Overfitting

Machine Learning – CSE546
Carlos Guestrin
University of Washington

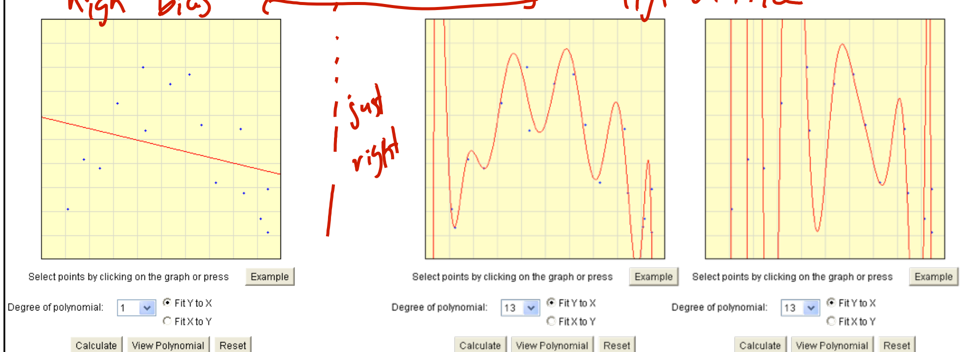
October 2, 2013

Bias-Variance Tradeoff

- Choice of hypothesis class introduces learning bias
  - More complex class → less bias
  - More complex class → more variance

high bias ← → high variance

just right



Training set error

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

- Given a dataset (Training data)
- Choose a loss function
  - e.g., squared error (L_2) for regression
- **Training set error:** For a particular set of parameters, loss function on training data:

squared

min w

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

hypothesis

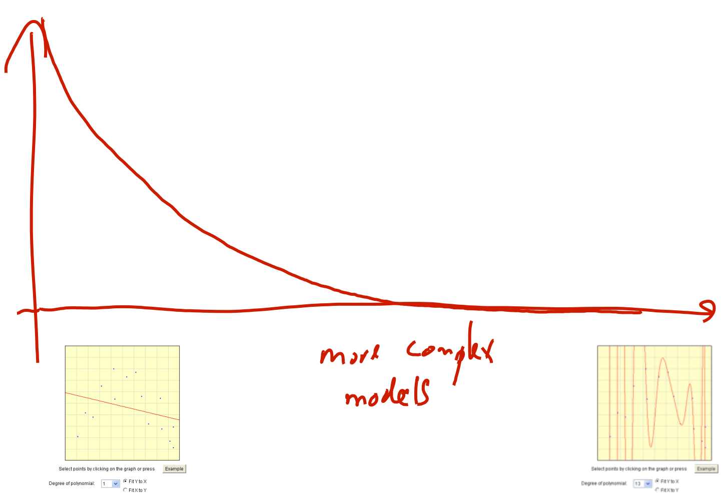
sum train data

target

prediction

Training set error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$



Prediction error

- Training set error can be poor measure of "quality" of solution
- **Prediction error:** We really care about error over all possible input points, not just training data:

want min

$$error_{true}(\mathbf{w}) = E_{\mathbf{x}}\left[\left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2\right]$$

all possible examples

$$= \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}$$

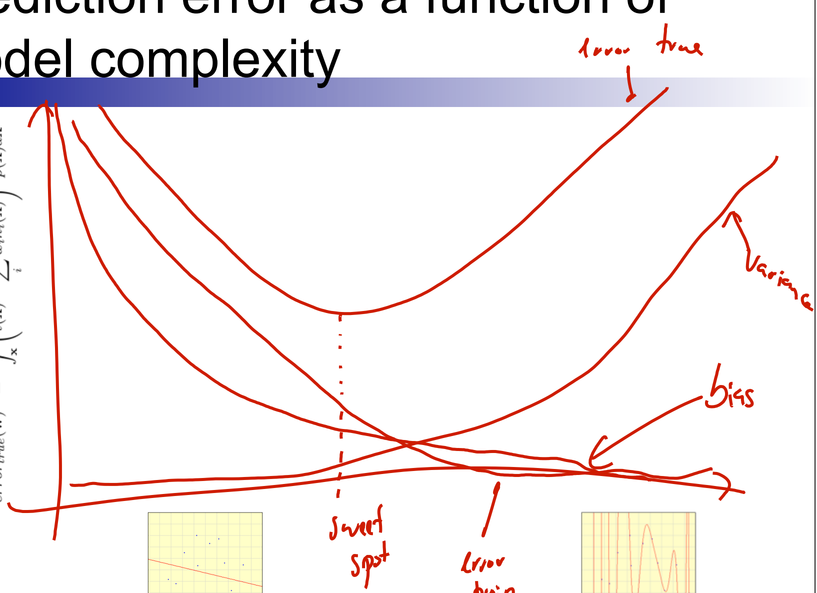
integral/average

weighted by likelihood of example

Prediction error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}}\sum_{j=1}^{N_{train}}\left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}$$



Computing prediction error

- Computing prediction
  - Hard integral
  - May not know t(**x**) for every **x**

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

Sample approx of true

$x_j \sim P(x)$

- Monte Carlo integration (sampling approximation)
  - Sample a set of i.i.d. points $\{\mathbf{x}_1,\ldots,\mathbf{x}_M\}$ from p(**x**)
  - Approximate integral with sample average

$$error_{true}(\mathbf{w}) \approx \frac{1}{M} \sum_{j=1}^{M} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

Why training set error doesn't approximate prediction error?

- Sampling approximation of prediction error:

$$error_{true}(\mathbf{w}) \approx \frac{1}{M} \sum_{j=1}^{M} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

- Training error :

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

- Very similar equations!!!
  - Why is training set a bad measure of prediction error???

Why training set error doesn't approximate prediction error?

Because you cheated!!!

Training error good estimate for a single **w,**
But you optimized **w** with respect to the training error, and found **w** that is good for this set of samples

Training error is a (optimistically) biased estimate of prediction error

- Very similar equations!!!
  - Why is training set a bad measure of prediction error???

Test set error

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

- Given a dataset, **randomly** split it into two parts:
  - Training data – $\{\mathbf{x}_1, \ldots, \mathbf{x}_{Ntrain}\}$
  - Test data – $\{\mathbf{x}_1, \ldots, \mathbf{x}_{Ntest}\}$
- Use training data to optimize parameters **w**
- **Test set error:** For the ***final output*** $\hat{\mathbf{w}}$, evaluate the error using:

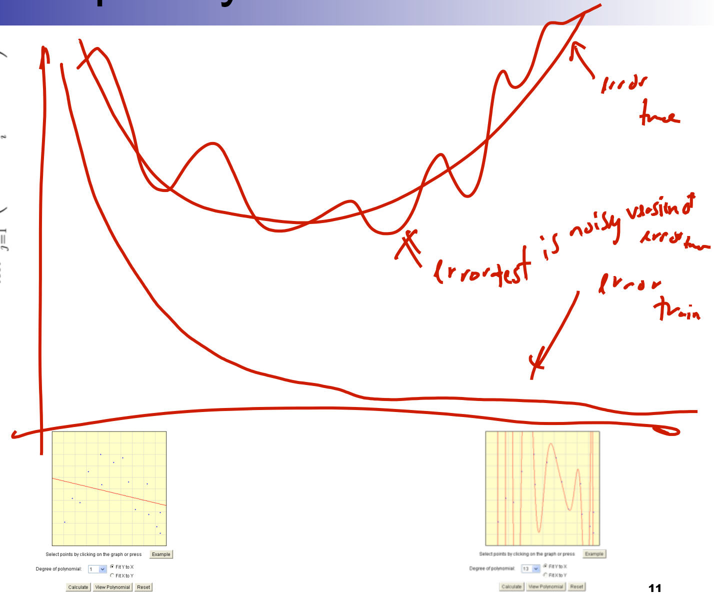
$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

Test set error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

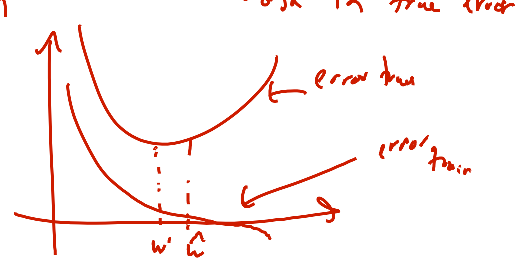
$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$



Overfitting

- **Overfitting:** a learning algorithm overfits the training data if it outputs a solution $\hat{\mathbf{w}}$ when there exists another solution **w'** such that:

$$\underbrace{[error_{train}(\hat{\mathbf{w}}) < error_{train}(\mathbf{w}')]}_{\text{better on train data}} \wedge \underbrace{[error_{true}(\mathbf{w}') < error_{true}(\hat{\mathbf{w}})]}_{\text{worse in true error}}$$



How many points to I use for training/testing?

- Very hard question to answer!
  - Too few training points, learned $\hat{\mathbf{w}}$ is bad
  - Too few test points, you never know if you reached a good solution
- Bounds, such as Hoeffding's inequality can help:

$$P(\mid \widehat{\theta} - \theta^* \mid \geq \epsilon) \quad \leq \quad 2e^{-2N\epsilon^2}$$

- More on this later this quarter, but still hard to answer
- Typically:
  - If you have a reasonable amount of data, pick test set "large enough" for a "reasonable" estimate of error, and use the rest for learning
  - If you have little data, then you need to pull out the big guns…
    - e.g., bootstrapping   cross validation

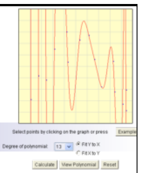
Error estimators

$$error_{true}(\mathbf{w}) \;=\; \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$ ← gold standard

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$ ← data for learning, optimistically biased solution

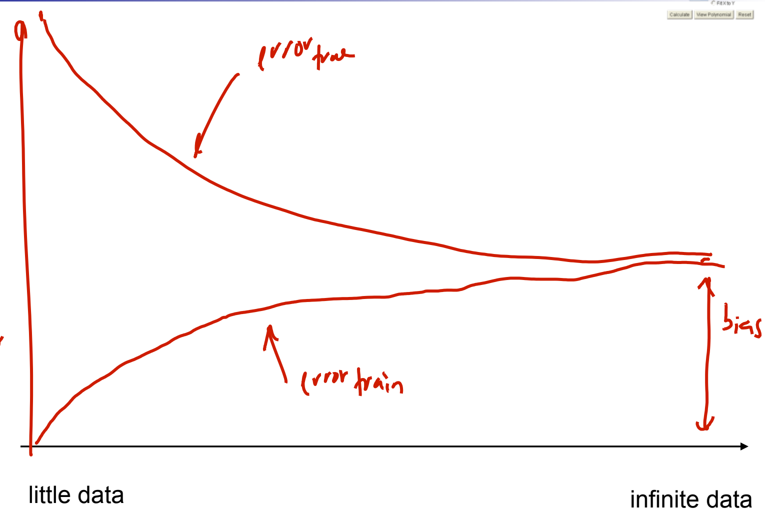
$$error_{test}(\mathbf{w}) \;=\; \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$ ← Evaluate final model

Error as a function of number of training examples for a fixed model complexity



$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$$error_{\cancel{test}\ true}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$



Error estimators

err

err

Be careful!!!

Test set only unbiased if you never never ever ever do any any any any learning on the test data

For example, if you use the test set to select the degree of the polynomial… no longer unbiased!!! (We will address this problem later)

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

What you need to know

- True error, training error, test error
  - Never learn on the test data
  - Never learn on the test data
  - Never learn on the test data
  - Never learn on the test data
  - Never learn on the test data
- Overfitting

"Regularization"

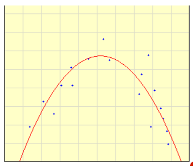
Machine Learning – CSE546
Carlos Guestrin
University of Washington

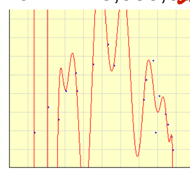
October 2, 2013

Regularization in Linear Regression

- Overfitting usually leads to very large parameter choices, e.g.:

$-2.2 + 3.1\,X - 0.30\,X^2$ $\qquad\qquad$ $-1.1 + 4{,}700{,}910.7\,X - 8{,}585{,}638.4\,X^2 + \ldots$





larger coeff

less variance → smaller coeffs

- ***Regularized*** or ***penalized*** regression aims to impose a "complexity" penalty by penalizing large weights
  - "Shrinkage" method

Ridge Regression

- Ameliorating issues with overfitting: penalize large weights
- New objective:

$$\underbrace{\sum_{j=1}^{N}\left(t(x_j)-\left(w_0+\sum_{i=1}^{k} w_i h_i(x_j)\right)\right)^2}_{\text{train error}} + \lambda \underbrace{\sum_{i=1}^{k} w_i^2}_{\|w_{1..k}\|_2^2}$$

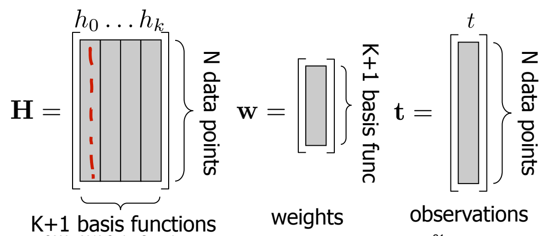
regularization coeff

$\lambda \geq 0$

Ridge Regression in Matrix Notation

$$\hat{\mathbf{w}}_{ridge} = \arg\min_w \sum_{j=1}^{N} \left(t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j)) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

$$= \arg\min_{\mathbf{w}} \underbrace{(\mathbf{Hw} - \mathbf{t})^T (\mathbf{Hw} - \mathbf{t})}_{\text{residual } \cancel{\text{error}}} + \lambda\, \mathbf{w}^T I_{0+k} \mathbf{w}$$



$$I_{0+k} = \begin{pmatrix} 0 & & & & \\ & 1 & & 0 & \\ & & 1 & & \\ & 0 & & 1 & \\ & & & & 1 \end{pmatrix}$$

Minimizing the Ridge Regression Objective

$$\hat{\mathbf{w}}_{ridge} = \arg\min_w \sum_{j=1}^{N} \left(t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j)) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

$$F(w) = (H\mathbf{w} - \mathbf{t})^T (H\mathbf{w} - \mathbf{t}) + \lambda\, \mathbf{w}^T I_{0+k} \mathbf{w}$$

$$\frac{d}{dw} \lambda w_i w = 2\lambda w_i$$

$$\nabla_w F(w) = 2H^T(Hw - t) + 2\lambda I_{0+k} w = 0$$

$$\hat{w}_{ridge} = (H^T H + \lambda I_{0+k})^{-1} H^T t$$ just matrix ops!!

adding λI_{0+k} can only make $H^T H$ "more invertible"
So regularization is also a numerical stabilization method

Shrinkage Properties

$$\hat{\mathbf{w}}_{ridge} = (H^T H + \lambda\ I_{0+k})^{-1} H^T \mathbf{t}$$

- If orthonormal features/basis: $H^T H = I$

$\hat{w}_{MLE} = (H^T H)^{-1} H^T t$

$\hat{w}_{ridge} = (I + \lambda I_{0+k})^{-1} H^T t$

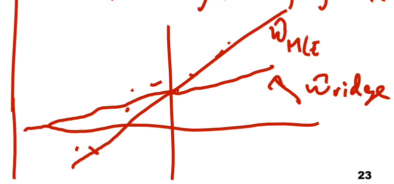
$= \begin{pmatrix} 1 & & \\ & 1+\lambda & 0 \\ & 0 & 1+\lambda \ \ddots \end{pmatrix}^{-1} H^T t$

$\begin{pmatrix} 1 & & \\ & \frac{1}{1+\lambda} & \\ & & \ddots \end{pmatrix}$

For $i > 0$

$\{\hat{w}_{ridge}\}_i = \dfrac{\{\hat{w}_{MLE}\}_i}{1+\lambda}$

For example: degree 1 polynomial:



Ridge Regression: Effect of Regularization

$$\hat{\mathbf{w}}_{ridge} = \arg\min_w \sum_{j=1}^{N} \left(t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j)) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

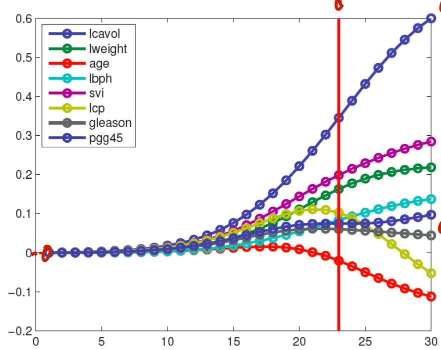
- Solution is indexed by the regularization parameter λ
- Larger λ high regularization
- Smaller λ low regularization
- As λ → 0 $\hat{w}_{ridge} \rightarrow \hat{w}_{MLE}$
- As λ →∞ $\hat{w}_{ridge} \rightarrow w_0 \leftarrow$ average $t(x_j)$
  flat line

Ridge Coefficient Path

"optimal" λ

huge coeff

value of w_i



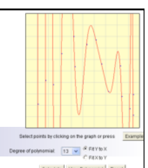
pretty close to $\emptyset$

From
Kevin Murphy
textbook

huge λ $\quad$ $1/\lambda$, $\|w\|$ $\quad$ λ decreasing

- Typical approach: select λ using cross validation, more on this later in the quarter

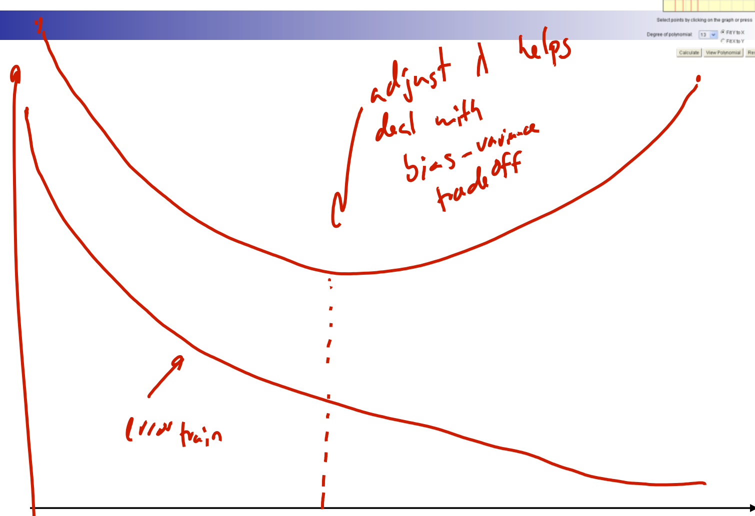
Error as a function of regularization parameter for a fixed model complexity



$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

adjust λ helps deal with bias-variance tradeoff



error train

λ=∞ $\qquad$ λ=0

λ_{opt}

What you need to know…

- Regularization
  - Penalizes for complex models
- Ridge regression
  - L_2 penalized least-squares regression
  - Regularization parameter trades off model complexity with training error

Cross-Validation

Machine Learning – CSE546
Carlos Guestrin
University of Washington

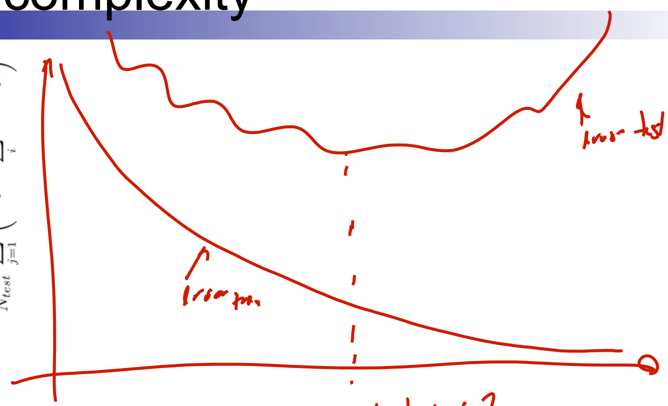
October 2, 2013

Test set error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

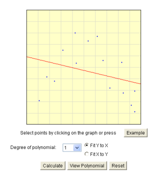
$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

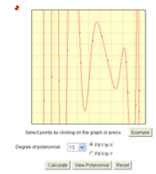
$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$



use test data?

no, no





How… How… How???????

- How do we pick the regularization constant λ…
  - And all other constants in ML, 'cause one thing ML doesn't lack is constants to tune… ☹
- We could use the test data, but…

1. don't learn on test data

2. may not have enough data to separate some to pick λ

(LOO) Leave-one-out cross validation

← choose λ on 1 training example

- Consider a **validation set with 1 example**:
  - D – training data ← N data points
  - $D\backslash j$ – training data with j th data point moved to validation set
- **Learn classifier $h_{D\backslash j}$ with $D\backslash j$ dataset** ← N-1 datapoints (Regressor)
- **Estimate true error** as squared error on predicting $t(\mathbf{x}_j)$:

  $error_{true}(h_{D\backslash j}) = E_x[(t - h_{D\backslash j})^2] \approx (t(x_j) - h_{D\backslash j}(x_j))^2$
  - Unbiased estimate of $error_{true}(h_{D\backslash j})$!
  - Seems really bad estimator, but wait!
- **LOO cross validation**: Average over all data points j:
  - **For each data point you leave out, learn a new classifier $h_{D\backslash j}$** (or regressor)
  - **Estimate error** as:

$$error_{LOO} = \frac{1}{N}\sum_{j=1}^{N}\left(t(\mathbf{x}_j) - h_{\mathcal{D}\backslash j}(\mathbf{x}_j)\right)^2$$

LOO cross validation is (almost) unbiased estimate of true error of h_D!

- When computing **LOOCV error, we only use *N-1* data points**
  - So it's not estimate of true error of learning with N data points!
  - Usually pessimistic, though – learning with less data typically gives worse answer
- **LOO is almost unbiased**!

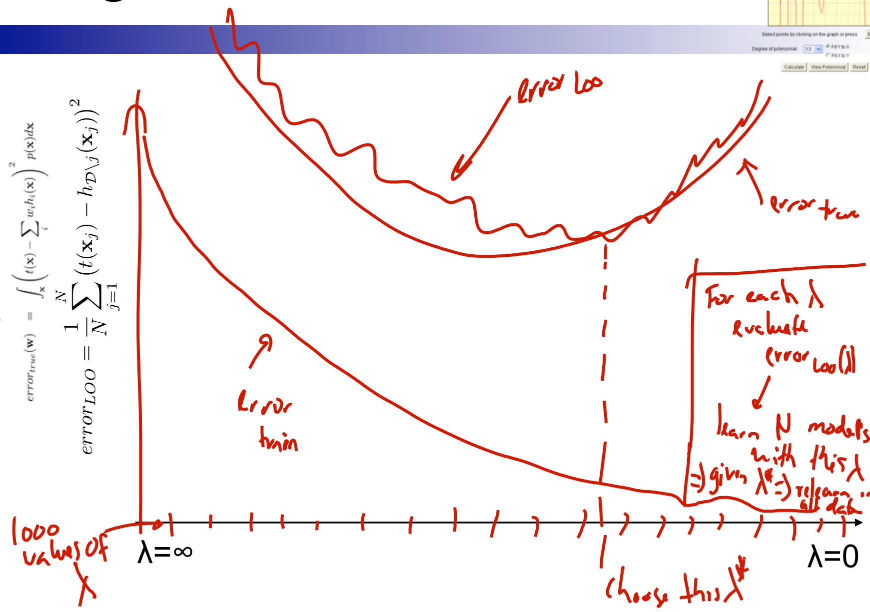
  $error_{true}(h_D) \approx error_{LOO}$
- **Great news!**
  - **Use LOO error for model selection!!!**
  - **E.g., picking λ**

Using LOO to Pick λ

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}$$

$$error_{LOO} = \frac{1}{N} \sum_{j=1}^{N} \left(t(\mathbf{x}_j) - h_{\mathcal{D}\setminus j}(\mathbf{x}_j)\right)^2$$

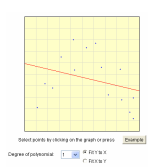


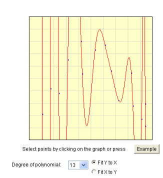
Using LOO error for model selection

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}$$

$$error_{LOO} = \frac{1}{N} \sum_{j=1}^{N} \left(t(\mathbf{x}_j) - h_{\mathcal{D}\setminus j}(\mathbf{x}_j)\right)^2$$





Computational cost of LOO

- Suppose you have 100,000 data points
- You implemented a great version of your learning algorithm
  - Learns in only 1 second
- Computing LOO will take about 1 day!!! of λ
  - If you have to do for each choice of basis functions, it will take fooooooreeeve'!!!
- Solution 1: Preferred, but not usually possible
  - Find a cool trick to compute LOO (e.g., see homework)

Solution 2 to complexity of computing LOO: (More typical) **Use *k*-fold cross validation**

LOO ≡ N-fold cross validation

- Randomly **divide training data into *k* equal parts**
  - $D_1, \ldots, D_k$
- For each i
  - **Learn** regressor or **classifier $h_{D \setminus D_i}$ using data point not in D_i**
  - **Estimate error of $h_{D \setminus D_i}$ on validation set D_i**:

$$error_{\mathcal{D}_i} = \frac{k}{N} \sum_{\mathbf{x}_j \in \mathcal{D}_i} \left(t(\mathbf{x}_j) - h_{\mathcal{D} \setminus \mathcal{D}_i}(\mathbf{x}_j)\right)^2$$

- ***k*-fold cross validation error is average** over data splits:

$$error_{k-fold} = \frac{1}{k} \sum_{i=1}^{k} error_{\mathcal{D}_i}$$

- *k*-fold cross validation properties:
  - **Much faster to compute** than LOO
  - **More (pessimistically) biased** – using much less data, only N *n(k-1)/k*
  - Usually, **k = 10** ☺ (the more the better)

What you need to know…

- Never ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever train on the test data
- Use cross-validation to choose magic parameters such as λ
- Leave-one-out is the best you can do, but sometimes too slow
  - In that case, use k-fold cross-validation