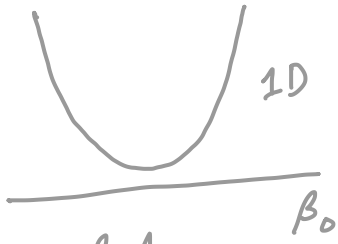
For robust implementation, see Sec. 7.5.2 of Kevin Murphy's book.

To do

$K = D$ or N (none of S_j are 0)

1. Revise linear algebra to understand why $\widetilde{\mathbf{X}}$ needs to have full rank. Read the Wikipedia page on rank of a matrix. $S_j \neq 0 \; \forall j$ → infinite sol^n
2. For details on the geometrical interpretation, see Bishop 3.1.2. However, better to read this after the lecture on "basis-function expansion". Also, note that notation in the book is different. This might make the reading difficult.
3. Understand matrix inversion robust implementation and play with it using the code for labs. Read Kevin Murphy's section 7.5.2 for details.
4. Understand ill-conditioning. Reading about the "condition number" in Wikipedia will help. Also, understanding SVD is essential. Here is another link provided by Dana Kianfar (EPFL) http://www.cs.uleth.ca/~holzmann/notes/illconditioned.pdf. Read this note
5. Work out the computational complexity of least-squares (use the Wikipedia page on computational complexity).

7 Challenge: Overfitting



Motivation

Linear model can be easily modified to obtain more powerful non-linear model. We can use basis function expansion to get a non-linear regression model, and then use a sequence of these models to construct a deep model.

$y_n \approx \beta_0 \cdot 1$

$y_n \approx \beta_0 \cdot 1 + \beta_1 \cdot x_n$

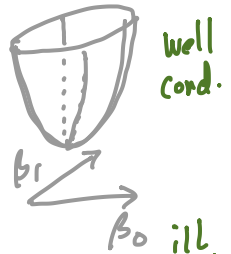
suppose $x_n = c$, a constant

we expect in 2D

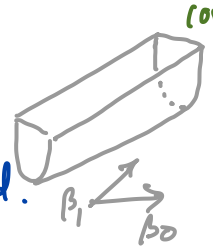
$y_n \approx \beta_0 \cdot 1 + \beta_1 c$

$= \beta_0^{new} \cdot 1$

where $\beta_0^{new} = \beta_0 + c\beta_1$



Adding more data "may" make the loss ill-conditioned.



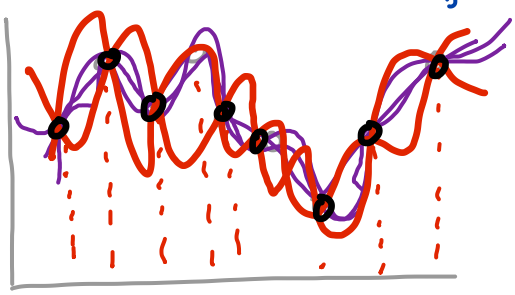
↳ interplay bet^n N & D, but what matters is K (rank) of X.

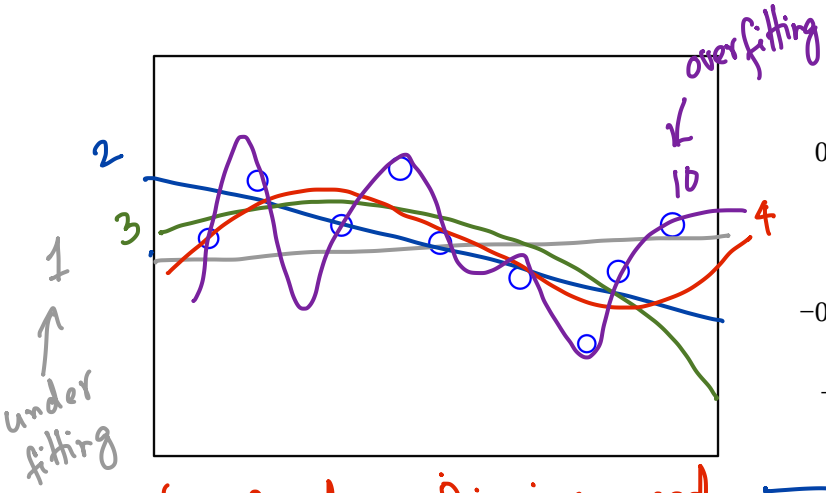
Consider simple linear regression. Given one-dimensional input x_n, we can generate a polynomial basis.

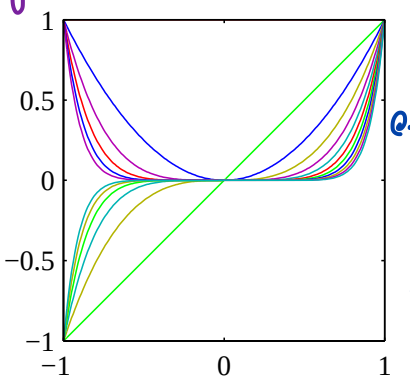
$$\boldsymbol{\phi}(x_n) = [1,\ \underline{x_n},\ x_n^2,\ x_n^3,\ \ldots,\ x_n^M]$$

Then we fit a linear model using the original *and* the generated features:

$$y_n \approx \beta_0 + \beta_1 x_n + \beta_2 x_n^2 + \ldots + \beta_M x_n^M$$







Infinite sol^n

↓

e.g. Some parameters (or their combinations)

In linear model whenever

$K < \min(N, D)$

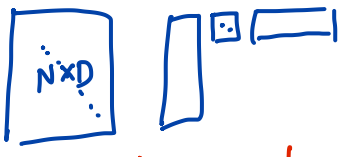
↓

there are some zero singular values!

$S_j = 0,\ j > K+1 \ldots$

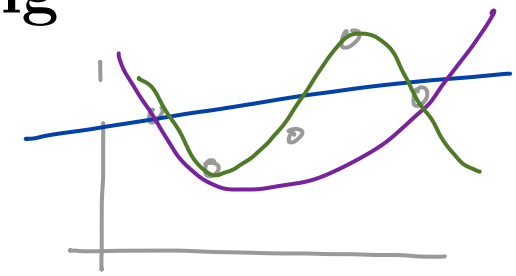
N is fixed, D is increased

Is that good?



Simplest fix: Process data, feature extraction/engineering.

Overfitting and Underfitting



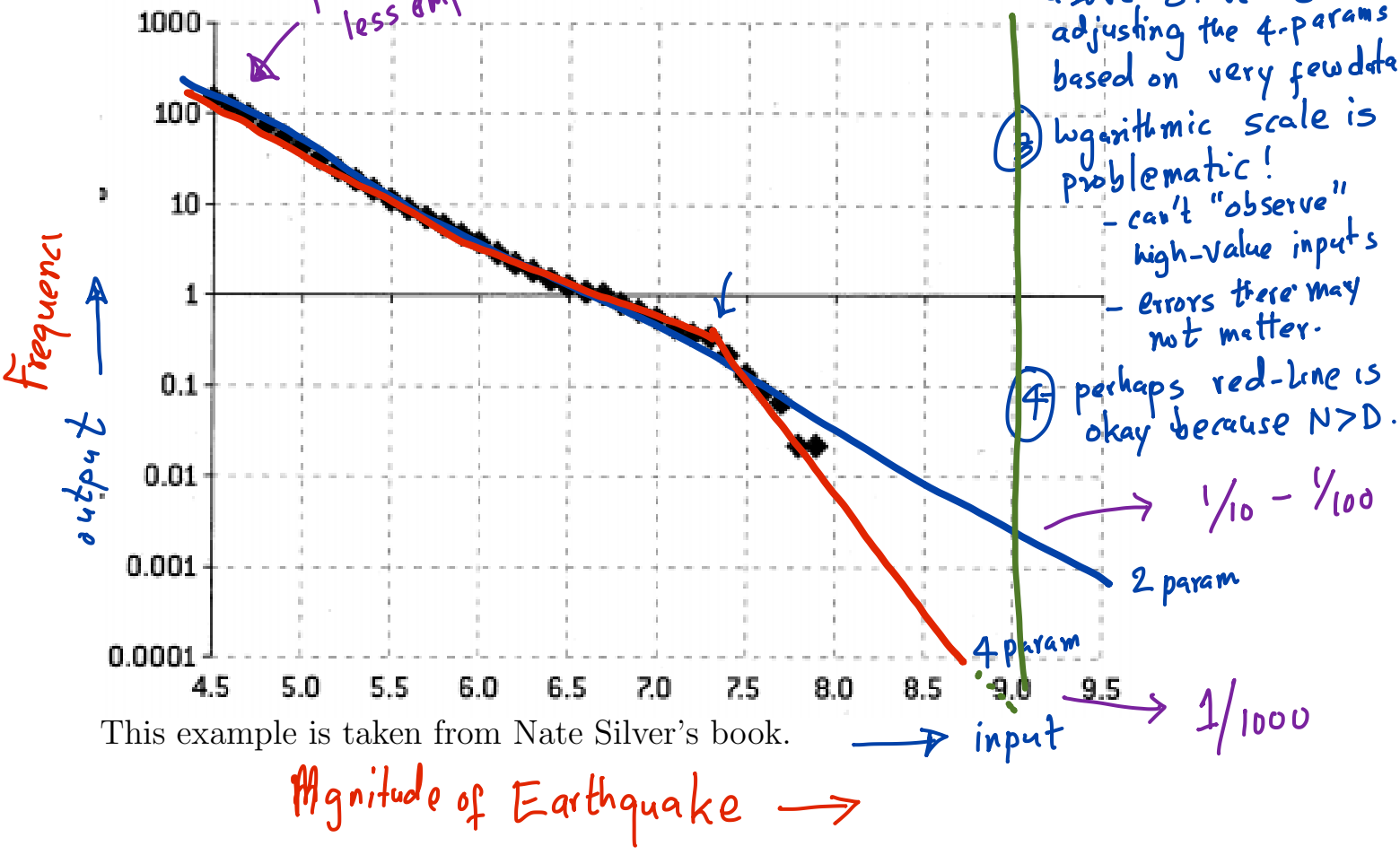
Overfitting is fitting the noise in addition to the signal. Underfitting is not fitting the signal well. In reality, it is very difficult to be able to tell the signal from the noise.

Which is a better fit?

Try a real situation. Below, y-axis is the frequency of an event and x-axis is the magnitude. It is clear that as magnitude increases, frequency decreases.

Tohoku

prediction is less important



① Cautious because don't know the data

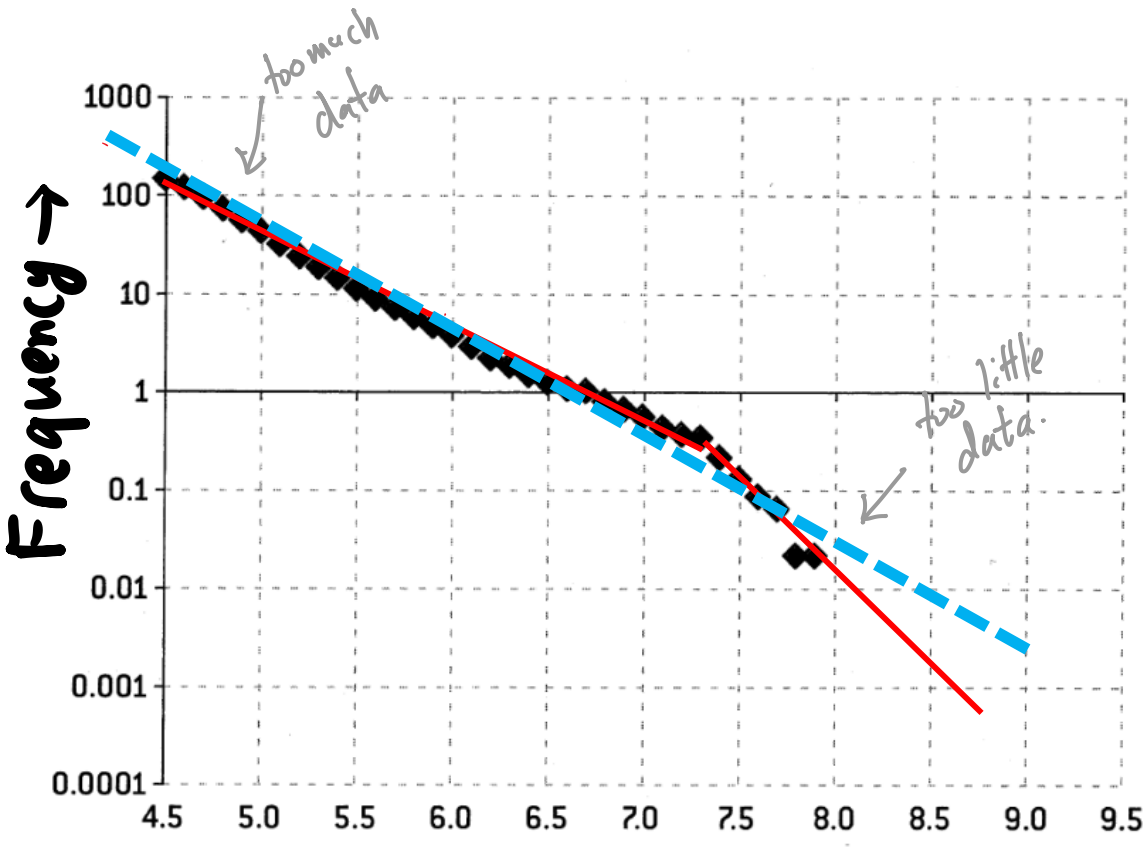
② Don't have much data above 8. We are adjusting the 4-params based on very few data

③ Logarithmic scale is problematic!
- can't "observe" high-value inputs
- errors there may not matter.

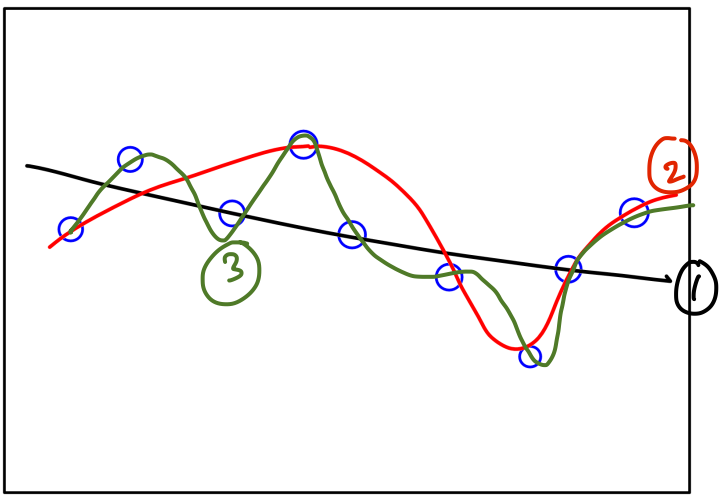
④ perhaps red-line is okay because N>D.

This example is taken from Nate Silver's book.

Which model is a better fit? blue or red?



Another example: Which model is a better fit? black or red? Data is denoted by circle.



MSE over "data in hand"

1	0.72
2	0.2
3	0

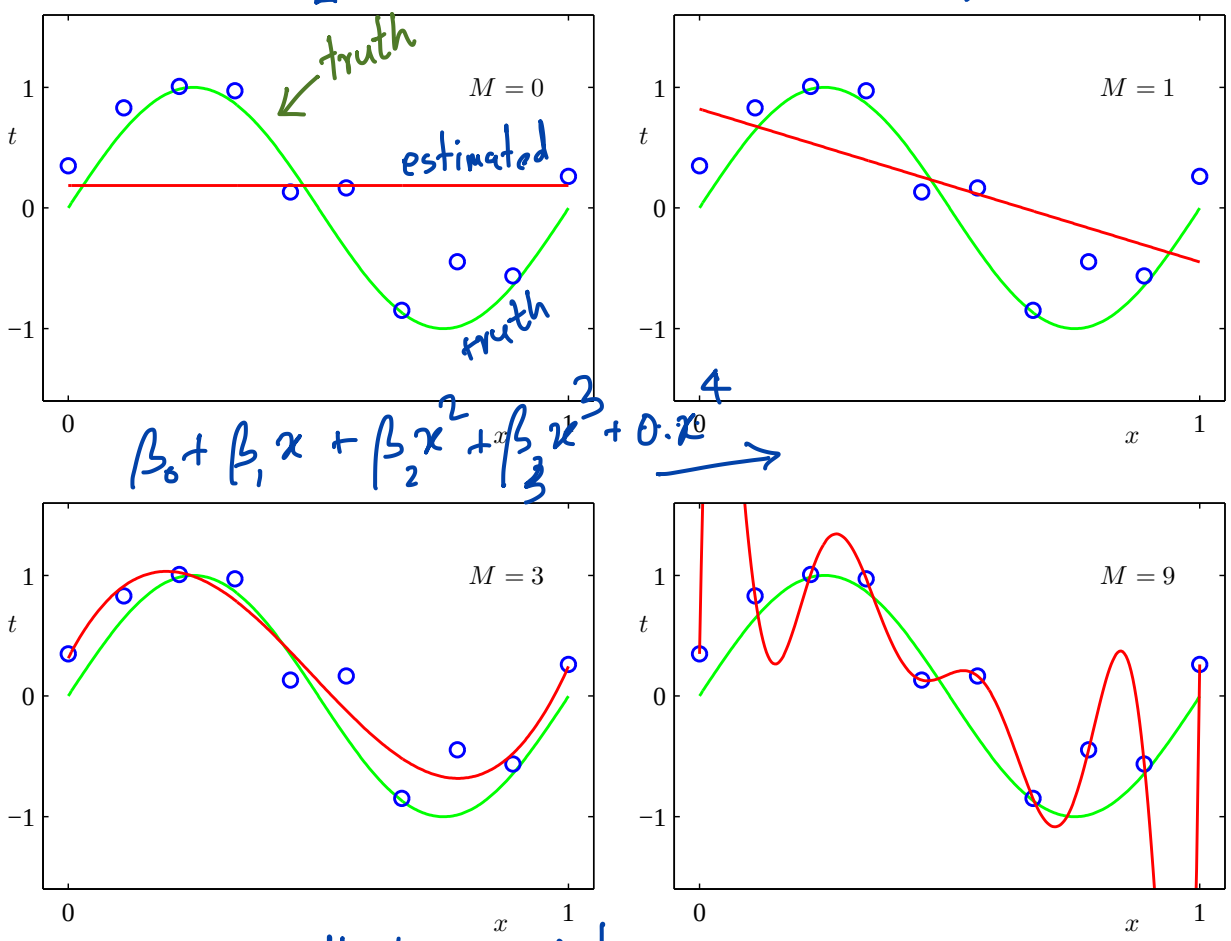
This is not the right criteria to look at.

Complex models overfit easily (see Bishop)

Circles are data points, green line is the truth & red line is the model fit. M is the maximum degree in the generated polynomial basis.

$\beta_0 + 0\cdot x + \cdots 0\cdot x^9$

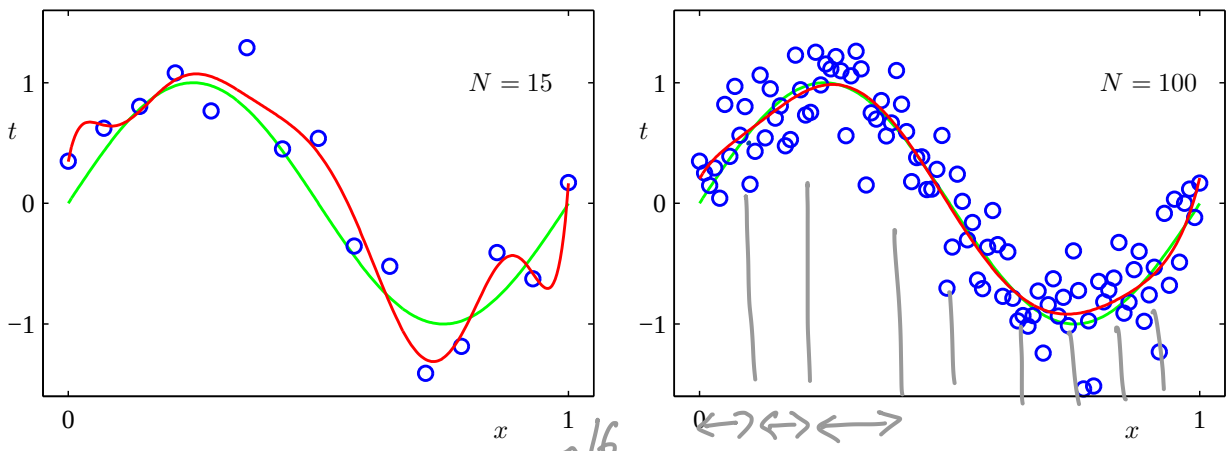
$\beta_0 + \beta_1 x + 0x^2 + \cdots 0\cdot x^9$

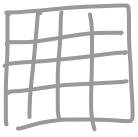


$\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + 0\cdot x^4$

When we can collect the data,

If you increase the amount of data, overfitting *might* reduce.





$4\times 4 = 16 \leftarrow \boxed{256}$ 16

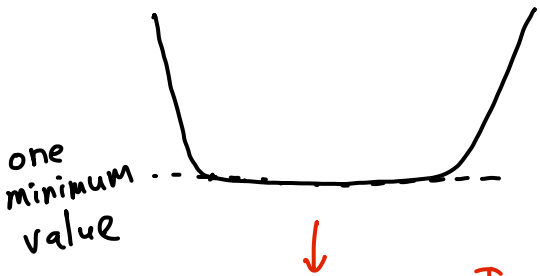
Occam's razor

One solution is dictated by Occam's razor which states that "Simpler models are better – in absence of certainty."

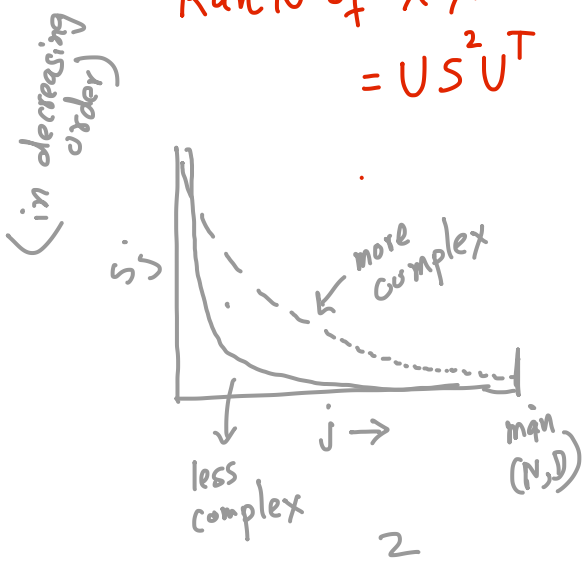
Sometimes, if you increase the amount of data, you might reduce overfitting. But, when unsure, choose a simple model over a complicated one.

Complexity

Linear model
+
MSE
+
any algorithm



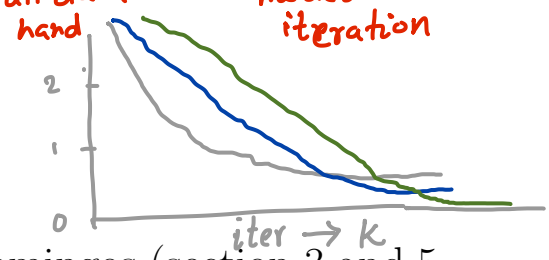
Rank of $X^T X = U S^2 U^T$



$$MSE_{tr} = \sum_{i \in D_{tr}} \left(y_i - \tilde{x}_i^T \beta^{(k)} \right)^2 \quad \forall i \text{ in data}$$

all data in hand

model at k'th iteration

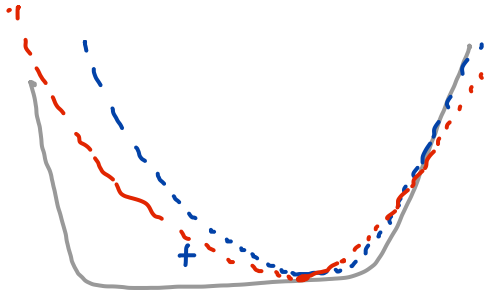


Additional Notes

Read about overfitting in the paper by Pedro Domingos (section 3 and 5 of "A few useful things to know about machine learning"). You can also read Nate Silver's book on "The signal and the noise" (the earthquake example is taken from this book).

Complex model → ill-conditioning → Overfitting

8 Solutions: Regularization



What is regularization?

Through regularization, we can penalize complex models and favor simpler ones:

$$\min_{\beta} \quad \underbrace{\mathcal{L}(\boldsymbol{\beta})}_{\text{MSE}} + \underbrace{\frac{\lambda}{2N}\sum_{j=1}^{M}\beta_j^2}_{\text{Regularizer}} \quad \text{— (A)}$$

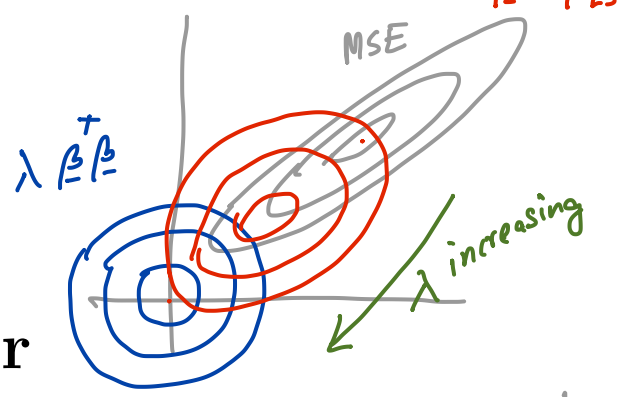
$$\sum_{i=1}^{N}\left(y_i - \beta_0 - \beta_1 x_{i1}\right)^2 + \lambda\left(\beta_0^2 + \beta_1^2\right)$$

Ridge Regression

Q1: What happens, $\lambda \to \infty$ $\quad \underline{\beta}^* = 0$

Q2: " " , $\lambda \to 0$ $\quad \underline{\beta}^* = \underline{\beta}^*_{LS}$

The second term is a regularizer (with $\lambda > 0$). The main point here is that an input variable weighted by a small β_j will have less influence on the output.



Regularization Parameter

The parameter λ can be tuned to reduce overfitting. But, how do you choose λ?

Write the SVD of $X = USV^T$

Q: Write an expression for the minimizer of (A)

$$\underline{\beta}^*(\lambda) = \left(X^T X + \lambda I\right)^{-1} X^T \underline{y}$$

Q: Why "simpler" features are favoured?

Q: How does the conditioning of the problem improved?

The generalization error

The generalization error of a learning method is the expected prediction error for "*unseen*" data, i.e. mistakes made on the data that we are going to see in the future. This quantifies how well the method *generalizes*.

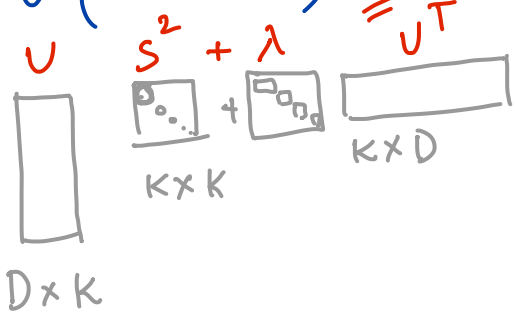
What is "unseen" data?
Tricky to define ...
nevertheless ....

$$X^T X + \lambda I$$

$$= U S \underbrace{V^T V}_{I_K} S U^T + \lambda I_D$$

$$= U S^2 U^T + \lambda U U^T$$

$$= U \left(S^2 + \lambda I \right) U^T$$



Q: Write an expression for the minimizer of (A)

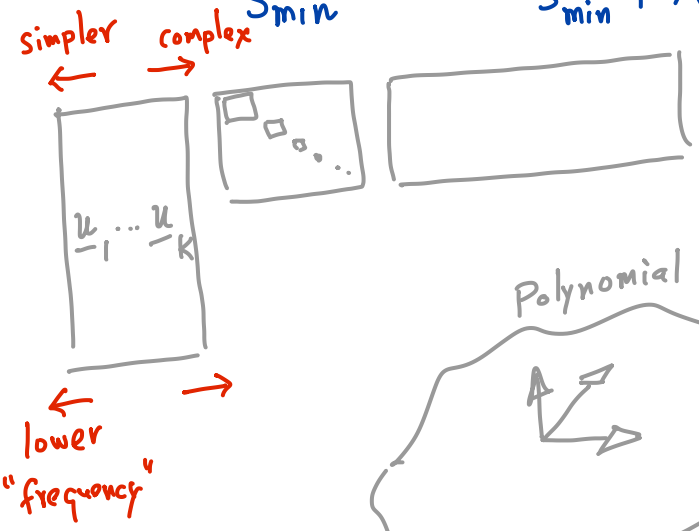
$$\beta^*(\lambda) = \left(X^T X + \lambda I \right)^{-1} X^T \underline{y}$$

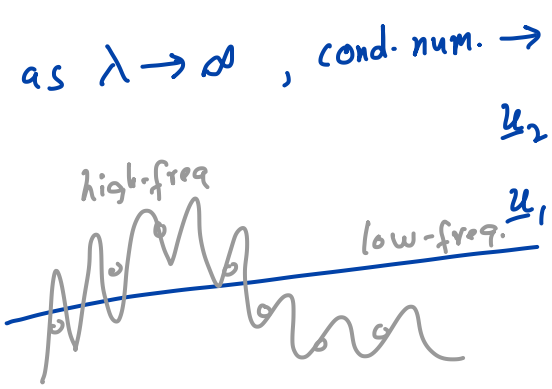
Q: How does the conditioning of the problem improved?

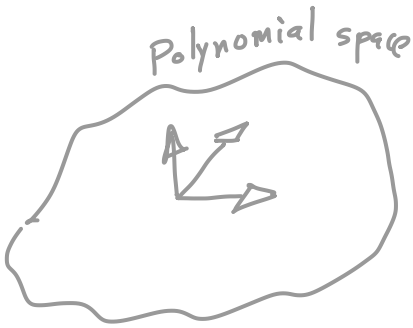
Write the SVD of $X^T = U S V^T$

Q: Why "simpler" features are favoured

Cond. num. : $\dfrac{S_{max}^2}{S_{min}^2} \longrightarrow \dfrac{S_{max}^2 + \lambda}{S_{min}^2 + \lambda}$, as $\lambda \to \infty$, cond. num. $\to 1$







$$\underset{D \times N}{X^T} = \begin{bmatrix} 1 & 1 & & 1 \\ x_1 & x_2 & \cdots & x_N \\ x_1^2 & x_2^2 & & x_N^2 \\ \vdots & & & \vdots \\ x_1^D & x_2^D & & x_N^D \end{bmatrix}$$

Simulating the future

Ideally, we should choose λ to minimize the mistakes that will be made in the future. Obviously, we do not have the future data, but we can always *simulate the future* using the data in hand.

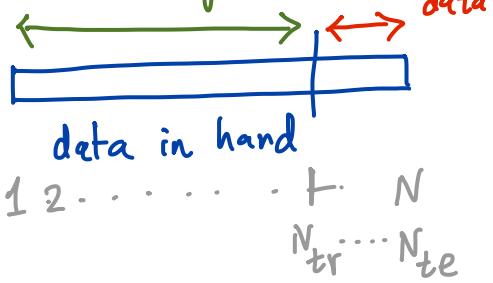
$\beta_0 + \beta_1 x + \cdots + \beta_{100} x^{100}$

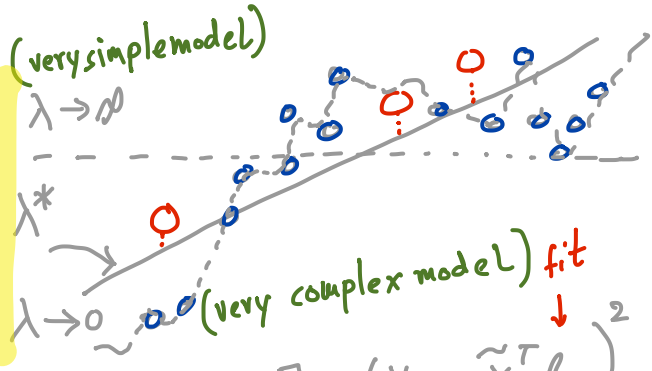
$\lambda \to \infty, \quad \beta_j \to 0$

Splitting the data

For this purpose, we split the data into train and validation sets, e.g. 80% as training data and 20% as validation data. We pretend that the validation set is the future data. We fit our model on the training set and compute a prediction-error on the validation set. This gives us an *estimate* of the generalization error (one instant of the future).

$D_{tr} = \{\underline{x}_i, y_i\}_{i=1}^{N_{tr}}$ $\quad D_{val} = \cdots$

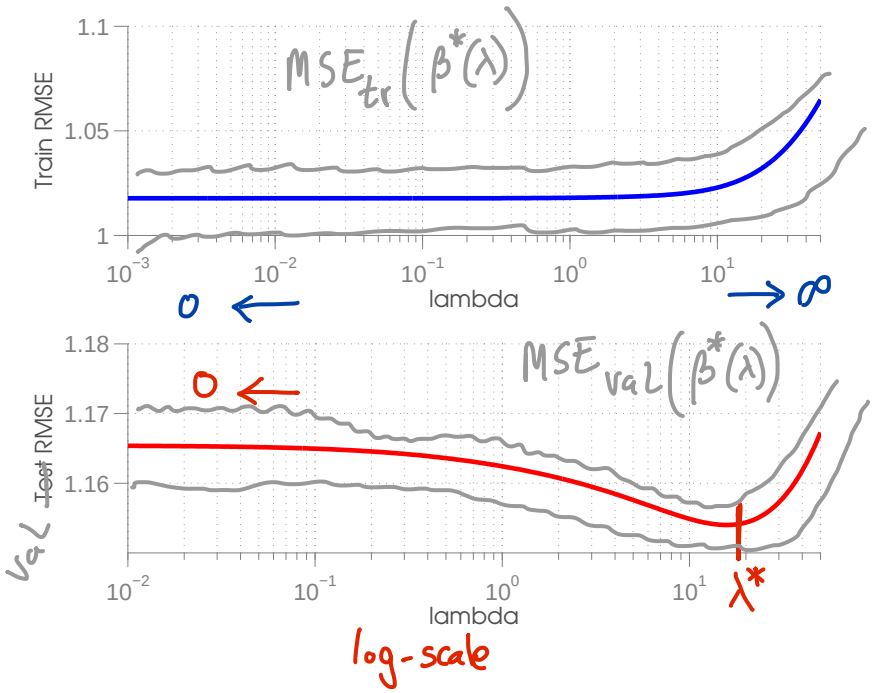




fit

$$MSE_{tr}(\beta) = \sum_{i \in D_{tr}} (y_i - \tilde{\underline{x}}_i^T \underline{\beta})^2$$

$$MSE_{val}(\beta) = \sum_{i \in D_{val}} (\quad '' \quad)^2$$



$$\beta^*(\lambda) = \min_{\beta} MSE_{tr}(\beta) + \frac{\lambda}{2} \underline{\beta}^T \underline{\beta}$$

Q: How should we choose the split?

A: randomly? but we should be careful.

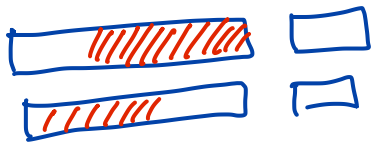
It should be "representive" of the problem/data.

Cross-validation

Random splitting (aka bootstrap) is not an efficient method.

for $s = 1, \dots, 100$
shuffle the data
pick the first x% as train
fit
evalute $MSE_{Val}(\beta_s^*(\lambda))$
end

Random shuffling is not efficient!



K-fold cross-validation allows us to do this efficiently. We randomly partition the data into K groups. We train on $K-1$ groups and test on the remaining group. We repeat this until we have tested on all K sets. We then average the results.

Validation sets are disjoint!
Estimates of valError is a bit better (low variance) but it's biased.

shuffle and
* Split in K subsets
* $\lambda = [10^{-5}\ 10^{-4} \dots 1\ 100\ 10^{3}]$
* for all λ

for $k = 1:K$
Fit model Train
test on the validation
$MSE_{val}^{\lambda}[k]$
end
$MSE_{val}^{\lambda} = \text{mean over } k=1:K \left[MSE_{val}^{\lambda}[k]\right]$

Cross-validation returns an estimate of the *generalization error*.

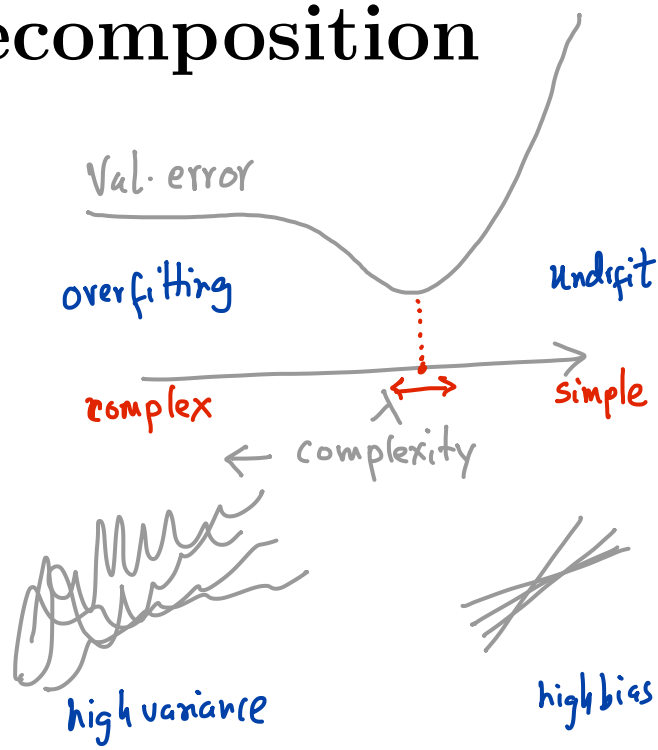
Additional Notes

Details on cross-validation are in Chapter 7 in the book by Hastie, Tibshirani, and Friedman (HTF). You can also read about bootstrap in Section 7.11 in HTF book. This method is related to random splitting and is a very popular method.

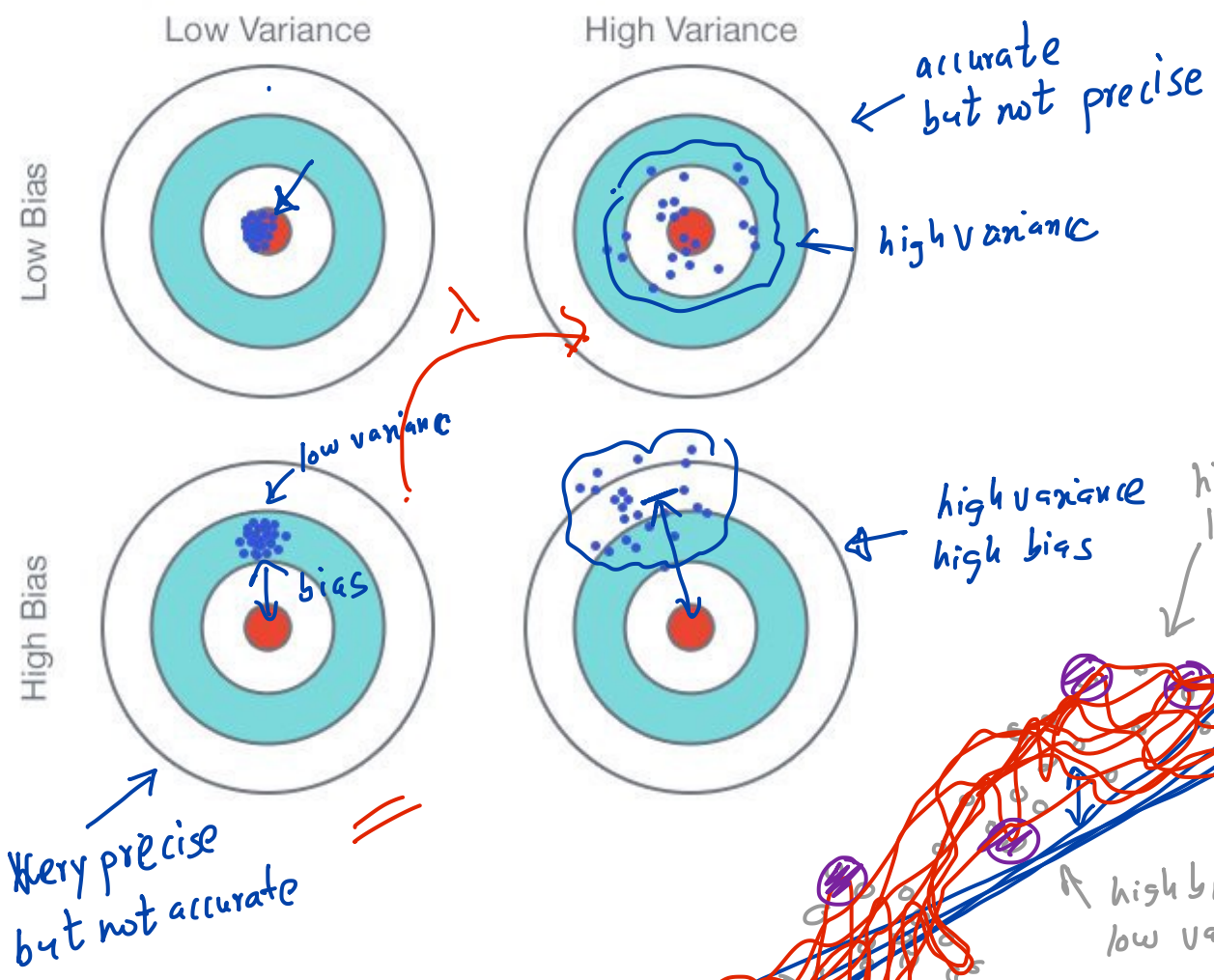
9 Bias-Variance Decomposition

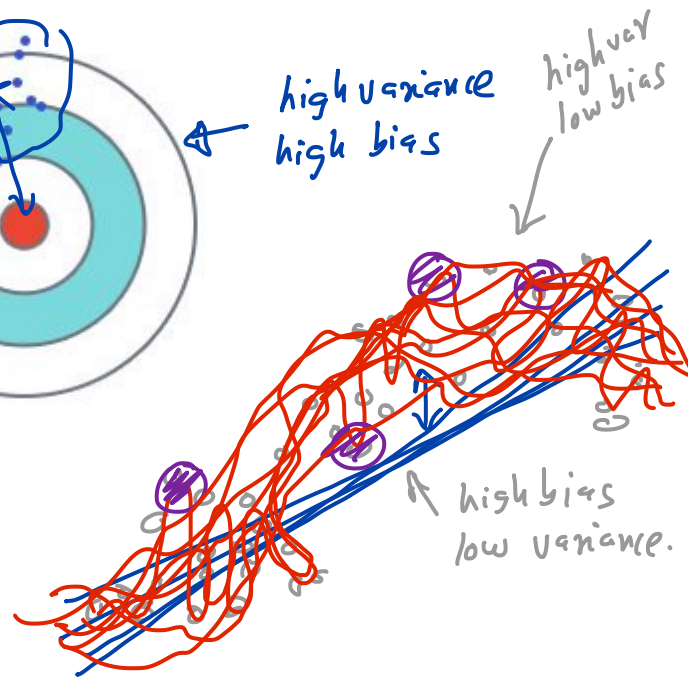
What is bias-variance?

One natural question is how does the test error vary wrt λ? When λ is high, the model underfits, while when λ is small, the model overfits. Therefore, a good value is somewhere in between.



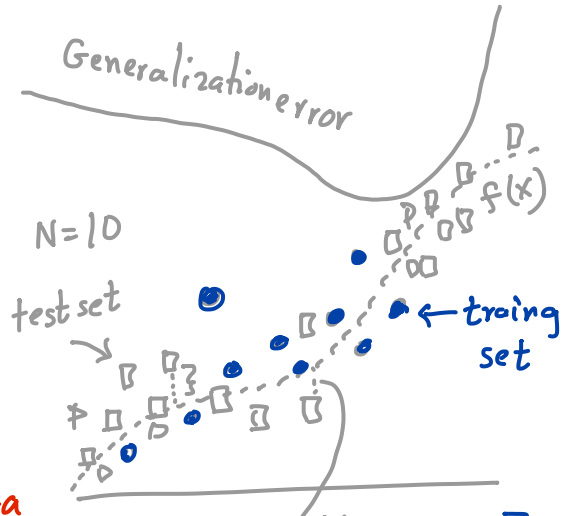
Bias-variance decomposition explains the shape of this curve.





Generalization error

Given training data $\boxed{\mathcal{D}_{tr} \text{ of size } N,}$ we would like to estimate the expected error made in future prediction. This error is the generalization error. Below is a definition suppose that we have infinite test data $\mathcal{D}_{te}$,



training data

training data

$$teErr(\mathcal{D}_{tr}) := \mathbb{E}_{\mathcal{D}_{te}}[\{y - f(\mathbf{x})\}^2] = \mathbb{E}_{y,x\sim D_{te}}\left[(y - \beta_*^T \tilde{\underline{x}})^2\right]$$

test data

best linear model obtained by

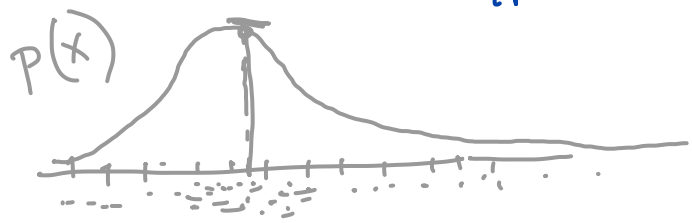
Generalization error is different from the training error which measures how well you fit the data.

$$\beta_*(D_{tr}) = \arg\min_{\beta} \frac{1}{N} \sum_{x_n, y_n \sim D_{tr}} \left[(y_n - \beta^T \tilde{\underline{x}}_n)^2\right]$$

$$trErr(\mathcal{D}_{tr}) := \underbrace{\sum_{n=1}^{N}}_{\text{function of } D_{tr}} [\{y_n - \underbrace{f(\mathbf{x}_n)}_{\beta_*^T \tilde{\underline{x}}_n}\}^2]$$

function of D_{tr}

function of training data D_{tr}



$$\lim_{n\to\infty} \frac{1}{N} \sum_{n=1}^{N} x_n \longrightarrow \underset{\text{mean}}{\mathbb{E}(x)} = \int x\, p(x)\, dx$$

$$D_{te} = \{y_n, \underline{x}_n\}_{n=1}^{\infty}$$

Mean - Squared error $\frac{1}{N} \sum_{n=1}^{N} \left(y_n - f(\underline{x}_n)\right]^2$

$\downarrow N \to \infty$

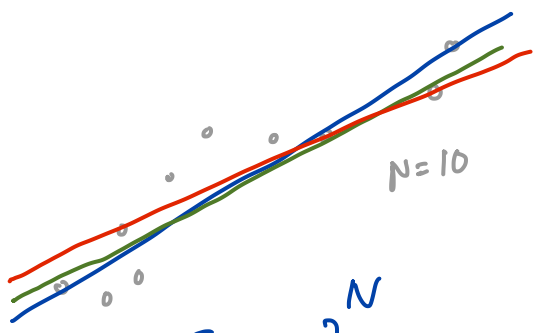
$$\mathbb{E}_{D_{te}}\left[(y - f(x))^2\right] = \int (y - f(x))\, p(y,x)\, dy\, dx$$

Errors vs model complexity

As we increase the model complexity, how do these errors vary? The blue line shows training error for a dataset with $\boxed{N = 50}$, while the red line shows the generalization error for that dataset.

Simple model have high train and generalization error since they have a high bias, while complex model have low train but high generalization error because they have high variance.

$teErr(D_{tr})$



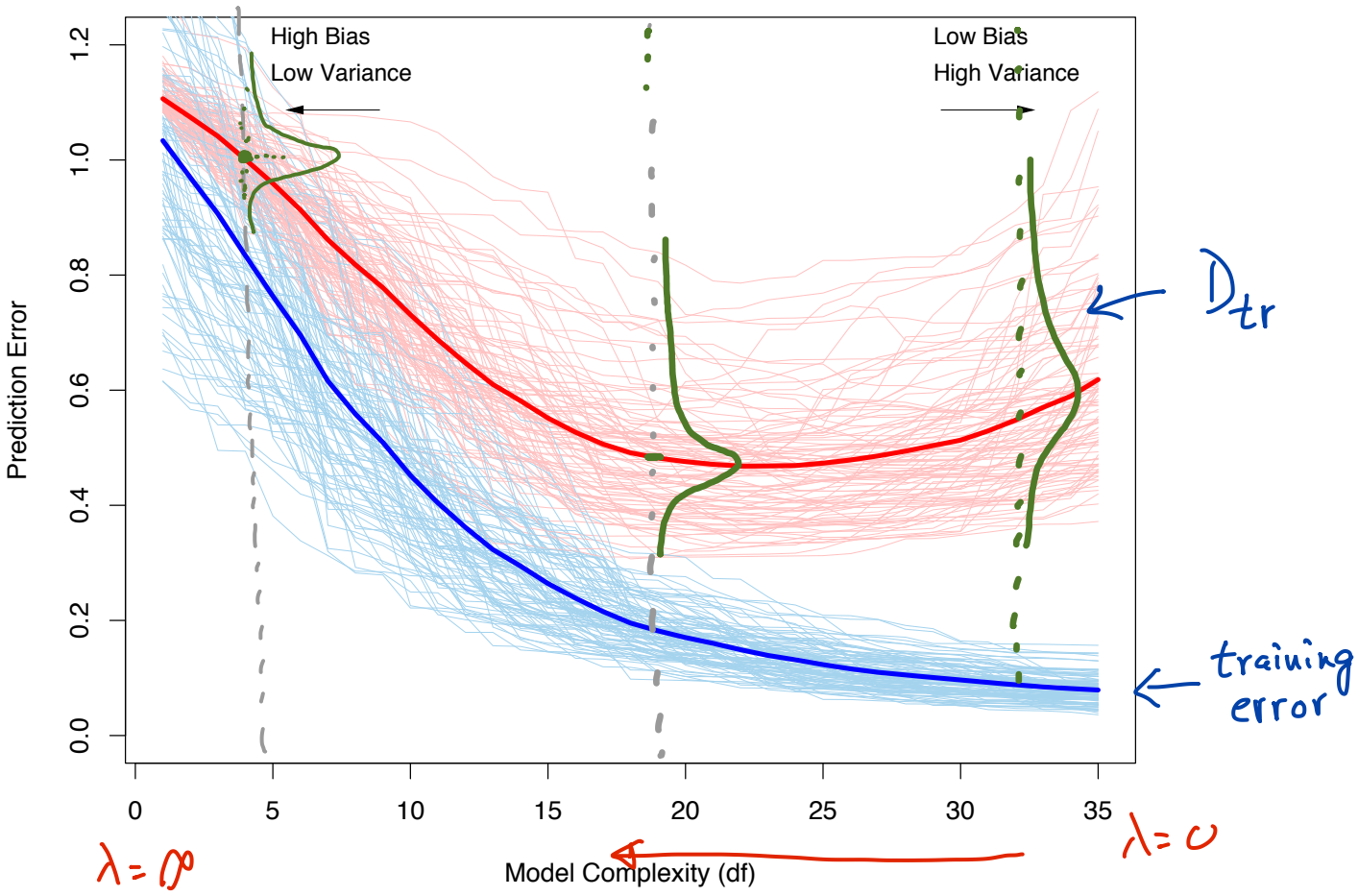
$D_{tr} = \{x_n, y_n\}_{n=1}^{N}$

where $x_n, y_n \sim P(x, y)$

$\beta_*(D_{tr}; \lambda)$

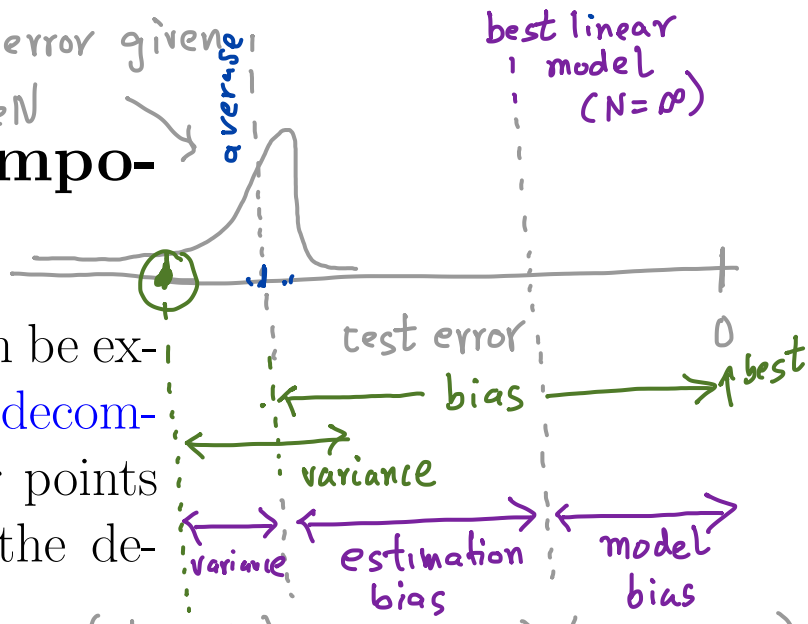
$\downarrow$

$teErr(D_{tr}; \lambda)$



Distribution of testerror given $D_t \sim P(X,Y)$ of size N

Bias-variance decomposition



The shape of these curves can be explained using bias-variance decomposition. The following four points can be explained by using the decomposition:

1. both bias and variance contribute to generalization error.

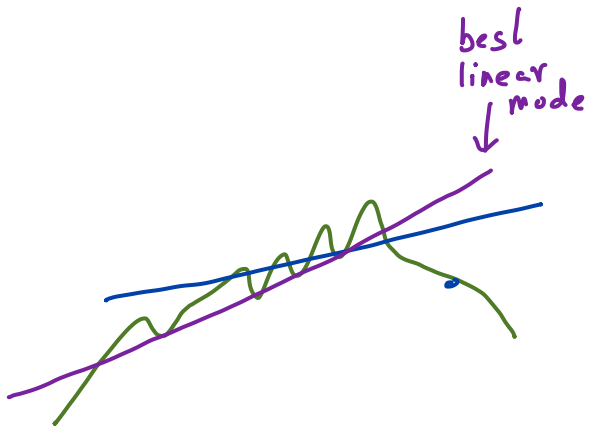
   (because of "given data at hand") (because my data size is not enough) (because my model is not good)

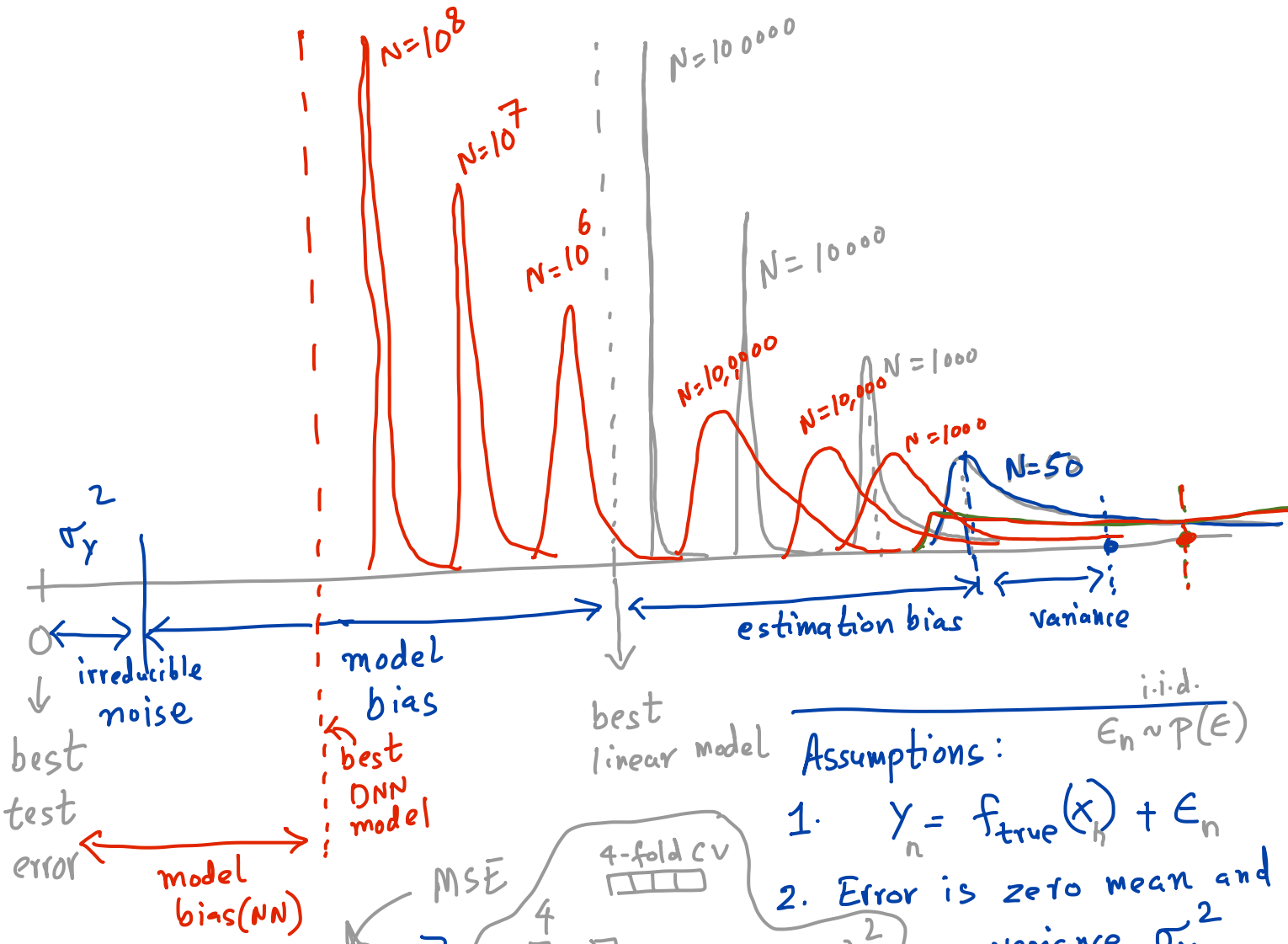
2. For bias, both model-bias and estimation-bias are important. When we increase model complexity, we increase generalization error due to increased variance.

   We will use "add-subtract" technique to analyze this:

$$y - \beta_*^T \tilde{x}$$
$$= \underbrace{y - f_{true}(x)} + \underbrace{f_{true}(x) - \beta_*^T \tilde{x}}$$
$$\underbrace{f_{true}(x) - \beta_{best}^T \tilde{x}}_{\text{model bias}} + \underbrace{\beta_{best}^T \tilde{x} - \beta_*^T \tilde{x}}_{\text{estimation bias + variance}}$$

3. Regularization increases estimation bias while reducing variance.





Assumptions:

1. $Y_n = f_{true}(x_n) + \epsilon_n$ $\quad \epsilon_n \overset{i.i.d.}{\sim} P(\epsilon)$
2. Error is zero mean and variance σ_y^2 (Doesn't need to be Gaussian)
3. x is "given", not stochastic

$$\mathbb{E}_{D_{te}}[\cdot\cdot] = \mathbb{E}_{P(y)}[\cdots]$$

4-fold CV

$$\mathbb{E}_{D_{tr}}\mathbb{E}_{D_{te}}\left[\left(y - f_{est}(x)\right)^2\right] \approx \sum_{i=1}^{4}\sum_{j\in D_{te}^{(i)}}\left(y_j - f(x_j)\right)^2$$

$D_{tr}^{(i)}$

MSE

$$= \mathbb{E}_{D_{tr}}\mathbb{E}_{D_{te}}\left[\left(y - f_{true} + f_{true} - f_{est}\right)^2\right]$$

$$= \underbrace{\mathbb{E}_{D_{te}}\left[(y - f_{true})^2\right]} + \mathbb{E}_{D_{tr}}\left[(f_{true} - f_{est})^2\right] + 2\,\cancel{\mathbb{E}_{D_{te}}\underbrace{(y - f_{true})}_{\epsilon}}\,\mathbb{E}_{D_{tr}}(f_{true} - f_{est})$$

$= 0$

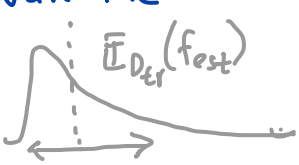
$$= \sigma_y^2 + \mathbb{E}_{D_{tr}}\left[(f_{true} - f_{est})^2\right]$$

$$= \sigma_y^2 + \mathbb{E}_{D_{tr}}\left(f_{true} - \mathbb{E}_{D_{tr}}(f_{est}) + \mathbb{E}_{D_{tr}}(f_{est}) - f_{est}\right)^2$$

$$= \underbrace{\sigma_y^2}_{\text{irreducible noise}} + \underbrace{\left(f_{true} - \mathbb{E}_{D_{tr}}(f_{est})\right)}_{\text{bias}} + \underbrace{\mathbb{E}_{D_{tr}}\left[\left(f_{est} - \overbrace{\mathbb{E}_{D_{tr}}(f_{est})}^{\text{mean}_{tr}}\right)^2\right]}_{\text{variance}} + 2\,\cancel{\mathbb{E}_{D_{tr}}\left(f_{est} - \mathbb{E}_{D_{tr}}(f_{est})\right)}\left(f_{true} - \mathbb{E}_{D_{tr}}(f_{est})\right)$$

$= 0$

$$\underbrace{f_{true} - \mathbb{E}_{D_{tr}}(f_{est}^{best})}_{\text{model bias}} + \underbrace{\mathbb{E}_{D_{tr}}(f_{est}^{best}) - \mathbb{E}_{D_{tr}}(f_{est})}_{\text{estimation bias}}$$



Understanding bias-variance "Trade-off" in Linear model.

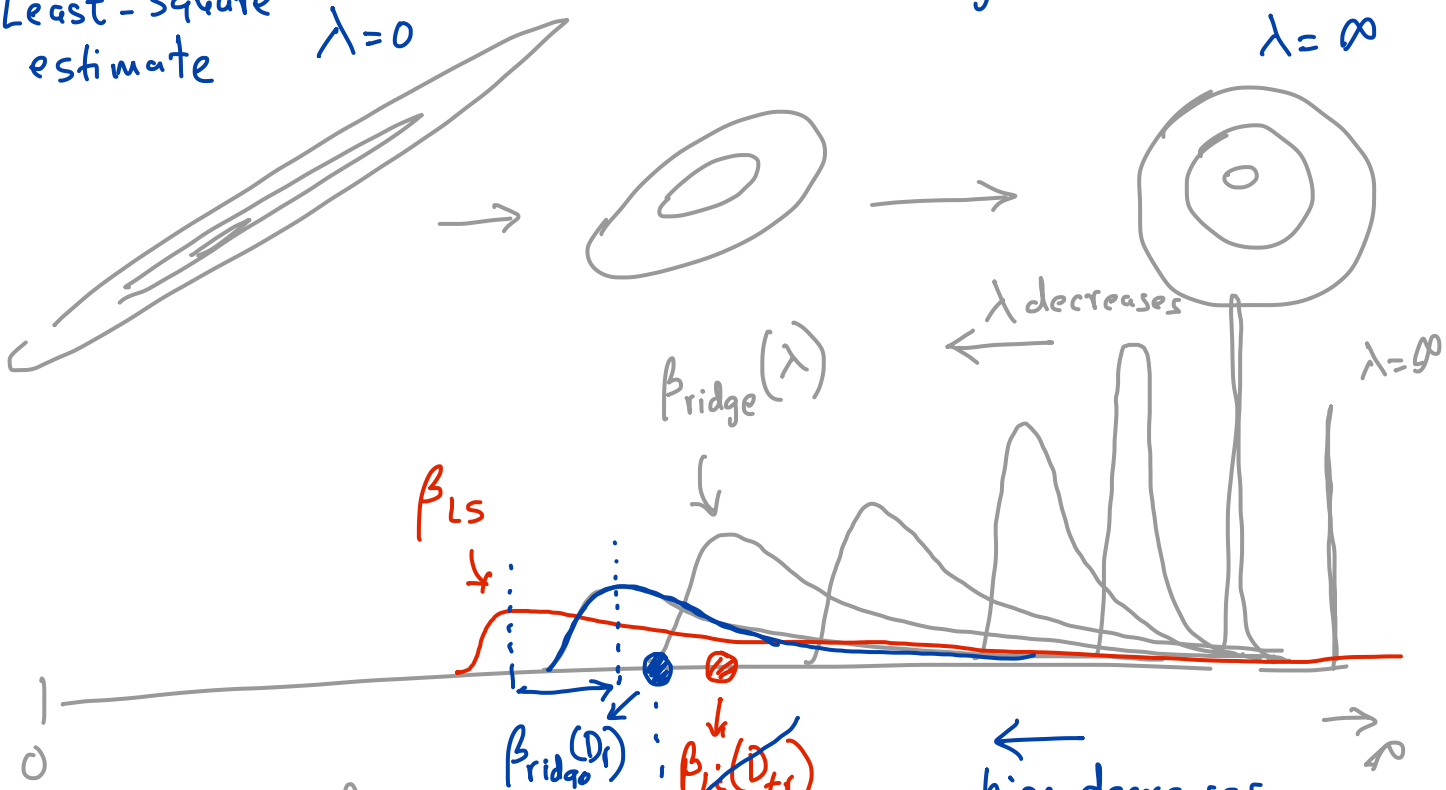
(Ridge regression)

$$\beta^*_{ls}(D_{tr}) = \underbrace{\left(\tilde{X}_{tr}^T \tilde{X}_{tr}\right)^{-1}} \tilde{X}_{tr}^T Y_{tr}$$

Least-square estimate

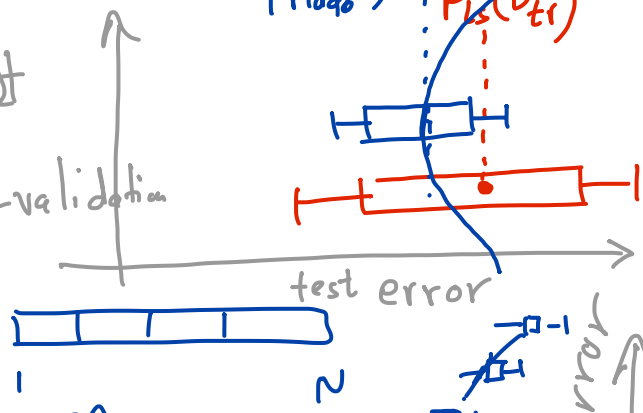
$$\beta^*_{ridge}(D_{tr}, \lambda) = \left(\tilde{X}_{tr}^T \tilde{X}_{tr} + \lambda I\right)^{-1} X_{tr}^T Y_{tr}$$

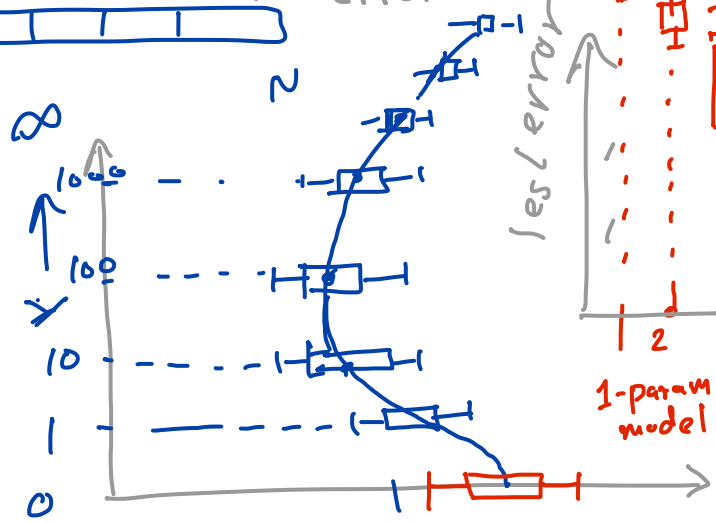
Ridge estimate

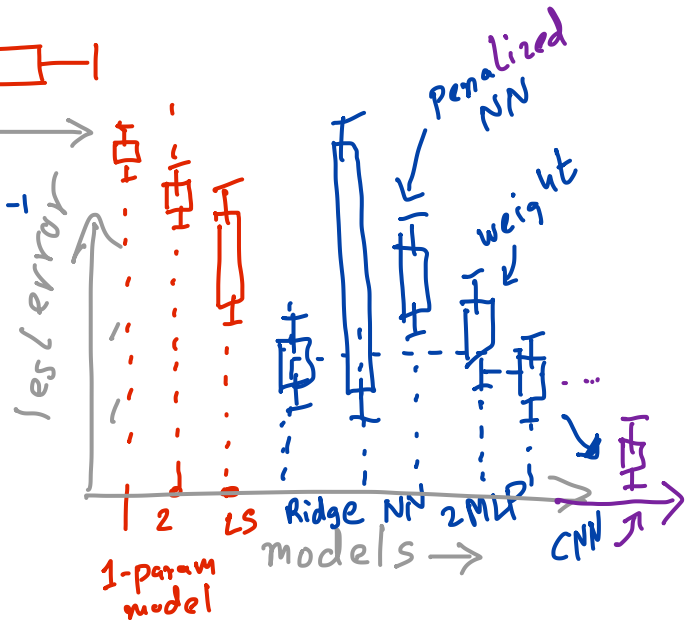


Result of Cross-validation

Fix D_{tr}







$Y_n \approx \beta_0$

$Y_n \approx \beta_0 + \beta_1 X_{n1}$

10 Recent Advances

Deep Learning & Overfitting

Deep learning has shown a new (but old) way to combat overfitting. For many applications, more data and deep architecture combined with stochastic gradient-descent is able to get us to a good minimum which generalizes well.

Challenges

There are many challenges ahead. Learning from nasty, unreliable data still remains a challenge (e.g. small sample size, redundant data, non-stationary data, sequential learning).

On the other hand, living beings - even young ones - are very good in dealing with such data. How do they do it, and how can we design ML methods that can learn like them?