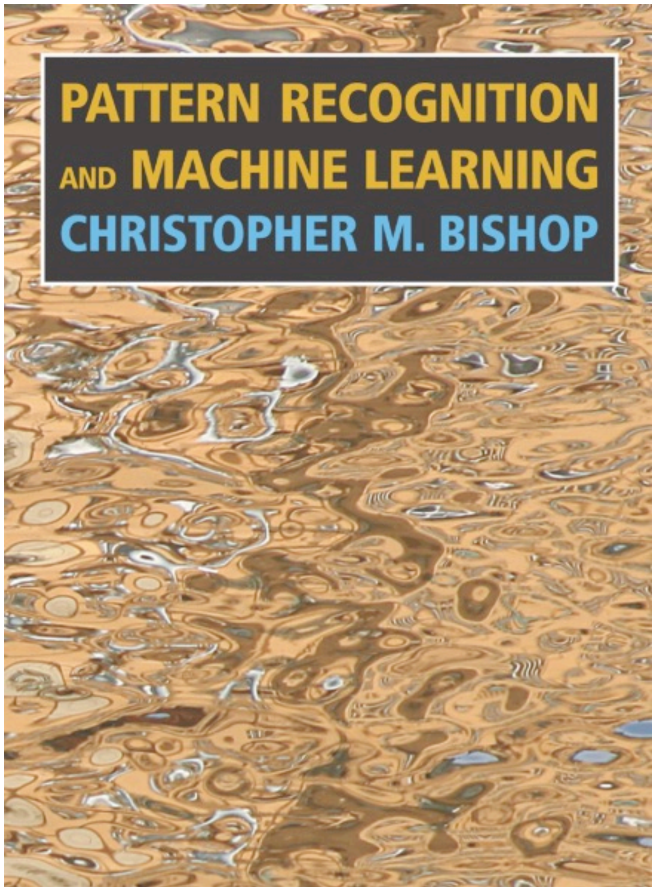
Regression: Overfitting and Curse of Dimensionality

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

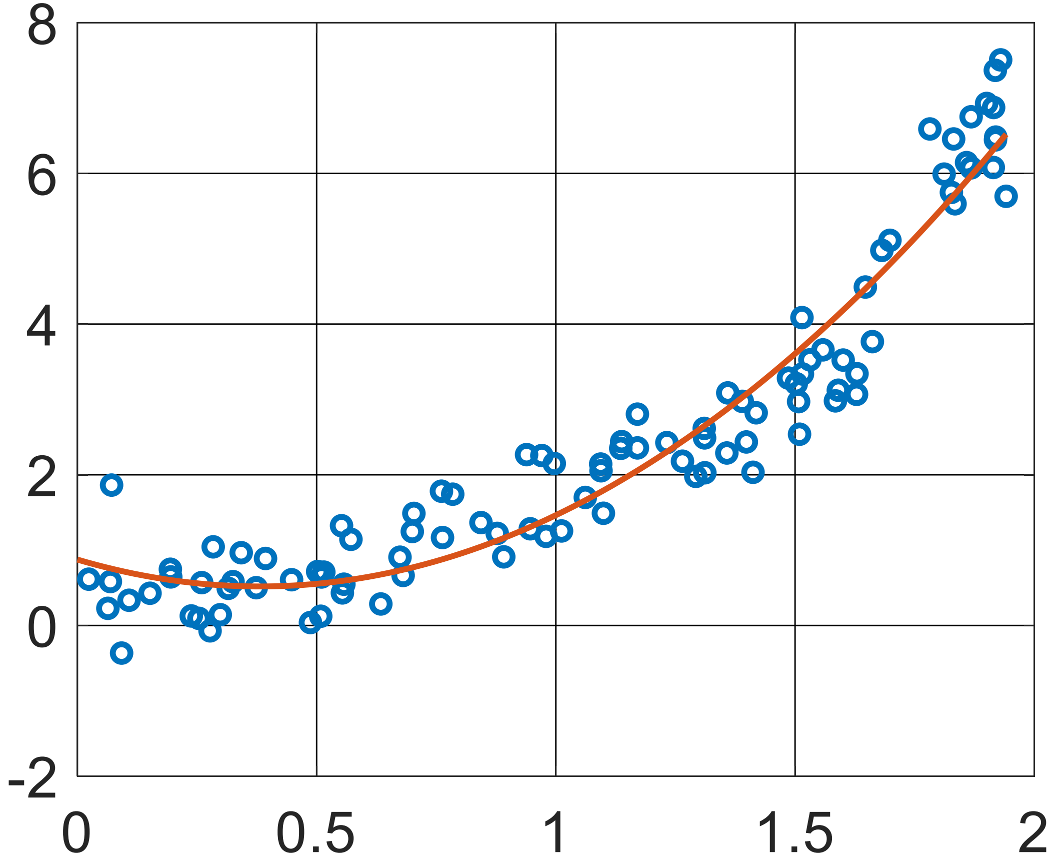
$$\boldsymbol{w}_{\mathrm{LS}} := \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i \in D_0} [y_i - f(\boldsymbol{x}_i; \boldsymbol{w})]^2$$

$$f(\boldsymbol{x}; \boldsymbol{w}) := \langle \boldsymbol{w}_1, \boldsymbol{\phi}(\boldsymbol{x}) \rangle + w_0, \boldsymbol{w} := [\boldsymbol{w}_1, w_0]^\top$$

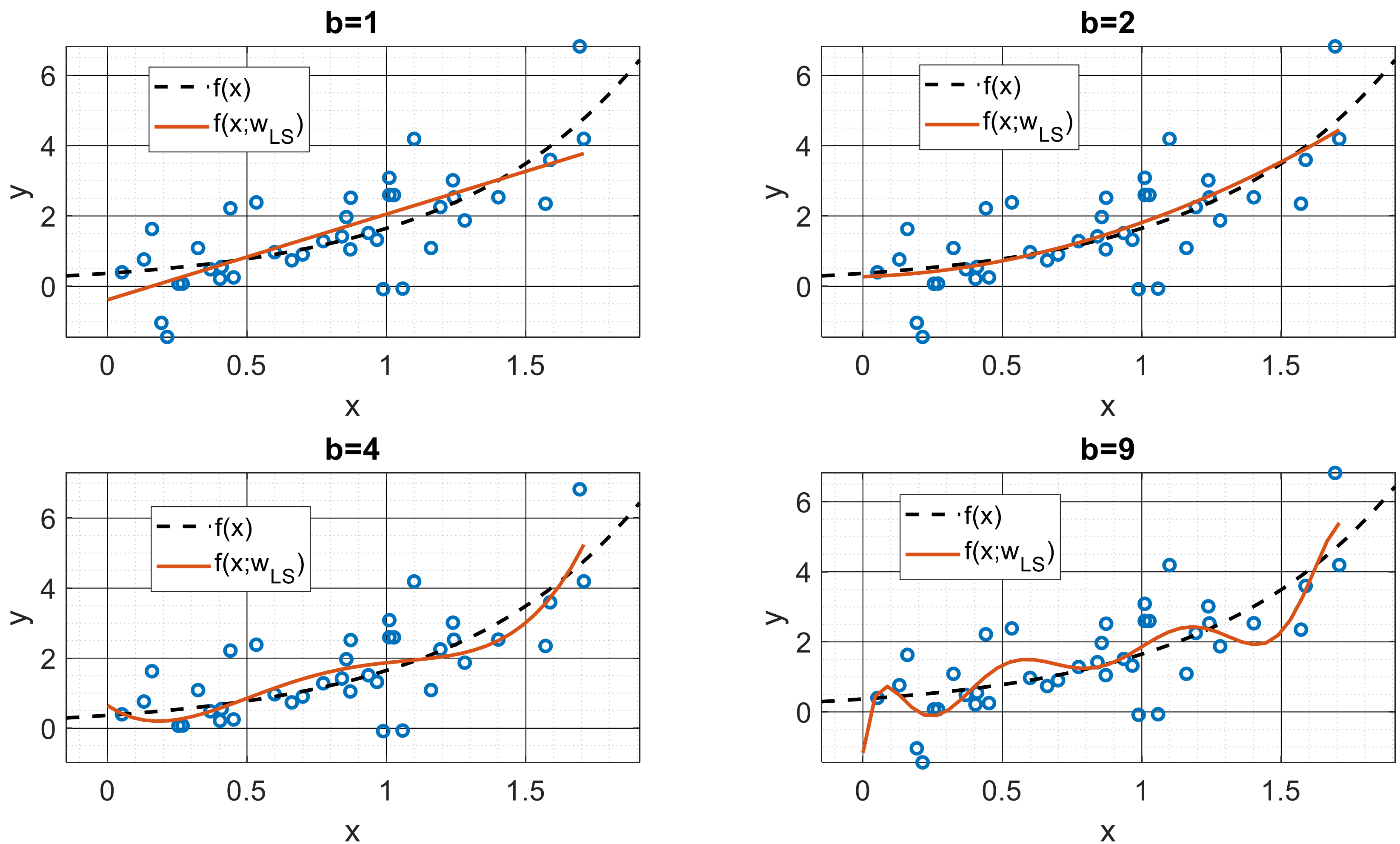
- $\boldsymbol{\phi}(x)$ can be a collection of polynomial functions:
- $\boldsymbol{\phi}(x) := [x^1, x^2, x^3 \dots x^b]^\top$.
- b is called the degree of $\boldsymbol{\phi}(x)$.

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 $f(x; \boldsymbol{w})$ is.

- However, when increasing b,
  - The fit of $f(x; \boldsymbol{w}_{\mathrm{LS}})$ first got better ($b = 2$).
  - then got worse ($b = 4, b = 9$).
  - $f(x; \boldsymbol{w}_{\mathrm{LS}})$ become too “squiggly”, when b is large.
  - $f(x; \boldsymbol{w}_{\mathrm{LS}})$ 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.
- $\boldsymbol{w}_{\mathrm{LS}}$ is fitted using D_0 only.

- Define an error $E(D', \boldsymbol{w}) = \sum_{i \in D'} [y_i - f(\boldsymbol{x}_i; \boldsymbol{w})]^2$.
- It tells how well $f(\boldsymbol{x}; \boldsymbol{w})$ fits a specific dataset D'.
- We can have **two performance metrics**:
- $E(D_0, \boldsymbol{w}_{\mathrm{LS}})$ is usually referred to as training error.
- $E(D_1, \boldsymbol{w}_{\mathrm{LS}})$ is usually referred to as testing error.

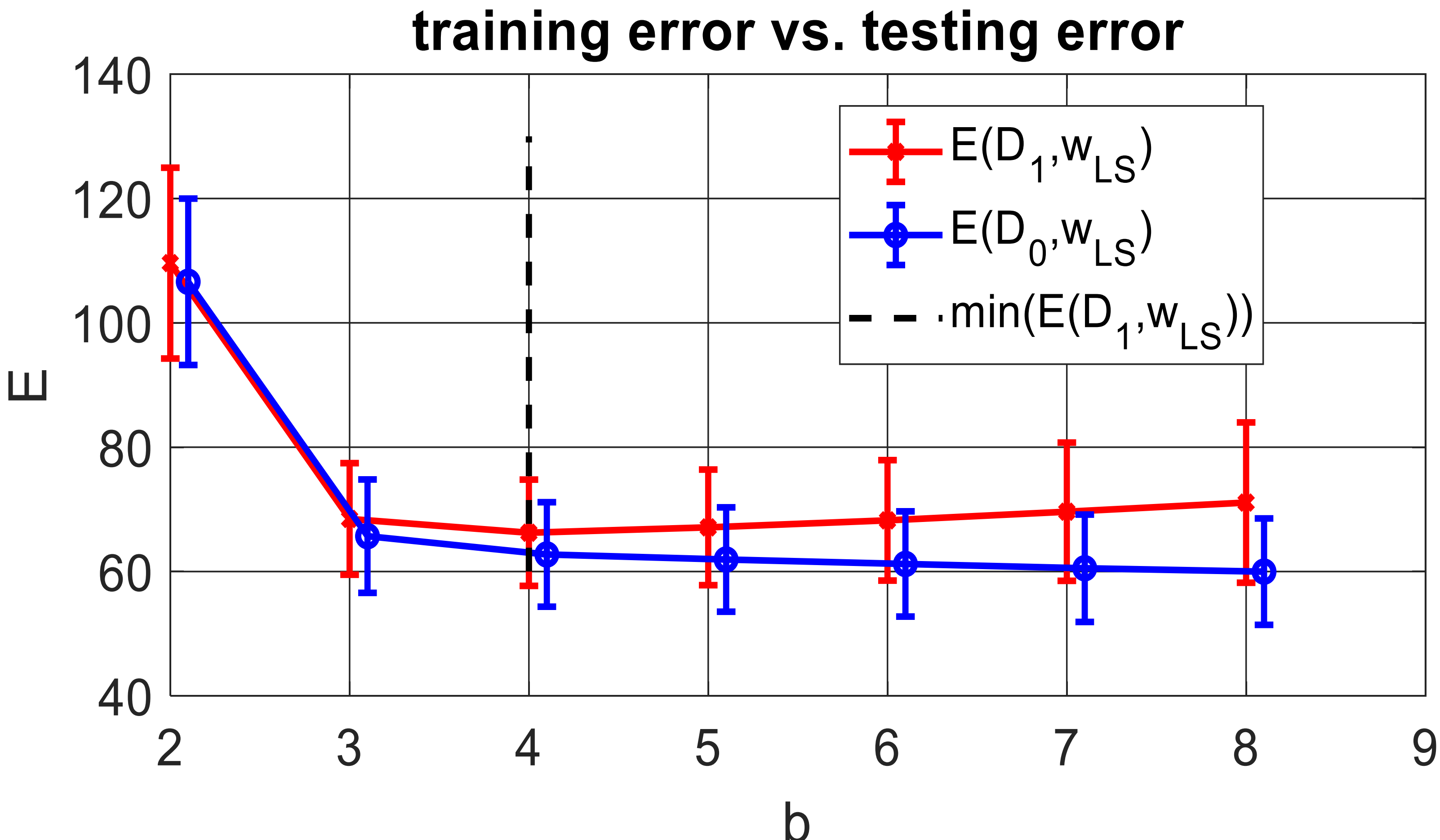
Training Error and Testing Error

- We do not care $E(D_0, w_{\mathrm{LS}})$!
- We have already seen the output in D_0 during the training.
- We care performance of $f(\boldsymbol{x}; \boldsymbol{w}_{\mathrm{LS}})$ on unseen dataset D_1!

- The ability of getting low $E(D_1, w_{\mathrm{LS}})$ is called generalization.
- Generalization is a key goal in statistical decision making.

- Go back to the example,
- As b increases, how $E(D_0, w_{\mathrm{LS}})$ and $E(D_1, w_{\mathrm{LS}})$ 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.
- $f(\boldsymbol{x}; \boldsymbol{w}_{\mathrm{LS}})$ fit D_0 better and better as b increases.
- Testing error drops then goes up again.
- $f(\boldsymbol{x}; \boldsymbol{w}_{\mathrm{LS}})$ does not fit unseen D_1 well, when b is too large.

- **The problem**:
- Generalization of $f(\boldsymbol{x}; \boldsymbol{w}_{\mathrm{LS}})$ deteriorates when b is too large.
- The phenomenon $f(\boldsymbol{x}; \boldsymbol{w}_{\mathrm{LS}})$ 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 $f_{\mathrm{LS}}(b)$ and use D_1 to compute $E(D_1, f_{\mathrm{LS}}(b))$.
- Select a b such that $E(D_1, f_{\mathrm{LS}}(b))$ is the lowest.
- Fit f_{LS} again using the selected b on the full dataset.

Selecting b (Efficiently!)

Problem of splitting D into D_0 and D_1:

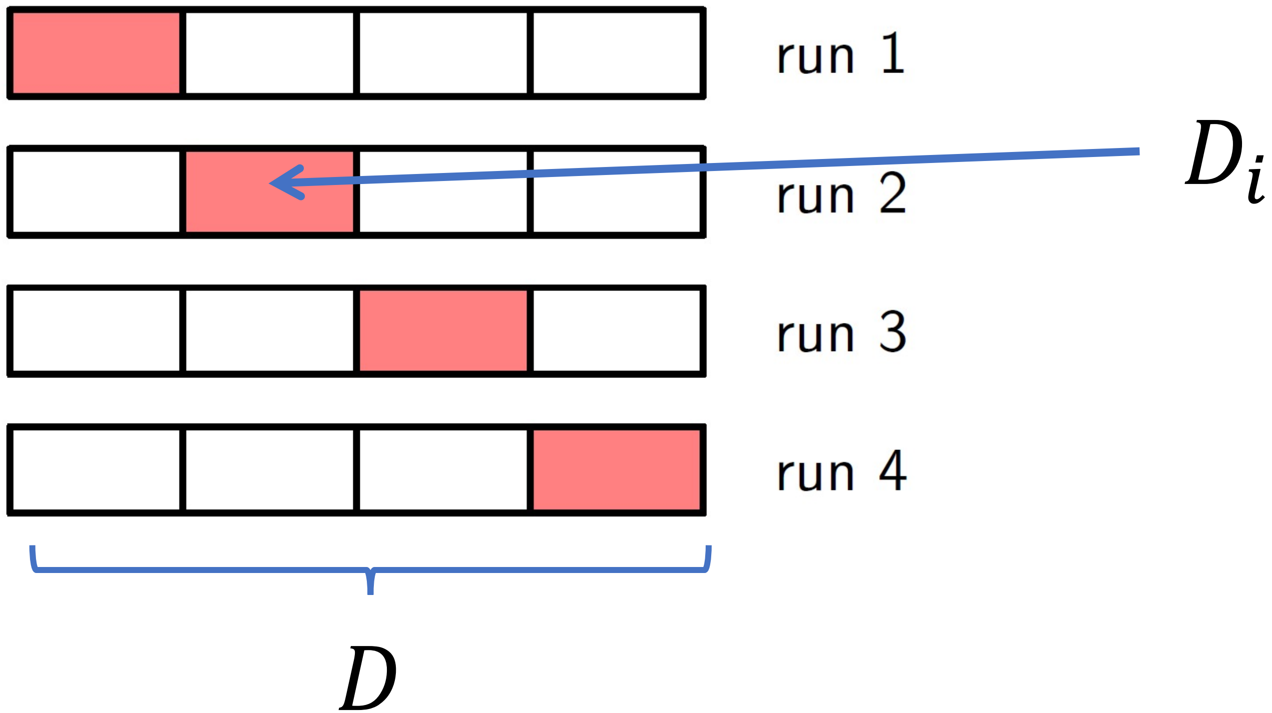
1. However, we have wasted D_1 for validation.
    - What if D_1 contains info that is beneficial for fitting a good f_{LS}?
2. Only computed $E\left(D_1, f_{\text{LS}}(b)\right)$ once, result may be random.

- Split D into D_0 and D_1, compute $E\left(D_1, f_{\text{LS}}(b)\right)$
- Swap the role of D_0 and D_1, compute $E\left(D_0, f'_{\text{LS}}(b)\right)$
    - $f'_{\text{LS}}(b)$ is fitted using D_1
- Select b that minimizes $\frac{E\left(D_1, f_{\text{LS}}(b)\right)}{2} + \frac{E\left(D_0, f'_{\text{LS}}(b)\right)}{2}$

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 $f_{\mathrm{LS}}^{(i)}(b)$ on all subsets but D_i, $\forall b$
  - Compute $E\left(D_i, f_{\mathrm{LS}}^{(i)}(b)\right)$, $\forall b$
- 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.
  - $f_{\mathrm{LS}}^{(i)}(x; \boldsymbol{w})$ 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 $\boldsymbol{x} \in R^d$, when the output y depends on multiple inputs?
- When $\boldsymbol{x} \in R^d$,
  - $\boldsymbol{\phi}(\boldsymbol{x}) := \left[\boldsymbol{h}\left(x^{(1)}\right), \boldsymbol{h}\left(x^{(2)}\right), \dots, \boldsymbol{h}\left(x^{(d)}\right)\right]^{\top}$.
  - $\boldsymbol{h}(t) := [t^1, t^2, \dots, t^b] \in R^b$.
  - $\boldsymbol{\phi}(\boldsymbol{x}) \in R^{db}$, which means $\boldsymbol{w}_1 \in R^{db}$.
- 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 $\boldsymbol{\phi}(\boldsymbol{x})$:
  - $\boldsymbol{\phi}(\boldsymbol{x}) := \left[\boldsymbol{h}\left(x^{(1)}\right), \dots, \boldsymbol{h}\left(x^{(d)}\right), \forall_{u<v} x^{(u)} x^{(v)}\right]$
  - $\boldsymbol{\phi}(\boldsymbol{x}) \in R^{db+\binom{d}{2}}$,
- Similarly, we can include all the **triplets**:
  - $\boldsymbol{\phi}(\boldsymbol{x}) :=$ $\left[\boldsymbol{h}\left(x^{(1)}\right), \dots, \boldsymbol{h}\left(x^{(d)}\right), \forall_{u<v} x^{(u)} x^{(v)}, \forall_{u<v<w} x^{(u)} x^{(v)} x^{(w)}\right]$
  - $\boldsymbol{\phi}(\boldsymbol{x}) \in R^{db+\binom{d}{2}+\binom{d}{3}}$,
- 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
  - $\binom{d}{1} + \binom{d}{2} + \binom{d}{3} + \binom{d}{4} + \cdots \binom{d}{d} = 2^d$
- We have not yet included cross terms like:
  - $\left[x^{(u)}\right]^2 x^{(v)} \ldots$
- The output dimension of $\boldsymbol{\phi}(\boldsymbol{x})$ 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 $\boldsymbol{\phi}(\boldsymbol{x})$, otherwise, we cannot obtain $\boldsymbol{w}_{\mathrm{LS}}$!

- 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 $\boldsymbol{x}$.
- Next week, we will introduce a way to reduce the flexibility of f to combat overfitting and the probabilistic idea behind it.

Computing Lab

- Download "Prostate Cancer dataset", description, dataset.
- Implement a Least-square solver using R. Do not use built-in functions.

- Fit $f(\boldsymbol{x}; \boldsymbol{w})$ 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?