

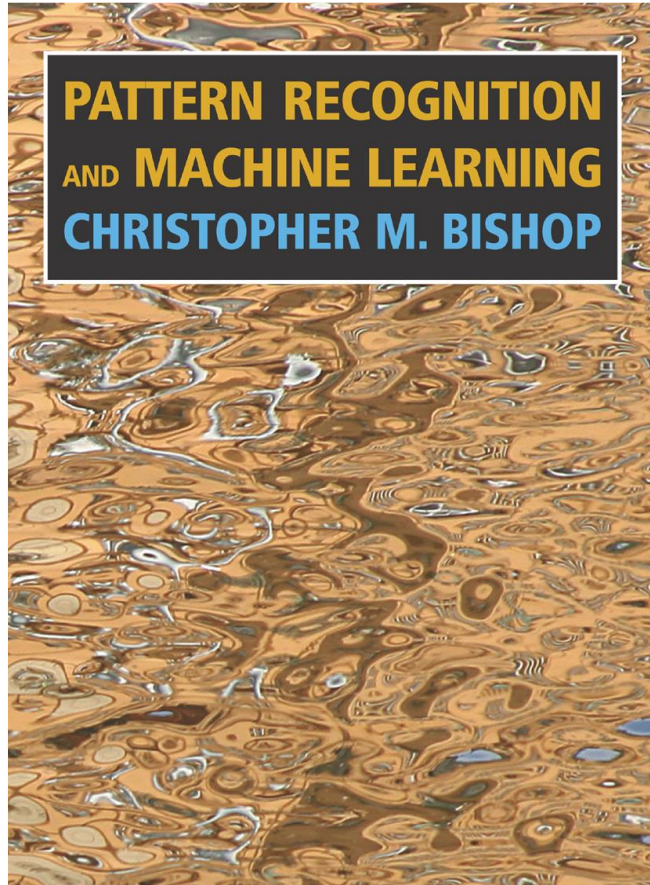
Regression: Overfitting and Curse of Dimensionality

Song Liu (song.liu@bristol.ac.uk)

Office Hour: 3-4pm Tuesday

Office: Fry Building GA 18

Reference



Today's class *roughly* follows Chapter 1.

Pattern Recognition and
Machine Learning

Christopher Bishop, 2006

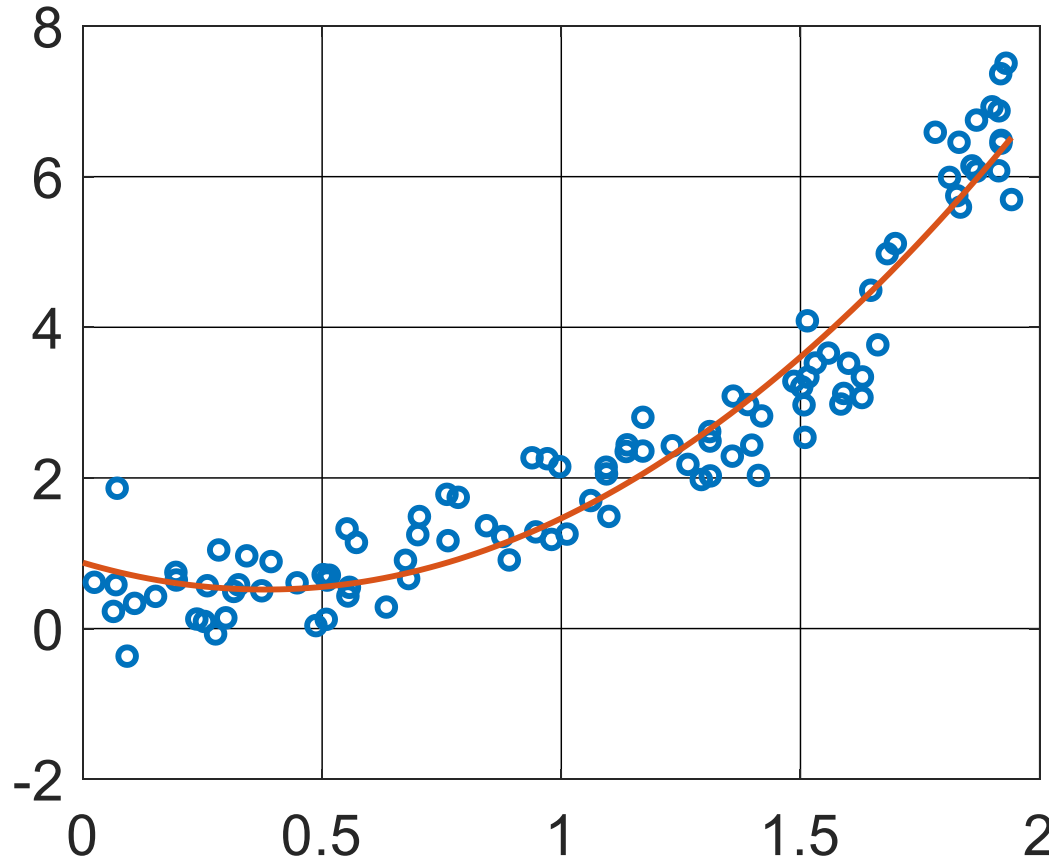
LS with Feature Transform

$$\mathbf{w}_{\text{LS}} := \operatorname{argmin}_{\mathbf{w}} \sum_{i \in D_0} [y_i - f(\mathbf{x}_i; \mathbf{w})]^2$$
$$f(\mathbf{x}; \mathbf{w}) := \langle \mathbf{w}_1, \boldsymbol{\phi}(\mathbf{x}) \rangle + w_0, \mathbf{w} := [\mathbf{w}_1, w_0]^\top$$

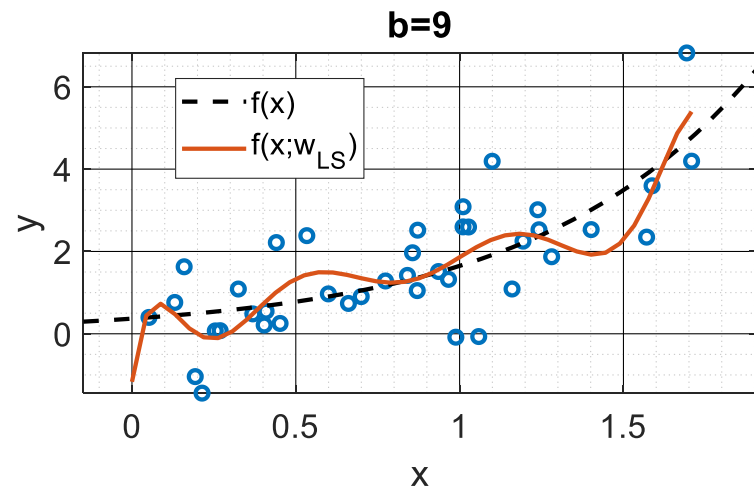
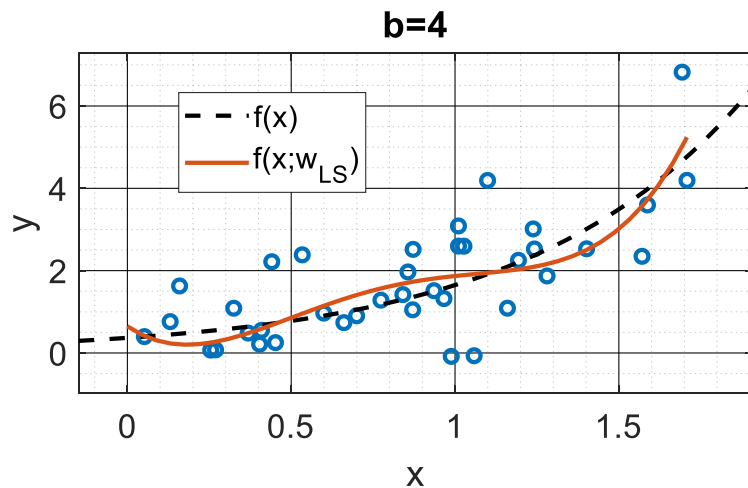
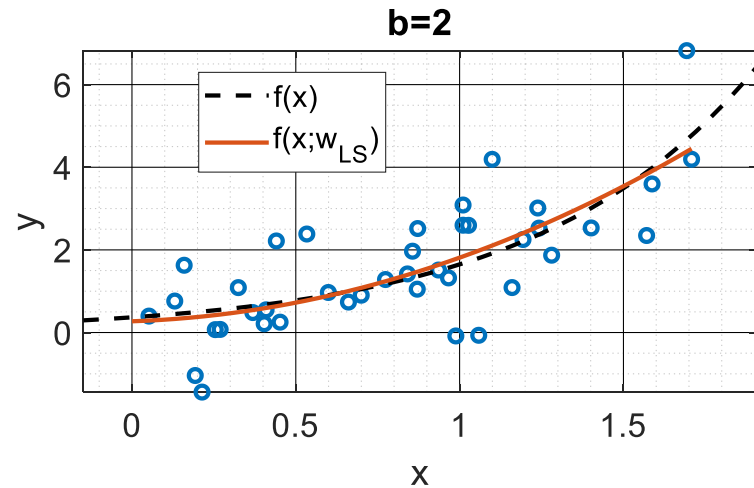
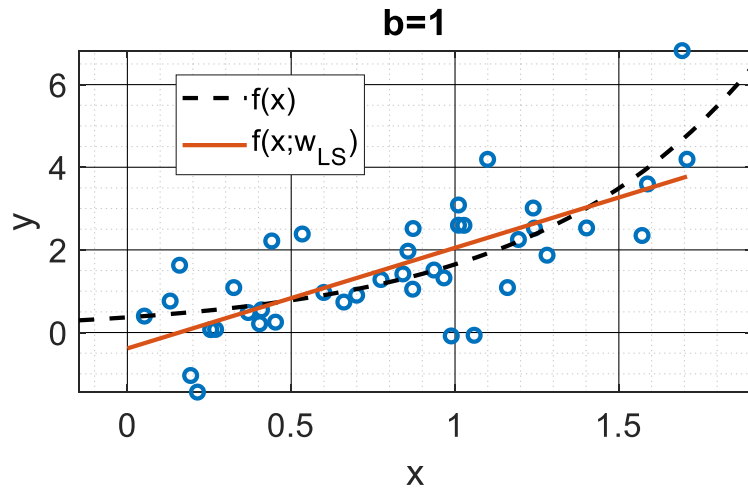
- $\boldsymbol{\phi}(x)$ can be a collection of polynomial functions:
- $\boldsymbol{\phi}(x) := [x^1, x^2, x^3 \dots x^b]^\top$.
- b is called the degree of $\boldsymbol{\phi}(x)$.

LS with Polynomial Transform ($b = 2$)

- $x \sim \text{uniform}(0,2)$
- $y = f(x) + \epsilon, f(x) = \exp(1.5x - 1), \epsilon \sim N(0, .64)$



Poly. Transform with various b



Poly. Feature with various b

- The higher the b , the more flexible our $f(x; \mathbf{w})$ is.
- However, when increasing b ,
 - The fit of $f(x; \mathbf{w}_{LS})$ first got better ($b = 2$).
 - then got worse ($b = 4, b = 9$).
 - $f(x; \mathbf{w}_{LS})$ become too “squiggly”, when b is large.
 - $f(x; \mathbf{w}_{LS})$ almost tried “too hard” to fit our data.
- Is this a general pattern?
 - We design an experiment to find out.

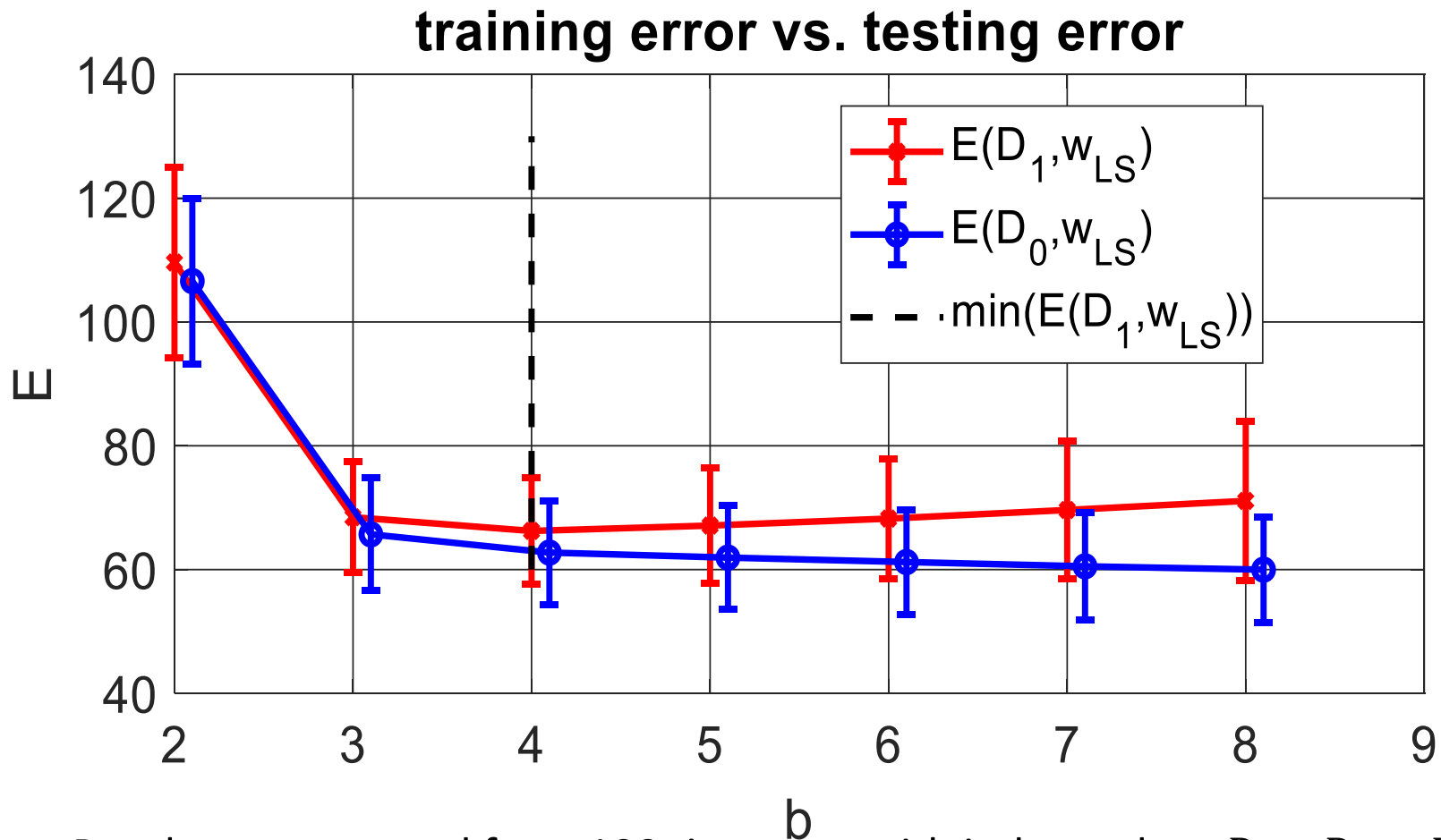
Training Error and Testing Error

- We randomly split our dataset D into D_0 and D_1 .
 - assuming D contains IID pairs.
- \mathbf{w}_{LS} is fitted using D_0 only.
- Define an error $E(D', \mathbf{w}) = \sum_{i \in D'} [y_i - f(\mathbf{x}_i; \mathbf{w})]^2$.
- It tells how well $f(\mathbf{x}; \mathbf{w})$ fits a specific dataset D' .
- We can have **two performance metrics**:
- $E(D_0, \mathbf{w}_{LS})$ is usually referred to as **training error**.
- $E(D_1, \mathbf{w}_{LS})$ is usually referred to as **testing error**.

Training Error and Testing Error

- We do not care $E(D_0, w_{LS})$!
- We have already seen the output in D_0 during the training.
- We care performance of $f(\mathbf{x}; \mathbf{w}_{LS})$ on unseen dataset D_1 !
- The ability of getting low $E(D_1, w_{LS})$ is called **generalization**.
- Generalization is a **key goal** in statistical decision making.
- Go back to the example,
- As b increases, how $E(D_0, w_{LS})$ and $E(D_1, w_{LS})$ change?

Training Error and Testing Error



Results are averaged from 100 times run with independent $D = D_0 \cup D_1$ generated by different random seeds, and are plotted with standard deviation

Training Error and Testing Error

- Training error keeps reducing.
- $f(\mathbf{x}; \mathbf{w}_{LS})$ fit D_0 better and better as b increases.
- Testing error drops then goes up again.
- $f(\mathbf{x}; \mathbf{w}_{LS})$ does not fit unseen D_1 well, when b is too large.
- **The problem:**
 - Generalization of $f(\mathbf{x}; \mathbf{w}_{LS})$ deteriorates when b is too large.
 - The phenomenon $f(\mathbf{x}; \mathbf{w}_{LS})$ fits too well on training set while underperforming on unseen datasets, is called **Overfitting.**

Selecting b

- b should not be too small, so f is **flexible enough**!
- b should not be too large, so f is **not too flexible**!
- How do we select?
- We can split full dataset D into D_0 and D_1 .
- Use D_0 to fit $f_{LS}(b)$ and use D_1 to compute $E(D_1, f_{LS}(b))$.
- Select a b such that $E(D_1, f_{LS}(b))$ is the lowest.
- Fit f_{LS} again using the selected b on the full dataset.

Selecting b (better approach)

Problem of splitting D into D_0 and D_1 :

1. However, we have wasted D_1 for validation.
 - What if D_1 contains info that is beneficial for choosing b ?
2. $E(D_1, f_{LS}(b))$ is random, the selection may be random.
 - We want to take average and reduce the randomness.

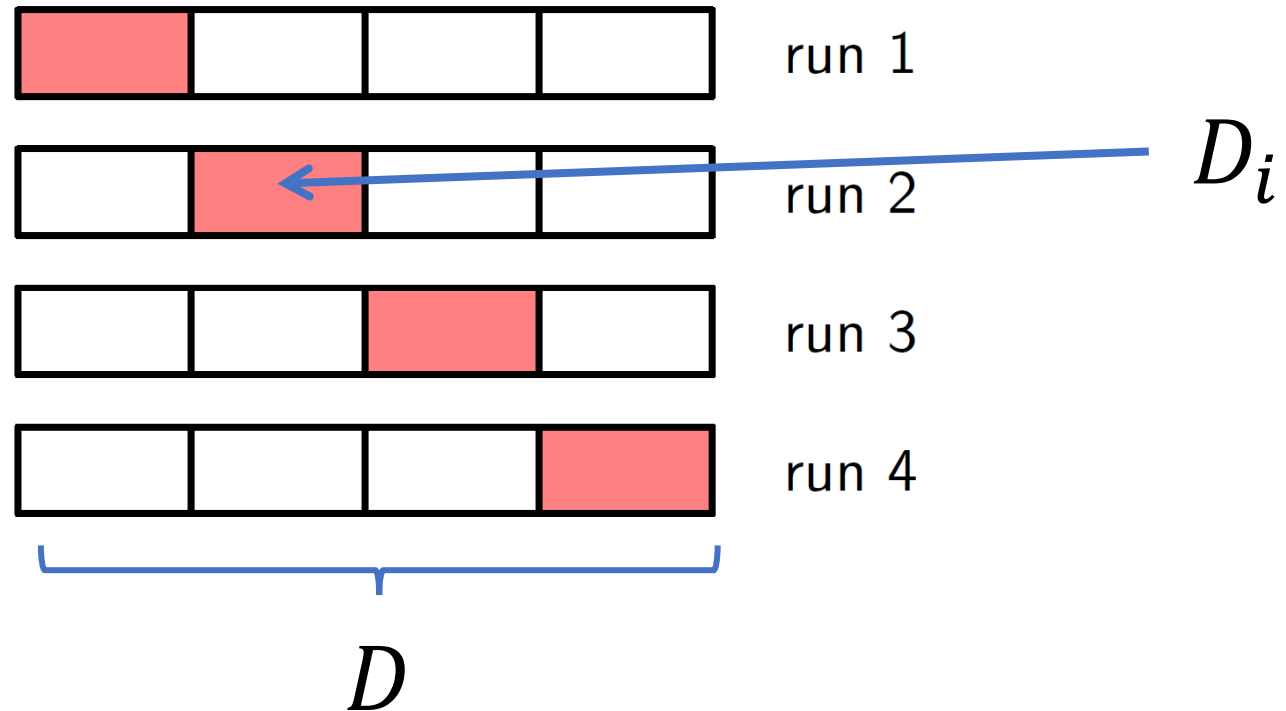
- Split D into D_0 and D_1 , compute $E(D_1, f_{LS}(b))$
- Swap the role of D_0 and D_1 , compute $E(D_0, f'_{LS}(b))$
 - $f'_{LS}(b)$ is fitted using D_1

- Select b that minimizes $\frac{E(D_1, f_{LS}(b))}{2} + \frac{E(D_0, f'_{LS}(b))}{2}$

Cross-validation

- The extension of above idea gives rise to a commonly used model selection method: **Cross-validation.**
- Split D into **disjoint** $D_0 \dots D_k$,
- For $i = 0$ to k
 - Fit $f_{\text{LS}}^{(i)}(b)$ on all subsets **but** D_i , $\forall b$
 - Compute $E\left(D_i, f_{\text{LS}}^{(i)}(b)\right)$, $\forall b$
- Select b that minimizes $\frac{\sum_i E^{(i)}}{k+1}$
- k can go as high as $n - 1$: **leave-one-out-validation**

Cross-validation



- PRML, Figure 1.18
- Read Chapter PRML 1.3

Problem of Cross-validation

- The implementation of cross-validation is easy,
- But the computational cost is high.
 - $f_{LS}^{(i)}(x; \mathbf{w})$ must be fitted and validated for all splits.
- The effectiveness of cross-validation depends on the IID assumption of our dataset D .
 - Validation set and the training set must be IID!
 - Which may not hold in reality: e.g. stock price dataset.
- Can we avoid overfitting without splitting our dataset for validation? We will discuss this in the future.

Polynomial Transform on Higher Dimensional Dataset

- So far, we only considered polynomial transform on one dimensional dataset, i.e., $x \in R$
- What about $\mathbf{x} \in R^d$, when the output y depends on multiple inputs?
- When $\mathbf{x} \in R^d$,
 - $\boldsymbol{\phi}(\mathbf{x}) := [\mathbf{h}(x^{(1)}), \mathbf{h}(x^{(2)}), \dots, \mathbf{h}(x^{(d)})]^\top$.
 - $\mathbf{h}(t) := [t^1, t^2, \dots, t^b] \in R^b$.
 - $\boldsymbol{\phi}(\mathbf{x}) \in R^{db}$, which means $\mathbf{w}_1 \in R^{db}$.
- This does **not** include cross-dimension polynomials.
 - e.g., $x^{(1)} x^{(2)}, x^{(1)} x^{(2)} x^{(3)}, \dots$
 - These can be useful as the output value may depends jointly on several inputs. e.g. blood pressure <- (weight,height)

Polynomial Transform on Higher Dimensional Dataset

- To include **pairwise** cross-dimension polynomials, we can slightly redesign $\phi(\mathbf{x})$:
 - $\phi(\mathbf{x}) := [\mathbf{h}(x^{(1)}), \dots, \mathbf{h}(x^{(d)}), \forall_{u < v} x^{(u)} x^{(v)}]$
 - $\phi(\mathbf{x}) \in R^{db + \binom{d}{2}}$,
- Similarly, we can include all the **triplets**:
 - $\phi(\mathbf{x}) := [\mathbf{h}(x^{(1)}), \dots, \mathbf{h}(x^{(d)}), \forall_{u < v} x^{(u)} x^{(v)}, \forall_{u < v < w} x^{(u)} x^{(v)} x^{(w)}]$
 - $\phi(\mathbf{x}) \in R^{db + \binom{d}{2} + \binom{d}{3}}$,
- and we can go on to include **quadruplets**...

Curse of Dimensionality

- We can include cross terms all the way up to d -plets.
- Unfortunately, we know
 - $\binom{d}{1} + \binom{d}{2} + \binom{d}{3} + \binom{d}{4} + \dots + \binom{d}{d} = 2^d$
- We have not yet included cross terms like:
 - $[x^{(u)}]^2 x^{(v)} \dots$
- The output dimension of $\phi(\mathbf{x})$ can grow exponentially with dimensionality d and this is a bad news...

Curse of Dimensionality

- We have seen in yesterday's homework, the number of observations n , needs to at least match the output dimension of $\phi(\mathbf{x})$, otherwise, we cannot obtain \mathbf{w}_{LS} !
- It means we need to grow n exponentially with d !
- Imagine a problem with $d = 100$.
 - A terabyte-data on hard-drive contains 2^{40} bytes.

Curse of Dimensionality

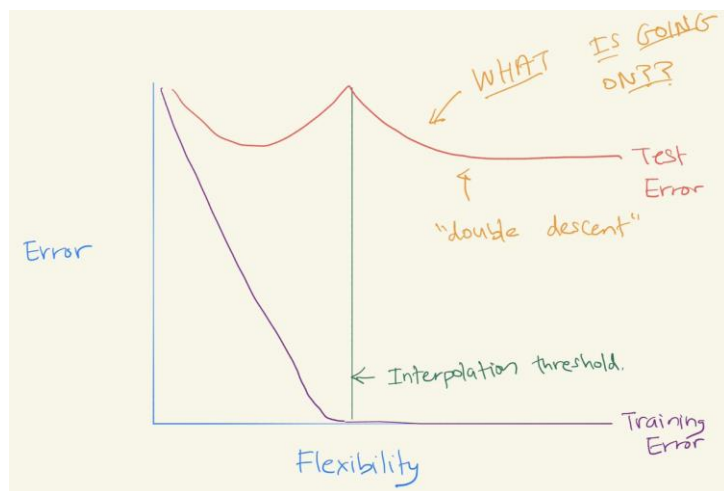
- The phenomenon, that the number of observations needed to solve a problem grows exponentially with d exists in many statistical learning tasks.
- They are collectively called “Curse of Dimensionality”.
- This phenomenon forbids us solving high-dimensional problems.

Conclusion

- We introduce poly. transform to our prediction func. f .
- This increases the flexibility of f , but we also see this additional complexity caused two major problems:
- **Overfitting**
 - The generalization of f is poor.
- **Curse of Dimensionality**
 - n needs to grows exponentially with the dimensionality of x .
- Next week, we will introduce a way to reduce the flexibility of f to combat overfitting and the probabilistic idea behind it.

Um... Actually...

- In recent years, people realize (particularly in the context of deep neural network models), there is a phenomenon called double descent:



The testing error **comes back and drops again** as the flexibility of your prediction function increases!



- Read this [twitter thread](#) by Daniela Witten.

Computing Lab

- Download “Prostate Cancer dataset”, [description](#), [dataset](#).
- Implement a Least-square solver using R. Do not use built-in functions.
- Fit $f(\mathbf{x}; \mathbf{w})$ using classic linear least squares.
- Calculate the cross-validation error.
- How does the cross-validation testing error change if you **remove one of the features**?
 - How do you explain this using what we have learned today?