Beyond Linear Regression Part I

Intro to Nonlinear Regression & Model Selection

송 준

고려대학교
통계학과 / 융합데이터과학 대학원

Recap: Goal of Regression Models

- **회귀 모형 (Regression Model)**:

$$\boldsymbol{Y} = \boldsymbol{f}(\boldsymbol{X}) + \boldsymbol{\epsilon}$$

- **Goal of Regression Models:**
  - **추정 (Estimation):** 관계를 나타내는 함수 f에 대한 추정
  - **예측 (Prediction)**: X 값이 주어졌을 때 대응되는 Y 값의 예측
  - **추론 (Inference)**: Further investigation
    - 예측이 "얼마나" 정확한가?
    - 함수 f() 가 얼마나 정확한가?
    - 예측변수가 여러 개 있을 때 모든 변수가 Y의 값에 영향을 주나?
    - 모형이 충분히 적합 됐나?

Recap: Goal of Regression Models

- **회귀 모형 (Regression Model)**:

$$Y = f(X) + \epsilon$$

- **Goal of Regression Models:**
  - **추정 (Estimation):** 관계를 나타내는 함수 f에 대한 추정
  - **예측 (Prediction)**: X 값이 x로 주어졌을 때 Y 값의 예측
  - **추론 (Inference)**: Further investigation of the data
    - 예측이 "얼마나" 정확한가?
    - 함수 f() 가 얼마나 정확한가?
    - 예측변수가 여러 개 있을 때 모든 변수가 Y의 값에 영향을 주나?
    - 모형이 충분히 적합 됐나?
  - **예측**만 목표로 할 시: 다양한 방법론 적용 가능
  - **추론**을 목표로 할 시: 관계를 나타내는 f()에 제약이 필요함
  - 단순한 모형부터 시작! **f 는 선형함수.**

Recap : Function Estimation

Optimal predictor:

$$f^* = argmin_f \mathbb{E}[(f(X) - Y)^2]$$

Empirical Risk Minimizer:

$$\widehat{f_n} = argmin_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} ((f(X_i) - Y_i))^2$$

Class of predictors *Empirical mean*

Recap : Linear Regression Algorithm

- **Input** : Dataset $Z = \{(x_1, y_1), \cdots, (x_n, y_n)\}$
- Compute

$$\hat{\beta}(Z) = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} L(\beta; Z)$$
$$= \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2$$

- **Output** : $f_{\hat{\beta}(Z)}(x) = \hat{\beta}(Z)^T x$
- Discuss algorithm for computing the minimal β later

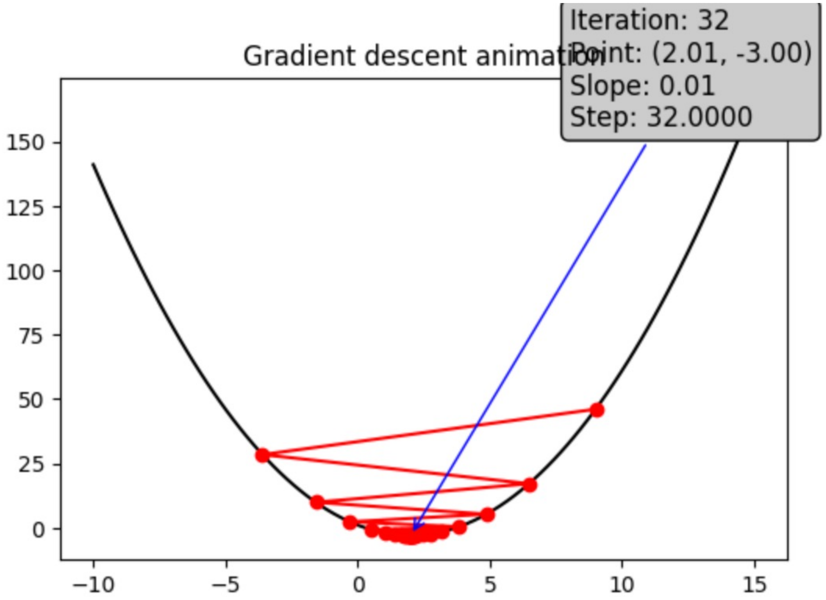
Recap : Solution to the Optimization Problem

- **Analytic Solution (Explicit form, closed form – solution): 미분=0**

$$L(\beta; x) = \beta^2 - 2x\beta + 10 = (\beta - x)^2 + 9$$

$$\arg\min_{\beta} L(\beta) = x$$

- **Numerical Solution (Optimization Algorithm)**



Recap: R^2

- **SST (총 편차제곱합)**

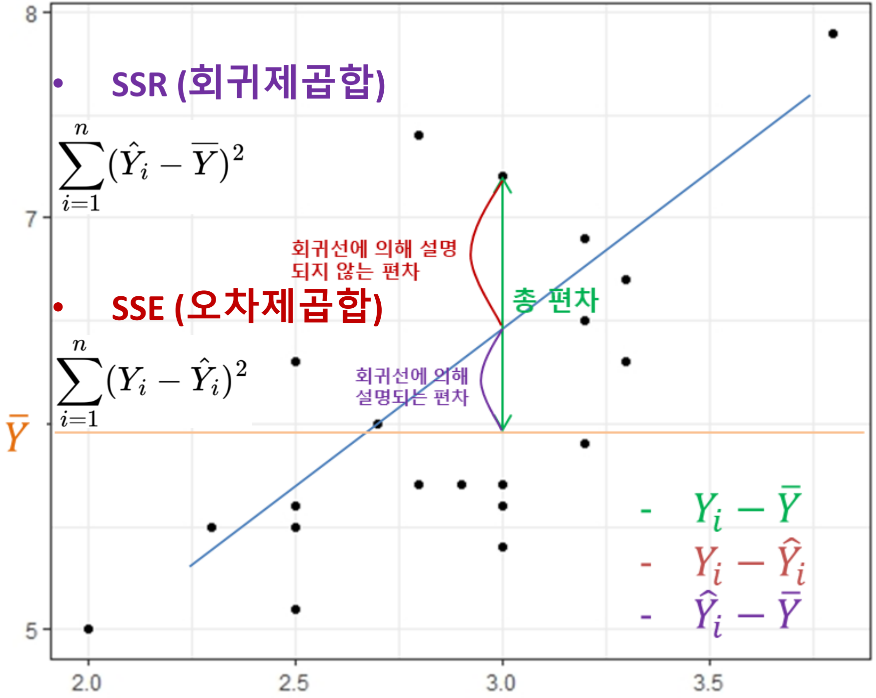
$$\sum_{i=1}^{n}(Y_i - \overline{Y})^2$$

- **SSR (회귀제곱합)**

$$\sum_{i=1}^{n}(\hat{Y}_i - \overline{Y})^2$$

- **SSE (오차제곱합)**

$$\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$



결정계수 $\boldsymbol{R^2}$ (coefficient of determination)

회귀직선의 적합도를 평가하는 방법

전체변동에서 회귀로 설명되는 부분이 차지하는 비율

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSR}{SST}$$

$$\sum_{i=1}^{n}(Y_i - \overline{Y})^2 = \sum_{i=1}^{n}(\hat{Y}_i - \overline{Y})^2 + \sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

SST(var) = **SSR** + **SSE** (training loss)

Feature Mapping

Function Approximation View of ML

ML algorithm outputs a model f that best “approximates” the given data Z

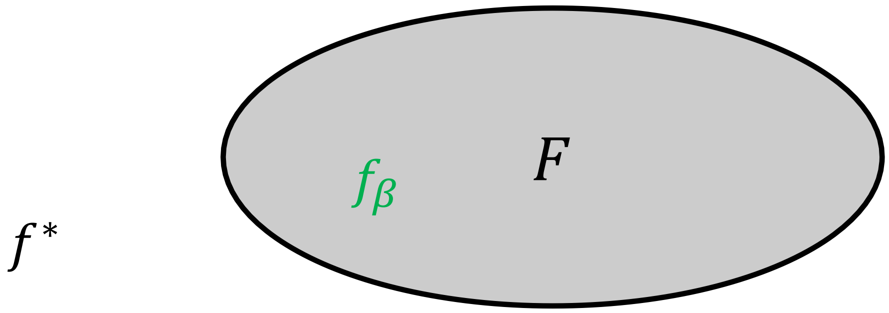
Function Approximation View of ML

- *Framework for designing machine learning algorithms*

- **Two design decisions**
  - *What is the family of candidate models* f*? (E.g., linear functions)*
  - *How to define "approximating"? (E.g., MSE loss)*

Aside : "True Function"

- **Input:** *Dataset* Z
  - *Presume there is an unknown function* f^* *that generates* Z
- **Goal:** *Find an approximation* $f_\beta \approx f^*$ *in our model family* $f_\beta \in F$
  - *Typically,* f^* *not in our model family* F



Function Approximation View of ML

- *Framework for designing machine learning algorithms*
- **Two design decisions**
  - *What is the family of candidate models* f*? (E.g., linear functions)*
  - *How to define "approximating"?(E.g., MSE loss)*
- *How do we specialize to linear regression?*

Loss Minimization

Data Z → *Machine learning algorithm* → *Model* f

Linear Regression

Data $Z = \{(x_i, y_i)\}_{i=1}^{n}$

$\hat{\beta}(Z) = \underset{\beta}{\operatorname{argmin}}\, L(\beta; Z)$

L encodes $y_i \approx f_\beta(x_i)$

Model $f_{\hat{\beta}(Z)}$

MSE loss

Model is a linear function $f_\beta(x) = \beta^T x$

Linear Regression

General strategy

- Model family $\mathrm{F} = \{f_\beta\}_\beta$
- Loss function $L(\beta; Z)$

Linear regression strategy

- Linear functions $F = \{f_\beta(x) = \beta^T x\}$
- MSE $L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)^2$

Linear regression algorithm

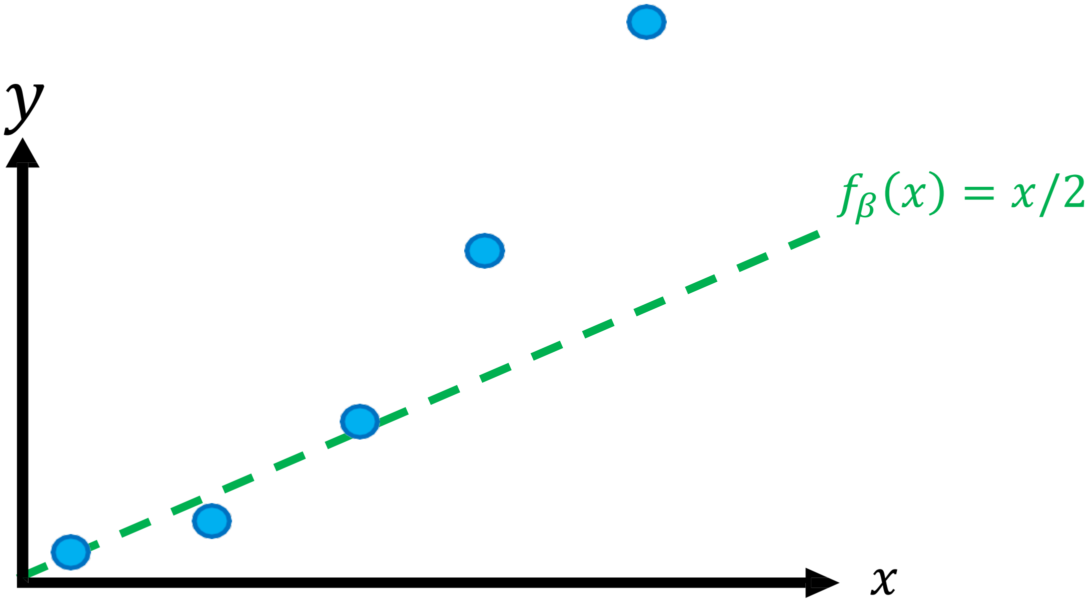
$$\hat{\beta}(Z) = \arg\min_\beta L(\beta; Z)$$

Agenda

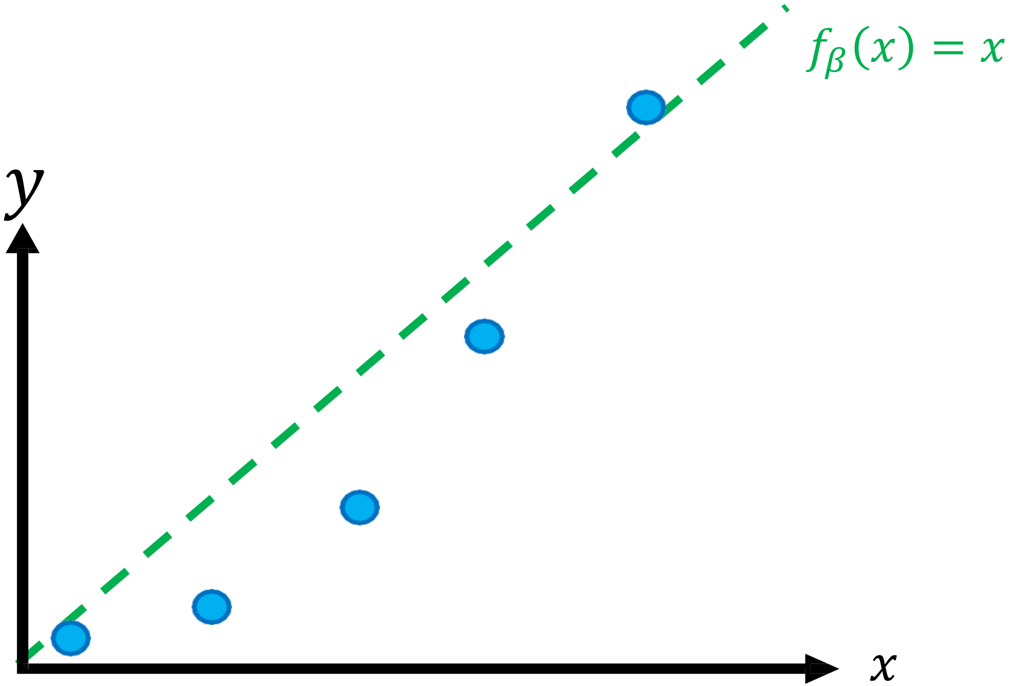
- **Function approximation view of machine learning**
  - *Modern strategy for designing machine learning algorithms*
  - **By example:** *Linear regression, a simple machine learning algorithm*

- **Bias-variance tradeoff**
  - *Fundamental challenge in machine learning*
  - **By example:** *Linear regression with feature maps*

Example: Quadratic Function



Example: Quadratic Function



Can we get a better fit?

Feature Maps

General strategy

- *Model family* $F = \{f_\beta\}_\beta$
- *Loss function* $L(\beta; Z)$

Linear regression with feature map

- Linear functions over a given **feature map** $\phi: X \to \mathbb{R}^d$

$$F = \{f_\beta(x) = \beta^T \phi(x)\}$$

- MSE $L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T \phi(x_i))^2$

Quadratic Feature Map

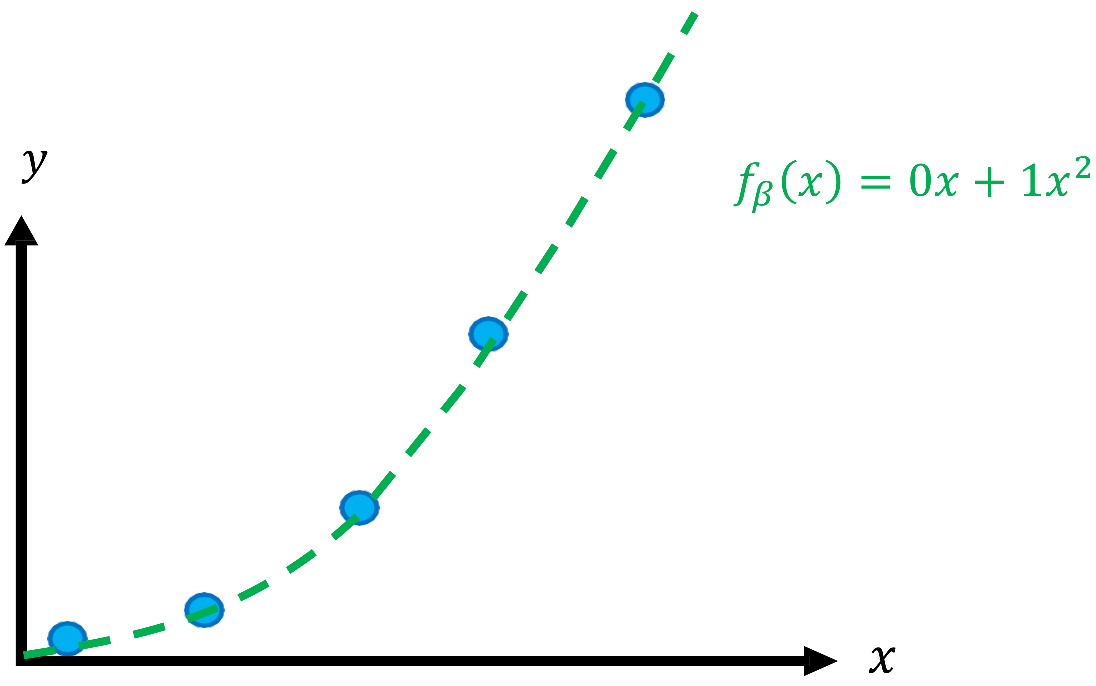
- *Consider the feature map* $\phi: \mathbb{R} \to \mathbb{R}^2$ *given by*

$$\phi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$$

- *Then, the model family is*

$$f_\beta(x) = \beta_1 x + \beta_2 x^2$$

Quadratic Feature Map



In our family for $\beta = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$!

Feature Maps

- *Powerful strategy for encoding prior knowledge*

- **Terminology**
  - x is *the* **input** *and* $\phi(x)$ *are the* **features**
  - *Often used interchangeably*

Examples of Feature Maps

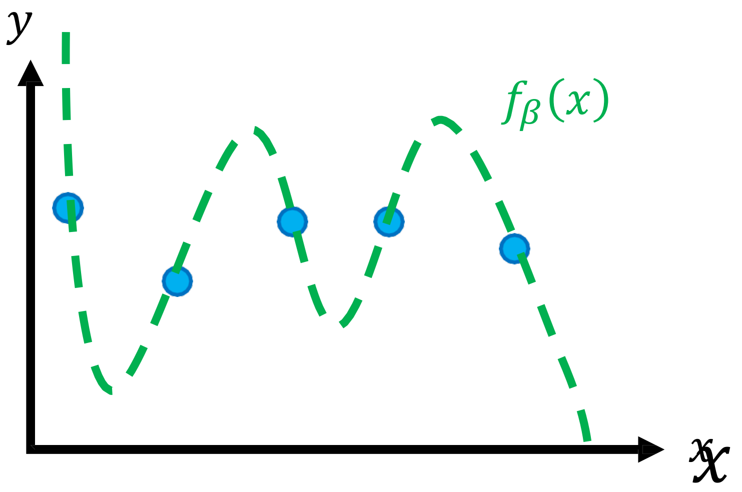
- **Polynomial features**
  - $f_\beta(x) = \beta_1 + \beta_2 x_1 + \beta_3 x_2 + \beta_4 x_1^2 + \beta_5 x_1 x_2 + \beta_6 x_2^2 + \cdots$
  - *Quadratic features are very common; capture "feature interactions"*
  - *Can use other nonlinearities (exponential, logarithm, square root, etc.)*
- **Basis expansion approach**
  - $f_\beta(x) = \beta_0 + \beta_1 \phi_1(x) + \cdots + \beta_d \phi_d(x)$
  - *Fit the data in a more general way*
- **Encoding non-real inputs**
  - *E.g.,* x = *"the food was good" and* y = *4 stars*
  - $\phi(x) = [1(\text{"}good\text{"} \in x) \quad 1(\text{"}bad\text{"} \in x) \quad \cdots]^T$

Algorithm

- *Reduces to linear regression*
- **Step 1:** *Compute* $\phi_i = \phi(x_i)$ *for each* x_i *in* Z
- **Step 2:** *Run linear regression with* $Z' = \{(\phi_1, y_1), \cdots, (\phi_n, y_n)\}$

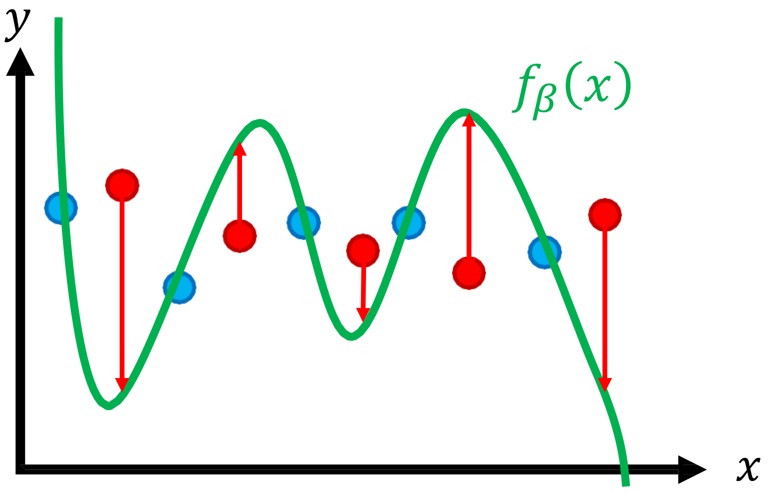
Question

- **Why not throw in lots of features?**
  - $f_\beta(x) = \beta_1 + \beta_2 x_1 + \beta_3 x_2 + \beta_4 x_1^2 + \beta_5 x_1 x_2 + \beta_6 x_2^2 + \cdots$
  - *Can fit any n points using a polynomial of degree n*



Question

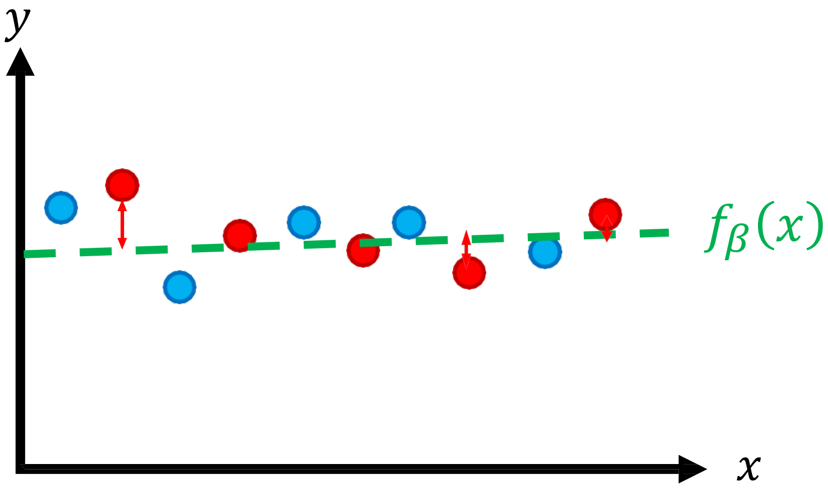
- **Issue: prediction with a new data**
  - *Given a* **new** *input* x, *predict the label* $\hat{y} = f_\beta(x)$



The errors on new inputs is very large!

Question

- **Issue: prediction with a new data**
  - *Given a* **new** *input* x, *predict the label* $\hat{y} = f_\beta(x)$



Vanilla linear regression actually works better!

Model Selection Basic

Training vs. Test Data

- **Training data:** *Examples* $Z = \{(x, y)\}$ *used to fit our model*

- **Test data:** *New inputs* x *whose labels* y *we want to predict*

- **Goal**: *Find a model that works well on the test data (unseen data)*

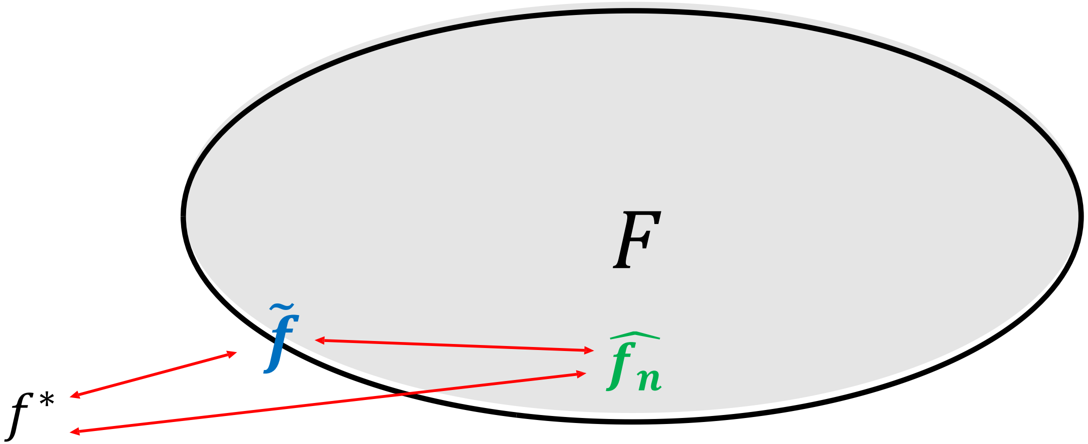
Recap : Function Estimation

Ideal goal: *Construct prediction rule* $f^* : \mathcal{X} \rightarrow \mathcal{Y}$

$$\boldsymbol{f}^* = \boldsymbol{argmin_f} \mathbb{E}_{\boldsymbol{XY}}[\boldsymbol{loss}(\boldsymbol{Y}, \boldsymbol{f}(\boldsymbol{X}))]$$

$$\tilde{\boldsymbol{f}} = arg \min_{f \in F} \mathbb{E}_{XY}[\text{loss}(Y, f(X))]$$

$$\widehat{\boldsymbol{f}_{\boldsymbol{n}}} = arg \min_{f \in F} \sum_{i=1}^{n} \text{loss}(Y_i, f(X_i))$$



Training Loss (MSE) v.s Test Loss (MSE)

Ideal goal: *Construct prediction rule* $f^* : \mathcal{X} \to \mathcal{Y}$

$$\boldsymbol{f^* = argmin_f \mathbb{E}_{XY}[loss(Y, f(X))]}$$

$$\tilde{\boldsymbol{f}} = \arg\min_{f \in F} \mathbb{E}_{XY}[\text{loss}(Y, f(X))]$$

$$\widehat{\boldsymbol{f_n}} = \arg\min_{f \in F} \sum_{i=1}^{n} \text{loss}\left(Y_i, f(X_i)\right)$$

It is obtained using the data

Given the estimated function from the data, $\widehat{\boldsymbol{f_n}}$,

$$\boxed{\textbf{Test MSE: } \mathbb{E}_{XY}\left[\text{loss}\left(Y, \widehat{f_n}(X)\right)\right]}$$

We need to minimize the Test MSE!

Training MSE: $\frac{1}{n}\sum_{i=1}^{n} \text{loss}\left(Y_i, \widehat{f_n}(X_i)\right)$

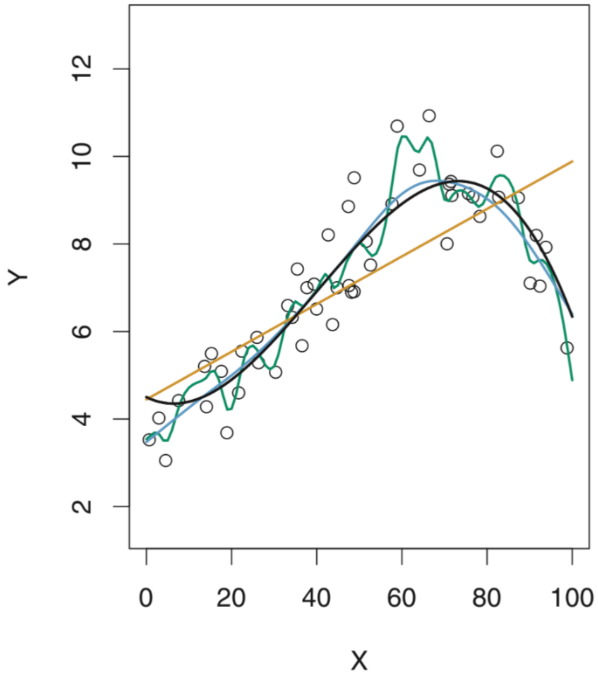
Training Loss (MSE) v.s Test Loss (MSE)

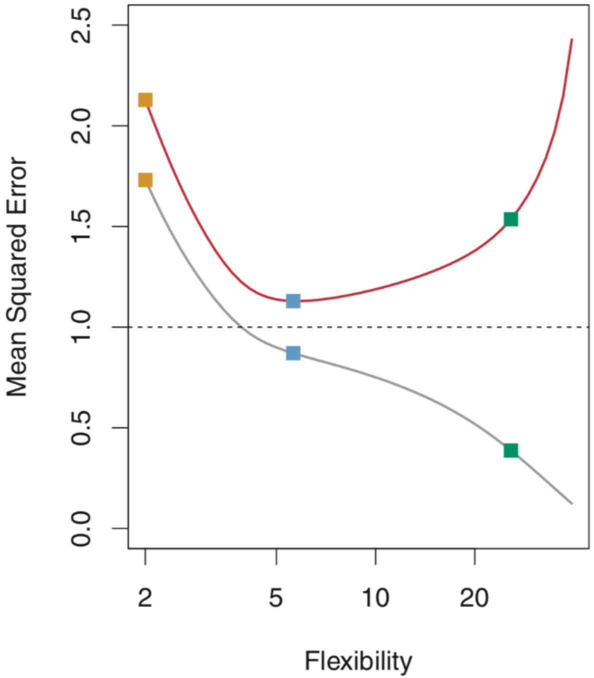
- *If no test observations are available? Should we choose the method that minimizes the training MSE?*

- *No! There is no guarantee that the method with the smallest training MSE will have the smallest test MSE*

Training Loss (MSE) v.s Test Loss (MSE)

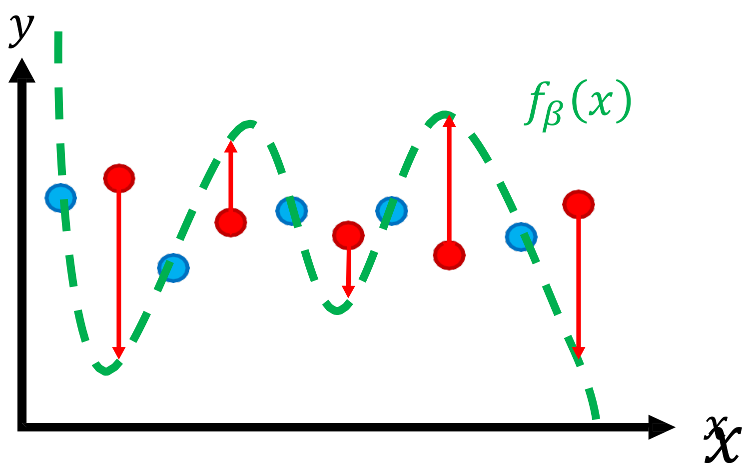
- *Training MSE (grey) decreases monotonically as the model flexibility increases and Test MSE (red) has U-shape*



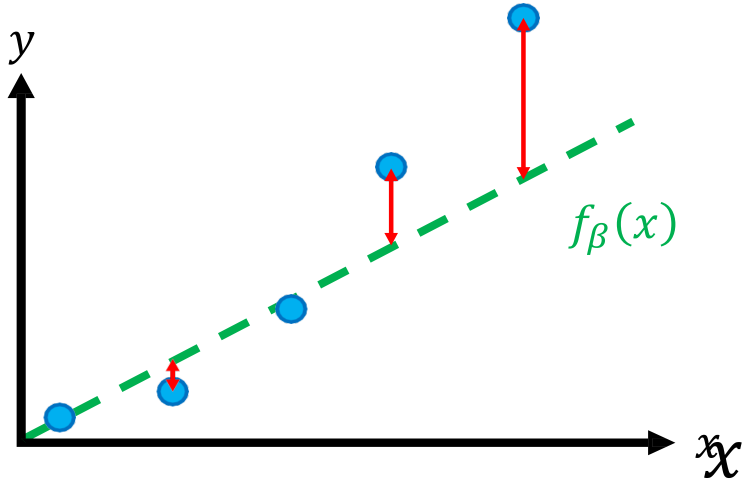


Overfitting v.s Underfitting

- **Overfitting**
  - *Fit the* **training data** Z *well*
  - *Fit new* **test data** (x, y) *poorly*



- **Underfitting**
  - *Fit the* **training data** Z *poorly*
  - *(Necessarily fit new* **test data**
  - (x, y) *poorly)*

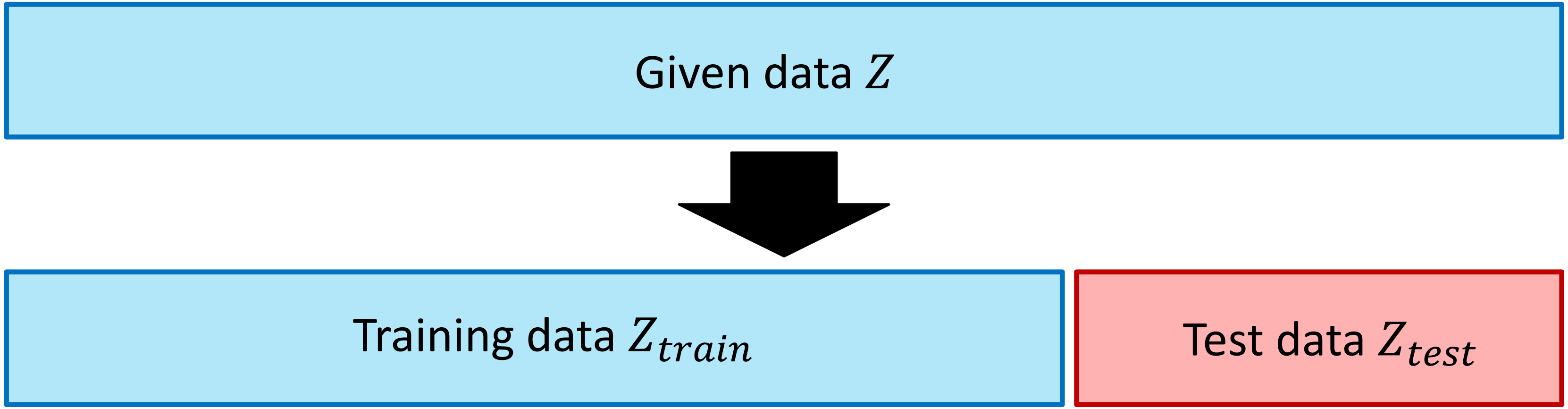


Training Loss (MSE) vs Test Loss (MSE)

- *See the <u>example</u>*
- *There is no way to have the true Test MSE*
- *Estimate the TEST MSE!*

Training/Test Split

- **Issue:** *How to detect overfitting vs. underfitting?*
- **Solution:** *Use* **held-out test data** *to estimate loss on new data*
  - *Typically, randomly shuffle data first*



Training/Test Split Algorithm

- **Step 1:** *Split* Z *into* Z_{train} *and* Z_{test}

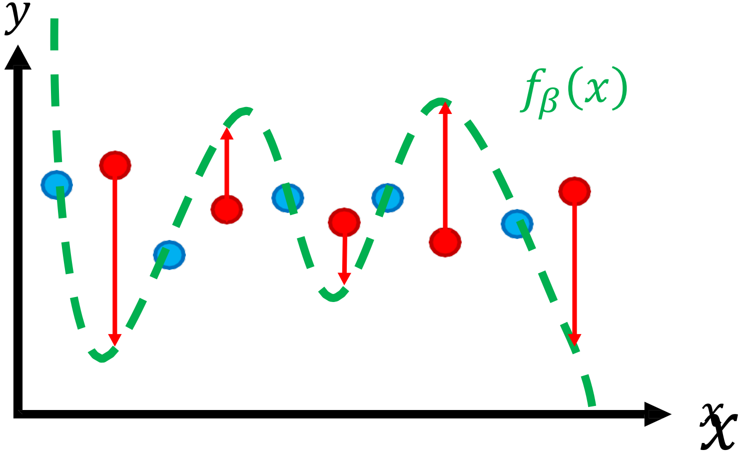
Training data Z_{train} | *Test data* Z_{test}

- **Step 2:** *Run linear regression with* Z_{train} *to obtain* $\hat{\beta}(Z_{train})$

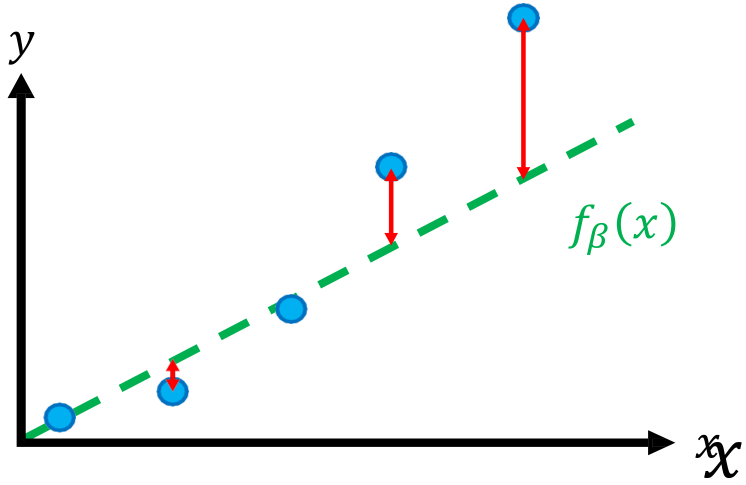
- **Step 3:** *Evaluate*
  - **Training loss**: $L_{train} = L\left(\hat{\beta}(Z_{train}); Z_{train}\right)$
  - **Estimated Test (or generalization) loss**: $L_{test} = L\left(\hat{\beta}(Z_{train}); Z_{test}\right)$

Training/Test Split Algorithm

- **Overfitting**
  - *Fit the* **training data** Z *well*
  - *Fit new* **test data** (x, y) *poorly*

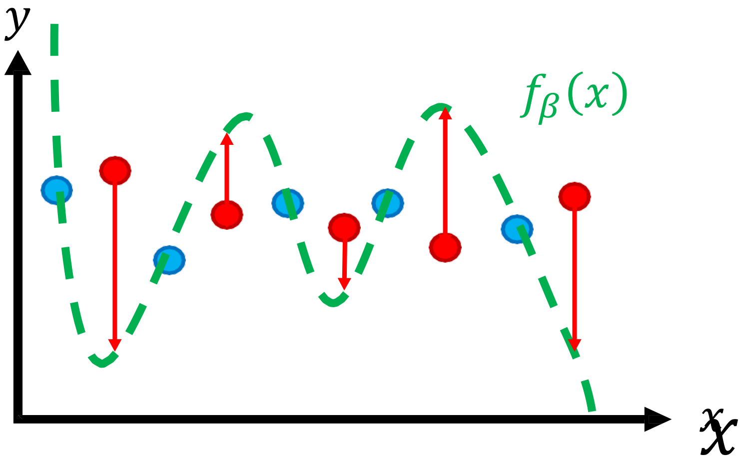


- **Underfitting**
  - *Fit the* **training data** Z *poorly*
  - *(Necessarily fit new* **test data** (x, y) *poorly)*

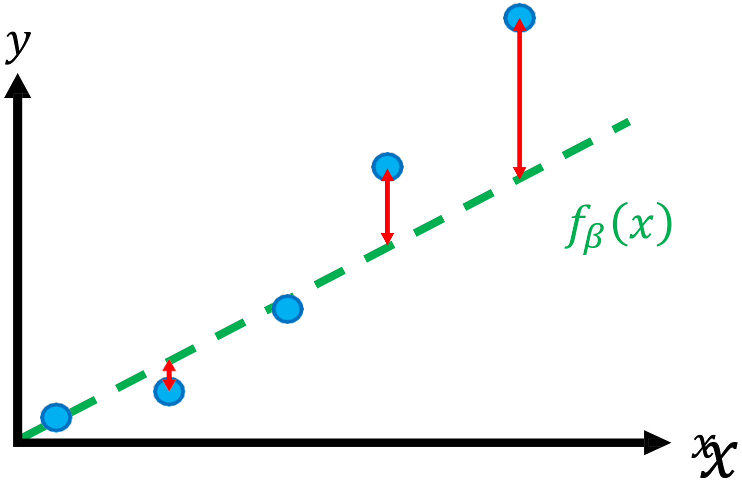


Training/Test Split Algorithm

- **Overfitting**
  - L_{train} *is small*
  - L_{test} *is large*

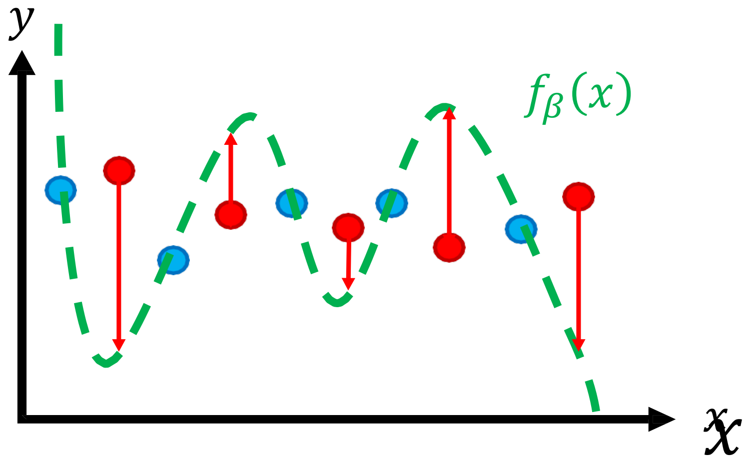


- **Underfitting**
  - *Fit the* ***training data*** Z *poorly*
  - *(Necessarily fit new* ***test data*** (x, y) *poorly)*

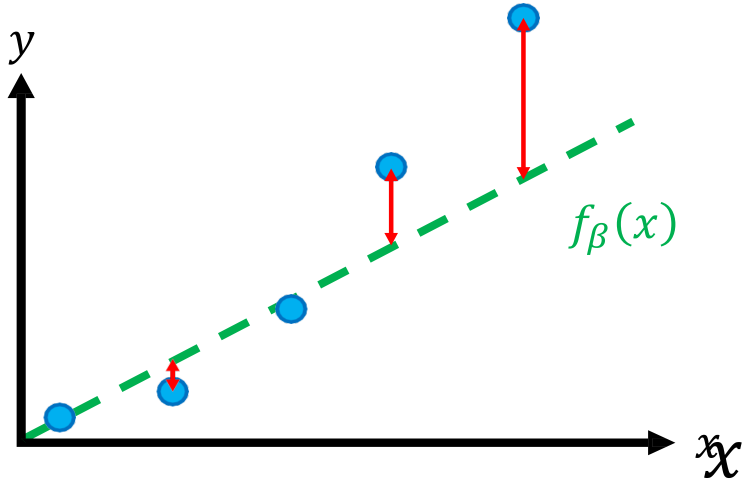


Training/Test Split Algorithm

- **Overfitting**
  - L_{train} *is small*
  - L_{test} *is large*



- **Underfitting**
  - L_{train} *is large*
  - L_{test} *is large*



Aside: IID Assumption

- **Underlying IID assumption**
  - *Future data are drawn IID from same data distribution* $P(x, y)$ *as* Z_{test}
  - *IID = independent and identically distributed*
  - *This is a strong (but common) assumption!*

- **Time series data**
  - *Particularly important failure case since data distribution may shift over time*
  - **Solution:** *Split along time (e.g., data before 2024.05.18 vs. data after 2024.05.18)*

How to fix Underfitting/Overfitting?

- *Choose the right model family!*

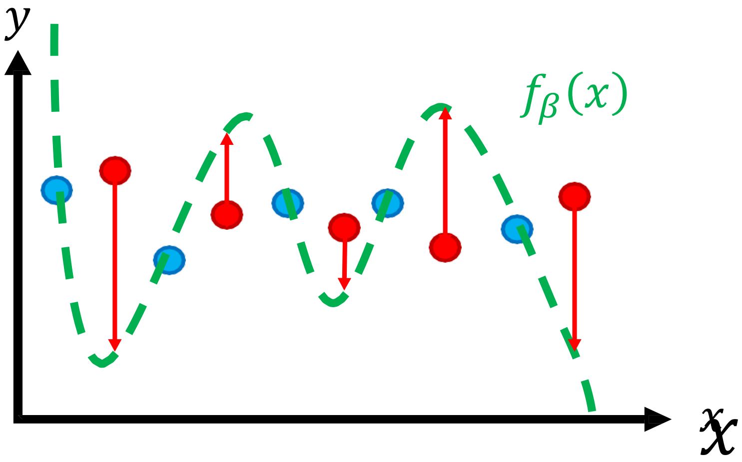
Role of Capacity

- **Capacity** *of a model family captures "complexity" of data it can fit*
  - *Higher capacity more likely to overfit(model family has high* **variance***)*
  - *Lower capacity more likely to underfit(model family has high* **bias***)*

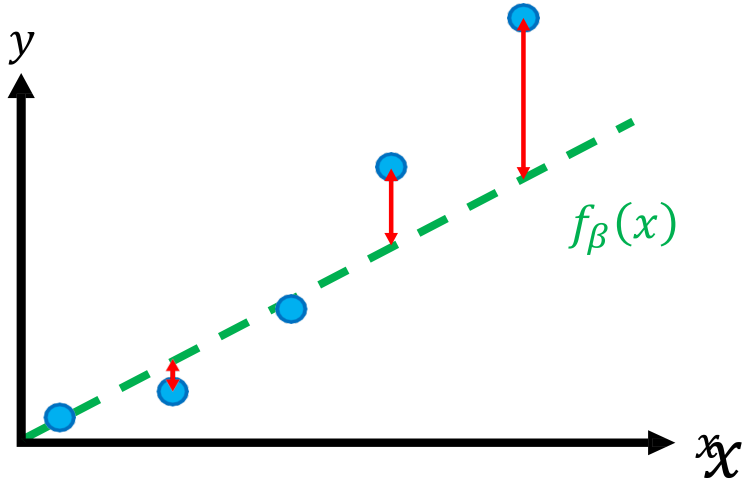
- *For linear regression, capacity corresponds to feature dimension* d
  - *I.e., number of features in* $\phi(x)$

Bias-Variance Tradeoff

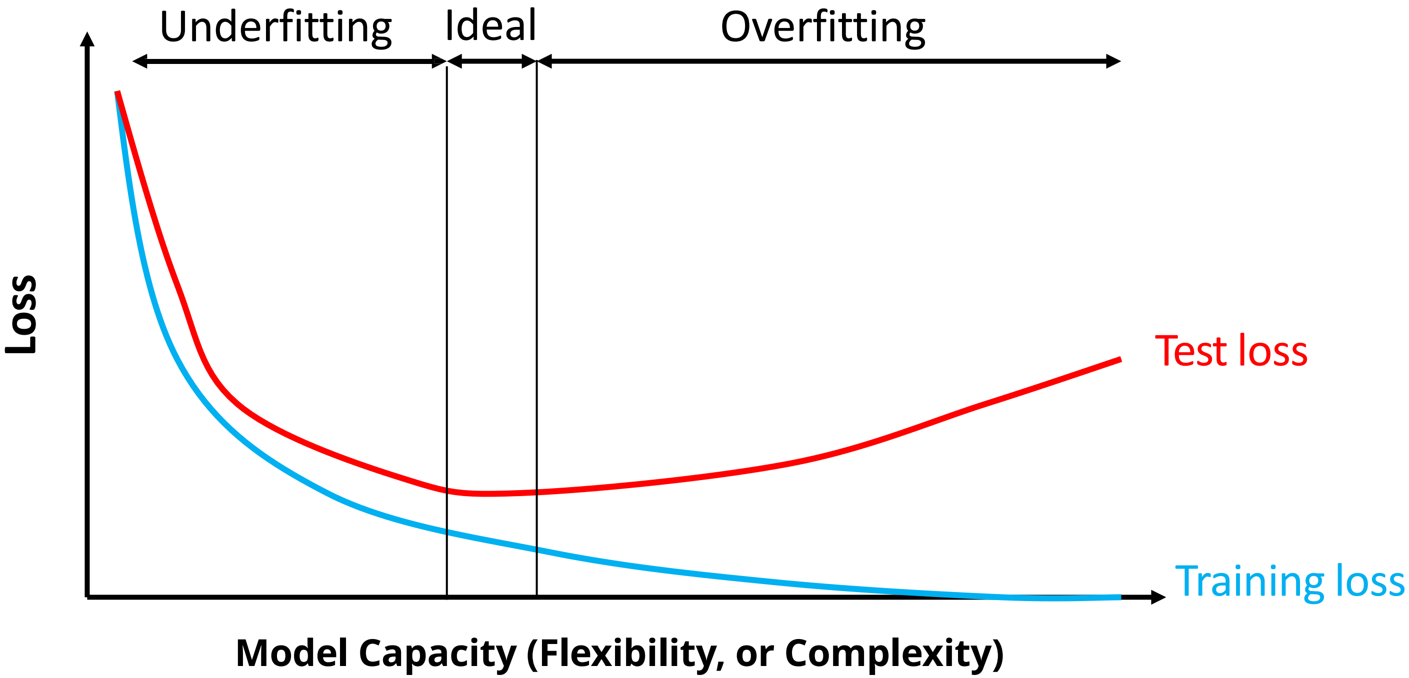
- **Overfitting(high variance)**
  - *High capacity model capable of fitting complex data*
  - *Insufficient data to constrain it*



- **Underfitting(high bias)**
  - *Low capacity model that can only fit simple data*
  - *Sufficient data but poor fit*



Bias-Variance Tradeoff



Slide by Padhraic Smyth, UCIrvine

Bias-Variance Tradeoff

- *For linear regression, increasing feature dimension d…*
  - *Tends to* **increase capacity**
  - *Tends to decrease bias but increase variance*
- *Need to construct ϕ to balance tradeoff between bias and variance*
  - **Rule of thumb***: $n \approx d \log d$*
  - *Large fraction of data science work is data cleaning + feature engineering*

Bias-Variance Tradeoff

- *Increasing number of examples n in the data…*
  - *Tends to* **increase bias** *and* **decrease variance**

- **General strategy**
  - **High bias:** *Increase model capacity d*
  - **High variance:** *Increase data size n (i.e., gather more labeled data)*

Bias-Variance Tradeoff

Ideal goal: *Construct prediction rule* $f^*: \mathcal{X} \rightarrow \mathcal{Y}$

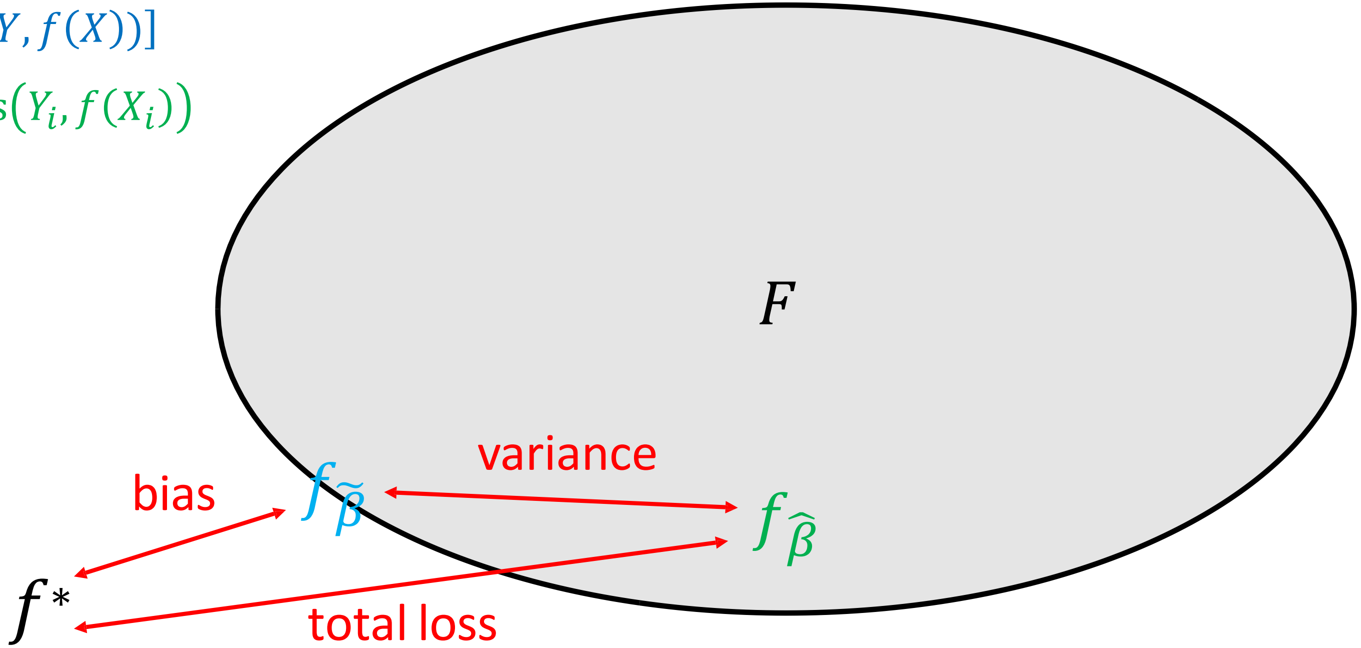
$$\boldsymbol{f^* = argmin_f \mathbb{E}_{XY}[loss(Y, f(X))]}$$

$$\tilde{\boldsymbol{f}} = \arg\min_{f \in F} \mathbb{E}_{XY}[\text{loss}(Y, f(X))]$$

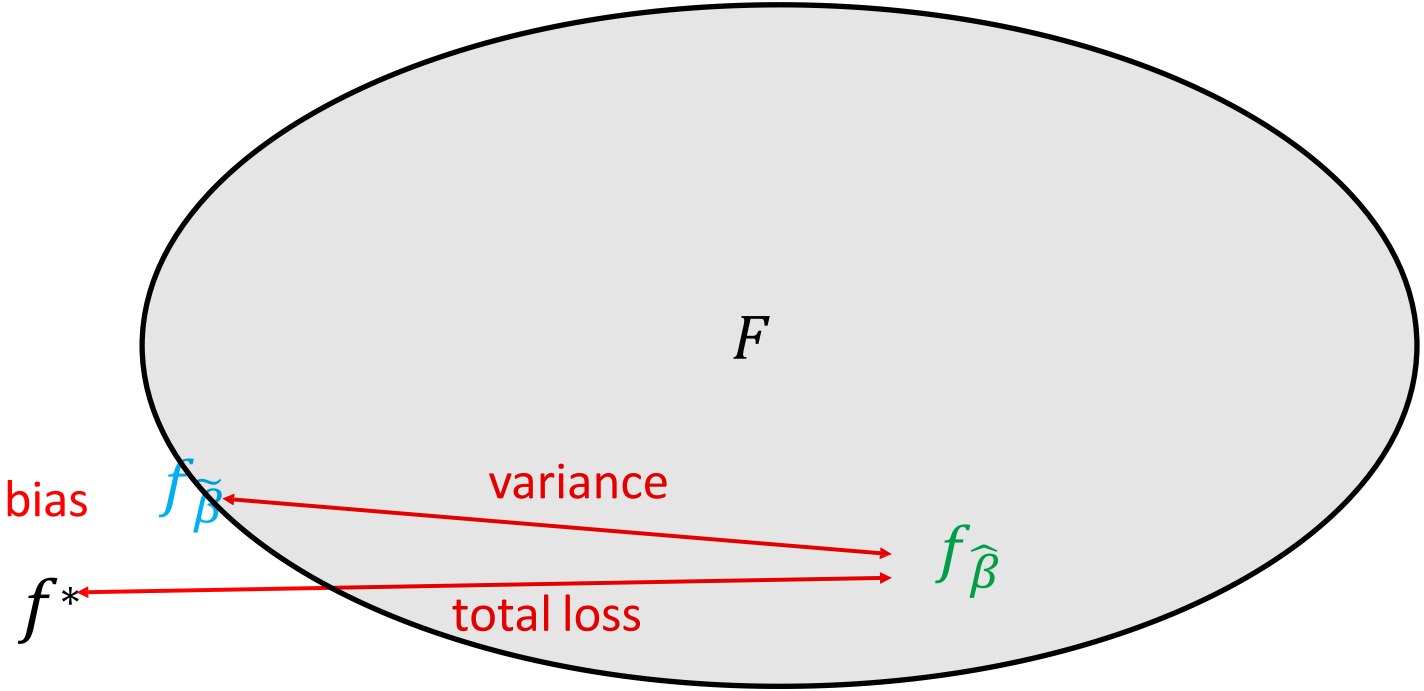
$$\widehat{\boldsymbol{f_n}} = \arg\min_{f \in F} \sum_{i=1}^{n} \text{loss}(Y_i, f(X_i))$$

Overfitting?

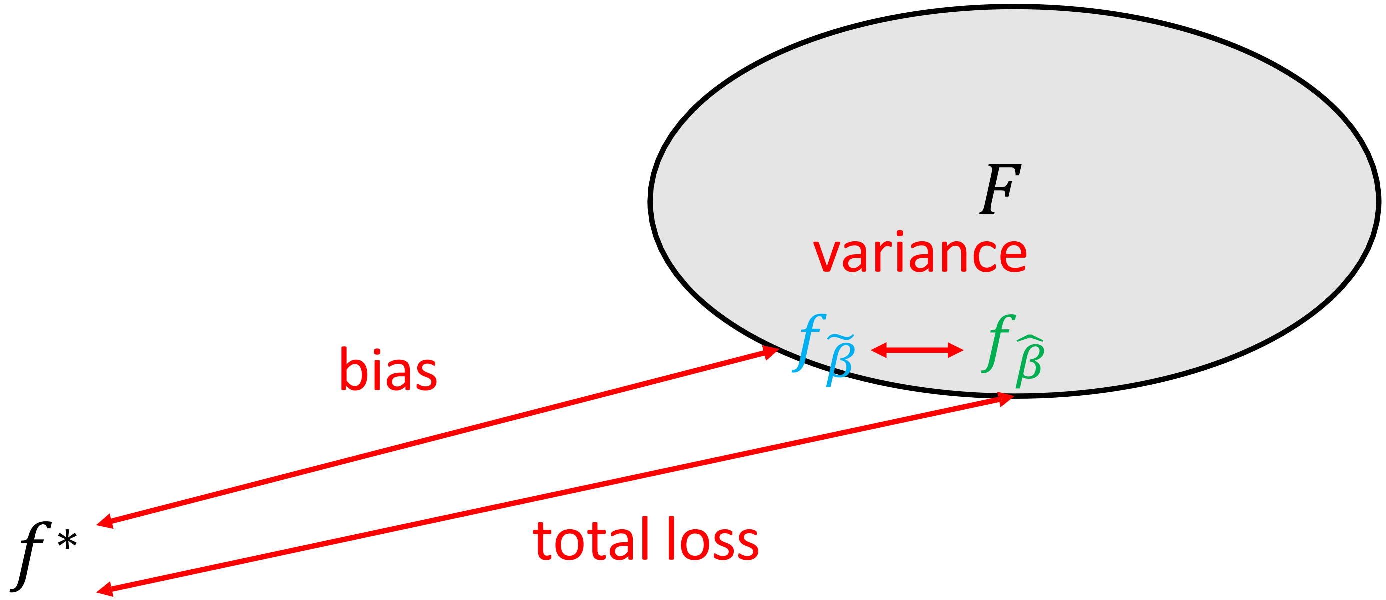
Underfitting?



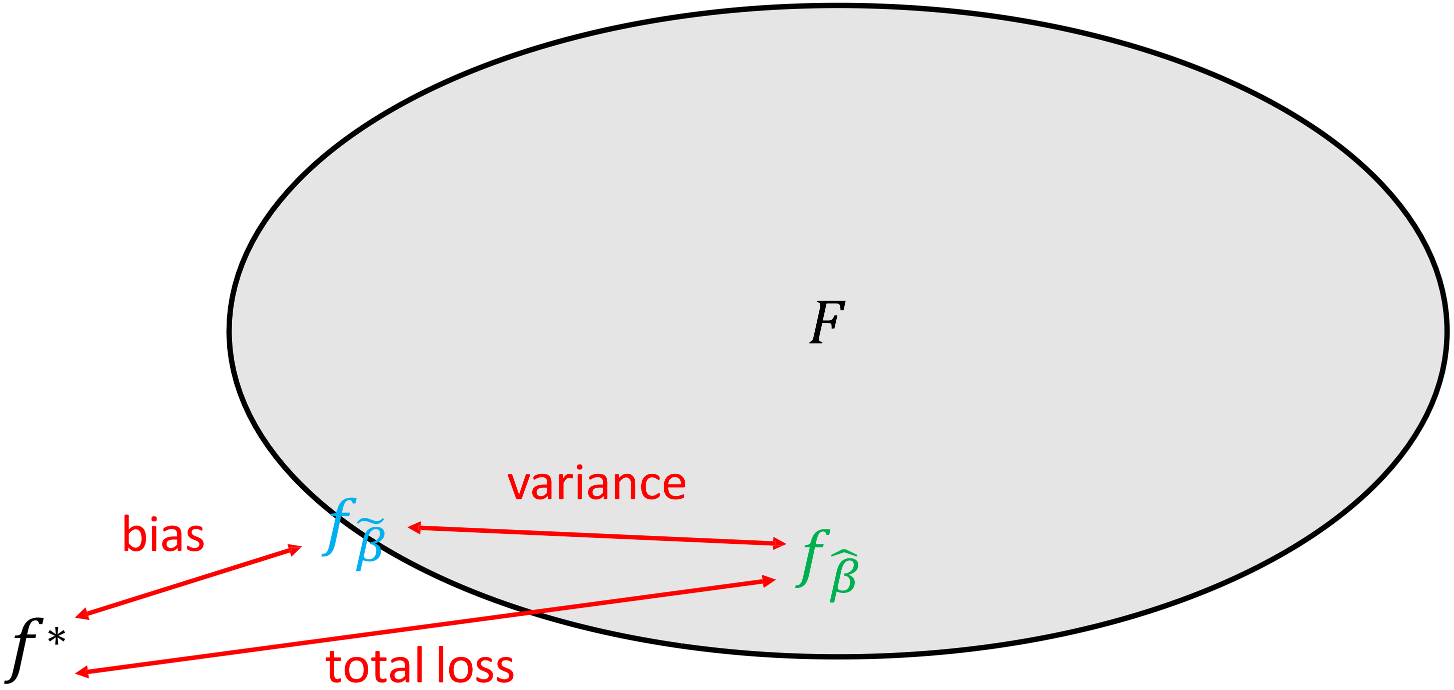
Bias-Variance Tradeoff(Overfitting)



Bias-Variance Tradeoff(Underfitting)



Bias-Variance Tradeoff(Ideal)

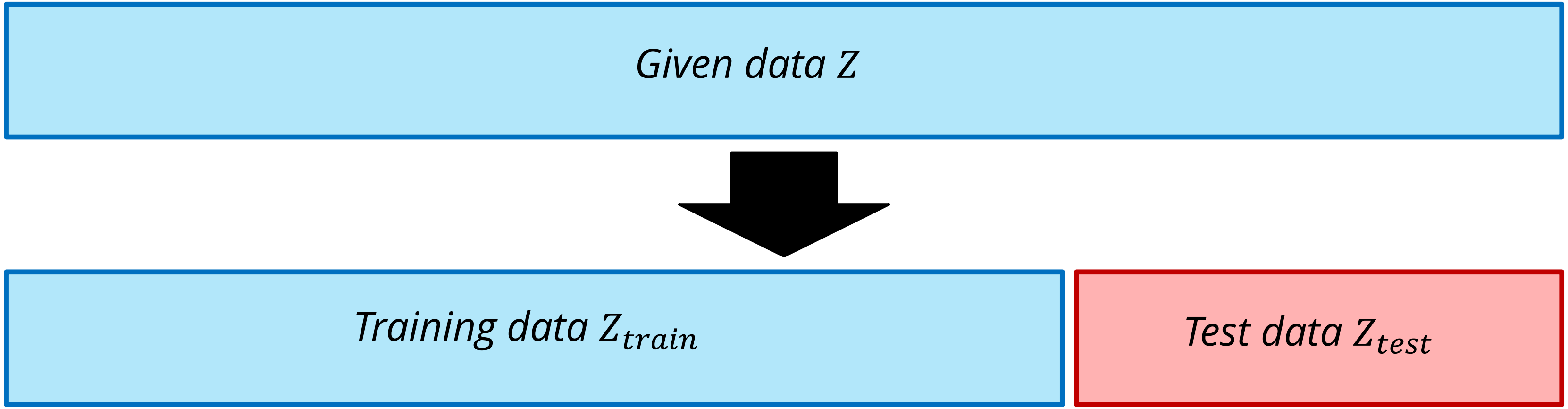


Cross Validation

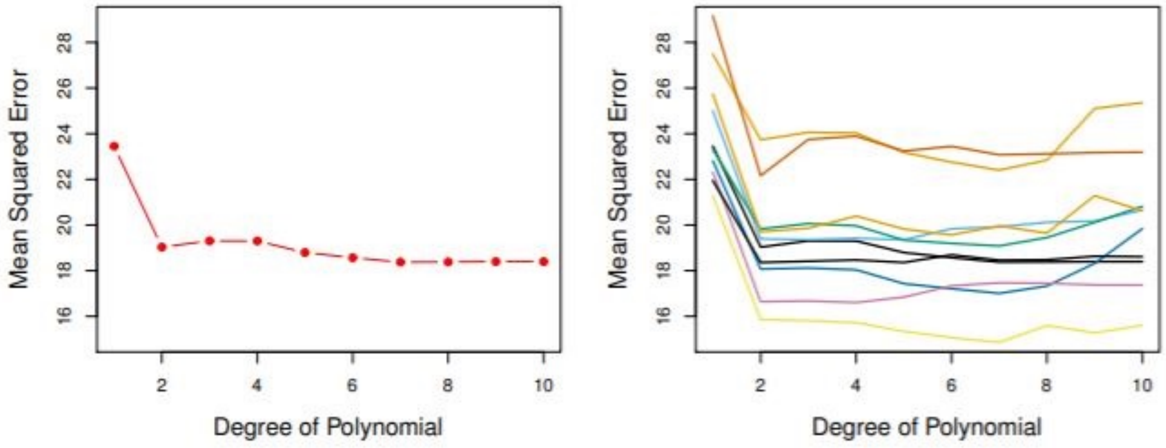
Training/Test Split

Validation Set Approach:

Is it enough?



Validation Set Approach

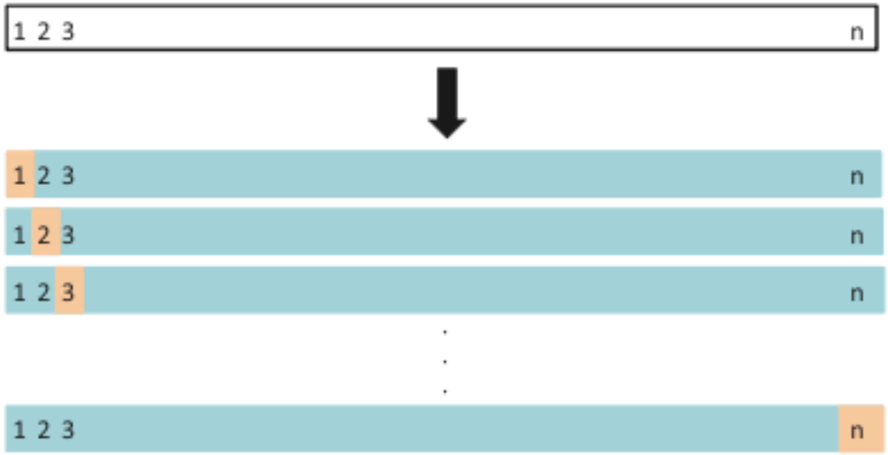


- *Left: Test MSE for a single split*
- *Right: Validation method repeated 10 times, each time split is done randomly*
- *There is a lot of variability among the MSE's and this is Not good!*
- *We need more stable methods!*

Validation Set Approach

- *Advantages:*
    - *Simple and computationally less expensive.*
- *Disadvantages:*
    - *The validation MSE can be highly variable.*
    - *Only a subset of observations is used to fit the model (training data).*
    - *Statistical methods tend to perform worse when trained* **on fewer observations***.*
    - *i.e., the test MSE tends to be over-estimated.*

Leave-one-out cross validation (LOOCV)



- Split the data set of size n into
  - training: n-1
  - validation: 1
- Fit the model using the training data
- Validate the model using the validation data: $MSE_{(i)}$(do this n times)
- Final test MSE = $\frac{1}{n}\sum_{i=1}^{n} MSE_{(i)}$

Leave-one-out cross validation (LOOCV)

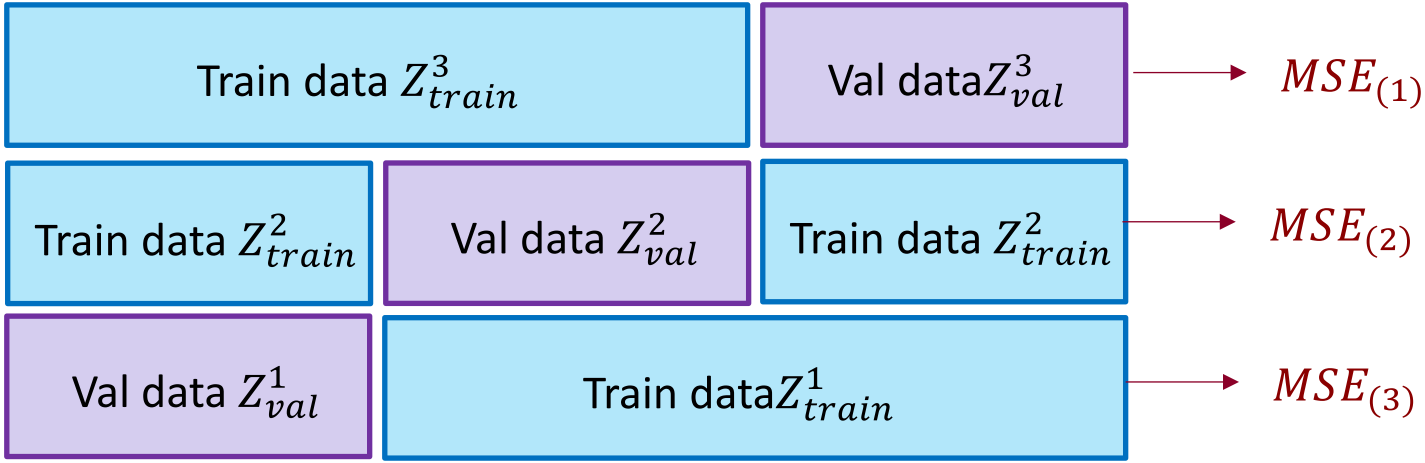
- *LOOCV has less bias*
  - *We repeatedly fit the statistical learning method using training data that contains $n - 1$ obs., i.e. almost all the data set is used.*

- *LOOCV is stable (no more randomness)*

- **Issue***: LOOCV is computationally expensive*
  - *We need to fit the model n times!*

k-fold CV

- Example: K=3 fold

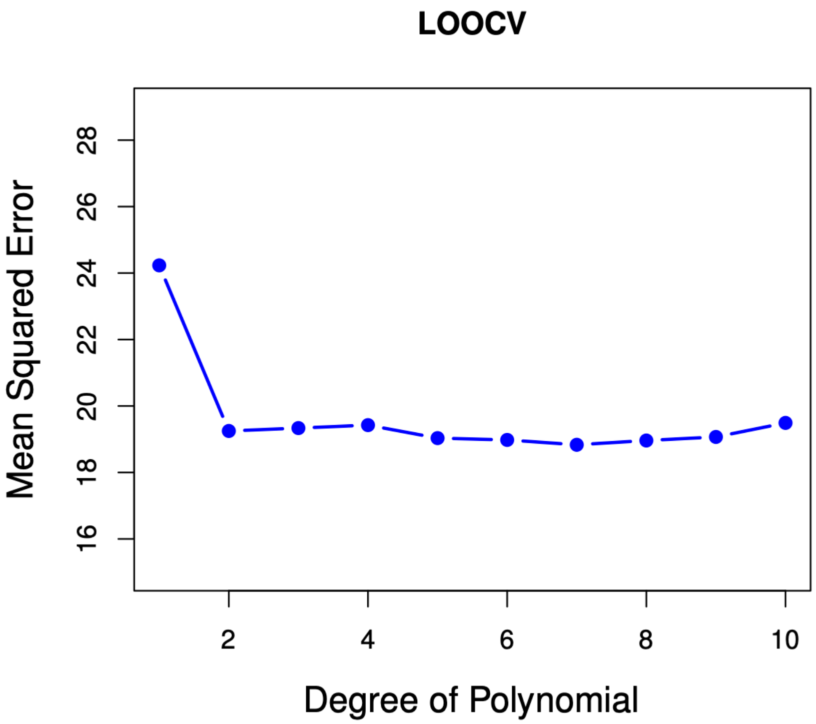


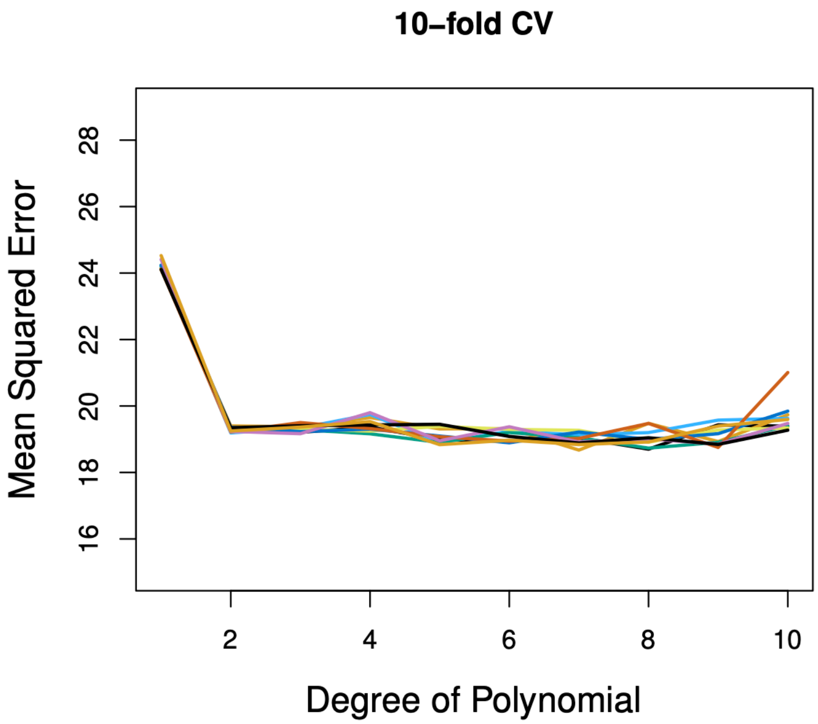
$$CV = \sum_{k=1}^{K} MSE_{(k)}$$

Choose the tuning parameter that minimizes the CV

k-fold CV

- *LOOCV v.s. 10-fold CV*





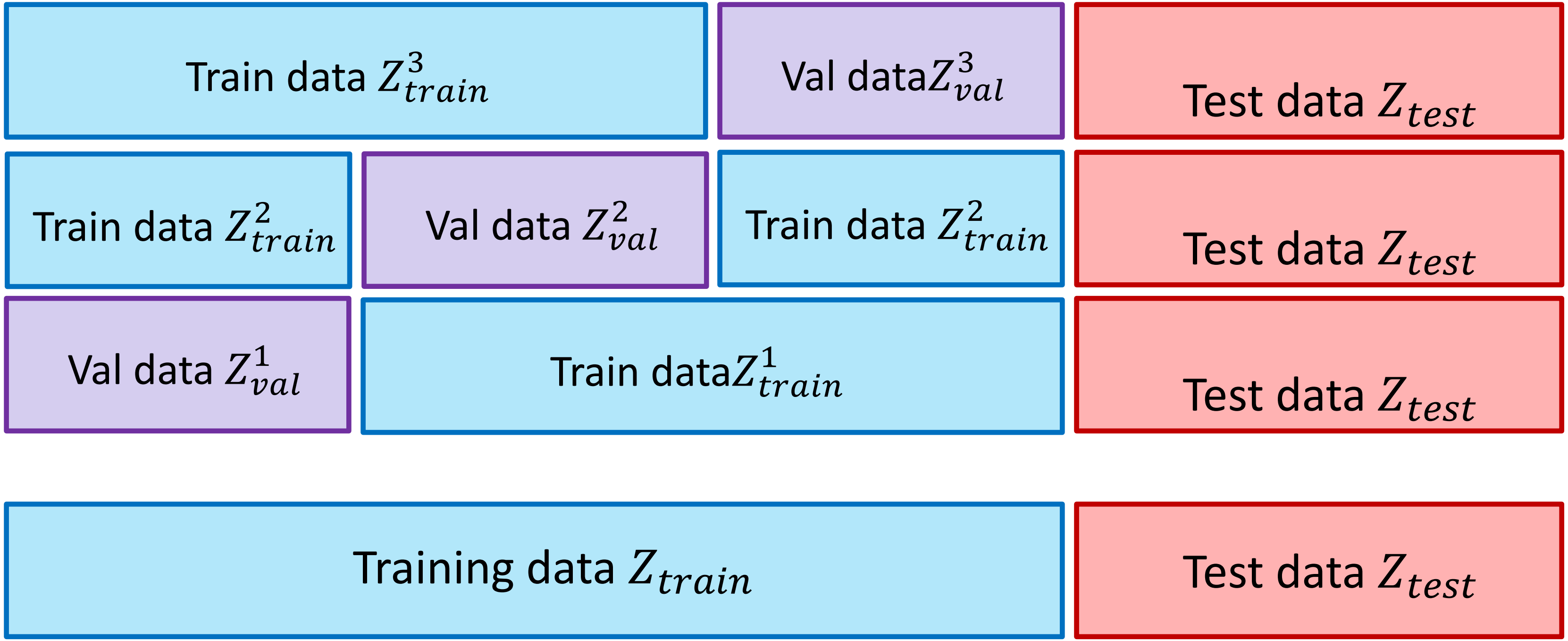
k-fold CV

- *LOOCV = n-fold CV*

- *k-fold CV*
  - *(good & bad) slightly biased / variable*
  - *(good) computationally efficient*

k-fold CV: in actual scenario

- ✓ Goal: Find a hyperparmeter (model complexity, some tuning parameter)
- ✓ Split train & test. Apply k-fold CV to train!



For reporting test loss

k-Fold Cross-Validation

Goal: Find λ

- **Alternative:** *k-fold cross-validation (e.g., k = 3)*
  - *Split Z into Z_{train} and Z_{test}*
  - *Split Z_{train} into k disjoint sets Z_{val}^{s} , and let $Z_{train}^{S} = \bigcup_{s' \neq s} Z_{val}^{S}$*
  - *Use λ' that works best on average across $s \in \{1, \dots, k\}$ with Z_{train}*
  - *Chooses better λ' than above strategy*
- **Compute vs. accuracy tradeoff**
  - *As $k \to N$, the model becomes more accurate*
  - *But algorithm becomes more computationally expensive*