<u>Question</u> How to guarantee statistically that minimizing $J^{(train)}$ minimizes $J^{(test)}$?

<u>IID Assumptions</u>: Both training set & test set are gotten by independently randomly sampling the same probability distribution p_{data}.

Usually T is accomplished by finding some parameters w^* which minimize cost function

$$J(w; E^{(train)})$$

($E^{(train)}$ means training data)

(measures the <u>average</u> error in $E^{(train)}$)

and measuring performance using $J(w^*; E^{(test)})$

<u>The process goes</u>:

① Randomly sample p_{data} to get $E^{(train)}$

② Find $w^* = \arg\min_w J(w; E^{(train)})$

(3) Sample p_{data} to get $E^{(test)}$

(4) Compute $J(w^*; E^{(test)})$

So $E_{p_{data}}\left[J(w^*; E^{(test)})\right]$

$$\geq E_{p_{data}}\left[\min_w J(w; E^{(test)})\right]$$

$$= E_{p_{data}}\left[\min_w J(w; E^{(train)})\right] \quad \text{by IID}$$

$$= E_{p_{data}}\left[J(w^*; E^{(train)})\right]$$

So value of test error is expected to be larger than the training error.

So performance of model depends on

(1) Minimizing training error

(2) Minimizing the gap between training error and test error

If you struggle with ① this is a problem of "underfitting"

② $\Rightarrow$ "overfitting"

(Informal) The capacity of a learning algo is the "size" of the $\underbrace{\text{space of functions}}_{\text{hypothesis space}}$ the algo can fit.

Note: To make this formal, we need statistical learning theory (e.g. VC dimension)

E.g. Linear regression with 1 variable

Case 1 no polynomial terms

predict with $\boxed{f(x)} = wx + b$ hypothesis function

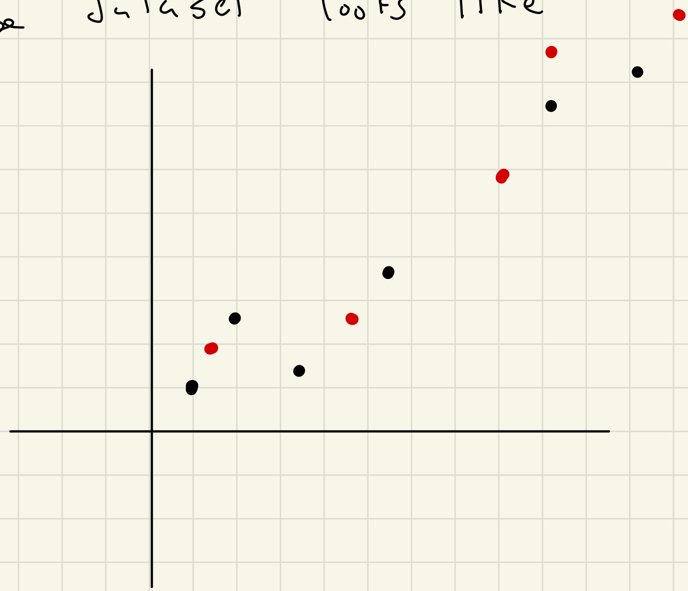
(capacity) the space of all such functions has dim 2

Case 2 poly terms up to degree 4

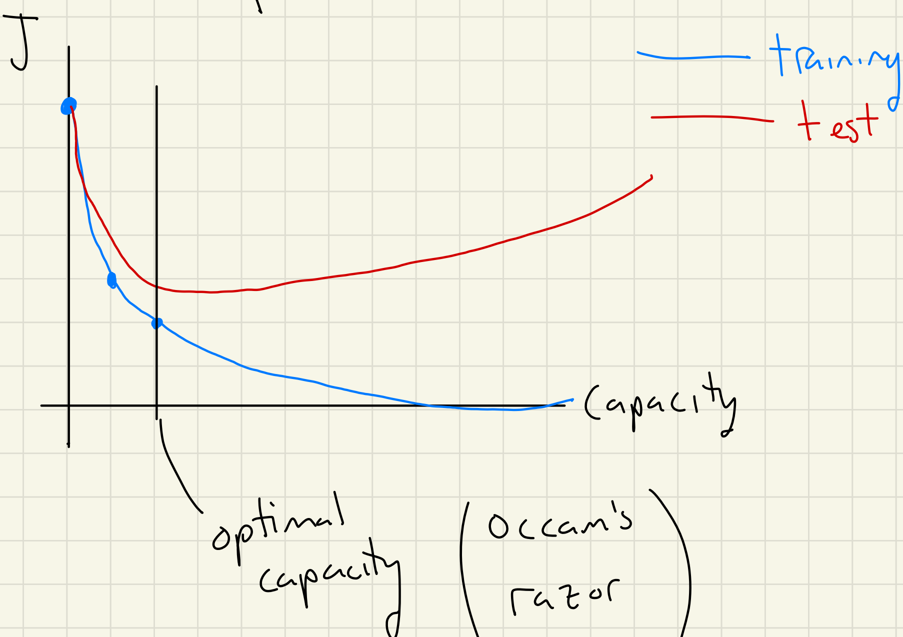
predict with $f(x) = w_4 x^4 + w_3 x^3 + w_2 x^2 + w_1 x + b$

Now capacity is 5

Suppose dataset looks like



Lets graph $\begin{pmatrix} \text{training error} \\ \text{test error} \end{pmatrix}$ versus capacity



(Occam's razor)

Bayes Error (Suppose we have p_{data})

Use p_{data} to make predictions

e.g. $f^B(\vec{x}) = \underset{y}{\operatorname{argmax}}\, p_{data}(y \mid \vec{x})$ (supervised algo)

Such a model provides a theoretical minimal cost for any prediction model. Define

$$\text{Bayes Error} = \min_f \mathbb{E}_{p_{data}}\left[J(f; \vec{x}, y)\right]$$

where $J(f; \vec{x}, y)$ is the cost of prediction $f(\vec{x}) = y$

<u>Note:</u> Bayes Error can be nonzero

- mapping $\vec{x}$ to y can be nondeterministic
- $\vec{X}$ can be mapped to y in a deterministic way but using more variables than we have in X.

<u>Regularization:</u>

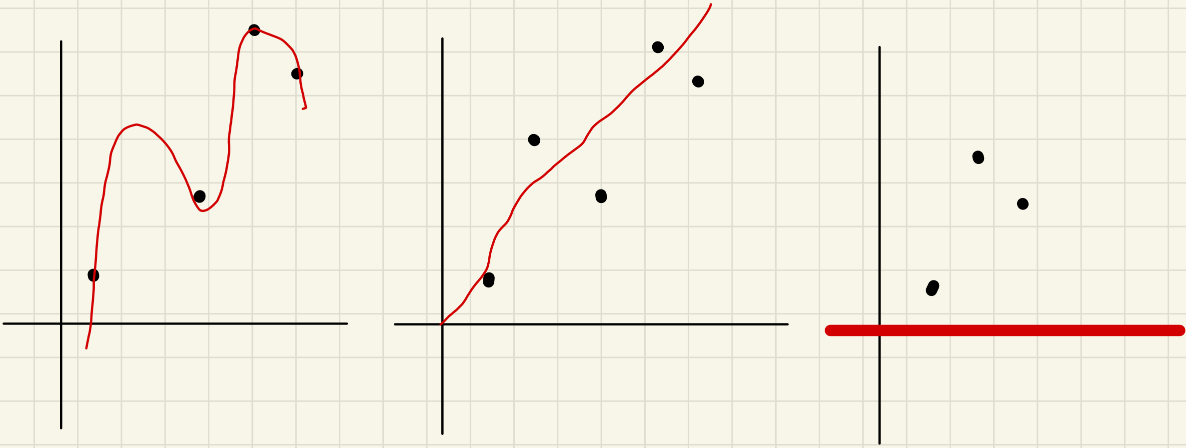
<u>Idea:</u> Add a new term to cost function with parameter λ J to further restricting the hypothesis space

<u>Example</u> (Linear regression)

Modify $J(\vec{w}) = \frac{1}{m} \| \hat{X} w - y \|_2^2 + \lambda \| w \|_2^2$

Since we minimize J to find w^*, this extra term promotes coefficients of w^* to be small. λ dictates how strongly this is promoted.

<u>E.g</u> Using poly terms up to degree 10



$\lambda = 0$ (overfit) — Somewhere in between (good fit) — λ Large (underfit)

Using regularization, can start with high capacity model and "tune" λ to avoid overfitting / underfitting.

Hyperparameters and Validation

<u>Hyperparameter</u>: Parameters which affect / control a learning algo's behavior but are not learned by the algorithm.

<u>E.g.</u> Polynomial regression w/ regularization

n, λ are hyperparameters

n: max degree of poly terms

Hyperparameters are usually tuned by hand to minimize test error.

Issue: When we do this, the test set is no longer a test set (model parameters are fit to the test set)

Solution Introduce a new "test set" called validation set

The process

Split data E into $\underset{80\%}{E^{train}}, \underset{10\%}{E^{val}}, \underset{10\%}{E^{test}}$

① Given hyperparameters $\vec{\lambda}$ train model with E^{train} by finding $\vec{w}^*_{\vec{\lambda}} = \underset{\vec{w}^*}{\operatorname{argmin}} J(\vec{w}, \vec{\lambda}; E^{train})$

② Evalute performance by looking at

$$J(\vec{w}^*_{\vec{\lambda}}; E^{val}) \longleftarrow \text{(without regularization term)}$$

(3) Repeat (1) & (2) with different choices of $\vec{\lambda}$ until find lowest value $J(\vec{w}^*_{\vec{\lambda}^*}; E^{val})$

$\longrightarrow \vec{\lambda}^*$ best $\vec{\lambda}$ (tuning $\vec{\lambda}$)

(4) Approximate generalization error by computing

$J(\vec{w}^*_{\vec{\lambda}^*}; E^{test})$ (without regularization term)

Tip: If E is too small, may want to try k-fold cross validation (allows for putting more in test set)

(read about this)

In python cross_val_score (sklearn)

Bias & Variance:

Def: A point estimator for a parameter Θ associated to some distribution p is any function of random variables $X_1, \ldots, X_n$ (distributed by p)

write $\hat{\theta} = g(x_1, \dots, x_n)$

$\uparrow$

estimation for θ

The bias of $\hat{\theta}$ is

$$\mathrm{bias}(\hat{\theta}) = E_p[\hat{\theta}] - \theta$$

The variance of $\hat{\theta}$ is

$$\mathrm{Var}(\hat{\theta}) = E_p\left[\left(\hat{\theta} - E_p[\hat{\theta}]\right)^2\right]$$

E.g. $X_1, \dots, X_n$ IID random variables distributed by Bernoulli distribution $\Omega = \{0, 1\}$

$$p(x=1) = \theta$$

$$p(X=x) = \theta^x (1-\theta)^{(1-x)}$$

Estimate θ by $\hat{\theta} = \frac{\#\{i \mid x_i = 1\}}{n} = \frac{\sum_{i=1}^n x_i}{n}$

$$\hat{\theta} = \frac{\sum_{i=1}^n X_i}{n}$$

$$\mathrm{bias}(\hat{\theta}) = E_p[\hat{\theta}] - \theta$$

$$= E_p\left[\frac{\sum_{i=1}^n x_i}{n}\right] - \theta$$

$$= \frac{1}{n}\sum_{i=1}^n E_p[x_i] - \theta$$

$$= \frac{1}{n}\sum_{i=1}^n \theta - \theta$$

$$= \frac{n\theta}{n} - \theta = 0$$

__Def__ If $\mathrm{bias}(\hat{\theta}) = 0$ call $\hat{\theta}$ an __unbiased estimator__.

$$\mathrm{Var}(\hat{\theta}) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^n x_i\right)$$

$$= \frac{1}{n^2}\sum_{i=1}^n V(X_i)$$

$$= \frac{1}{n^2}\sum_{i=1}^n \theta(1-\theta)$$

$$= \frac{\theta(1-\theta)}{n} \longrightarrow 0 \quad \text{as} \quad n \to \infty$$

Exercises ① Gaussian distribution $\mathcal{N}(x; \mu, \sigma^2)$

estimate μ by IID random variables $X_1, \ldots, X_n$

$$\hat{\mu} = \frac{X_1 + \cdots + X_n}{n}$$

Show $\mathrm{Bias}(\hat{\mu}) = 0$

② Estimate σ^2 by $\widehat{\sigma^2} = \frac{1}{n} \sum_{i=1}^{n} (X_i - \hat{\mu})^2$

Show $\mathrm{Bias}(\widehat{\sigma^2}) = -\frac{\sigma^2}{n}$

$\mathrm{Var}(\widehat{\sigma^2}) = ?$

③ Estimate σ^2 by $\widehat{\sigma^2} = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \hat{\mu})^2$

Show $\mathrm{bias}(\widehat{\sigma^2}) = 0$ $\quad \mathrm{Var}(\widehat{\sigma^2}) = ?$

(called the unbised variance estimator)

<u>Def</u>: The <u>mean squard error</u> (MSE) of $\hat{\theta}$

is $\mathrm{MSE}(\hat{\theta}) = E_p[(\hat{\theta} - \theta)^2]$

Lemma $\mathrm{MSE}(\hat{\theta}) = \mathrm{Var}(\hat{\theta}) + \mathrm{Bias}(\hat{\theta})^2$

Proof: $\mathrm{MSE}(\hat{\theta}) = E_p\left[(\hat{\theta} - \theta)^2\right]$

$$= E_p\left[(\hat{\theta} - E_p[\hat{\theta}] + E_p[\hat{\theta}] - \theta)^2\right]$$

$$= E_p\Big[(\hat{\theta} - E_p[\hat{\theta}])^2 + (E[\hat{\theta}] - \theta)^2 + 2(\hat{\theta} - E[\hat{\theta}])(E[\hat{\theta}] - \theta)\Big]$$

$$= E\left[(\hat{\theta} - E[\hat{\theta}])^2\right] + E\left[(E[\hat{\theta}] - \theta)^2\right] + 2E\Big[(\hat{\theta} - E[\hat{\theta}])(E[\hat{\theta}] - \theta)\Big]$$

Note $E[\hat{\theta}] - \theta$ is constant

$$= \mathrm{Var}(\hat{\theta}) + (E[\hat{\theta}] - \theta)^2 + 2(E[\hat{\theta}] - \theta)\,\underbrace{E\left[(\hat{\theta} - E[\hat{\theta}]\right]}_{E[\hat{\theta}] - E[\hat{\theta}] = 0}$$

$$= \mathrm{Var}(\hat{\theta}) + \mathrm{bias}(\hat{\theta})^2$$

$\blacksquare$

Maximum Likelihood Estimators

Back to usual setup: p_{data} generating

$X = X^{train}, X^{test}$ [X has rows $\vec{x}^{(1)}, \ldots, \vec{x}^{(m)}$ treated as random variables distributed by p_{data}]

Goal: Approximate $p_{data}(\vec{x})$. Let

$p_{model}(\vec{x}, \vec{\theta})$ be an approximation for

p_{data} depending on parameter $\vec{\theta}$

E.g. If we expect p_{data} to be gaussian

$$p_{model}(\vec{x}, \theta) = \mathcal{N}(\vec{x}; \vec{\mu}_0, \Sigma_0)$$

Def The maximum likelihood estimator for $\vec{\theta}$ is

$$\vec{\theta}_{ML} = \underset{\vec{\theta}}{\operatorname{argmax}}\, p_{\text{model}}\left(\vec{x}^{(1)}, \ldots, \vec{x}^{(m)}; \vec{\theta}\right)$$