Introduction to Machine Learning

Shizheng Wen

shiwen@student.ethz.ch

Table of contents

1 Introduction

1.1 Useful Links

- Course Website
- Python Tutorials

1.2 Logistics

1. Q&A is presented by Zoom (homework solutions)
2. Tutorials (Math Recap)

1.3 Technical Problems

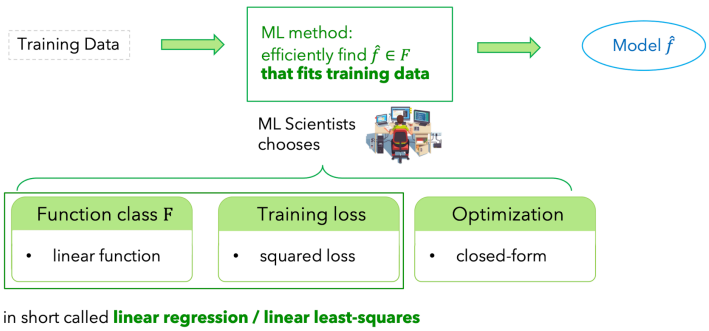
1.3.1 Building environments for testing demos

Problems:

1. Unable to execute `pip install -r requirements.txt`
   (a) solutions: CSDN + the version of matplotlib is compatible with that of Python with version of 3.9
2. Unable to clone the code

2 Supervised Learning: Linear Regression

Machine Learning Pipeline



- Define your train loss (Goal) and find a best approximaiton from the function class by using relative optimizaiton methods.
- Here, loss is ofter referred as the distance between the $\hat{y}$ and y. From this prospective, we can use norm to characterize such properties.

Function class: Linear function

- Parameterized linear function with several variables. From the prospective of LSE (#equation→data > #variable→parameters in this function class), we can feel that it is impossible for function class with parameters larger than the training dataset. However such intution is violated in DL.
- In multiple regression, the linear function is a parameterized hyperplane.

Training loss: L_2 norm

Our final "learnt rule" is the function $\hat{f}$ that minimizes the training loss (squared loss on average) L_2 norm in vector space: $\hat{f} = \underset{f \in F_{lin}}{\operatorname{argmin}}\ L(f) = \underset{f \in F_{lin}}{\operatorname{argmin}}\ \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

- $\hat{f}$ is called the solution of the ML method linear regression.

Furthermore, by substitution, $\hat{w} := (\hat{w}_0, \hat{w}_1) = \underset{w_0, w_1 \in \mathbb{R}}{\operatorname{argmin}}\ L(w_0, w_1) = \underset{w_0, w_1 \in \mathbb{R}}{\operatorname{argmin}}\ \frac{1}{n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2$

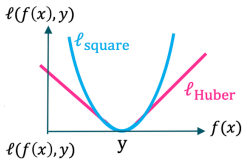
Side comment: other losses

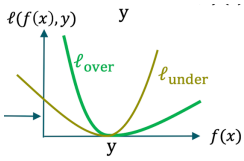
The limit of the squared loss:

- weighs over- and underestimation the same. 也就是predicted value在accurate value左右两侧对loss的贡献是一致的。但有些时候，我们想要体现出这种不一致性。比如说预测汽车发车时间，我们希望predicted value > accurate value会造对loss function造成更大的贡献，从而来反向抑制使得我们的模型尽可能不预测出大于accurate value的值。这种情况也叫underestimation。
- Costs grows quadratically (large errors hugely penalized). 这会导致某些偏离数据集的点(离群值，异常值outliers)严重影响回归曲线的走向。

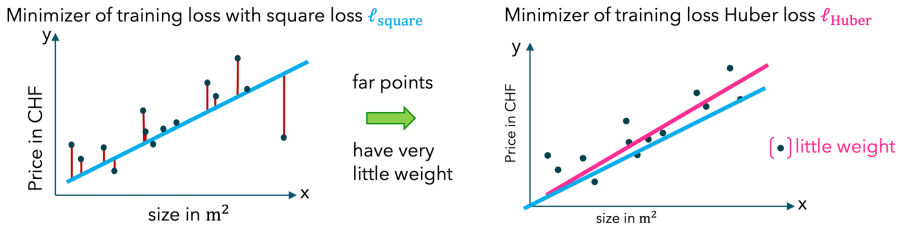
Instead might want:

- Ignore outliers (ones with very large penalty) → **Huber loss**
- Weigh over- and underestimation differently → **asymmetric losses**



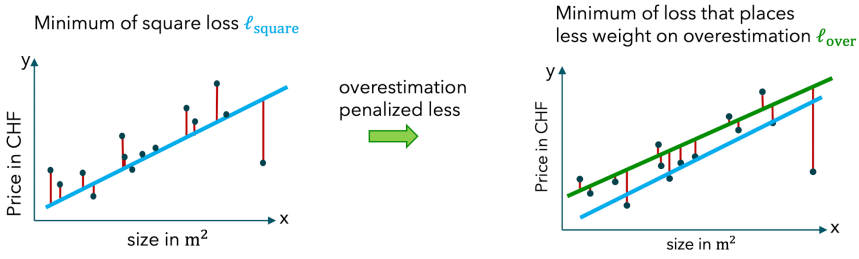


See the following example for Huber loss



Perhaps better choice if there are outliers in your training data not representing current market

See the following example for asymmetric regression loss



Perhaps better choice if you care more about maximizing profit than how fast you can sell

Optimization: Closed-form

Two different ways to derive the optimal solution → normal equation

1. Algebric perspective → stationary point condition (gradient = 0): because it is a convex problem, a global minimum $\hat{w}$ must satisfy $\nabla_w L(\hat{w}) = 0$

   (a) $$\hat{w} = \underset{w_0 \in \mathbb{R},\, w \in \mathbb{R}^d}{\operatorname{argmin}} L(w_0, w) = \underset{w_0 \in \mathbb{R},\, w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n}||y - Xw||^2$$

   i. Taking derivate of $\frac{1}{n}||y||^2 - \frac{2}{n}y^\top Xw + \frac{1}{n}w^\top X^\top Xw$ Yields gradient $\nabla_w L(0,w) = \frac{2}{n}(X^\top Xw - X^\top y)$ and Hessian $D^2 L(0,w) = \frac{2}{n}X^\top X$

   ii. minimum must be a stationary points. Because the Hessian matrix is positive semi-definite (psd), all stationary points of this quadratic loss are minima. [convex problem]

   iii. Gradient expression together with stationary point condition yields the condtion (normal equation) → $X^T y = X^T X\hat{w}$

   (b) It should be noted that the #solutions depends on $\{(x_i, y_i)\}_{i=1}^n$

   i. From the prospective of matrix vector notation, we know that the solution is determined by whether the data matrix (tall matrix) satisfies full-rank condition (FRC).

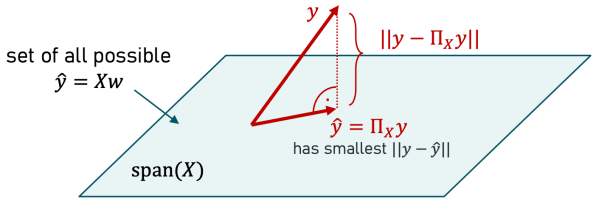
2. Geometry perspective/argument → orthogonal projection:

   (a) First, rewrite the training loss L in matrix vector notation (in multiple regression)

   $$L(w_0, w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w_0 - w^\top x_i)^2 = \frac{1}{n}||y - 1w_0 - Xw||^2$$

   $\mathbf{1} = (1, \ldots, 1) \in \mathbb{R}^d$ is the all-ones vector

   Furthermore, we have $\hat{w} = \underset{w_0 \in \mathbb{R},\, w \in \mathbb{R}^d}{\operatorname{argmin}} L(w_0, w) = \underset{w_0 \in \mathbb{R},\, w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n}||y - Xw||^2$

   (b) $y - X\hat{w} \perp Xw$ gives rise to $X^\top y = X^\top X\,\hat{w}$

Remarks about the normal equations:

- view the math recap for sorting more information

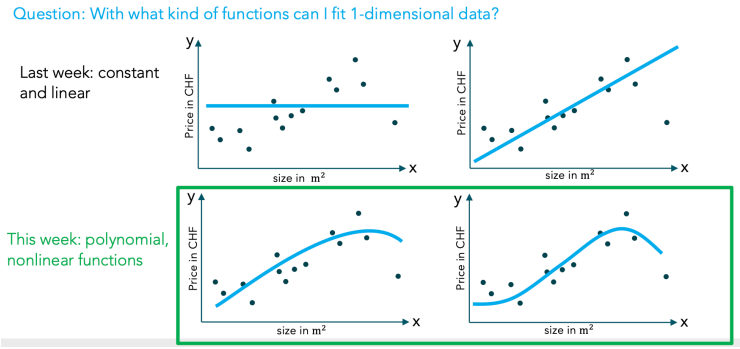
3 Optimization

Motivation: In last chapter, we talked about the closed-form (normal equation derived by computing stationary points or using a projection argument) for linear regression. However, the closed-form may have following drawbacks:

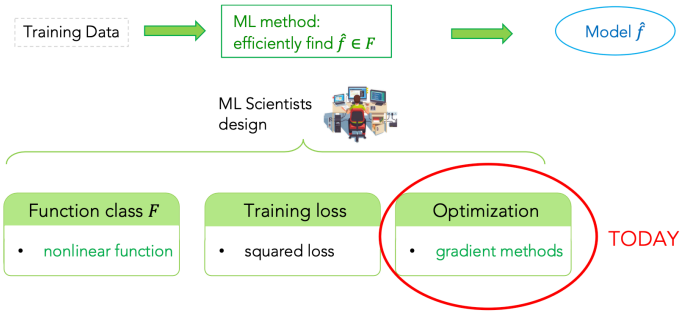
- may have high cost: have to compute inverse of a $d \times d$ matrix (see homework)

- lack of generality: How to find $\hat{f}$ when the closed form cannot be directly calculated? (e.g. for the absolute value or Huber loss) The closed-form discussed before is only for squared loss.

Therefore, we will discuss a generic iterative optimization procedure to find minimizer $\hat{w}$. By the way, using such optimization method, we can consider more complex function classes F



The optimization methods discussed in this course are all based on gradients.



General workflow of interative methods of optimization for multidimension:

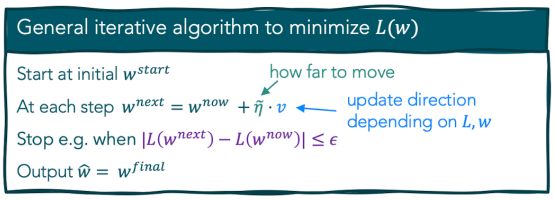
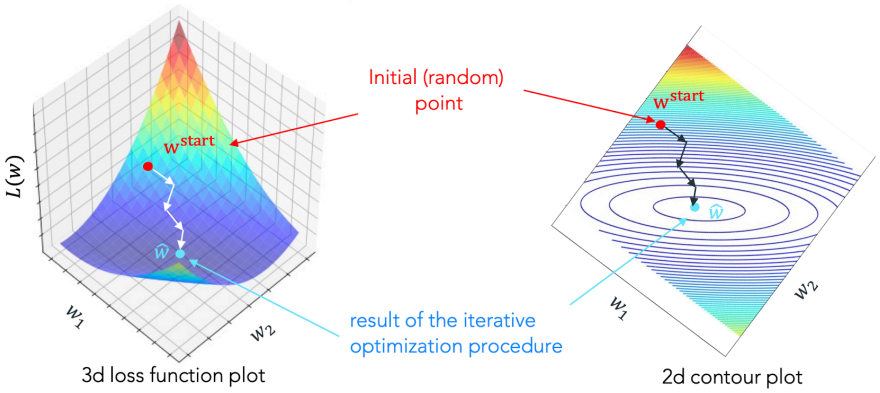


Figure out following questions:

- Q1: which direction v?
- How far ($\tilde{\eta} > 0$) to go in that direction? $\rightarrow$ not too far
- When to stop? $\rightarrow$ e.g. when $\|L(w^{next} - L(w^{now}) \leq \epsilon\|$
  - **Stopping creterion:** $L(w^{now}) - L(w^{previous})$; $w^{now} - w^{previous}$; $L'(w^{now}) < 10^{-5}$ $\Rightarrow$ This also means we stop at a point close to stationary point. By the way, we could also check Hessian.

Next, we will use a 2d case for intuition understanding. Then, we will rigorously derive the actual form for direction and how far to move.

Visualizaiton of losses in 2d via contour plots

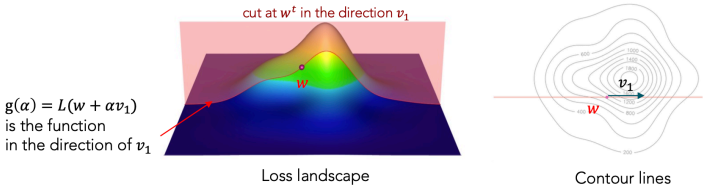


Question:

- Which direction would you follow if you wanted to go down the shortest possible route?
  - the steepest descent direction which is orthogonal to the contour lines!
- Which direction would you follow if you wanted to go down fast but also technically more easily?
  - At least on average following the steepest descent direction (stochastic methods – next lecture)
  - NOT so clear about the technically more easily?

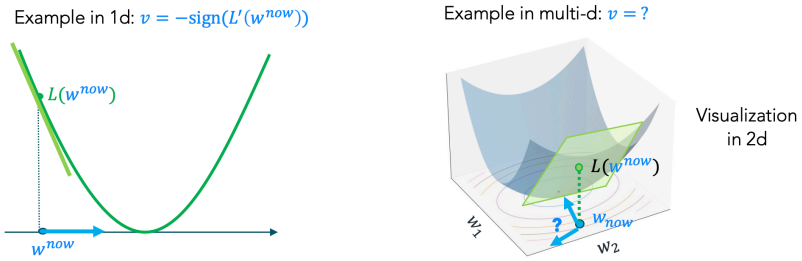
Negative gradient as steepest descent direction

Following, all discussions are centered with **steepest descent direction is negative gradient. We will prove it from different perspectives.**

- Graphically using orthogonality to contour lines
  - we can find that steepest descent direction is orthogonal to contour lines. → we want to concretely compute it for a loss L at a given point w^{now}?
    - Compute the direction of contour lines v_0. → by definition, is such that $g(\alpha, v_0) = L(w + \alpha v_0)$ doesn't change much for small α. And $g(\alpha, v_0) = L(w + \alpha v_0)$ has stationary point at $\alpha = 0$
    - 

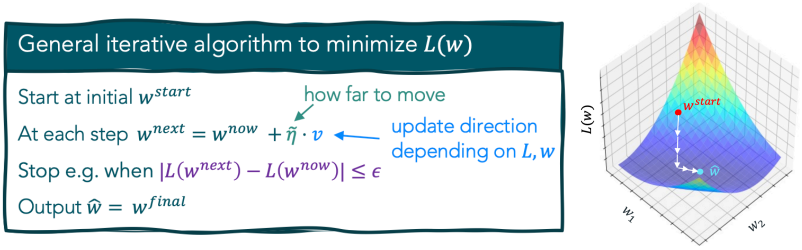
    - $g(\alpha, v_0) = L(w + \alpha v_0)$ has stationary point at $\alpha = 0$ → by chain rule, $g'(0, v_0) =< \nabla L(w), v_0 >= 0$. where $\nabla L(w)$ is the gradient. → By coincidence, we can find that $\nabla L(w)$ is orthogonal to countour line at w.
- Arithmetically using linear approximation
  - Reminder of general update formula: $w^{next} = w^{now} + \tilde{\eta}\, v$
  - Intuition: for small $\tilde{\eta}$, steepest direction on linear approx. (easier to compute) $\approx$ steepest on the true loss. Assume that L is differentiable → best linear approximation of the loss around current point w^{now} is: $L(w^{next}) = L(w^{now} + \tilde{\eta}\, v) \approx L(w^{now}) + \tilde{\eta}\, \langle \nabla L(w^{now}), v \rangle$.

- If we want to $L(w^{next})$ is as small as possible, the v should be the negative direction of the gradient at this point w^{now}.
- This is called the steepest direction on the linear approximation



  - Derivation of steepest descent direction on tangent plane ($\perp$ to the line in contour)

Smmary of gradient descent method



At each step, depending on L, w^{now}:

- the direction are $\rightarrow$ negative direction gradient
- setting $\eta = \frac{\tilde{\eta}}{||\nabla L(w^{now})||}$ (called learning rate), we arrive at gradient descent update: $w^{next} = w^{now} - \eta \nabla L(w^{now})$
  - ⚠改形式仍旧是方向加步长，但是依照本章节所讨论的上述形式，步长保持不变如果$\tilde{\eta} = \text{const}$ 。但是在实际中,η是个常数，也就是步长会随着梯度的下降越来愈小。从而无法override
- Moreover, according to the theory, if η is small enough, the negative gradient direction is always a descent direction! $L(w^{t+1}) = L(w^t - \eta \nabla L(w^t)) \approx L(w^t) - \eta \langle \nabla L(w^t), \nabla L(w^t) \rangle < L(w^t)$

Negative gradient descent for linear regression

Then, Let's apply negative gradient descent in linear regression problem:

- by substuting $\nabla L(w) = -X^\top(y - Xw)$ and $X^\top X w_{min} = X^\top y$ (normal equation) into $w^{t+1} = w^t - \eta \nabla L(w^t)$, we can get:

$$
\begin{aligned}
w^{t+1} - w_{min} &= w^t + \eta\, X^\top (y - Xw^t) - w_{min} \\
X^\top X w_{min} = X^\top y \longrightarrow \quad &= w^t - w_{min} + \eta (X^\top X w_{min} - X^\top X w^t) \\
&= (I - \eta X^\top X)(w^t - w_{min})
\end{aligned}
$$

  By appling norm in the both side, we can derive that

$$
\begin{aligned}
||w^{t+1} - w^*||_2 &\leq \underbrace{||I - \eta X^\top X||_{op}}_{=:\rho} ||w^t - w_{min}||_2 \\
&\leq ||I - \eta X^\top X|\Big|_{op}^{t+1} ||w^0 - w_{min}||_2
\end{aligned}
$$

  and operator is defined as follows:

- operator norm: $||A||_{op} = \sup_z \frac{||Az||_2}{||z||_2} = \sqrt{\lambda_{max}(A^\top A)}$
- For any real symmetric matrix A, the operator norm is $||A||_{op} = \max\{|\lambda_{max}(A)|, |\lambda_{min}(A)|\}$

- Summary:

$$||w^t - w_{min}||_2 \leq \overbrace{\left[||I - \eta X^\top X||_{op}\right]^t}^{\rho^t} \overbrace{||w^0 - w_{min}||_2}^{\text{initial distance (independent of t)}}$$

- If $\rho < 1$, the we have contraction $||w^t - w_{min}||_2 < ||w^0 - w_{min}||_2$ leading to convergence.
- and we say the distance converges $\to 0$ as $t \to \infty$ linearly / exponentially
- In the homework, you'll prove that $\eta \leq 2/\lambda_{max}(X^\top X)$ is sufficient for $\rho < 1$
  - 注意，这里的ρ又不是一个随梯度变化的数，而是一个常数，这个常数在对general funciton的优化时无法估计收敛值，通常是个hyperparameter to chooose.
  - The speed depends on the maximum/minimum eigenvalues and stepsize.

Well-conditioned problem and ill-conditioned problem

Taking a 2d problem as an example:

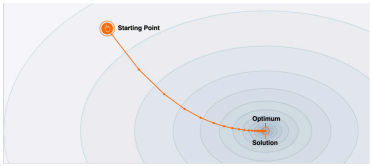
Assuming $X^\top X = \begin{bmatrix} \lambda_{min} & 0 \\ 0 & \lambda_{max} \end{bmatrix}$, we found that contour lines with $L(w) = c$ are ellipses around w_{min} with wide $\propto \frac{1}{\sqrt{\lambda_{min}}}$ and height $\propto \frac{1}{\sqrt{\lambda_{max}}}$. (See the derivation in page 45 of lecture notes)

Then, we can define two different conditions with different landscape:

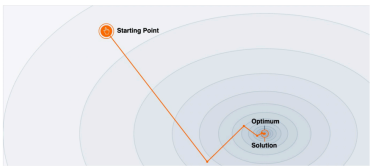
⇒ Well-conditioned

When $\lambda_{max} \approx \lambda_{min}$ (well-conditioned)

- contour lines are ellipses close to spheres
  → similar steepness in all directions
- gradient descent converges fast and
- can take pretty big steps!



small stepsize



large stepsize

⇒ Ill-conditioned

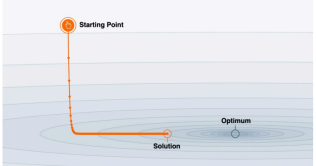
… vs. "ill-conditioned" landscape (same stepsizes)

When λ_{max} and λ_{min} are very different:

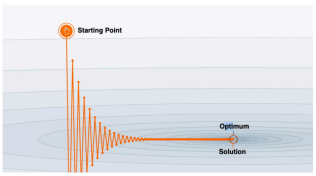
- contour lines/ellipses stretched very differently in different directions
  → there are very steep vs. very flat directions
- for small step size: slow in flat area
- for larger step size: there are large oscillations jumping over ravine

→ even with the best stepsize it takes forever to converge!

Note: Even though these simulations are for linear regression, the intuition is the same for general loss functions and eigenvalues of the corresponding Hessians



small stepsize



large stepsize

Other gradient based methods

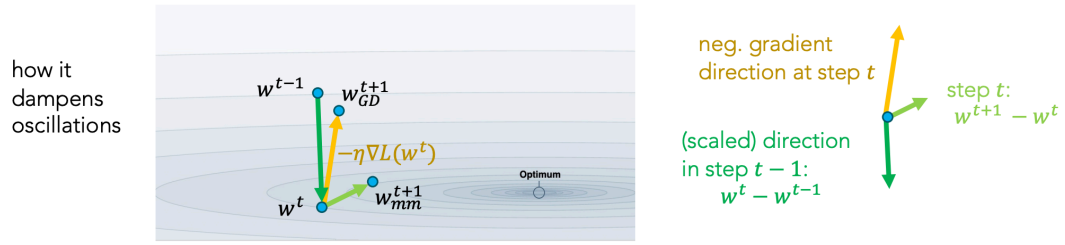
Goal: Large steps in flat areas, small steps in high curvature ones, dampened oscillations

如果在flat areas步长太小，会导致loss下降的太慢。在curvature的地方步长太大，则会导致震荡。

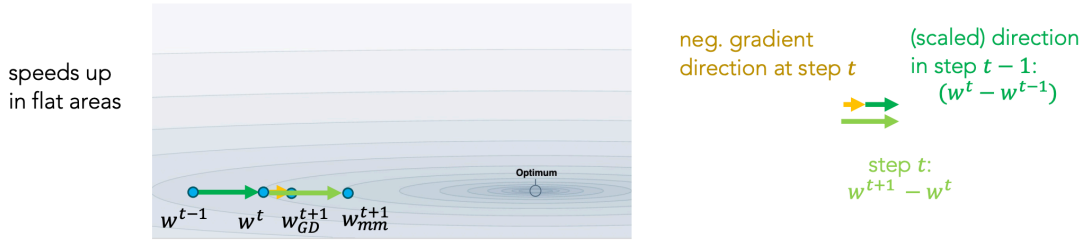
1. Momentum/accelerated gradient descent: combine previous direction with neg. gradient direction

   (a) $$w^{t+1} - w^t = \alpha\,(w^t - w^{t-1}) - \eta \nabla_w L(w^t)$$

   (b)



   (c)



2. adaptive methods (AdaGrad, RMSProp, Adam etc.): Different stepsize for different elements $w_{[i]}$ intuitively: the elements i which already changed a lot, have smaller stepsize

   (a) $$w_{[i]}^{t+1} = w_{[i]}^t - \frac{\eta}{\sqrt{\text{previouschange}_i + \gamma}} \frac{\partial L}{\partial w_{[i]}}(w^t)$$

3. 2nd order methods (Quasi-Newton, L-BFGS): https://fa.bianp.net/teaching/2018/eecs227at/newton.html

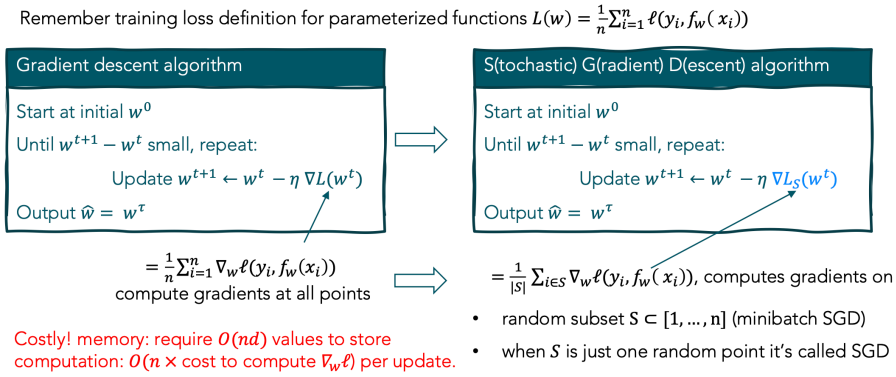
4 Optimization II

From last chapters, we know that the steepest descent direction which is orthogonal to the contour lines! For **linear regression** data affects the learning procedure (eigenvalues of the data matrix $\mathbf{X}^\mathbf{T}\mathbf{X}$) in terms of the convergence speed. See tutorial 2 for more information about the effect of data for convergence and convergent speed of linear regression. This week, we will introduce topics as follows:

- stochastic gradient methods
- how data affects the point gradient descent converges to
  - Definition of convexity, strong convexity
  - optimality of stationary points (also for regression)
- How data affects how good global minimum (of training loss) is for prediction
  - Dependence on sample size for linear regression, in low-dimensional case $n > d$
  - Dependence on sample size for linear regression, in high-dimensional case $n < d$
- Going beyong linear functions

Stochastic gradient method (Often called "SGD")

motivation: gradient descent algorithm is costful! 求梯度的时候每个数据点都得求。考虑SGD



Properties of (minibatch) SGD

如果我们用SGD,也就是每次梯度下降的方法不能保证是垂直于contours line的。也就是梯度下降的速度不是最快的。但是，通过实际情况我们发现，在多次采集样本，我们发现可以保证期望是相等的。也就是多次进行minbatch SGD，其总的梯度下降量是和全样本的gradient descent一致。

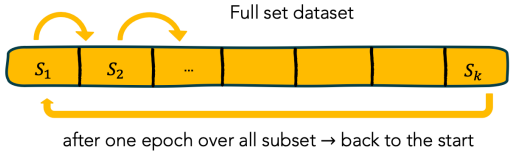
- If we sample S with replacement, the expectation $\mathbb{E}$ over the draws of the subset S

  of the minibatch gradient $\nabla L_S(w^t)$ reads

$$\mathbb{E}[\nabla L_S(w^t)] = \mathbb{E}\frac{1}{|S|}\sum_{i \in S} \nabla_w \ell(y_i, f_w(x_i)) = \nabla L(w^t)$$

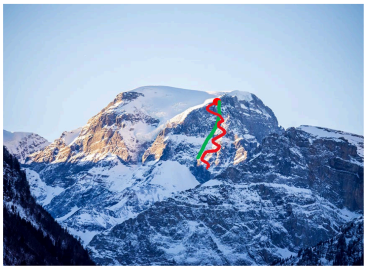
- In practice, S_t for every iteration is pre-set and we cycle through the minibatches in the data

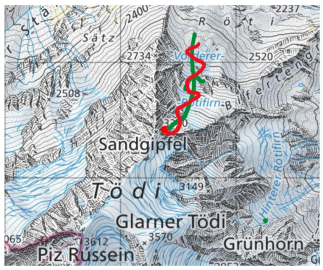
One iteration/update:
$w^t = w^{t-1} + \eta \nabla L_{S_t}(w^{t-1})$



Question: Question: Is the stochastic gradient direction $\nabla_w L_S(w^t)$ still a descent direction in the sense that $L(w^{t+1}) \leq L(w^t)$ size is small enough? **On average yes**. 也就是我们并不能保证每次都是最速下降的方向，甚至还有可能导致loss function上升，但是，从统计学意义上来讲，我们可以保证总的期望是等于全样本的梯度下降方法的。下面那个下山图有助于理解：

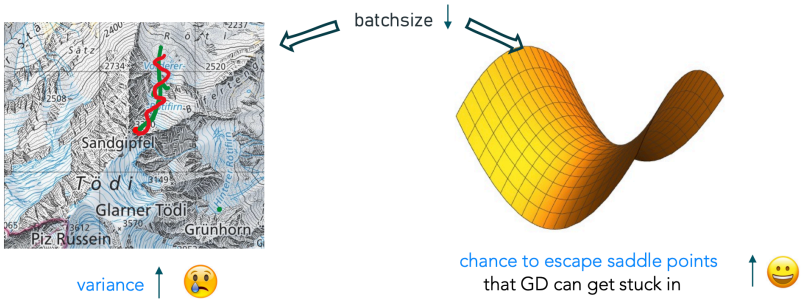
Which direction would you follow if you wanted to go down fast but also technically more easily?



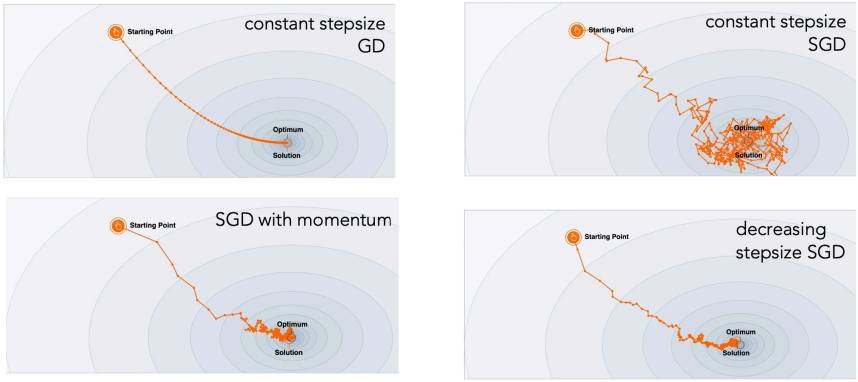


- Only on average following the steepest descent direction

⇓ Another properties is to escape saddle points.



Comparing the methods for linear regression



- We can see that SGD will introduce oscillations inevitable（因为它不能收敛到最低点，即使loss很小了也会继续在周围振荡. Therefore, we should reduce the stepsize with the marching.

How data affects the point gradient descent converges to

Motivation

Gradient descent can converge to a stationary point, but we want to get to minimum! Remember (see math recap notes): For differentiable losses,

w is local minimum $\xrightarrow{\text{always}}$ w is stationary point

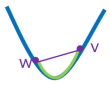
w is global minimum $\xrightarrow{\text{always}}$ w is local minimum

$L(w) < L(v)$ for all $v \neq w \in \mathbb{R}^d$ $\xrightarrow{\text{always}}$ w is global minimum

- Now, we want to know if we can derive $L(w) < L(v)$ for all $v \neq w \in \mathbb{R}^d$ From w is stationary point.
  - Depends on ⇒ $L(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_w(x_i))$ ⇒ Data! 你的数据将决定损失函数(如果是data-driven而不是physics-informed的话）从而决定你的优化问题

Convexity

Mathematically, convexity is the function property that guarantees local = global minimum. We can use following definitions (0-th, 1st and 2nd order) to define our convex function:



- 0-th order condition (if and only if): $L(\lambda w + (1-\lambda)v) \leq \lambda\, L(w) + (1-\lambda)L(v)$

  true function connecting any w, v $\leq$ linear function connecting two points x, y

- 1st order condition (if and only if): $L(v) \geq L(w) + \nabla L(w)^\top (v - w)$

  value at any v of linear approximation at any w $\leq$ true value at v

- 2nd order condition (if and only if): Hessian $\nabla^2 L(w) \succcurlyeq 0$

  non-negative curvature at every w

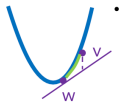
- 零阶最容易判断。⚠️上述定义不能保证global minimu unique，因为会出现如下这种情况。因此我们进一步定义strong convexity. Mathematically, strong convexity is the function property that guarantees **unique** global minimum

  –

  –

- 0th order condition $L(w)$ (if and only if for some $m > 0$): $L(w) - \frac{m}{2}||w||^2$ is convex



- 1st order condition (if and only if for some $m > 0$): $L(v) \geq L(w) + \nabla L(w)^\top (v - w) + \frac{m}{2}||v - w||^2$

  value at any v of linear approximation at any w $<$ true value at v

- 2nd order condition (if and only if for some $m > 0$): Hessian $\nabla^2 L(w) \succcurlyeq mI$

  (strictly) positive curvature at every w

Properties of convexity → Operations that preserve convexity

We care about minimizing the training loss $L(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_w(x_i))$ and want to check convexity fast.

✏️ You can either: (i) check for the 0th to 2nd order conditions, or (ii) look at a list of standard convex functions and show that it is obtained using a known convex function and some operations that preserve convexity:

- The sum $\alpha f + \beta g$ is convex for any $\alpha, \beta \geqslant 0$ if f, g are convex.
- $f \circ g$ is convex if f convex and g affine OR f non-decreasing and g convex (notation: $f \circ g(x) = f(g(x))$)
- $h(x) = \max\{f(x), g(x)\}$ is convex (pointwise maximum) if f, g are convex

Let's pick the square loss $\ell(y, f_w(x)) = (y - f_w(x))^2$. Which of the following statements is true?

(a) $L(w)$ is convex (b) $L(w)$ is convex for $f_w(x) = w^\top x$ (c) $L(w)$ is convex for $f_w(x) = \text{relu}(w^\top x) \coloneqq \max(0, w^\top x)$

– ◦ (a) is not generally true. (b) is strong convex. (c) is only locally convex.

Summary:

A more complete picture for differentiable functions

w is local minimum $\xrightarrow{\text{always}}$ w is stationary point (if convex)

w is global minimum $\xrightarrow{\text{always}}$ w is local minimum (if convex)

$L(w) < L(v)$ for all $v \neq w \in \mathbb{R}^d$ $\xrightarrow{\text{always}}$ w is global minimum (if strongly convex) } The global minimum is unique

Hence for linear regression we see directly how data determines possible minimizers of training loss:

If Hessian $X^\top X > 0 \rightarrow$ only one unique minimum, else ($X^\top X \geq 0$) $\rightarrow$ many global minima!

- ⚠:  This is convex, but not unique.

Data affects model quality: How data affects how good global minimum (of training loss) is for prediction

We have already seen conditions on the data for good prediction on the training data. However, what matters is prediction performance during test time. Therefore, below, we will see how data (via sample size) affects prediction at test time.

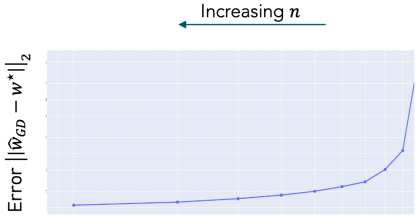
Metric First, we should know how to formalize a good model for regression. $\Rightarrow$ have low loss (average error over possible test inputs x) or close in parameter ($||\hat{w} - w^*||^2$) $\Rightarrow$ $\ell\left(\hat{f}(x), f^*(x)\right) = \left(\hat{f}(x) - f^*(x)\right)^2 = \left((\hat{w} - w^*)^\top x\right)^2$

But recall from our training dataset. Assume that our data follow specific rules. However, due to the noise in real world. the observed data often follows: $y_i = f^*(x_i) + \epsilon_i = w_0^* + \langle w^*, x_i \rangle + \epsilon_i$

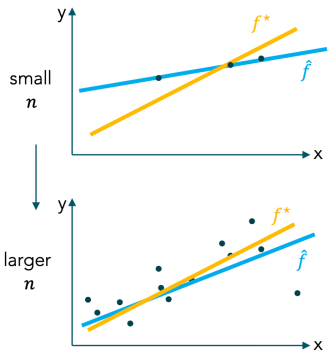
⇓ Next, we will discuss the relation between d and n for the influence of the test dataset.

Increasing n for low-dimensional case $d < n$

Fix $d = 1$, increase sample size starting at small n observing noisy data. What happens with the error of gradient descent solution / minimum of loss?



$n > d$: Underparameterized (unique solution)

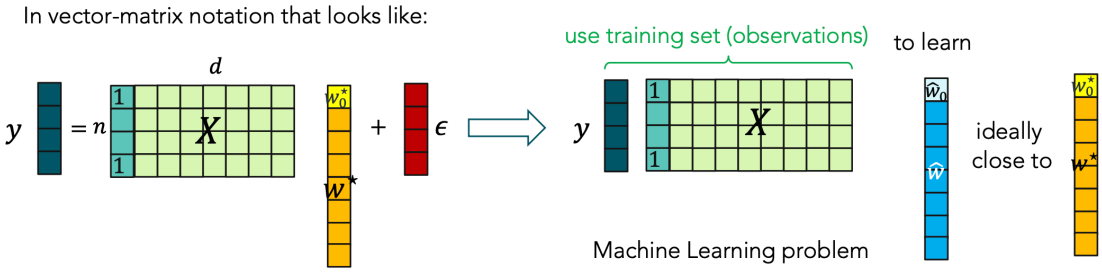


n grows → the effect of noise "gets averaged out"

- The effect of noise "gets averaged out"

⚠: **However, in the the age of big data, a lot of attributes $d \gg n$:**

- Scenario: Recent data from few (n) houses in a small village, but many (d) attributes Or think of the multi-omics example with few (n) patients but many (d) attributes

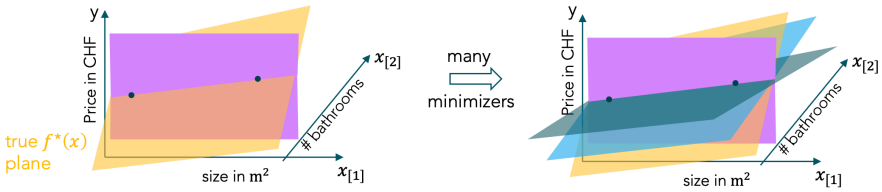


- A fat matrix, not a tall matrix. The number of parameters in our ML model are larger than that of training dataset. 这样的模型能训练成功有点违背直觉，但深度学习里面他们的效果往往出奇意料的好

⇓ 下面我们开始研究这种情况

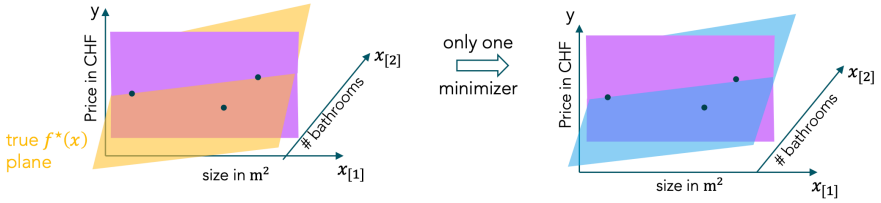
Sample size n increase for noiseless case and $d \gg n$

Let's first assume we have no observation noise (noiseless) $y_i = w_0^\star + \langle w^\star, x_i \rangle$



all inputs of training points • have fixed $x_{[2]}$ only learn how function changes with $x_{[1]}$

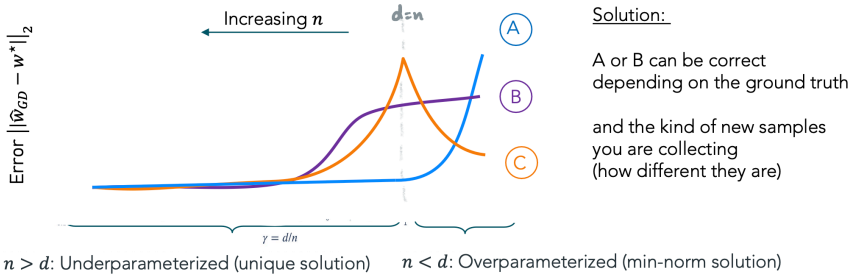
- ⚠ 我们考虑的是训练数据没有误差的情况
- 模型的维度是3，数据的数量只有2的话。我们发现根本无法有效的学习，可能在数据的部分维度上学的很好，但是在有些维度上学的很差。会有many minimizers存在



all inputs of training points • have fixed $x_{[2]}$ now you learn about both directions

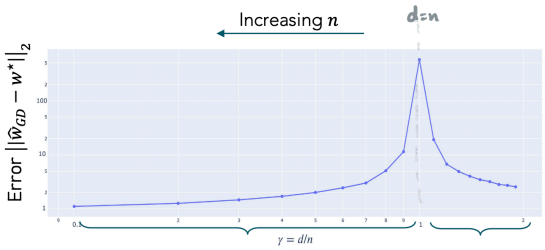
- 模型的维度是3，数据也是3。Now you learn about both directions.

Summary:



- 之所以会出现A, B两种情况是因为我们刚才是在理论上去讨论，但是在实际中，我们是通过测试test data从而画出曲线，如果test data的采样样本不在训练样本的分布下，确实会出现B这种不太理想的情况。

So far, we studied noisy data and saw it helps gradient descent solution on linear regression for both over parameterized and underparameterized situations, which satisfies the most fundamental intuition in statistics and machine learning. Next, we want to know does this carry over to noisy data in the high-dimensional case, $d \gg n$



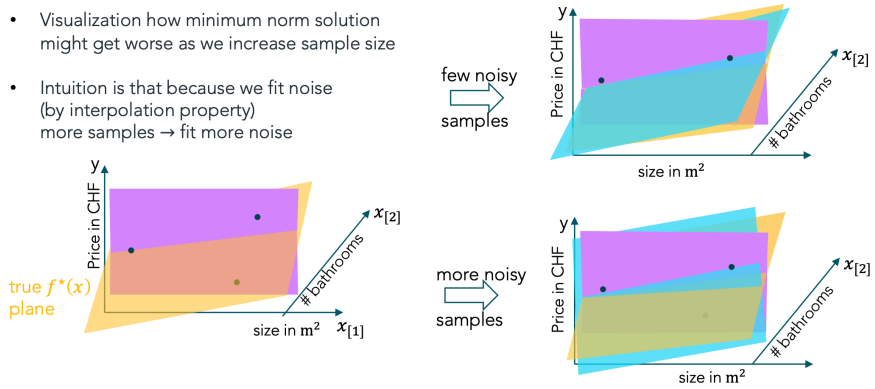
Solution:

C (because of intuition in the next slides)

Bonus: If you think about fixing n and increasing d, the C-curve corresponds to double descent

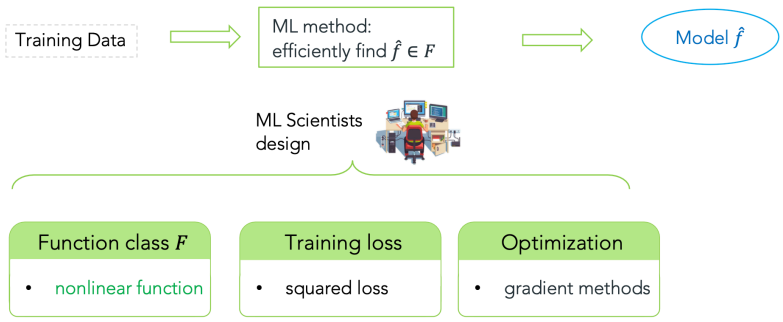
$n > d$: Underparameterized (unique solution) $n < d$: Overparameterized (min-norm solution)

- Violate our intuition!! A new intuition to explain:



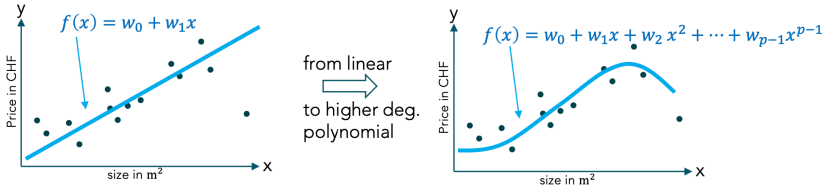
- – Fit more noise give rise to the pertubation of the trained hyperplane.

Going beyond linear functions: a simple way to fit nonlinear functions with tools so far



Example: polynomial functions

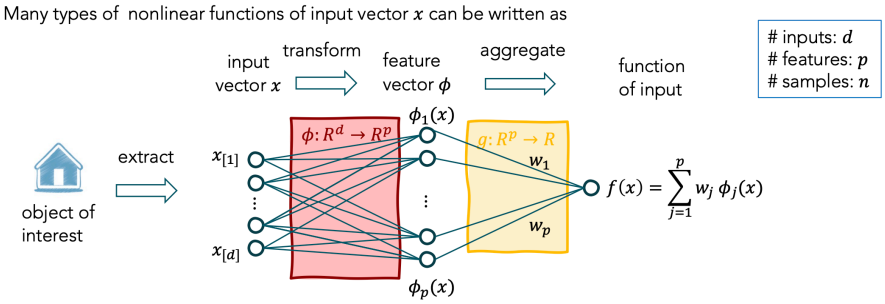
Both can be written as $f(x) = \sum_{j=1}^{p} w_j \phi_j(x)$ with feature vector $\phi(x) = \left(\phi_1(x), \ldots, \phi_p(x)\right) \in \mathbb{R}^p$



Linear function with $\phi_1(x) = 1, \phi_2(x) = x$

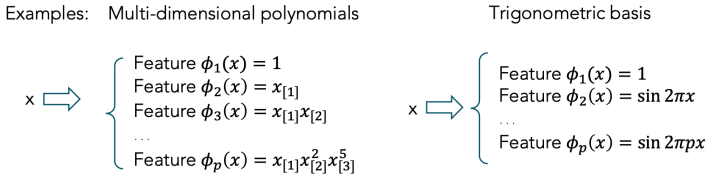
uses additional monomials $\phi_3(x) = x^2 \ldots \phi_p(x) = x^{p-1}$

-

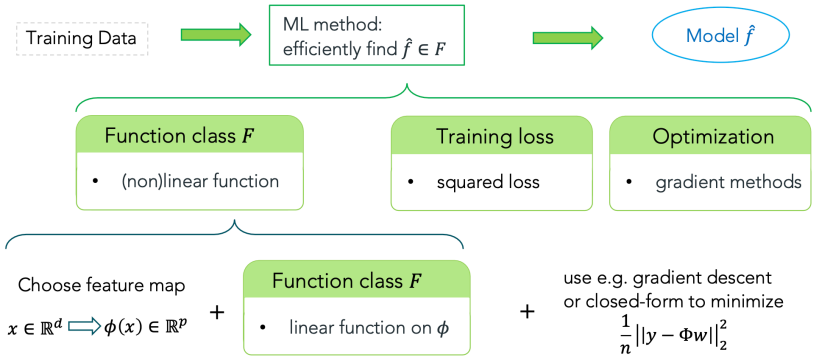


- Linear models are more powerful than you think. 我们可以看到nonlinear regression的本质就是线性回归，只不过它先将input 数据做了一次特征映射，映射到基底上去，从而得到feature vector，接着在做linear regression
  - Some ML models fit data with fixed mappings ϕ: Linear regression, polynomial and kernel regression
  - Others also learn the mapping ϕ when fitting the data: neural networks NN不需要做特征工程

- 还可以考虑其他的basis：



Summary

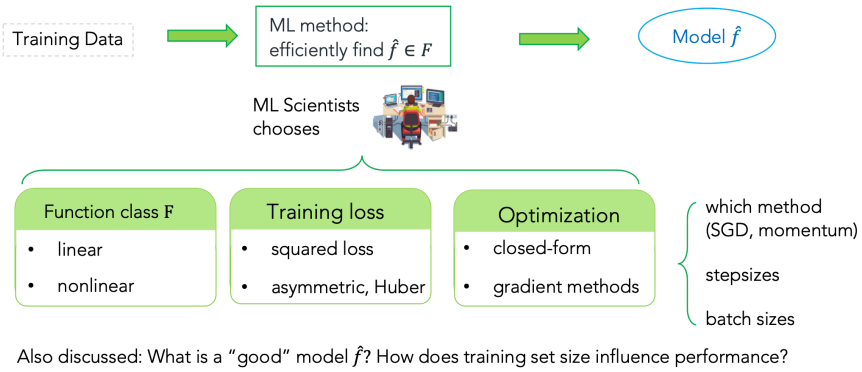


5 Model Selection

In this chapter, we want to discuss what is a good model? Metrics? And how does the model complexity (hyperparameters) influence performance of such model.

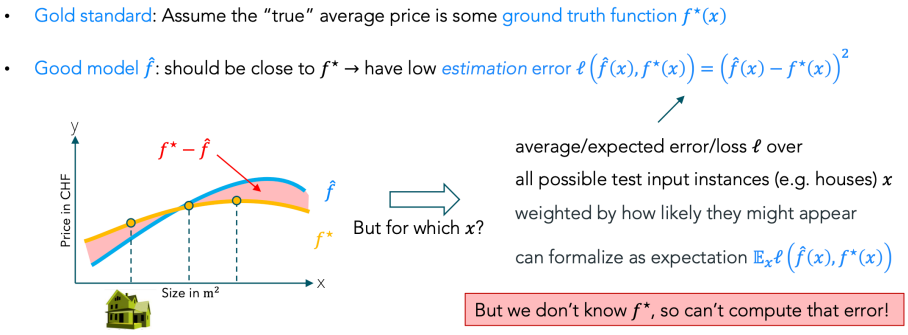
Recap

ML for regression



- Function class F and parameters in the optimization process are both called hyperparameters, which should be tuned during the training process.

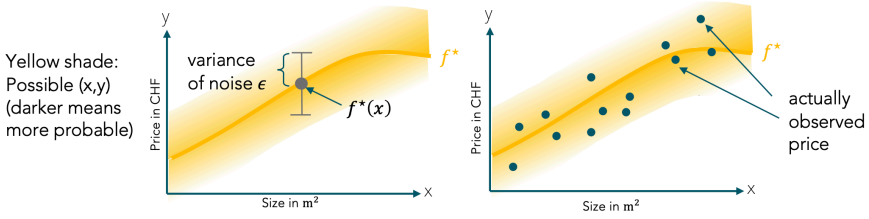
Formalizing "good" model via ground truth



- Seen above. If we can get the expection error of the model under test dataset, we will be able to know the performance of the model. But the brutal thing is that we don't know f^*. It is unable to compute such error! We have to do something else. Let's see if we can use data to do some predictions.

⇓ Firstly, let's better learn that data we use: observed data includes noise

Often, can model observed data as $y_i = f^*(x_i) + \epsilon_i$ where ϵ_i is "noise" and usually zero-mean that could arise from unseen attributes not accounted for, or random measurement error



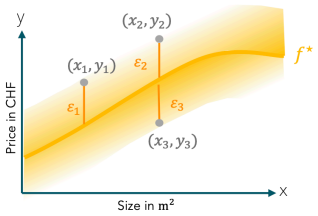
- Truth, our observed data have noise and usually zero-mean. 这时，描述我们的数据集分布就不应该用曲线，而应该用以上波浪
- 因为数据中引入了随机变量，则数据集的获取其实是在这个范围内进行一个采样。

So far, how well we can learn ground truth can be determined by data points (via convexity, via sample size). But other optimization hyperparameters like step size and choice of nonlinear features (function class) will influence our learned function. 在这一章节中，我们还是集中数据对模型的判断和影响，不过我们的重心不在数据对模型的训练上，而在数据对模型performance的估计上。

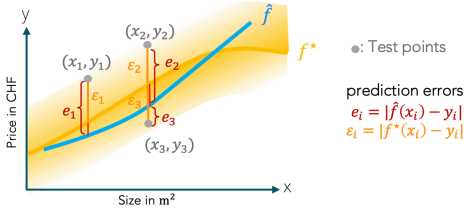
Today: Use data to answer following questions

i) How good is $\hat{f}$ approximately? **ii)** How to choose hyperparameters / the method?

Average prediction error for evaluation: Recall: cannot compute $\ell\left(\hat{f}(x), f^\star(x)\right) = \left(\hat{f}(x) - f^\star(x)\right)^2$ that compares prediction with true value, but can compute prediction error/loss that compares prediction with observed value $\ell(\hat{f}(x), y) = \left(\hat{f}(x) - y\right)^2$



Even $\hat{f} = f^\star$ has unavoidable error ϵ

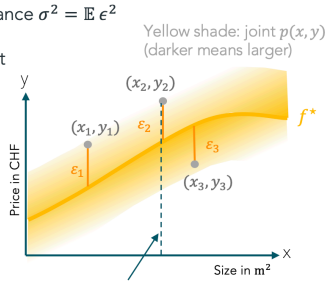


All other $\hat{f} \neq f^\star$ additionally have error $f^\star(x) - \hat{f}(x)$

Statistical model for the training (and test) data

- Recall we assume training points (x_i, y_i) follow $y_i = f^\star(x_i) + \epsilon_i$ with noise ϵ_i that is independent of any (x_i, y_i), mean zero $\mathbb{E}\,\epsilon = 0$. Let's call its variance $\sigma^2 = \mathbb{E}\,\epsilon^2$
- if x have certain probabilities of occurring & points independent → we say points (x_i, y_i) drawn i.i.d. from joint probability $\mathbb{P}$
- For simplicity assume $\mathbb{P}$ has density p, then can write $p(x, y) = p(y|x)p(x)$ and for above model conditional prob. $p(y|x) = p_\epsilon(y - f^\star(x))$, where p_ϵ is density of noise ϵ

MORE LATER...

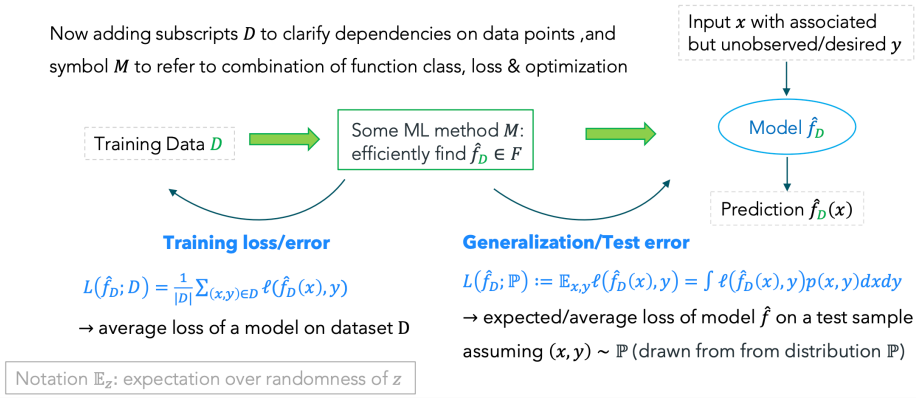


cut of yellow shade along $x = x_2$: $p(y|x_2)$

- 从统计的角度来理解数据集的生成（包括训练数据集训练模型和测试数据集测试模型performance）
- 数据集的生成其实就是从一个联合概率分布$\mathbb{P}$中进行采样： $p(x, y) = p(y|x)p(x)$

Generalization/test error for evaluation

⚠ From discussion before, we know how to evaluate a model. (用模型在test observed data上的期望来表示。这里我们进行总结，更出了一个更一般的形式，不仅要考虑模型在数据集上的表示，还需要考虑数据集的分布。)



- model training 和model evaluation是两回事，要分开去研究

Outline of this chapter

- Methods of approximating generalization error using data
  - training loss
  - simple train vs. test split
  - cross-validation

第一部分的核心就是求generalization error。因为generation error还设计到采样数据集的分布，当我们不知道分布时，我们如何去近似generation error，这里给出了三种方案，一种就是training loss方案，另一种就是train test split方案，最后一种则是交叉验证方案。而generalization error是模型好坏的一个度量，而之所以要度量模型的好坏是为了使得我们选择合适的超参数从而实现最好的模型，这也是本文的主旨：model selection。

- Effect of increasing model complexity
  - on the training error
  - on the generalization error (under- vs. overfitting)

接着，我们进行更深一步的讨论，关于模型的选择这一超参数对其traing error和generalization error的影响，讨论这一章也会有助于我们如何去进行模型的选择。

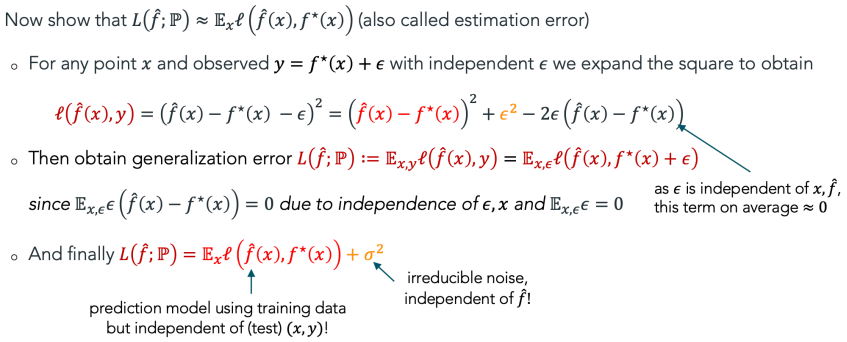
Methods of approximating generalization error using data

Way 1: Training loss

So far, our methods found minimizer of training loss $\hat{f}_D = \text{argmin}_{f \in F} L(f; D) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$ The training loss $L(\hat{f}_D; D) = \frac{1}{n}\sum_{i}^{n} \ell(\hat{f}_D(x_i), y_i)$ of the training loss minimizer is in general a ? approximation of the generalization error $L(\hat{f}_D; \mathbb{P}) = \mathbb{E}_{x,y}\ell(\hat{f}_D(x), y)$

- For small to moderate sample size, training error is not a good estimate!
  - It usually is too optimistic (the chosen model is meant to minimize it)
  - The ranking on the training error does not generalize to ranking on on the generalization error!
- But if sample size includes close to whole population and small noise, then it can be relatively reliable!
- THE Slide has some examples of the training loss ~ generalization error and training loss is too optimstic for generalization error.

Average prediction error for evaluation - BONUS

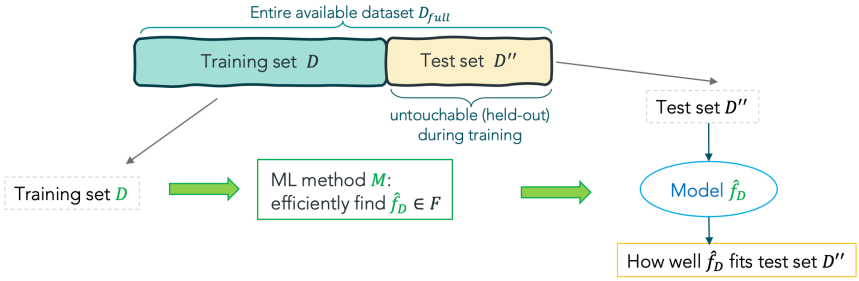


Way 2: simple train vs. test split

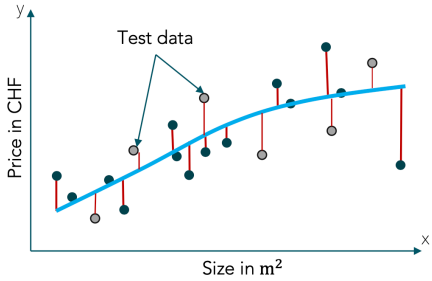
Model evaluation with held-out data

Wanted: A good model $\hat{f}_D$ and expected prediction error for $\hat{f}_D$ (how well it predicts y for expected x)?

Main idea: hold out part of the available data (called held-out or test data) and train on the rest



- 1d example:



During training:

- Only minimize loss on "training points"
- Ignore test points!

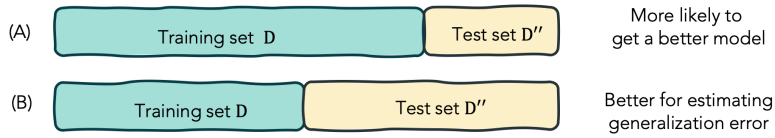
During testing:

Approximate gen. error $L(\hat{f}_D; \mathbb{P}) = \mathbb{E}_{x,y}\ell(\hat{f}_D(x), y)$ by computing *(average) test error* evaluated on the test set D'' drawn from $\mathbb{P}$:

$$L(\hat{f}_D; D'') = \frac{1}{|D''|}\sum_{(x,y)\in D''} \ell(\hat{f}_D(x), y)$$

What's a good train vs. test split?

- How would you split your available data into train and test set?

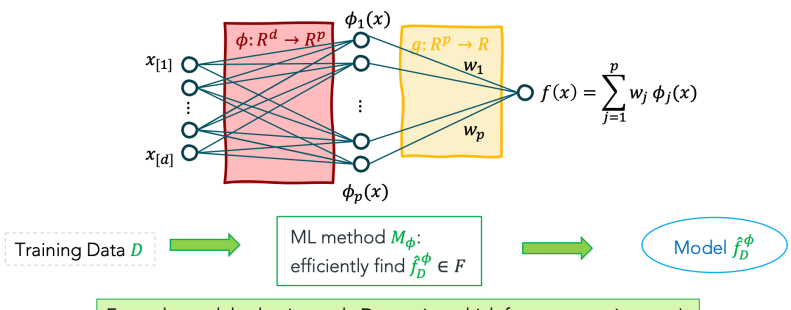


Way 3 Cross-validation

Motivation:

We want to achieve model (method) selection problem

Consider scenario for regression: No prior knowledge that informs which feature map ϕ to choose



Example model selection task: Determine which feature map ϕ to use!

If we achieve model selection using the test error, we will meet some questions:

Assume you use linear regression on different features to fit your data.
Which model would perform best on new data?

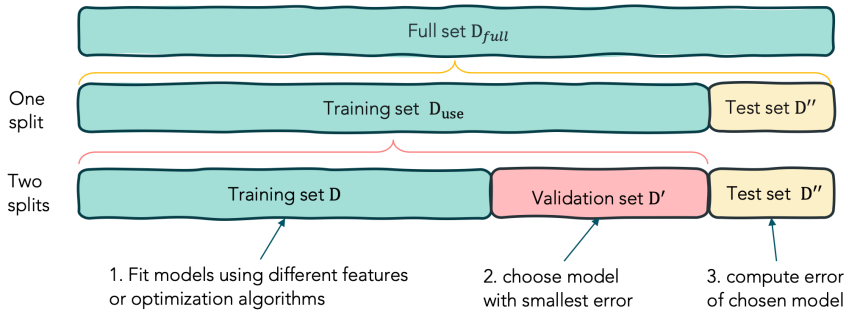
Used features ϕ:	(A) Linear	(B) Polynomial of degree 4	(C) Neural network
Error on test set:	2	0.8	0.5

- – Solution: Depends on test set and error size (and complexity of the neural network)! If test set small, relatively speaking, there would be test set overfitting.

⇓ Therefore, our trick is Train vs. validation vs. test split

Method is chosen to minimize test set error → test error might be too optimistic

Solution: More stability if we use one set to choose and one set to test! 这一部分有一个very tricky的地方，就是test dataset存在的意义就是为了评估generalization error，因此，为什么还需要再进一步划分validation dataset，然后通过validation dataset实现模型的选择后，再在test dataset上苹果model的generalization error。 Solution: 如果我们只进行traing和test的split，然后我们用test的结果来帮助我们进行特征的选择，那么我们如何exptect这个模型在新的test的表现，也就是我们是否又会遇到像上面那种只用training dataset来评估的情况，我们可能会too optimistic。因此这里要明白，知道模型在test dataset上的表示只是度量我们这个模型一个性能的好坏，我们对其的expection。而不是单单地基于此去选择进行model selection。因为基于数据进行model selection的时候，我们默认将此数据也考虑为我们的训练部分了。我们不知道模型在未知数据上的表现。因此最好的方法就是将原先数据集划分三份。

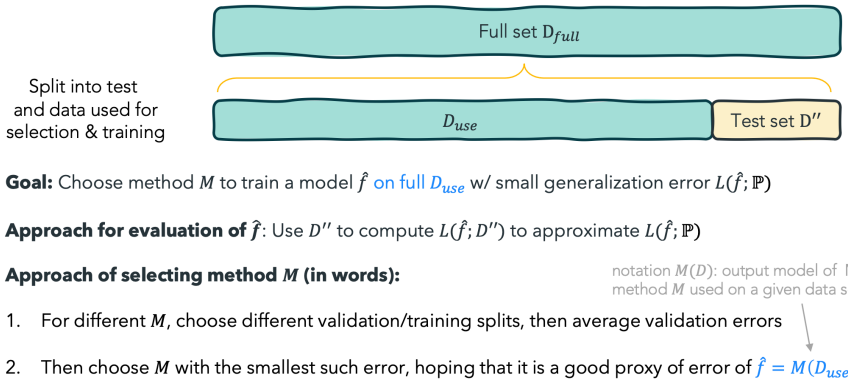


-

Typical splits:	80% 50%	10% 25%	10% 25%

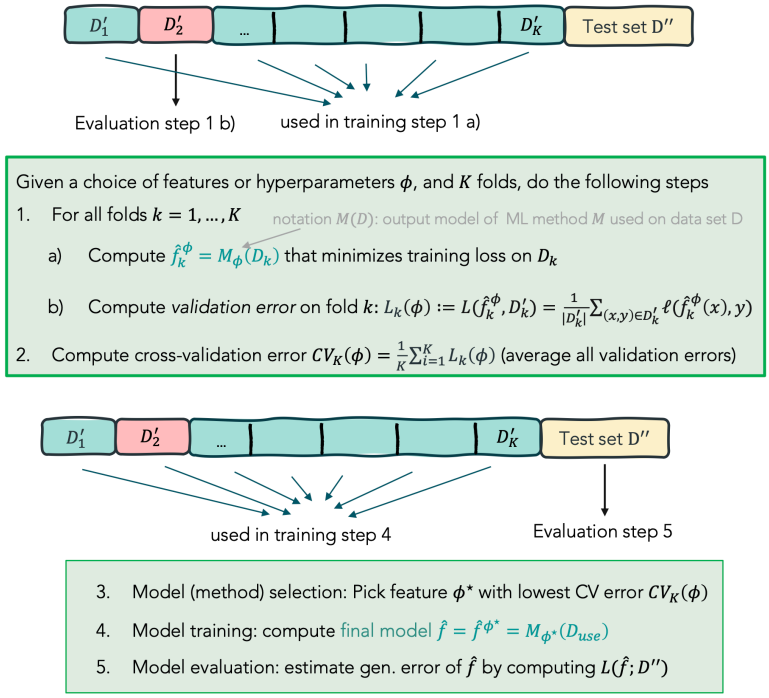
⇓ Next, we want discuss how to choose validation set. And we know that by we have many ways to choose the validation/training split which all yield slightly different models. BUT the problem is that Often dataset is quite small (remember e.g. multi-omics data with few patients) and it's wasteful to set aside both hold-out validation set AND test set. Therefore, that is the reason why we propose cross-validation.

Idea behind cross-validation



- Validation dataset用来进行model selection，如果没有交叉验证，同样可以用validation dataset来做early stop，从而防止过拟合。而test dataset则是用来给generalization error，用来给我们一个直观的认识，模型的表示能力如何。

Specific procedures of K-fold CV for model selection



Note I: This general framework works for comparing any design choices for methods M and losses ℓ
Note II: Aim of model (method) selection is to find a method such that generalization error is small!

- **Aim of model (method) selection is to find a method such that generalization error is small!** 让模型具有很好的泛化性。Aim of CV is to find a method M such that generalization error $L(\hat{f}; \mathbb{P})$ is small where $\hat{f} = M(D_{use})$
  - More details in the lecture notes about how (well) can cross-validation work? and the influence of different K to the model selection. 视频57 minitue

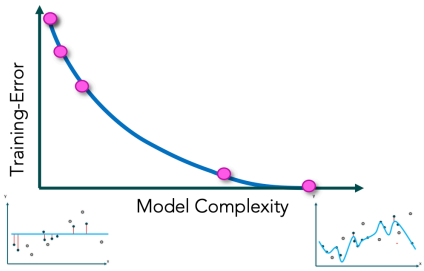
Effect of increasing model complexity

The following slides should develop intuition for how the choice of model complexity affects

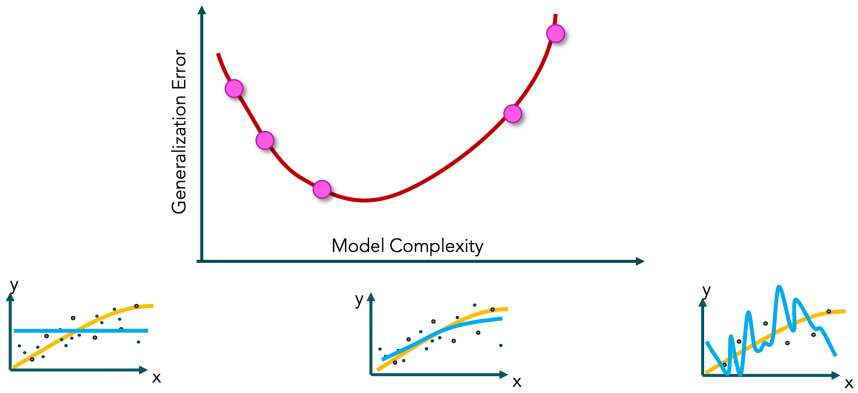
- training error (ability to fit difficult functions)
- generalization error (ability to find the “right” function through finite noisy samples)

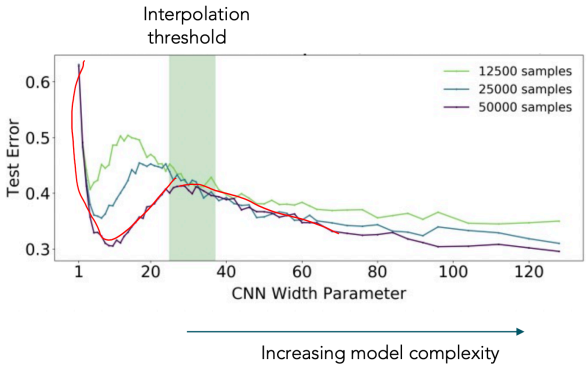
In the next example, we illustrate it back to 1-d polynomial regression and model model complexity by the polynomial degree

Model complexity and the training error



Model complexity and the generalization error

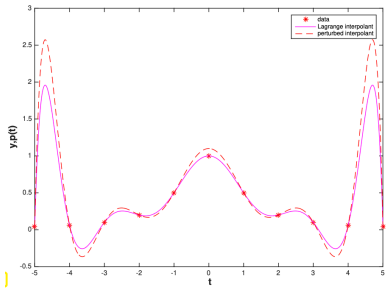


- 在深度神经网络中，我们会发现，随着模型复杂度的进一步上升，generalization error又会进一步下降，这确实太令人迷惑了。
  - **Double descent in practice:**
  - 

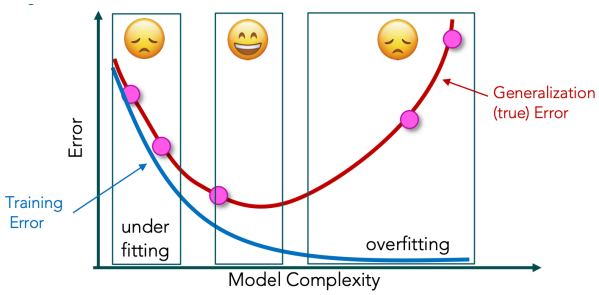
    Nakkiran et al. '20
    - Experiments on image dataset with 10% label noise

Some complementaries

对于有些模型（polynomial approximation），当模型的参数（model complexity）达到训练数据集的数量时，这时候训练的结果（使得误差函数最小）会使得模型实现线性插值的效果，即在训练误差上为0。但是，在数值中，我们知道，算法的稳定性概念，即当插值点中有些数据发生一些变化时，拟合后的模型会发生巨大的变化。在数值插值中，我们证明了equidistant node插值会导致巨大的不稳定性，而chebyshelv nodes则有更好的稳定性。这里回顾这个概念其实就是对算法generalization的另一个角度理解。这也是为什么我们在拟合数据时，一定不要使得模型达到了过拟合的效果。



Summary: Phenomenon of under- and overfitting



underfitting models predict	"right" models predict	overfitting models predict
• training data not well • test data not well	• training data well • test data well	• training data very well • test data not well

6 Bias-variance tradeoff & Regularization

Recap

In the last lecture, we talked about that the observed data will have random noise independent ((x, y)), mean zero, average size $\mathbb{E}\epsilon^2 = \sigma^2$. If inputs x follow some distribution, both training and test data are i.i.d. draws from the above distribution, that is $(x_i, y_i) \sim \mathbb{P}$. But y is dependent to x. And the machine learning algorithm is to approximate this $\mathbb{P}(y|x)$. If we want to approximate the generalization error, we can compute average loss on test set D'' drawn from $\mathbb{P}$. And we can use K-fold cross-validation to select the model achieving the low generalization error. By the way, you can also use **bagging trick** after conducting CV:

K-fold cross-validation is NOT doing the following, which one could also do

- After model selection using cross-validation error, instead of outputting model $\hat{f}_{D_{use}} = M(D_{use})$

  for a new test input x, could simply output the average of the ensemble as a prediction

$$\hat{f}(x) = \frac{1}{K}\sum_{k=1}^{K} \hat{f}_{D_i}(x)$$

- This is called bagging, comes from bootstrap – aggregating (will hear later what bootstrap is).

In today, we will talk about following question:

- Why does generalization error follow a u-curve? [the orgin of error]
  - Bias-variance decomposition of the squared error in the noisy and noiseless case
- How to control model complexity to find min of u-curve? → regularization
  - Methods: polynomial degree, Lasso regression, ridge regression
  - Intuition behind the Lasso
  - Bias and variance as a function of regularization strength

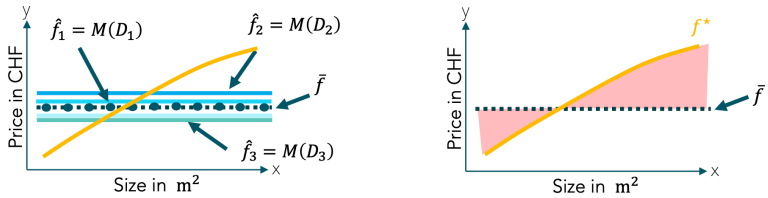
- Cross-validation to determine regularization strength

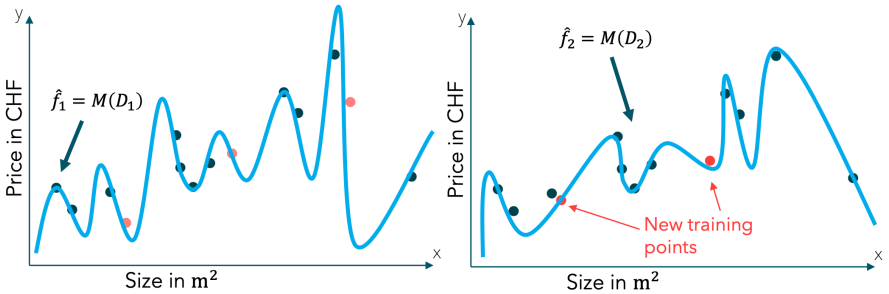
Definition of Bias and Variance

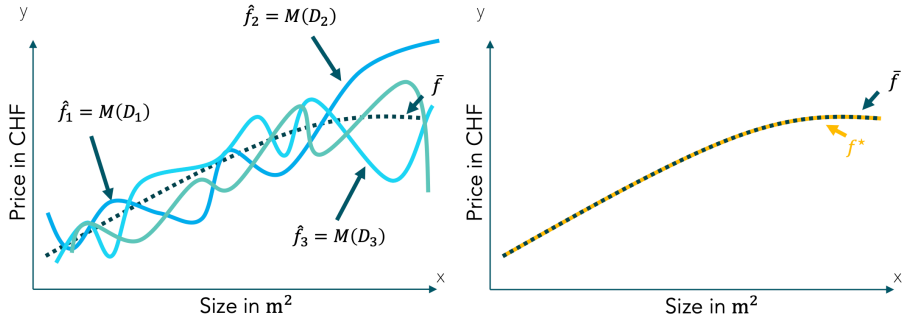
Bias and Variance都是针对模型/方法层面。主要用来表达你采用的这种机器学习模型/方法效果如何，对generalization error有什么影响！

Use dataset from same distribution to train infinity models. Calculate the averaged model. **Bias of method** M: distance of average model $\bar{f} = \frac{1}{J}\sum_{j=1}^{J}\hat{f}_j$ to the ground truth $\mathbb{E}_x\left(f^\star(x) - \bar{f}(x)\right)^2$. [⚠rigorous definition应该是$J \to \infty$情况下得到的平均模型，近似于在该分布上采样数据集的期望。此外，CV的几个fold也可以看成是average model.]

Example

- Bias of too simple models
  - 

  - Even if I average as much as I can, I won't get the ground truth. 这涉及模型/方法的表示能力
- Bias of complex models
  - 

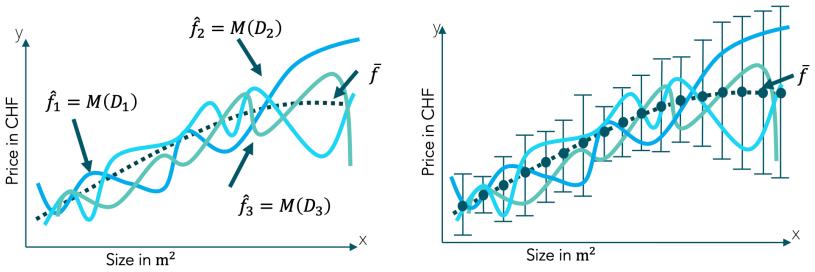
  - 不同的dateaset训练得到的模型变化巨大，在所有数据集上平均下来反而比较接近ground truth
  - 

    - Complex functions can model the ground truth well on average!

Variance of method M: average distance of individual models to average model $\mathbb{E}_x\frac{1}{J}\sum_{j=1}^{J}\left(\hat{f}_j(x) - \bar{f}(x)\right)^2$. the rigorous definition of variance refers to the limit $J \to \infty$ and reads $\mathbb{E}_x\mathbb{E}_D\left(\hat{f}_D(x) - \bar{f}(x)\right)^2$ with $\bar{f} = \mathbb{E}_D\hat{f}_D$

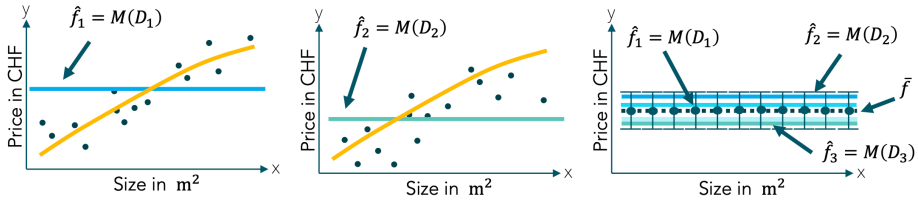
Example:

- Variance of complex models



  - Variance (error bars) is large for complex models!

- Variance of too simple models

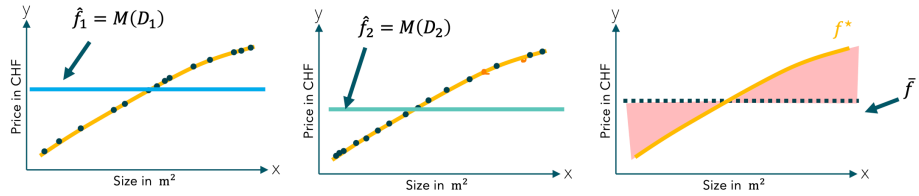


  - Variance for simple models is small!

❗ We know that the generalization error is closely related to the model complexity, and we want to decompose the effect of model complexity on the generalization error.

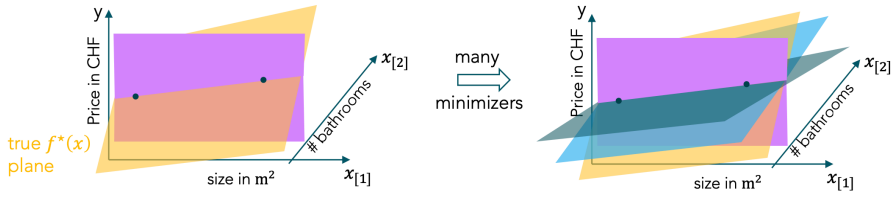
❓ Can the generalization error> 0 if there's no observation noise in regression, that is, $y_i = f^*(x_i)$?

- Bias of too simple models in the noiseless case
  -



  - Even if I train simple model using many noiseless points, the bias persists (but no variance!)

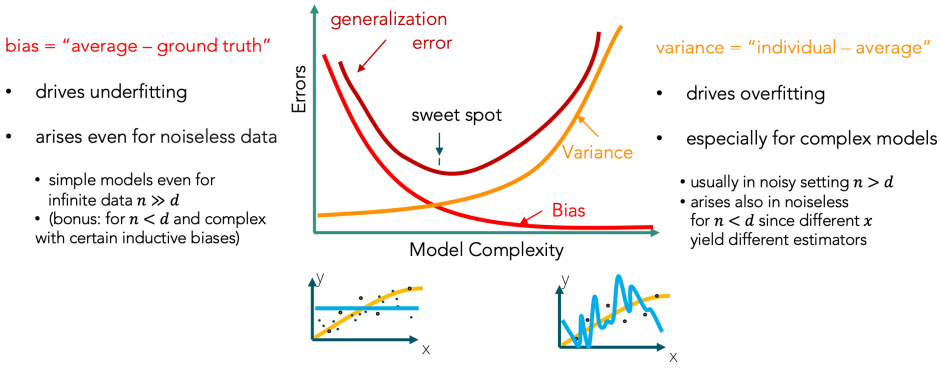
- Variance in the overparameterized noiseless case
  - Back to the example of noiseless linear regression with $d > n$
  -



  - Even if I train a complex model using few noiseless points, variance persists.
  - 复杂模型的情况就是你的模型一定能完美拟合你的数据。最后下来不会有bias error，而是variance error。

- 这个地方稍微有点绕，我的intuition如下:当你的模型非常复杂的时候，数据上很小的扰动都会造成你模型很大的变化，所以你模型非常容易收到数据的影响，不同的数据集去训练最后的得到的模型天差地别，这就是模型的variance很大。对于复杂的模型，即使你用的是non-noise的data，但当你的模型复杂度超过你的数据时，你的模型都能最小话training error，但是最小化的training error不是唯一的，无穷多种选择，这也导致了你模型的variation很大。因此，这样理解：所谓的variance就是数据集变动是模型是否会受到强烈的影响。而bias则是模型的表示能力如何，是否具有表达近似此问题的能力。

⚠️：Summary

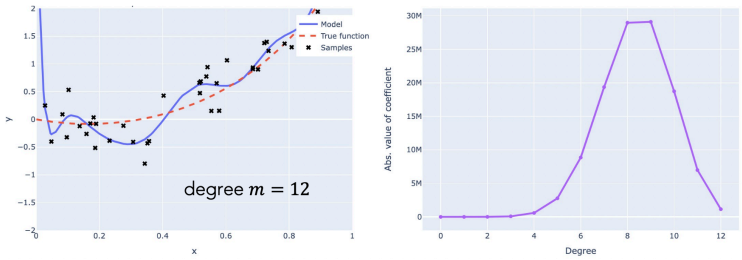


- 作业题中我们会证明generalization error的具体形式如下：generalization error=bias +variance （当然使用严格定义论证的。基于期望，极限的情况。而不是实际情况中我们选择的优先情况）。一个是average - ground truth的期望，另一个是individual - avarage的期望。

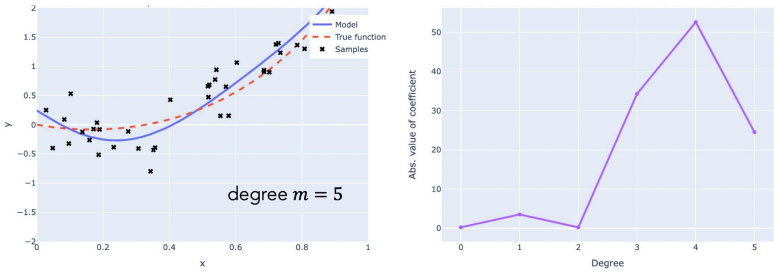
⇓ Next, we will discuss how to control model complexity to find min of u-curve. That is regularization!! We will use polynomial to illustrate our method: lasso regression and ridge regression

Example: Our task is to use polynomial features for regression

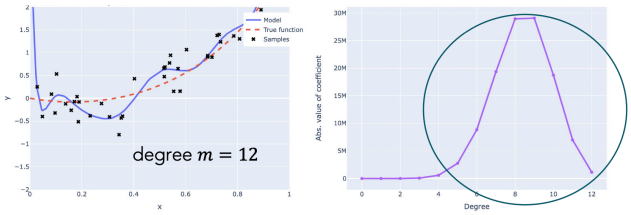
- *Option 0*: Fit polynomial of many degree



- *Option 1*: Reduce polynomial degree manually (run linear regression with features to degree $m = 5$)



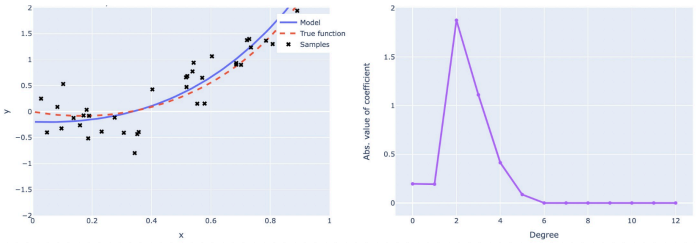
- *Option 2 (ideal)*: Only use a few of these features (enforce sparsity) – not knowing which!
  - 也就是虽然degree很多，但是我们强制降低模型复杂度，使得很多的degree为零，这里我们称之为sparsity。



  - Observe: Complex model uses too many features! Can we force it to use fewer?
  - Ideally: $\underset{w \in \mathbb{R}^d}{\text{argmin}} \; ||y - \Phi w||_2^2 \quad s.t. \quad ||w||_0 \leq k$ solution with limited number of non-zeros (sparse)
    - need to search for best subset of size $k \rightarrow$ combinatorial, infeasible for large $d \rightarrow$ use l_1- relaxation
  - Only use a few of these features, encourage sparsity via limiting l_1-norm: Lasso regression
  -

$$\underset{w \in \mathbb{R}^d}{\text{argmin}} \; ||y - \Phi w||_2^2 \quad s.t. \quad ||w||_1 \leq R \longleftrightarrow \underset{w \in \mathbb{R}^d}{\text{argmin}} \; ||y - \Phi w||_2^2 + \lambda ||w||_1$$

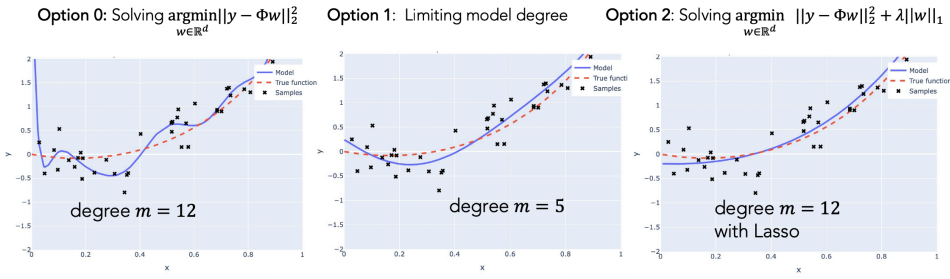
Remember $||w||_1 = \sum_{i=1}^{d} |w_i|$



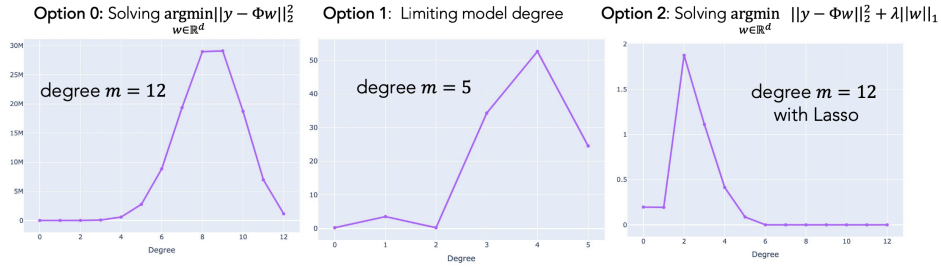
choosing $\lambda = 0.001$

  - 绿色方框中的是等价的，在优化书上有，注意这里的λ也是一个超参数。会影响模型训练效果。

- *Options 0 vs. 1 vs. 2*



  - Lasso does at least as well as manually limiting the polynomial degree!

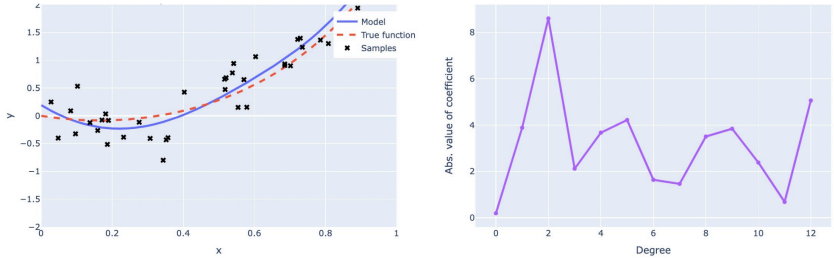


  - In fact, Lasso automtically focuses on lower degree polynomials!

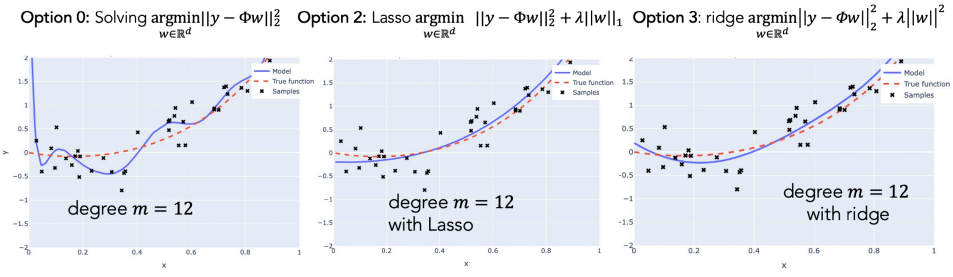
- *Option 3* Only choose models with limited l_2-norm (ridge regression)

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \ ||y-\Phi w||_2^2 \quad \text{s.t.} \quad ||w||_2^2 \le R \qquad \overset{\text{given } R}{\underset{\text{equivalent for some } \lambda}{\longleftrightarrow}} \qquad \hat{w}_\lambda = \underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \ ||y-\Phi w||_2^2 + \lambda||w||_2^2$$

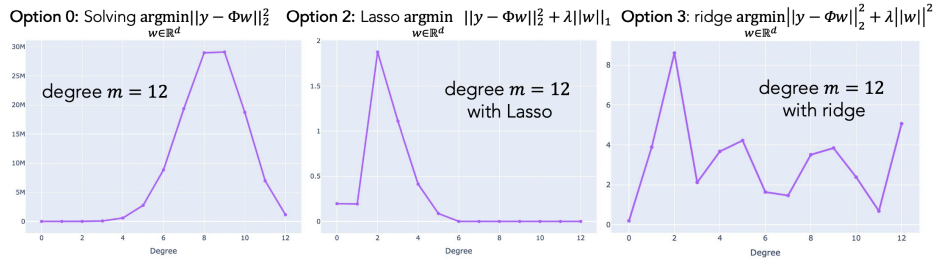
- Equivalence established via Lagrange duality (→ e.g. Optimization in Data Science)
- Closed-form solution is simple by finding stationary point: $\hat{w}_\lambda = (X^\top X + \lambda I)^{-1} X^\top y$
- Can also use gradient methods directly. In next homework you'll see: adding ridge penalty also helps to converge faster, as problem becomes better conditioned via changing max and min eigenvalue λ_{max} vs. λ_{min} of the Hessian.



- *Option 0 vs. 2 vs. 3*



- Lasso and ridge estimators look kind of similar in terms of closeness to the ground truth.



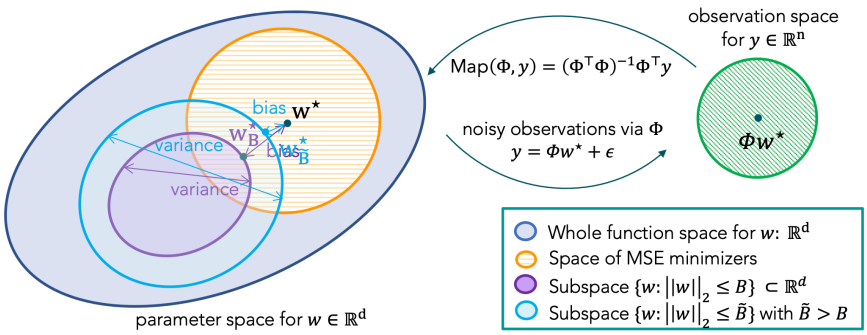
- In fact, ridge puts more weight on lower degrees but doesn't set higher ones to zero!

⚠ Next, we will give the intuition of the difference behind the Lasso and ridge. See the lecture for more details. I will complement it in the future

Bias and variance as a function of regularization strength

改变regularization strength改变的其实是能够决定的function space，其会导致限制增加，从而使得模型复杂性降低，则bias增加，variance降低。there is less flexibility to fit noise → less variance, but also less flexibility to fit the ground truth → more bias

Cartoon: Varying model complexity reflected via sizes of ellipses (representing norm balls) for $n > d$



I will complement **Effect of ridge penalty on the generalization error** in the future

Cross-validation to determine regularization strength

Instead of feature maps ϕ, question now is how to "optimally" choose λ?

- In the plots we chose the one that minimizes generalization error – that is of course cheating as the generalization error is uncomputable

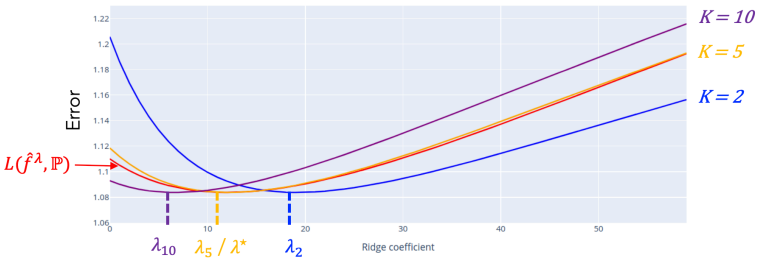
The regularized estimator $\hat{f}^\lambda = \langle \hat{w}_\lambda, \cdot \rangle = M_\lambda(D)$ where $\hat{w}_\lambda$ is the minimizer of the regularized training loss $\hat{w}_\lambda = \text{argmin}_w \, L(w; D) + \lambda \, ||w||_2^2$ with $L(w, D) = \frac{1}{|D|} \sum_{(x,y) \in D} \ell(w^\top x, y)$.

✏️ Now: in yesterday's model selection workflow with cross-validation, replace ϕ by λ

- Same tricks as last chapters. Details see the lecture notes

Best hyperparameters using K-fold CV vs. oracle

How does best hyperparameters (method) chosen by cross-validation compare with truly best hyperparameters chosen by generalization errors? → dependence on K see demo



- We plot the cross-validation errors of K-fold CV and the generalization error of "full" model $\hat{f}^\lambda$
- Here λ_K is the λ that achieves smallest cross-validation error for a given K
- We see that the CV errors for $K = 5$ and generalization errors $L(\hat{f}^\lambda, \mathbb{P})$ match well.

7 Classification

- Binary classification pipeline
  - Prediction and evaluation
  - Function class: Linear classifiers

  - Training loss: Comparison of surrogate losses
- Linear classification
  - logistic regression, hard-margin SVM and max-margin classifiers

we focused on linear classifier or linear classifier with nonlinear features which is nonlinear.

When we conduct the classification, the first thing that we need to do is to assign number to the labels. The tool that you used to classify is a function: $\hat{f}: \mathbb{R}^d \to \mathbb{R}$. The final prediction is $\hat{y} = \text{sign}(\hat{f}(x))$. 相当于是两个function套起来。

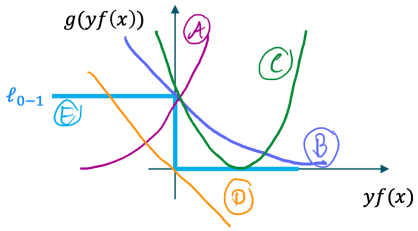
The prediction error is $\mathbb{I}_{y \neq \text{sign}\hat{f}(x)}$(indicator function), whose output is 1 or 0.

Definition of indicator fct $\mathbb{I}_{\{A\}}$ for event A:

$$\mathbb{I}_{\{A\}} = \begin{cases} 1, & A \text{ is true} \\ 0, & A \text{ is false} \end{cases}$$

$$\widehat{R}_{\mathcal{D}}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{\{y_i f(\boldsymbol{x}_i) \leq 0\}} = \frac{1}{n}\sum_{i=1}^{n} \ell_{0-1}(y_i f(\boldsymbol{x}_i)).$$

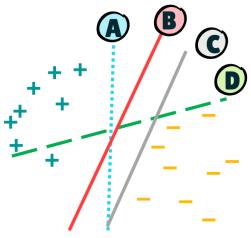
- How we evaluate a model during the test time. [Zero-one loss 至少可以用来在test dataset上做评估，但是 discontinuous and non-convex, hence difficult to minimize using usual optimization methods. To ease the optimization procedure, it is common practice to replace the zero-one loss with some well-behaved upper-bound, generally referred to as a *surrogate loss function*.]



  - Furthermore, we discuss compare this surrogate loss and focus on the **logistic regression** → gradient descent on the logistic loss. After training our model, during test time, we use zero-one loss as our prediction error.

所有的机器学习训练理论上都得基于数据的概率分布求期望。但是这在实际中是不可能实现的。因此实际中我们都是基于有限的数据集进行学习。就好比我们是先采样。基于采集的样本去估计整体的分布。

For linearly separable data on the right, which of the decision boundaries may we end up with if we run GD long enough?



- Use logistic loss for optimization, we will reach to the solution of boundary B. It is hard to explain.
- This further leads to SVM.

- Any question is that dose the minimizer converge to the minimum? [This is difficult to explain, we will see it at homework.]
  - Because the norm of the weight can be infinity. And the boundary didn't change.
  - 但是有一点我们可以确定，至少direction can be converge。也就是w的direction can be converge。至于norm是多少，我们真不能确定。然后四五年前有一拨人证明过当你用logistics loss，你最后会收敛到B这个方向。
    - 稍微给了点证明，就是说Assuming linearly separable data, gradient descent (initialized at 0) on the logistic loss eventually directionally aligns with the unit-norm vector w_{MM} that maximizes the minimum l_2-distance (among all points) to the boundary (inductive bias* of GD on logistic loss if data is linearly separable).
      - In particular, we can write $w_{MM} = \text{argmax}_{||w||_2=1} \min_i \underbrace{y_i \langle w, x_i \rangle}$

        equal to distance of point (x_i, y_i) to boundary for w with $||w||_2 = 1$ that linearly separates the data, i.e. for all i: $y_i \langle w, x_i \rangle = |\langle w, x_i \rangle|$
      - $\text{margin}(w) = \min_i y_i \langle w, x_i \rangle$ 这个称之为margin。如果norm of $w = 1$时margin就是距离。
      - $$w_{MM} = \text{argmax}_{||w||_2=1} \min_i y_i \langle w, x_i \rangle =: \text{argmax}_{||w||_2=1} \text{margin}\,(w)$$
    - 但是真实情况w的norm不等于1。这个时候我们就是在一个大的参数空间去搜索🔍。最后我们解决的是这个优化问题：
      - $$w_{MM} \parallel \underbrace{\text{argmax}_w \min_i y_i \left\langle \frac{w}{||w||_2}, x_i \right\rangle = \text{argmax}_w \frac{\text{margin}(w)}{||w||_2}}_{\text{(note that solution is not unique anymore, similar to log. reg.)}}$$ (parallel)
      - 但是这个优化问题的值就不唯一了。但是其最后w的方向是平行于w_{MM}，这也是我们为什么称其w的方向可以收敛，但是最后的解不收敛。
      - 后面引出了这个与SVM的关系这里顺带提一句，其实我们使norm = 1其实就是强制使优化问题收敛，解唯一w_{MM}。但是我们也可以不在这么tight的space去找解，也可以在更大一点的空间。就是这里的svm
        - For general vectors w that correctly separate the data, the min distance to the decision boundary is

          $$\frac{\min_i |w^\top x_i|}{||w||_2} = \frac{\text{margin}(w)}{||w||_2} = \frac{\text{margin}(\alpha w)}{||\alpha w||_2} \text{ for any } \alpha \in \mathbb{R}$$
        - We can hence rescale any unit norm w by $\alpha = \frac{1}{\text{margin}(w)}$ i.e. $\widetilde{w} = \alpha w$ such that

          $\text{margin}(\widetilde{w}) = \min_i y_i \widetilde{w}^\top x_i = \alpha \,\text{margin}(w) = 1$ with min. distance to dec. boundary $\frac{\text{margin}(\widetilde{w})}{||\widetilde{w}||_2} = \frac{1}{||\widetilde{w}||_2}$
        - Hence, instead of search within unit norm w to find w_{MM} with maximum min-distance $\text{margin}(w)$

          we can search within all $\widetilde{w}$ with $\text{margin}(\widetilde{w}) = 1$ to find the one that maximizes min-distance $\frac{1}{||\widetilde{w}||_2}$

          $w_{MM} \parallel \text{argmin}_{\widetilde{w}} ||\widetilde{w}||_2$ s.t. $y_i \widetilde{w}^\top x_i \geq 1$ for all $i = 1, \ldots, n$ Support Vector Machine (SVM)!

8 Classification II

Note: in max-margin problem we fix norm and maximize margin, in the SVM we fix margin minimize norm. [如果我们不考虑max-margin和SVM，则最后的解不一定唯一。]

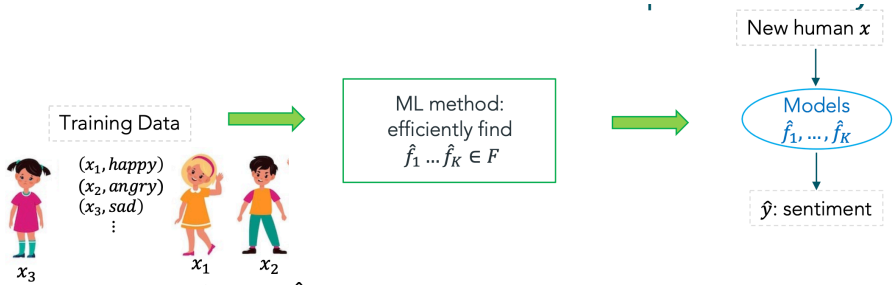
- They actually find the same solution.

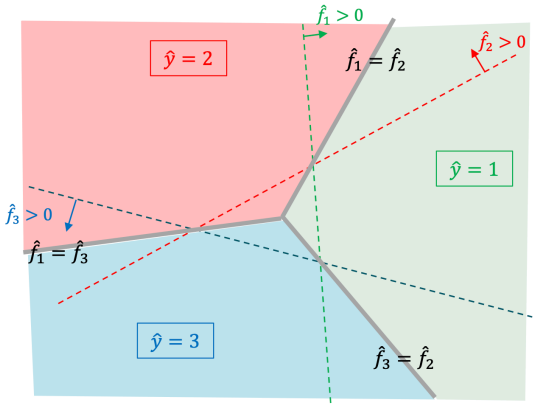
not every convex function has a unique minimizer take $\min_z \exp(-z)$. But if there is a minimizer, it is global. (a good property)

- multi-class classification
  - prediction with multiple models
  - training: one-vs-rest and cross-entropy loss
- asymmetric losses

Given one function $\hat{f}$ how do we obtain $\hat{y}$?

- If the labels are related, e.g. no vs. small vs. severe risk, one $\hat{f}$ could be enough...
- Often, labels are not monotonically related → need different models $\hat{f}_i$!
  - 也就是说如果是相关的样本，我们可以用一个f的大小，来表示不同程度从而奖励threshold。就是疾病是无，还是轻微，还是严重。但是，如果这些样本是毫无关系的，比如要分类的是猪还是猴子，你很难通过单个函数找到一个标准建立threshold来区分他们。由此看来，在二分类中，我们只存在是或不是两类，通过以0对称确实是一个非常好的方法。





Linear example with $K = 3$ for 2-dimensional x

- Say we map class $- \rightarrow y = 1$,
  class $+ \rightarrow y = 2$, class $* \rightarrow y = 3$
- Assume we are given $K = 3$ linear models $\hat{f}_k$
- We can predict $\hat{y}(x) = \text{argmax}_{k=1...K}\ \hat{f}_k(x)$
  → decision boundaries (gray solid)
  between classes i, j at $\hat{f}_i = \hat{f}_j$
  (illustration for the same slopes for all $\hat{f}_k$)

- 实际上，我们通过三个函数的相等来划分我们的分界线是一种非常方便的做法，因为预测结果是，三个函数运行得到的结果，谁的结果大，则是哪一类。
- 其实很好奇为什么不能通过三个函数是否大于0这种来分解，但这样在我的视角中的问题就是预测的结果很不方便，要逐一笔记才能得到结论

⇓ 如何设置我们的loss function

一个naive approach就是：one-vs-rest （也就是三个模型逐一从行标签。然后按照二分类来进行训练。最后训练得到的模型就有点像上图，最后我们人为规定的实现$\hat{f} \rightarrow \hat{y}$的转换会稍微与最开始训练得到的边界有些不同，虚线就是我们用one-vs-rest训练得到的边界。

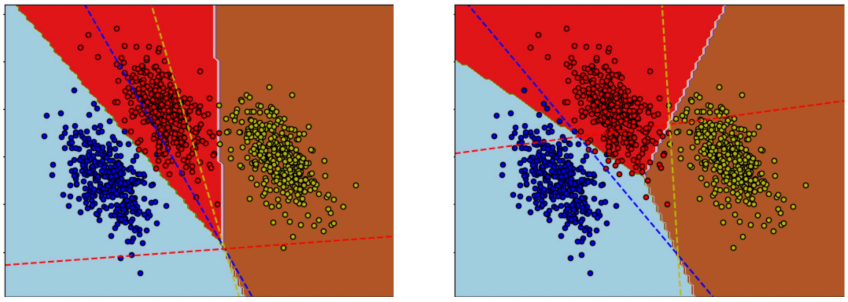
但是，这样做会遇到一些困难：Have to run K separate optimization procedures! Can we train them simultaneously?

因此我们就提出了simultaneous approach: Cross-entropy loss

- The most popular approach is to run e.g. gradient descent on the cross-entropy loss:

here, $f(x) = (f_1(x), ..., f_K(x)) \in \mathbb{R}^d$ → $\ell_{ce}(f(x), y) = -\log\left(\frac{e^{f_y(x)}}{\sum_{k=1}^{K} e^{f_k(x)}}\right)$ ← $f_y = f_i$ where $i = y$

  - 上图是在每一个训练数据点上的交叉熵。这个交叉熵其实挺合理的，其最后的效果就是在哪个$f(x)$上表现最好，最后的预测结果就是什么。



(A) (B)

Derivation of cross—entropy as a natural extension of the logistic loss in detail later

- In this lecture, we have some derivations about how to derive the cross-entropy via the softmax.

- asymmetric losses

- asymmetric errors: false positives, false negatives
- ROC curves and AUROC
- more asymmetric metrics: precision, recall, F1

这一张讨论的是，我们前面讨论了很多的surrogate loss for the 0-1 loss。But is 0-1 error really a good metric?

- 关注的重点是在per-class上我们是否有想要的performance（或许是equal，或许是differnt）

When errors count differently for binary classification, you often hear terms like false/true positives, false/true negative, ROC Curve, precision, recall reminiscent of **hypothesis testing** in statistics.

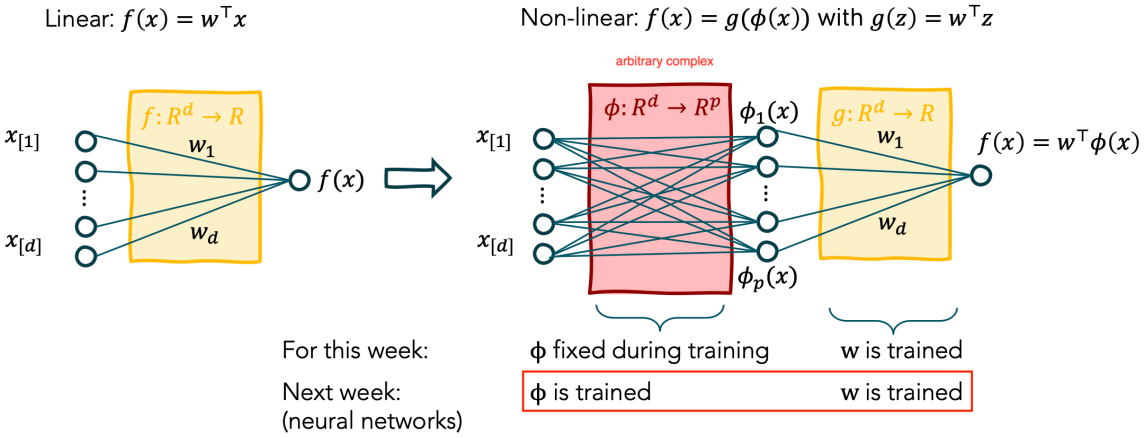
9 Classification & Kernels methods

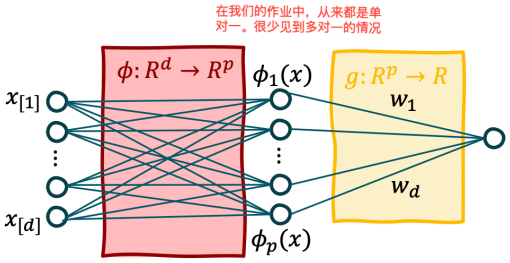
9.1 From non-linear features to kernels

Outline

- Complexity of polynomial regression
- kernel trick/kernelization
- Kerneled (ridge) regression for polynomial features

For some cases, we need to some non-linear functions. We can transfer our linear functions → nonlinear functions via feature maps (imposed on the original training data)





- If we choose at least ϕ_i as nonlinear feature maps

  → the final function $f(x) = w^\top \phi(x)$ can be nonlinear!

- Examples
  - polynomials $\phi(x) = [1, x_{[1]}, \dots, x_{[d]}, x_{[1]}^2, \dots, x_{[d]}^2, x_{[1]}x_{[2]}, \dots]$
  - Fourier features $\phi(x) = [1, \sin x, \cos x, \sin(2x), \cos(2x), \dots]$
  - neural network $\phi(x) = \sigma(W_n \sigma(W_{n-1} \dots \sigma(W_1 x)))$

    elementwise activation functions

Specifically, we solve the corresponding optimization problem: Solving $\hat{w} = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \ ||y - \Phi w||_2^2$ and outputting model $\hat{f}(x) = \langle \hat{w}, \phi(x) \rangle$

9.1.1 Complexity of polynomial regression

The reason why we don't want to use polynomial features is that it is too costful. We will encounter the feature explosion for polynomials when the dimensional our data is huge and we need large degree of polynomials m.

- Polynomials of degree m can be written as a linear combination of m-th degree monomials $1^{\alpha_0} x_{[1]}^{\alpha_1} \dots x_{[d]}^{\alpha_d}$

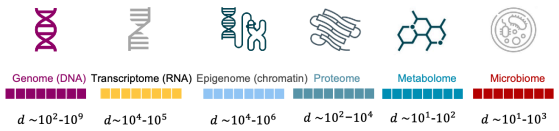
  with $\alpha_j \in N$ (natural number) and $\sum_{j=0}^{d} \alpha_j = m$ → feature size p = number of monomials of such form!

- (see bonus slide why) each monomial corresponds to picking a set of m from $d + m$ numbers

  yielding a total number of monomials of $p = \binom{d+m}{m} \approx \frac{(d+m-1)\cdots d}{m \cdots 1} \approx$
  - $O(d^m)$ for large enough d grows polynomially in d
  - $O(m^d)$ for large enough m grows polynomially in m

- 见下面举的具体的例子，这种问题在生物科学领域尤为常见：
  - In our complexity discussion, we care about "curse of dimensionality" for high-dimensional data

    i.e. how complexity increases with d for large d

  - In that regime, $p \in O(d^m)$ and the featurized training data set $\{(\phi(x_i), y_i)\}_{i=1}^n$ would be of size $O(nd^m)$
  - For $d \sim 10^5$, $n \sim 10^2$, even choosing $m = 3$ yields $O(nd^m) \sim 10^{17}$

    → prohibitively large memory and computational requirement!

  - 当我们对original data做feature space映射之后，这面大的数据存都存不下，数据的维度被进一步放大。

9.1.2 Kernel trick/kernelization

Overview

The kernel trick / kernelization - overview

- After featurization with a transformation ϕ, we solve $\hat{w} = \text{argmin}_{w \in R^p} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, w^\top \phi(x_i))$ and output function $\hat{f}(x) = \sum_j \hat{w}_j \phi_j(x)$ where $\phi_j(x)$ is the j-th element of the feature vector
- What is the **kernel trick** / what does it mean to **kernelize** a featurized problem? Assume $n \ll p$:

  Instead of parameterizing by $w \in R^p$, can "parameterize" f via $\alpha \in R^n$ via $f(x) = \alpha^\top \Phi \phi(x)$

  I. Solve $\hat{\alpha} = \text{argmin}_{\alpha \in R^n} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \alpha^\top \Phi \phi(x_i))$

  II. Output $\hat{f}(x) = \hat{\alpha}^\top \Phi \phi(x)$ depends on $x, x_1, \ldots, x_n$ only via featurized inner products of the form $\langle \phi(x), \phi(z) \rangle$

  III. Can then replace inner products by kernel functions $k(x, z)$ yielding $\hat{f}(x) = \sum_{i=1}^{n} \hat{\alpha}_i k(x, x_i)$ that fully depends $x, x_1, \ldots, x_n$ via the kernel function evaluation

- kernel完之后我们的数据维度是p。然后我们参数化的是p维的weight vector。而这里，kernel的trick就是我们不是参数化p的weight vector，而是n维，这里的n值得的是数据的大小。话说这里如果用batch的话，那岂不是n取决batch size的大小?
- 我们可以看到对于某一个具体的数据，其features是这个数据和所有数据之间的一个inner product。因此我们的feature维度就是n了。也就是inner product之后再parameterize。然后我们可以将这个inner product换成kernel function $k(x, z)$。
- 注意，这里的inner product是先做完feature变换后，由于维度太大，我们采用inner product从而将数据维度减少。
- 就是在test data上需要理解一下，你做inference的时候也是test data和所有的training data之间做inner product。

Furthermore, our optimization problem is simplified to calculate such kernel matrix

- Remember $\hat{\alpha} = \text{argmin}_{\alpha \in R^n} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \alpha^\top \Phi \phi(x_i)) = \text{argmin}_{\alpha \in R^n} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \sum_{j=1}^{n} \alpha_j \langle \phi(x_j), \phi(x_i) \rangle)$
- Now let's just define the inner products $\langle \phi(x), \phi(z) \rangle = k(x, z)$ as a kernel function $k: X \times X \to R$
- And define the kernel matrix $K \in R^{n \times n}$ on the training inputs $x_1, \ldots, x_n$ as

the domain X of inputs x

$$K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix} = \begin{pmatrix} \langle \phi(x_1), \phi(x_1) \rangle & \cdots & \langle \phi(x_1), \phi(x_n) \rangle \\ \vdots & \ddots & \vdots \\ \langle \phi(x_n), \phi(x_1) \rangle & \cdots & \langle \phi(x_n), \phi(x_n) \rangle \end{pmatrix}$$

- Then training loss and hence $\hat{\alpha}$ only depends on $\phi(x_1), \ldots, \phi(x_n)$ via elements of K!

- Memory: $O(np) \to O(n^2)$ 帮我们解决了内存存储的问题，但进一步增加了计算复杂度。
-

However, we still need to compute K in beginning – per entry/inner product, require p multiplications and additions, hence for n^2 entries, computational complexity $O(n^2 p)$!

对于computational complexity，其核心本质是我们算polynomial features时就很费力。再做n^2个inner product之后就更费力了。我们的手段是，能不能把polynomial features inner product这个过程替换成一个算的快的过程？

这里我们的手段是： Think of using higher-order polynomial features vs. kernel $(1+\langle x,z\rangle)^m$

- For m-th degree polynomial features in any dimension d (!), we can define

$$k(x,z) = (1+\langle x,z\rangle)^m$$

⇒ now compute time per evaluation $O(d+m)$ instead of $O(p) \sim O(d^m)$!

- 可以看到，computational effort大大缩减。

Step I: reparameterization to n-dim subproblem

Step II: kernel functions replacing inner products

10 Kernel & Other Methods

11 Neural Networks

11.1 Neural Networks I

11.2 Neural Networks II

11.3 Neural Networks III

Normalization and Standarization for input data: Reference

Non-convex:

- local minima
- flat area
- step

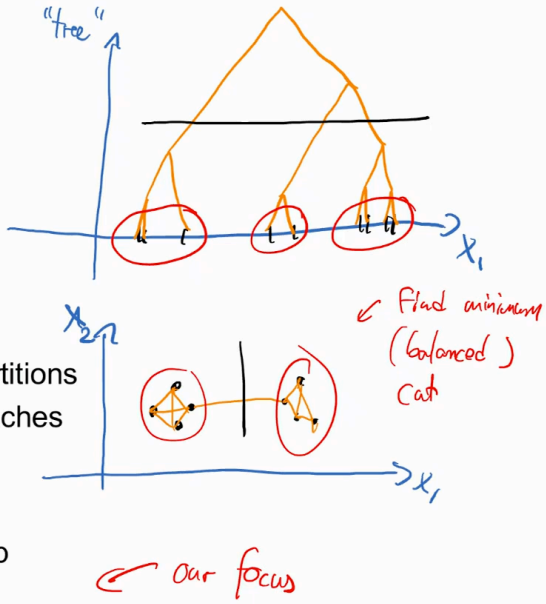
Multi-global

11.4 Neural Networks IV

12 Clustering

In supervised learning, it is hard to distinguish what we want.

Standard approaches to clustering



Hierarchical clustering

Build a tree (bottom-up or top-down), representing distances among data points

Example: single-, average- linkage clustering

Partitioning approaches

Define and optimize a notion of "cost" defined over partitions

Example: Spectral clustering, graph-cut based approaches

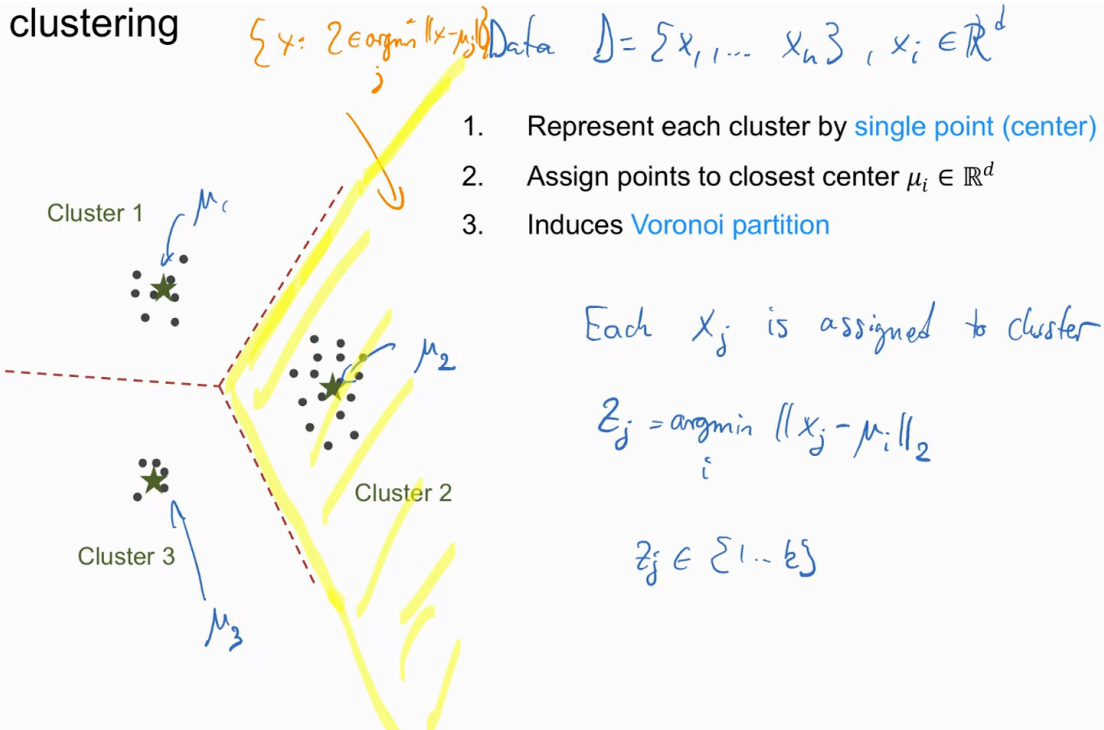
Model-based approaches

Maintain cluster "models" and infer cluster membership (e.g., assign each point to closest center)

Example: k-means, Gaussian mixture models, …

- 1,2 fixed dataset. 也就是分完之后就不能用来做推断了。这也是为什么我们需要model-based approaches。
- Our focus是model-based model,我们先会讨论很简单的模型，就像我们在supervised learning里面见到的那样。接着我们会讨论k-means以及更复杂的有kernel的k-means来处理更复炸的分类，其中我们会借用partitioning里面的spectral clustering的方法。

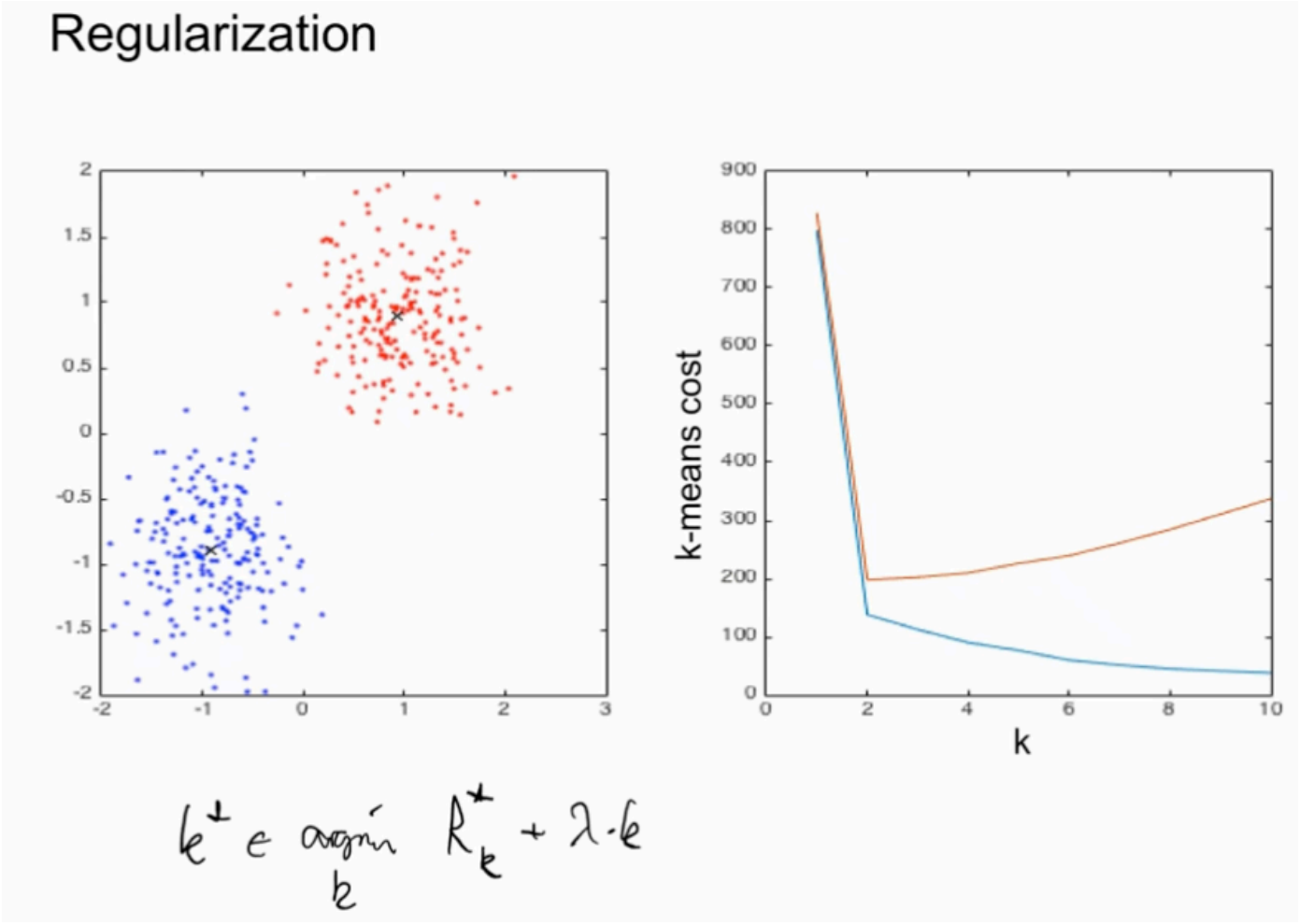
k-Means clustering



- Three means/ centers - induced a classifier.
- Because we used euclidean distance. the decision boundary is piece-wise linear. - Moroni partition
- Core is to find these three clusters

Non-convex problem, initialization is important.

57 minute!



所谓的regularization就是你需要做这样一个tradeoff。准确性和模型复杂度之间的tradeoff。也就是说，虽然你准确性提高，但是你的模型相应变得更复杂，所有会有一个penlty term。

- 我们想找到一个好的模型的同时有较低的模型复杂度
- 在这个问题里面就是模型复杂性增加带来的准确性上的收益很小了，但是造成的penalty却增大。所以总的收益很差了。

The inductive bias given by k-means is not suitable for 1:26:16

13 Dimension Reduction

Motivation:

- Visualization (k=1,2,3): main goal. Can be further implemented by clustering.
- Regularization:
- Unsupervised feature discovery (i.e., determine features from data!)

Here, we focus: model-based approaches

Given: data $D = \{\boldsymbol{x_1}, \dots, \boldsymbol{x_n}\} \subseteq \mathbb{R}^d$

Goal: Obtain mapping $f: \mathbb{R}^d \to \mathbb{R}^k$ where usually $k \ll d$

Can distinguish

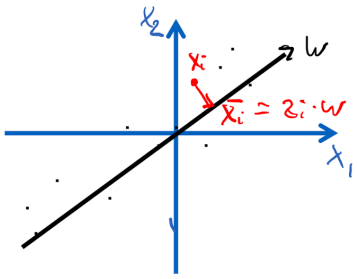
Linear dimension reduction: $f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$

Nonlinear dimension reduction (parametric or non-parametric)

Key question: Which mappings should we prefer?

- Model-based 方法的好处就是不仅仅处理当前数据集，当有新的数据输入的时候我们仍旧能够generalize it。

所谓的low dimension，即我们的数据原来在high-dimension中。现在我们找到一个low dimension space，通过一些映射，从而将数据压缩到low dimension中 $\mathbf{x_i} \approx z_i \mathbf{w}$ with coefficients $\mathbf{w} \in \mathbb{R}^d$



接着，我们的问题就抽象出来了：

Given data set $D = \{\boldsymbol{x_1}, \dots, \boldsymbol{x_n}\} \subseteq \mathbb{R}^d$

Want $\boldsymbol{x}_i \approx z_i \boldsymbol{w}$, e.g., minimizing $\|z_i \boldsymbol{w} - \boldsymbol{x}_i\|_2^2$

To ensure uniqueness, normalize: $\|\boldsymbol{w}\|_2 = 1$

Why? $(\alpha \cdot z_i)\left(\frac{1}{\alpha} w\right) = z_i w \quad \forall \alpha \neq 0 \Rightarrow$ w.l.o.g. $\alpha = \|w\|$

整个问题被变化成了一个优化问题

Optimize jointly over $\boldsymbol{w}, z_1, \ldots, z_n$

$$\min_{w, \|w\|_2=1, z_{1:n}} \underbrace{\sum_{i=1}^{n} \|x_i - z_i \cdot w\|_2^2}_{L(w; z_{1:n})}$$

数据的covariance matrix，之所以称之为matrix，是因为数据可能有多个维度，每个维度可以看作一个随机变量。

Covariance matrix

这有当covariance matrix follows from the fact that Σ is s.p.d matrix，才可用principal eigenvector进行展开

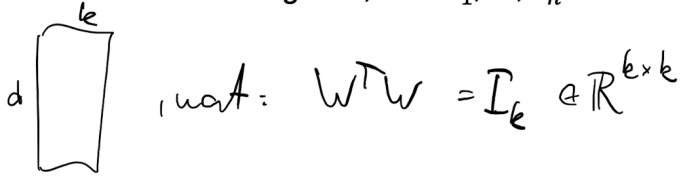
对于k dimension space,通过制定W is an orthogonal matrix, we can achieve the uniqueness of the problem.

Suppose we wish to project to more than one dimension. Thus we want:

$$(\boldsymbol{W}, \boldsymbol{z}_1, \ldots, z_n) = \arg \min_{\boldsymbol{W}^T \boldsymbol{W} = I_k, \boldsymbol{z}} \sum_{i=1}^{n} \|\boldsymbol{W}\boldsymbol{z}_i - \boldsymbol{x}_i\|_2^2$$

where $\boldsymbol{W} \in \mathbb{R}^{\boldsymbol{d}\times\boldsymbol{k}}$ is *orthogonal*, and $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n \in \mathbb{R}^k$

15:39 normalization



1. This is called the Principal Component Analysis problem
2. Its solution can be obtained in closed form even for $k > 1$

14 Dimension Reduction II

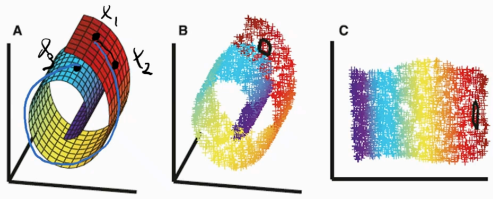
Core idea: minimize the reconstruction error. But we should determine what kind of function do we need?

For PCA, we have the optimized closed-form solution.

PCA, after project, it will preserve the euclidean distance, which will give rise to the following problem, we will loss the structure of these data.

Nonlinear Dimension Reduction

1. Motivating Example: „Swiss Roll“



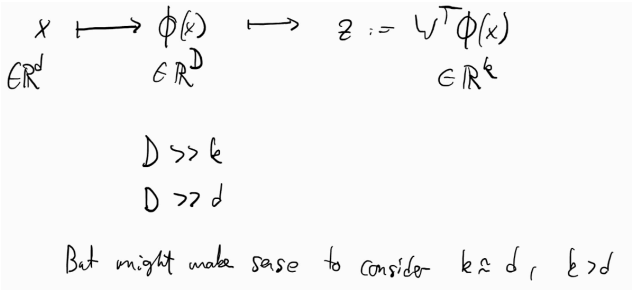
2. What is the result of a linear projection?

Use Kernels!

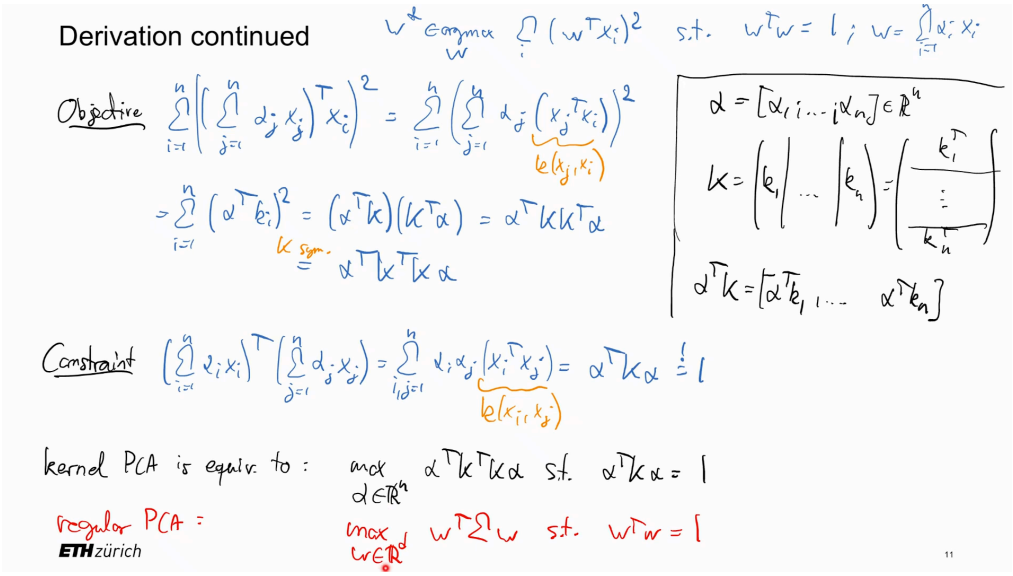
1. *Recall*: In supervised learning, kernels allowed us to solve non-linear problems by reducing them to linear ones in high-dimensional (implicitly represented) spaces
2. Can take the same approach for unsupervised learning!

仿照Linear regression，我们同样有两种trick：

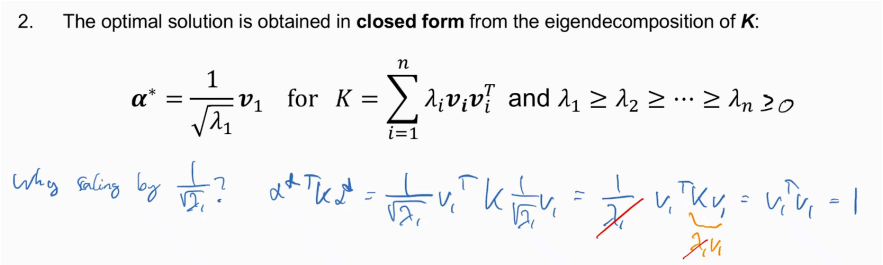
- 手工设定nonlinear features kernel PCA
- 自动学习这些features： Auto-encoder



- 这个地方真是让人迷惑，我们实在feature space做PCA。做完PCA后的维度甚至可能大于original space。这个地方回应了知乎上遇到的那个问题

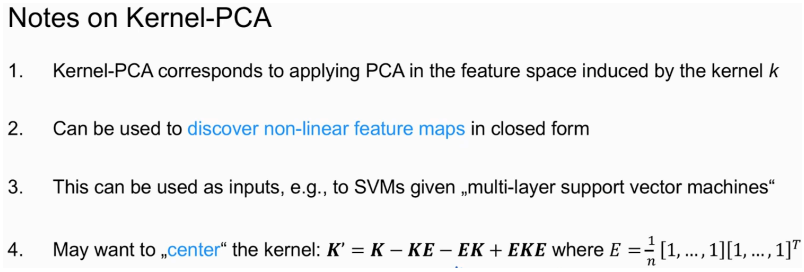


- 对于kernel PCA，we still have the optimized closed-form solution. not going to derive it here.

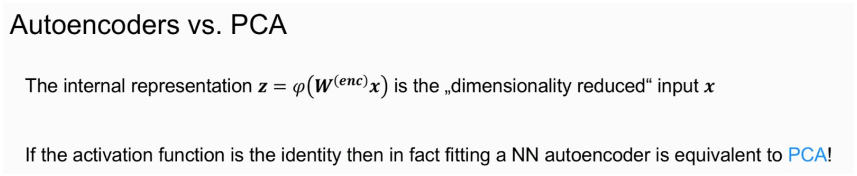


- scaling ensure the constraint.

这里做dimension reduction的一个用途是为了downstream的任务。这里好处是你下游任务的模型变得很容易，不用学很多不相关的东西。



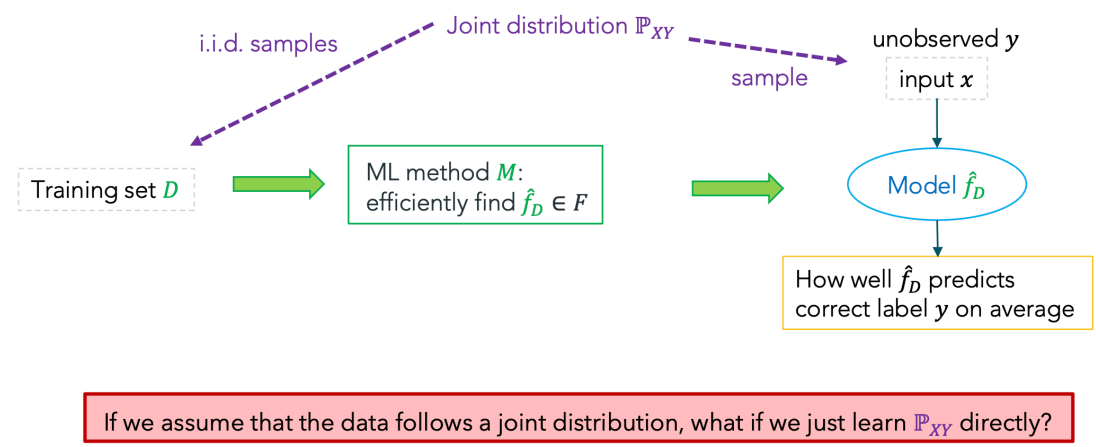
- 第4点我们会遇到一些问题，就是通常情况下，我们做clustering是要求我们的data必须是center的。但是，当我们变化到feature space的时候，data不在是centering了，因此我们可以用centering的centering‘s kernel
- 不太理解这里面的推导



- That particular function is non-convex as before, but you still see the closed-form solution as before.

Actually, wider the bottle neck, you will get the better reconstruction of the data.

15 Probabilistic modeling



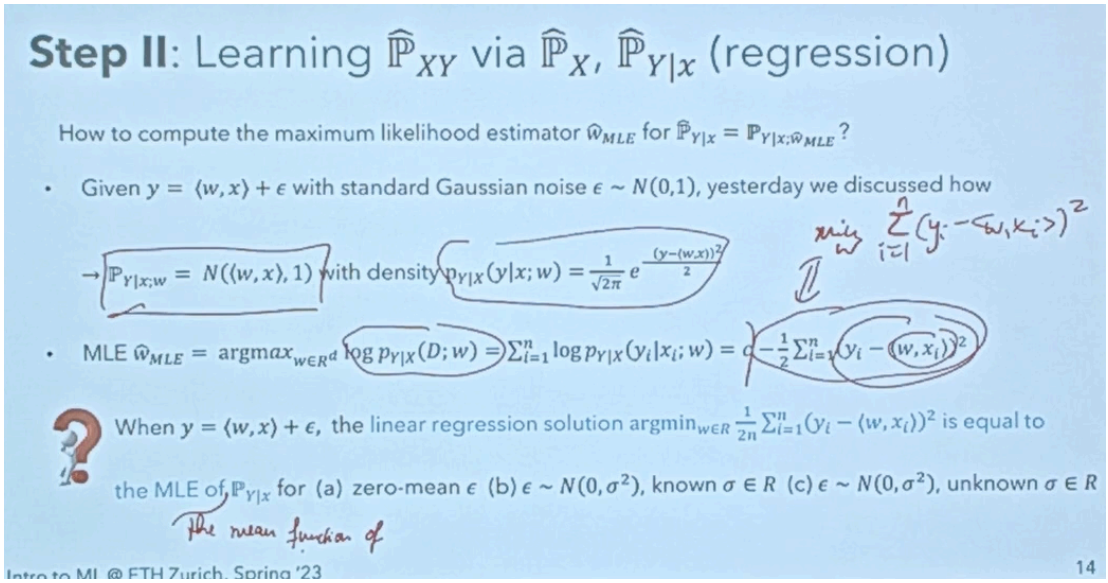
一般来说我们有training data，我们知道这个training data来自于一个joint distribution。那么我们为什么不直接学习这个joint distribution?毕竟我们有关于这个distribution的samples。这不比找到一个合适的模型去拟合数据更好?

- 在原来的范式中，我们通常是从一个function class里面去找一个好的function来拟合你的data，在这个过程中我们需要定义我们的training loss（目标）以及optimization method（方法）。不同的function class对应不同的ML method。

16 Probabilistic modeling II

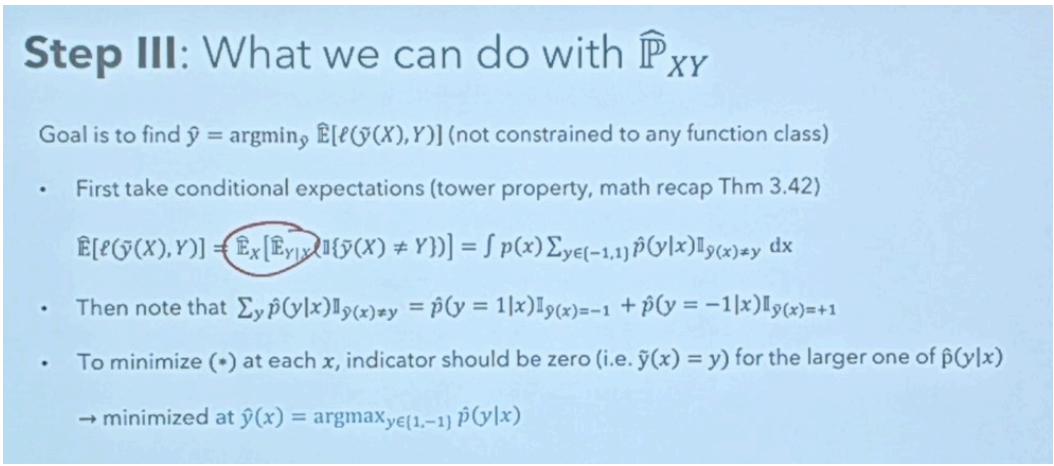
- Simple 1-d example for supervised learning $\mathbb{P}_{xy}$
  - MLE for Gaussian noise model (regression)
  - MLE for logistic nosie mdel (classification)
- Generative (vs. discriminative) modeling of $\mathbb{P}_{xy}$

First, we discussed what we could do if we have $\mathbb{P}_{xy}$



- 这个地方可以看出，我们再第一章做linear regression的时候其实是默认nosie是gaussian distribution的。如果不是Gaussian distribution，这里我们得到的MLE就会是另一个形式。这个地方涉及到linear regression背后的统计学本质

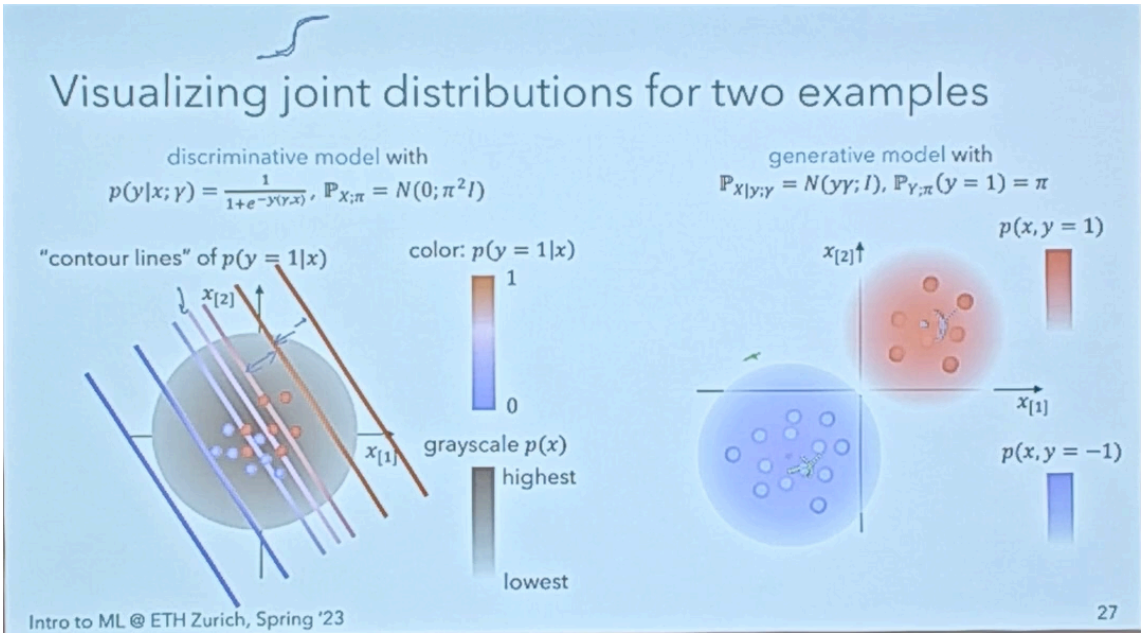
下面，我们看classification



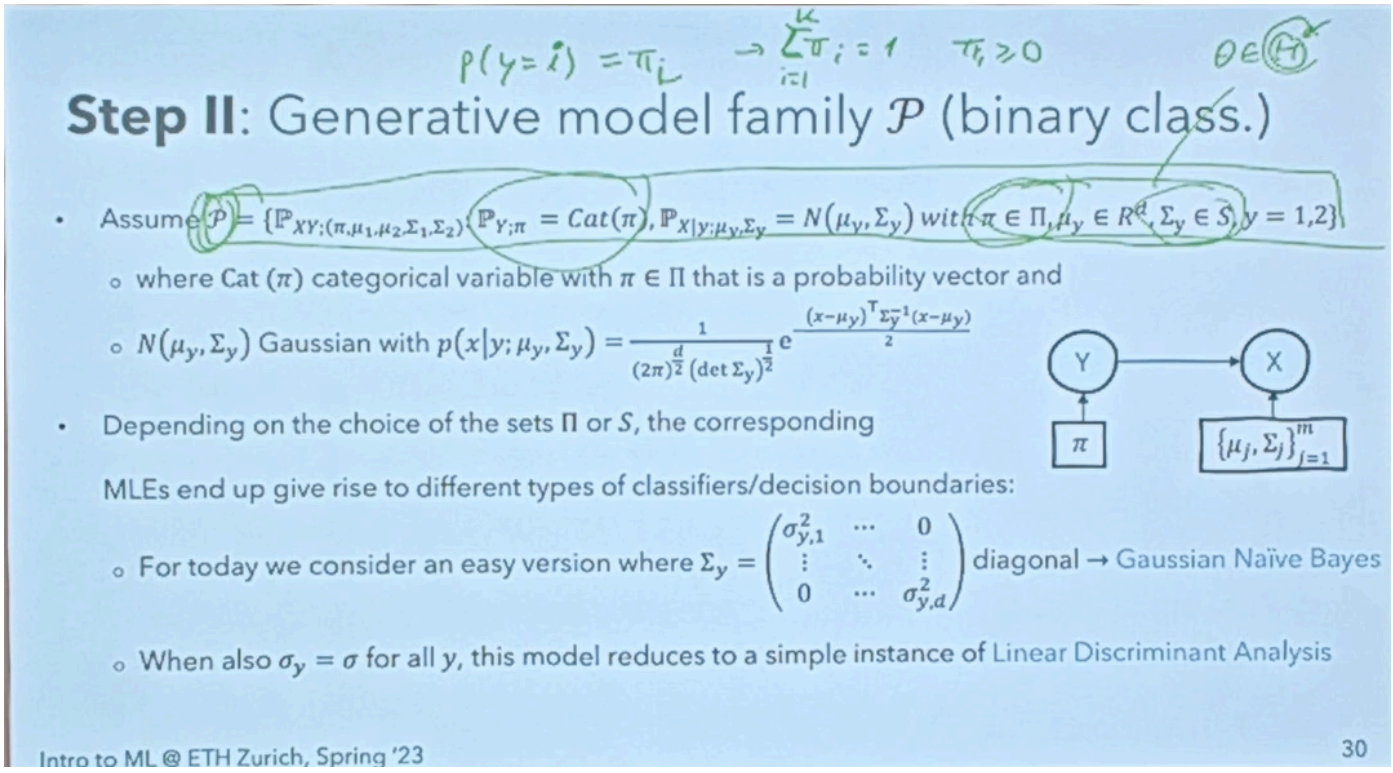
- classification中关于这部分的推导我们回去自己理解

$\mathbb{P}_{Y|X}$ 这里的assumption有点看不懂

接下来我们换个角度考虑分布，不考虑$\mathbb{P}_x$ and $\mathbb{P}_{y|x}$，而是$\mathbb{P}_y$ and $\mathbb{P}_{x|y}$.



- the weights of the label.
- 不太清楚这里的图



- 这里不太懂

17 Gaussian Mixture Models (I&II&III)

Recap:

Discreminative models aims to estimate $p(y|x)$ (conditional distribution)

Generative models aim to estimate joint distribution $p(y,x) = p(y) \cdot p(x|y)$

- Can derive conditional from joint distribution, but not vice versa!

Typical approach to generative modeling for classification

generative modeling通常用在classification问题中，因为常常在分类问题中，$p(\mathbf{x}|y)$比$p(y|\mathbf{x})$更容易建模。

Estimate $p(y)$ and $p(\boldsymbol{x} \mid y)$

In order to predict label y for new point x, use

$$P(y \mid \mathbf{x}) = \frac{1}{Z} P(y) P(\mathbf{x} \mid y) \qquad Z = \sum_{y} P(y) P(\mathbf{x} \mid y)$$

1. Predict using Bayesian decision theory.
2. E.g., in order to minimize misclassification error, predict

$$y = \arg\max_{y'} P(y' \mid \mathbf{x})$$

$= \operatorname{argmax}_{y'} \frac{1}{Z} p(y') p(x|y')$

$= \operatorname{argmax}_{y'} \log p(y') + \log p(x|y')$

- 整个过程中我们没有依赖于任何外界的模型假设，仅仅只是推导出我们的概率分布即可。但是，得到这些具体的联合概率密度分布函数可能需要我们参数化进行一些假设，这个过程中设计一些建模。但一旦这个模型建立完成，后续的prediciton不依赖于任何的modeling部分。

$$y = \arg\max_{y'} P(y' \mid \boldsymbol{x}) = \arg\max_{y' \in \{1 \ldots c\}} \log P(y') + \log P(\boldsymbol{x} \mid y')$$

eg. $\mathcal{N}(x \mid \mu_{y'}, \Sigma_{y'})$

$= \frac{1}{\sqrt{2\pi|\Sigma_{y'}|}} \cdots$

而建立这个joint distribution的过程可以用Gaussian Naive Bayes classifiers。不同的classifiers其实指的就是在建模$p(\mathbf{x}|y)$时我们假设什么样的分布模型。

Example: *Gaussian* Naive Bayes classifiers

Model class label as generated from categorical variable

$$P(Y = y) = p_y \qquad y \in \mathcal{Y} = \{1, \ldots, c\}$$

Model features by (conditionally) independent Gaussians

$$P(x_i \mid y) = \mathcal{N}(x_i \mid \mu_{y,i}, \sigma^2_{y,i}) \rightsquigarrow P(x \mid y) = \prod_{i=1}^{d} P(x_i \mid y)$$

- Estimate parameters via maximum likelihood estimation. 因为是由标签的数据，所以我们在建立概率密度分布时依照的原则是在这个假设的分布下，标签数据出现的likehood最大化。

为了更感性的认识假设的模型不同参数的影响，我们这里回顾一下多变量的高斯分布：

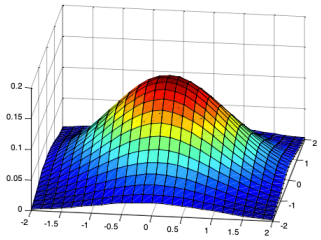
Multivariate Gaussian

$$p(x) = \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

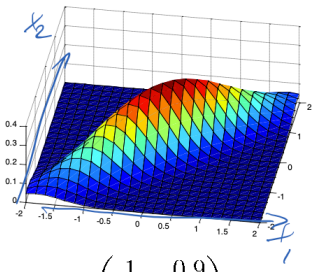
Example in 2 dimensions:

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix}$$

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$$



$$\Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$



$$\Sigma = \begin{pmatrix} 1 & 0.9 \\ 0.9 & 1 \end{pmatrix}$$

ETH zürich

- 而我们在建模时往往假设的就是这个variation的形式。

高斯navie是假设**x**的各个维度之间满足no correlation，因此variance是对角矩阵。但是在真实情况中这种情况不满足。理解上，我们可以认为如果是naive Bayes classifier，可以看成是对每一个feature单独用一个gaussian distribution进行建模。而general的情况则是多变量高斯分布，因此一种更general的形式是：

More general: Gaussian Bayes classifiers

$p_y \geq 0$

$\sum_y p_y = 1$

Model class label as generated from categorical variable

$$P(Y = y) = p_y \qquad y \in \mathcal{Y} = \{1, \ldots, c\}$$

Model features as generated by multivariate Gaussian

$$P(\mathbf{x} \mid y) = \mathcal{N}(\mathbf{x}; \mu_y, \Sigma_y)$$

- 同样 Estimate parameters, e.g., via Maximum Likelihood Estimation。但只不过参数搜索的空间更大了。

MLE for Gaussian Bayes Classifier

Given data set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$

MLE for class label distribution

$$P(Y = y) = \hat{p}_y$$

$$\hat{p}_y = \frac{\text{Count}(Y = y)}{n}$$

MLE for feature distribution

$$P(\mathbf{x} \mid y) = \mathcal{N}(\mathbf{x}; \hat{\mu}_y, \hat{\Sigma}_y)$$

$$\hat{\mu}_y = \frac{1}{\text{Count(Y=y)}} \sum_{i:y_i=y} \mathbf{x}_i$$

$$\hat{\Sigma}_y = \frac{1}{\text{Count(Y=y)}} \sum_{i:y_i=y} (\mathbf{x}_i - \hat{\mu}_y)(\mathbf{x}_i - \hat{\mu}_y)^T$$

- 这里似乎给出了我们用这种假设求解的形式。是一个解析的解

18 Generative Model with Neural Network

Motivation: So far, we looked at relatively simple parametric generative models (Gaussians, mixtures of Gaussians). How can we use the expressive power of neural network models to address more complex data sets (images, text, programs, molecules, ...)?

Much recent work on this topic:

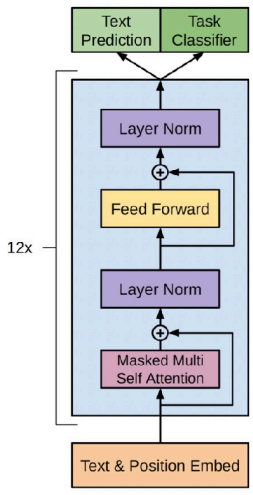
- Generative adversarial networks (GAN)
- Diffusion models
- Transformer models

The motivation for transformer model is from that we want to model

$$P(X_t = x|X_{1:t-1} = x_{1:t-1}) = P(X_t = x|X_{t-k:t-1} = x_{t-k:t-1}, \theta) := \text{Cat}(x|\text{softmax}(f(x_{t-k:t-1}), \theta))$$

with neural network architectures. And we can use sequence modeling like:

1. Recurent neural networks (LSTMs, GRUs)
2. Transformer models



Computational model / graph:

$$\boldsymbol{Z}_0 = \boldsymbol{X}\boldsymbol{W}_e + \boldsymbol{W}_p$$

$$\boldsymbol{Z}_l = \text{transformer\_block}(\boldsymbol{Z}_{l-1}) \; \forall i \in \{1, \dots n\}$$

$$\boldsymbol{P} = \text{softmax}(\boldsymbol{Z}_n \boldsymbol{W}_e^T)$$

Hereby, $X = (x_{t-k}, \dots, x_{t-1})$ represents the word (token) indices for the context of the k previous words (tokens);
W_e is a (learnable) word embedding matrix;
W_p is a (fixed) position embedding matrix

[Radford et al, 2018 "Improving Language Understanding by Generative Pre-Training"]

- The word embedding matrix can used pretrained other models, which can transfer every word with a embedding.

Word	age	gender	...
king	1	0	...
queen	1	1	...
prince	0	0	...
princess	0	1	...
...	...	...	...

Each word (token) is represented by a feature vector through a learnable embedding matrix W_e

- The reason why we use position embedding is due to the reason that in the transformer model, we don't have the sequence information like RNN. The whole sequenece is proceed at once, if we don't encoding the position information. The model don't know which word is front and back. And this positional embedding matrix is also fixed. We often use sine and cosine function to construct.
- In the transformer block, the most important part is "masked multi-head self attention". And it is consist of three parts:
  - Self-attention
    - we need to calculate the attention score of every word for others, which construct a attention matrix. The parameters during this process is trained.
  - Multi-head (self)-attention
    - We can seperate the features of every word for multi-parts. Each part is called a head. And we do the self-attention for each head. Because each head contains specific features. Therefore, we can learn the attention of different features.
  - Masking
    - During the training process, we want our model only to calculate the attention for specfic part. For other parts, we will mask them. For example, padding masking and lookahead masking.

Details can see the conservation with ChatGPT.

Model	Params	Layers	Dim	Context
GPT1	117M	12	768	512
GPT2	1.5B	48	1600	1024
GPT3	175B	96	12888	2048
GPT4	170T	?	?	?

  - Modern large language models have MANY parameters, trained on LARGE amounts of data.