CSE 559A: Computer Vision

Fall 2018: T-R: 11:30-1pm @ Lopata 101

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Zhihao Xia, Charlie Wu, Han Liu

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Oct 25, 2018

GENERAL

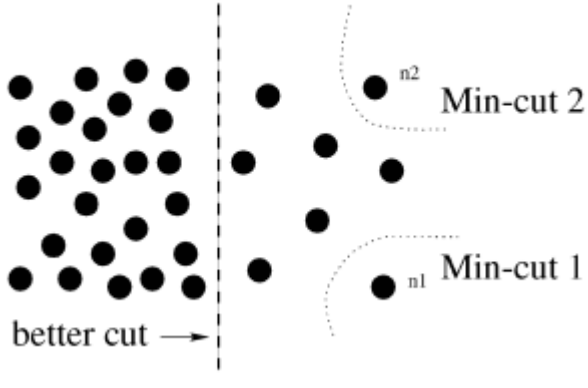
- Problem Set 3 Due tonight
- Problem Set 4 will be posted shortly
- Reminder: Tomorrow's Office Hours will be in Lopata 103

GROUPING AND SEGMENTATION

Graph Based Approaches

- Define a set of vertices V, edges E with weights $w(v_1, v_2)$
- Cut = partition of vertices into two sets of nodes A and B
  - $A \cup B = V$
  - $A \cap B = \Phi$
- Cut$(A, B) = \sum_{u \in A, v \in B} w(u, v)$

Can lead to isolated points



GROUPING AND SEGMENTATION

Graph Based Approaches

- Define a set of vertices V, edges E with weights $w(v_1, v_2)$
- Cut = partition of vertices into two sets of nodes A and B
  - $A \cup B = V$
  - $A \cap B = \Phi$
- $\text{Cut}(A, B) = \sum_{u \in A, v \in B} w(u, v)$
- Use normalized measure
  - $\text{Assoc}(A, V) = \sum_{u \in A, v \in V} w(u, v)$
  - $\text{Assoc}(B, V) = \sum_{u \in B, v \in V} w(u, v)$

$$\text{NCut}(A, B) = \frac{\text{Cut}(A, B)}{\text{Assoc}(A, V)} + \frac{\text{Cut}(A, B)}{\text{Assoc}(B, V)}$$

$$= 2 - \left(\frac{\text{Assoc}(A, A)}{\text{Assoc}(A, V)} + \frac{\text{Assoc}(B, B)}{\text{Assoc}(B, V)} \right) = 2 - \text{NAssoc}(A, B)$$

GROUPING AND SEGMENTATION

Graph Based Approaches

$$\mathrm{NCut}(A,B) = \frac{\mathrm{Cut}(A,B)}{\mathrm{Assoc}(A,V)} + \frac{\mathrm{Cut}(A,B)}{\mathrm{Assoc}(B,V)}$$

$$= 2 - \left(\frac{\mathrm{Assoc}(A,A)}{\mathrm{Assoc}(A,V)} + \frac{\mathrm{Assoc}(B,B)}{\mathrm{Assoc}(B,V)} \right) = 2 - \mathrm{NAssoc}(A,B)$$

- Instead of minimizing cut, minimize normalized cut, or maximize normalized association.
- Note that for this case, you might not even need a "unary" cost.
  - The solution of assigning all nodes to A or B is sub-optimal, because now the denominator goes to 0.
  - So this can be viewed as a form of clustering
- NP-hard even for the binary case. But there are approximate solvers based on "real-valued" relaxations.

GROUPING AND SEGMENTATION

Normalized Cuts: Sketch

Say we say, $x_i = 1$ if node i is in A, and -1 if it's in B.

Then we can show that the NCut expression simplifies as:

$$\text{NCut}(A, B) = \frac{\sum_{x_i>0, x_j<0} -w_{ij}x_i x_j}{\sum_{x_i>0} d_i} + \frac{\sum_{x_i<0, x_j>0} -w_{ij}x_i x_j}{\sum_{x_i<0} d_i}$$

where $d_i = \sum_j w_{ij}$.

Then, you can construct two $N \times N$ matrices D and W such that D is diagonal with entries d_i on its diagonal, and W is symmetric with $W_{ij} = w_{ij}$. Stack x_is into an N-dimensional vector x.

And you can show that

$$\text{NCut}(x) = \frac{y^T(D - W)y}{y^T Dy}$$

where $y = (1 + x) - b(1 - x), b = (\sum_{x_i>0} d_i)/(\sum_{x_i<0} d_i)$and you have $y^T D1 = 0$.

You can turn this into a minimization problem, but $x \in \{-1, 1\}$. This is still combinatorial. But if you let y take on real values, then this turns into an eigen-value problem.

GROUPING AND SEGMENTATION

$$\arg\min \mathrm{NCut}(x) = \frac{y^T(D-W)y}{y^T D y}$$

- Find lowest eigenvectors z of $D^{-0.5}(D-W)D^{-0.5}$, and get $y = D^{-0.5}z$.
- Lowest eigenvalue will be zero. Ignore that.
- Eigenvector corresponding to second lowest eigenvalue will give you a partition
  - Compute $z \rightarrow y \rightarrow x$, threshold by sign to figure out label
- Huge Matrices, but Sparse. Eigen-decomposition can be done efficiently by Lanczos method.
- Can apply this recursively, or look at next eigen-vectors to further sub-partition image.
- For details
  - Shi and Malik, "Normalized Cuts and Image Segmentation," PAMI 2000.

MACHINE LEARNING

So far, given an input X and desired output Y we have

- Tried to explain the relationship of how X results from Y
  - X = observed image(s) / Y = clean image, sharp image, surface normal, depth
  - Noise, photometry, geometry, ...
- Often put a hand-crafted "regularization" cost to compute the inverse
  - Depth maps are smooth
  - Image gradients are small
- But sometimes, there is no way to write-down a relationship between X and Y ?
  - X = Image, Y = Does the image contain a dog ?
- Even if there is, the hand-crafted regularization cost is often arbitrary.
  - Real images contain far more complex and subtle regularity.

MACHINE LEARNING

Instead, we are going to assume that there is some underlying joint probability distribution $P_{XY}(x, y)$

- And our goal is to compute:
  - The best estimate of y conditioned on a specific value of x,
  - To minimize some notion of "risk" or "loss"

Define a loss function $L(y, \hat{y})$, which measures how much we dislike $\hat{y}$ as our estimate,
when y is the right answer.

Examples

- $L(y, \hat{y}) = \|y - \hat{y}\|^2$
- $L(y, \hat{y}) = \|y - \hat{y}\|$
- $L(y, \hat{y}) = 0$ if $y = \hat{y}$, and some C otherwise.

MACHINE LEARNING

Instead, we are going to assume that there is some underlying joint probability distribution $P_{XY}(x, y)$

- And our goal is to compute:
  - The best estimate of y conditioned on a specific value of x,
  - To minimize some notion of "risk" or "loss"

Ideally,

$$\hat{y}(x) = \arg\min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \; dy$$

$$P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y')dy'}$$

MACHINE LEARNING

$$\hat{y}(x) = \arg\min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \; dy$$

$$P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y')dy'}$$

- So we have a loss (depends on the application)
- We can compute $P(y|x)$ from P_{XY}.
- But we don't know P_{XY} !

Assume we are given as training examples, samples $(x, y) \sim P_{XY}$ from the true joint distribution.

MACHINE LEARNING

$$\hat{y}(x) = \arg\min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \; dy$$

$$P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y')dy'}$$

Given $\{(x_i, y_i)\}$ as samples from P_{XY}, we could:

- Estimate P_{XY}
  - Choose parametric form for the joint distribution (Gaussian, Mixture of Gaussians, Bernoulli, ...)
  - Estimate the parameters of that parametric form to "best fit" the data.
  - Depending again on some notion of fit (often likelihood)

$$P_{XY}(x, y) = f(x, y, ; \theta)$$

$$\theta = \arg\max_{\theta} \sum_{i} \log f(x_i, y_i; \theta)$$

Maximum Likelihood Estimation

MACHINE LEARNING

$$\hat{y}(x) = \arg\min_{\hat{y}} \int L(y, \hat{y}) P(y|x) \; dy$$

$$P(y|x) = \frac{P_{XY}(x, y)}{\int P_{XY}(x, y')dy'}$$

$$P_{XY}(x, y) = f(x, y, ; \theta)$$

$$\theta = \arg\max_{\theta} \sum_{i} \log f(x_i, y_i; \theta)$$

So that's one way of doing things ...

- You're doing a minimization for learning P_{XY}, but then also a minimization at "test time" for every input x.
- You're approximating P_{XY} with some choice of the parametric form f.
- And it's possible that the best θ that maximizes likelihood, may not be the best θ that minimizes loss.

MACHINE LEARNING

Given a bunch of samples $\{(x_i, y_i)\}$ from P_{XY},

we want to learn a **function** $y = f(x)$, such that

given a typical x, y from P_{XY}, the **expected** loss $L(y, f(x))$ is low.

$$f = \arg\min_f \int \left(\int L(y, f(x)) p(y|x) dy \right) p(x) dx$$

MACHINE LEARNING

Given a bunch of samples $\{(x_i, y_i)\}$ from P_{XY},

we want to learn a **function** $y = f(x)$, such that

given a typical x, y from P_{XY}, the **expected** loss $L(y, f(x))$ is low.

$$f = \arg\min_f \int \int L(y, f(x)) \; p_{XY}(x, y) dx dy$$

What we're going to is to replace the double integration with a summation over samples !

MACHINE LEARNING

Given a bunch of samples $\{(x_i, y_i)\}$ from P_{XY},

we want to learn a **function** $y = f(x)$, such that

given a typical x, y from P_{XY}, the **expected** loss $L(y, f(x))$ is low.

$$f = \arg\min_f \sum_i L(y_i, f(x_i))$$

What we're going to is to replace the double integration with a summation over samples !

Empirical Risk Minimization

- So instead of first fitting the probability distribution from training data, and then given a new input, minimizing the loss under that distribution ...
- We are going to do a search over possible functions that "do well" on the training data, and assume that a function that minimizes "empirical risk" also minimizes "expected risk".

MACHINE LEARNING

Formally

- Given inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we want to learn a function $y = f(x)$, i.e., $f : \mathcal{X} \to \mathcal{Y}$
- Function should be a "good" predictor of y, as measured in terms of a risk or loss function: $L(y, \hat{y})$.
- Ideally, we want to find the best $f \in \mathcal{H}$, among some class or space of functions $\mathcal{H}$ (called the hypothesis space), which minimizes the expected loss under the joint distribution $P_{XY}(x, y)$:

$$f = \arg\min_{f \in \mathcal{H}} \int_{\mathcal{X}} \int_{\mathcal{Y}} L(y, f(x)) \; P_{XY(x,y)} \; dy \, dx$$

- But we don't know this joint distribution, but we have a **training set** $(x_1, y_1), (x_2, y_2), \ldots (x_T, y_T)$, which (we hope!) are samples from P_{XY}.
- So we approximate the integral over the P_{XY} with an average over the training set (x_t, y_t),

$$f = \arg\min_{f \in \mathcal{H}} \frac{1}{T} \sum_{t} L(y_t, f(x_t))$$

You're given data. Choose a loss function that matches your task, a hypothesis space $\mathcal{H}$, and minimize.

MACHINE LEARNING

$$f = \arg\min_{f \in \mathcal{H}} \frac{1}{T} \sum_t L(y_t, f(x_t))$$

Consider:

- $x \in \mathcal{X} = \mathbb{R}^d$
- $y \in \mathcal{Y} = \mathbb{R}$
- $\mathcal{H}$ is the space of all "linear functions" of $\mathcal{X}$.
  - $f(x; w, b) = w^T x + b, \quad w \in \mathbb{R}^d, b \in \mathbb{R}$
  - Minimization of $f \in \mathcal{H}$ becomes a minimization of w, b
- Consider $L(y, \hat{y}) = (y - \hat{y})^2$

And then we have our familiar least-squares regression!

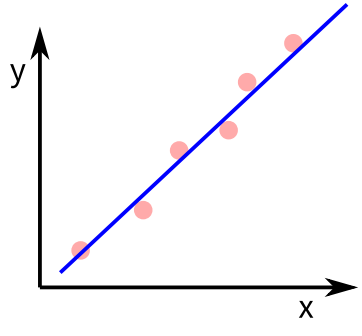
MACHINE LEARNING

$$f = \arg\min_{f \in \mathcal{H}} \frac{1}{T} \sum_t L(y_t, f(x_t))$$

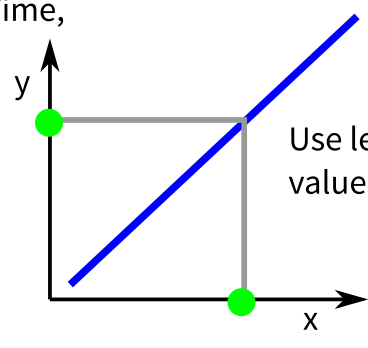
$$w, b = \arg\min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{T} \sum_t (y_t - w^T x_t - b)^2$$

So we know how to solve this: take derivative and set to 0.

Training



At Test Time,



Use learned function to predict values of new inputs

Not just for fitting "lines". Imagine x is a vector corresponding to a patch of intensities in a noisy image. y is corresponding clean intensity of the center pixel. You could use this to "learn" a linear "denoising filter", by fitting to many examples of pairs of noisy and noise-free intensities.

$$w, b = \arg \min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{T} \sum_t (y_t - w^T x_t - b)^2$$

Define $\tilde{x}_t = [x_t^T, 1]^T$

$$w = \arg \min_{w \in \mathbb{R}^{d+1}} \frac{1}{T} \sum_t (y_t - w^T \tilde{x}_t)^2$$

$$w = \left(\sum_t \tilde{x}_t \tilde{x}_t^T \right)^{-1} \left(\sum_t \tilde{x}_t y_t \right)$$

Now, let's say we wanted to fit a polynomial instead of a linear function.

For $x \in \mathbb{R}$,

$f(x; w_0, w_1, w_2, \ldots, w_n) = w_0 + w_1 x + w_2 x^2 + \ldots w_n x^n.$

This is our hypothesis space. Same loss function $L(y, \hat{y}) = (y - \hat{y})^2$.

MACHINE LEARNING

$$w = \arg\min_{w \in \mathbb{R}^{n+1}} \frac{1}{T} \sum_t (y_t - w_0 - w_1 x_t - w_2 x_t^2 \ldots - w_n x_t^n)^2$$

Set $\tilde{x}_t = \left[1, x_t, x_t^2, x_t^3, \ldots x_t^n\right]^T$.

And you get exactly the same equation !

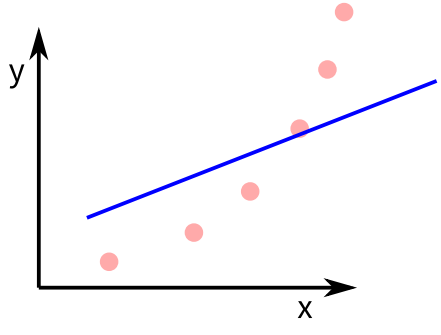
$$w = \left(\sum_t \tilde{x}_t \tilde{x}_t^T \right)^{-1} \left(\sum_t \tilde{x}_t y_t \right)$$

- But now, inverting a larger matrix.
- Can apply the same idea to polynomials of multi-dimensional x.
- Can apply least-squares fitting to any task with an $L2$ loss, and where the hypothesis space is linear in the parameters (not necessarily in the input).

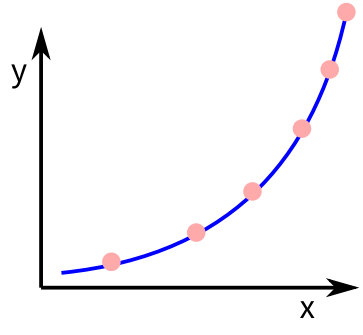
E.g: $f(x; w_0, w_1, w_2, \ldots, w_n) = w_0 + w_1 x + w_2 x^2 + \ldots w_n x^n$.

MACHINE LEARNING

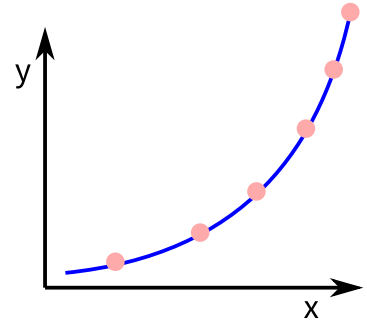
Why not just fit the more complex model ?



Linear Fit



Quadratic Fit



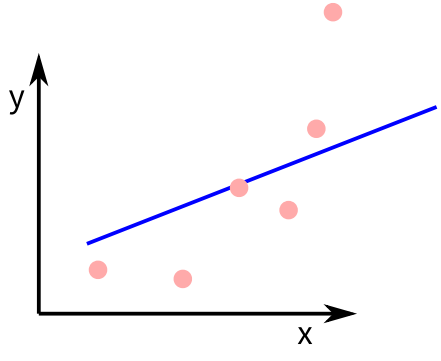
12th Order Polynomial

nth Order Polynomials can fit all orders upto n.

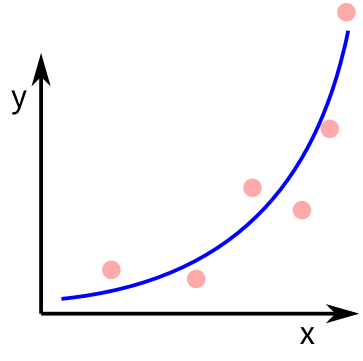
Why not just choose the most complex inclusive hypothesis space ?

MACHINE LEARNING

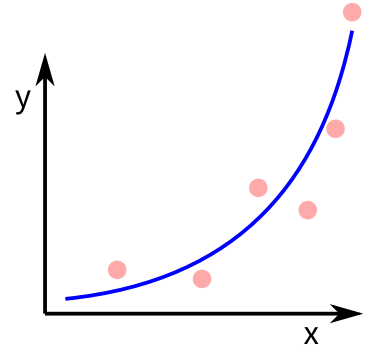
Why not just fit the more complex model ?



Linear Fit



Quadratic Fit



12th Order Polynomial

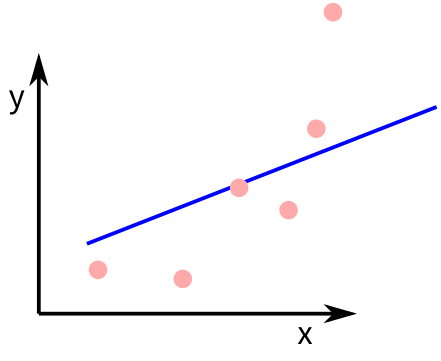
nth Order Polynomials can fit all orders upto n.

Why not just choose the most complex inclusive hypothesis space ?

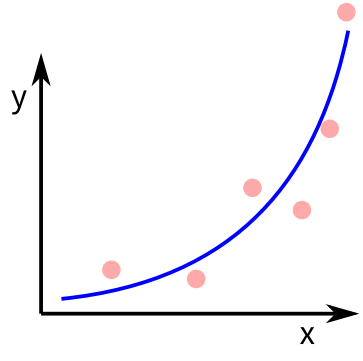
Because, y may have noise (P(y|x) is not deterministic).

MACHINE LEARNING

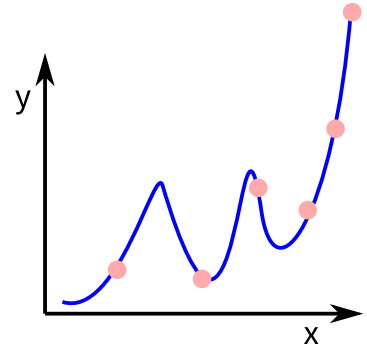
Why not just fit the more complex model ?



Linear Fit



Quadratic Fit



12th Order Polynomial

Given enough capacity, f will hallucinate structure in the noise

nth Order Polynomials can fit all orders upto n.

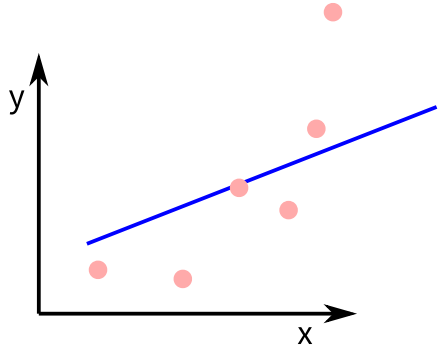
Why not just choose the most complex inclusive hypothesis space ?

Because, y may have noise (P(y|x) is not deterministic).

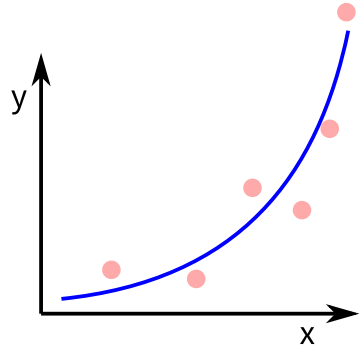
MACHINE LEARNING

Why not just fit the more complex model ?

Too simple

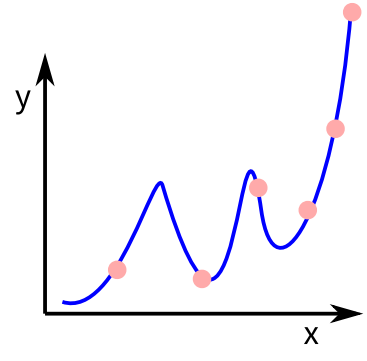


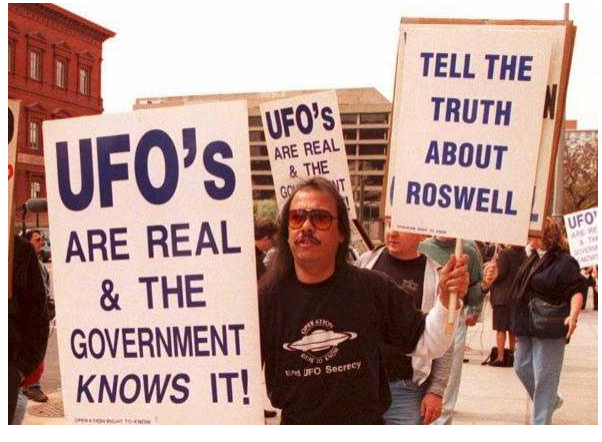
Just Right



Quadratic Fit

Too complex



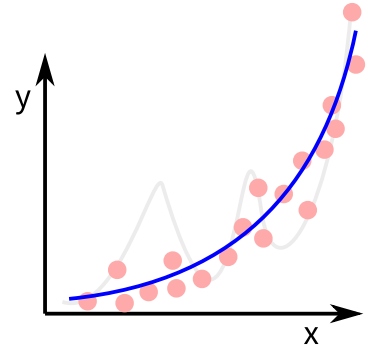


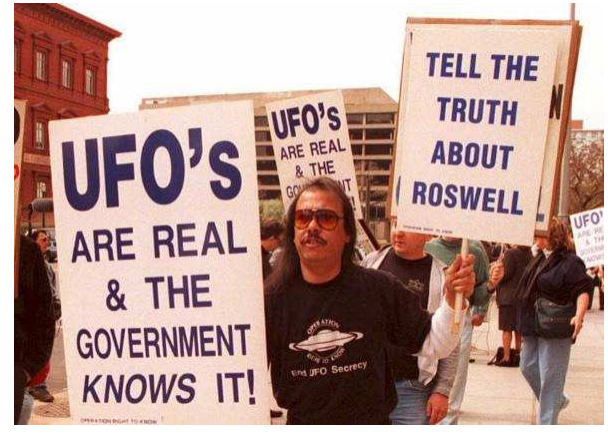
MACHINE LEARNING

Why not just fit the more complex model ?

Too complex

Can be fixed if we had a lot more data.





MACHINE LEARNING

While we train on empirical loss,

$$\frac{1}{T}\sum_{t} L(y_t, f(x_t))$$

we care about the actual expected loss:

$$\int_{\mathcal{X}}\int_{\mathcal{Y}} L(y, f(x))\ \ P_{XY(x,y)}\ \ dy\ dx$$

Why ? Because we don't want to explain the training set. We want to do well on new inputs. We want to "generalize" from the training set.

This is why a more complex function that exactly fits the training set is bad, when it "generalizes" poorly.

MACHINE LEARNING

Error = Bayes Error + Approximation Error + Estimation Error

- Bayes Error: This is the error due to the uncertainty in $p(y|x)$. This is the error you would have even if you knew the exact distribution and could craft a function f with infinite complexity.

$$\text{Bayes Error} = \int \left(\int \min_{\hat{y}} L(y, \hat{y}) P_{XY}(x, y) dy \right) dx$$

- Approximation Error: This is the error due to the limited capacity of our hypothesis space $\mathcal{H}$. It is the error of the true best function $f \in \mathcal{H}$, minus the Bayes error, assuming we had perfect knowledge of P_{XY}.
  Also called the "Bias"^{2}.
- Estimation Error: This is the remaining component of error, caused by the fact that we don't know the true P_{XY}, but only have a limited number of samples.
  This depends on the size of our training set (and also, how well we are able to do the minimization). Called "variance".

MACHINE LEARNING

Error = Bayes Error + Approximation Error + Estimation Error

Bias-Variance Tradeoff

Choosing a simple function class: higher approximation error, lower estimation error.

Choosing a complex function class: lower approximation error, higher estimation error.

How do I decrease Bayes Error ? By getting better inputs!

MACHINE LEARNING

Overfitting

- Definitions of complexity of a function or hypothesis space $\mathcal{H}$: VC-dimension, Rademacher complexity
- Try to capture that one function or function space provides a "simpler" explanation than another
- Useful as intuition. But often don't "work" for very complex functions and tasks.
- But the idea is:
  - Given two functions with the same error, you want to choose one that's simpler.
  - You never want to consider a class of functions that can fit 'random' T pairs of (x, y), where T is the size of your training set.
- Choose hypothesis space based on size of training set.
- Add a "regularizer" (for example, a squared penalty on higher order polynomial coefficients).

Public Service Announcement: Any regularizer is biased by what you think is "simple". There is no universal definition of simple.

Overfitting: Good ML "Hygiene"

Remember, you can overfit not just the parameters, but your design choices !

For any given task:

- Have train, dev, val, and test set.
- Train your estimators on the train set.
- Choose hyperparameters based on dev set.
  - Function class
  - Regularization weight
  - Number of iterations to train, etc.
- Keep periodically checking to see if it generalizes to val set.
- Look at the test set only at the "end" of the project.