Regularization, Ridge Regression

Machine Learning – CSEP546

Carlos Guestrin

University of Washington

January 13, 2014

The regression problem

$h_3(x) = \sin x \ln x$

$h_4(x) = e^{\cos x}$

(GPA, grade) salary

$h_1(x) = x$ $h_2(x) = x^2$ ...

e.g. polynomials $h_5 = \mathbb{1}(x > 0)$

- **Instances:** <$\mathbf{x}_j$, t_j>
- **Learn:** Mapping from x to t(**x**)
- **Hypothesis space:**
  - Given, basis functions
  - Find coeffs **w**={w_1,…,w_k}

$$H = \{h_1, \ldots, h_K\}$$

$$\underbrace{t(\mathbf{x})}_{\text{data}} \approx \widehat{f}(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x})$$

linear combination

  - Why is this called linear regression???
    - model is linear in the parameters

loss function

- Precisely, minimize the residual squared error:

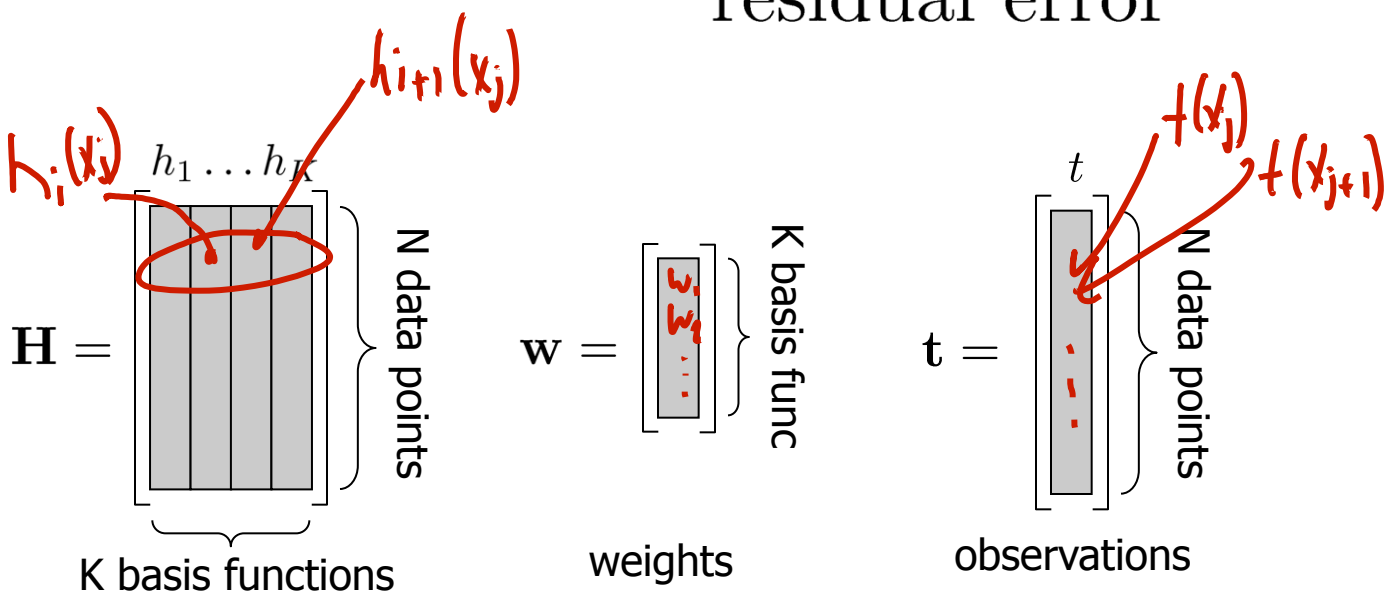
sum data fruth estimate squared

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{j=1}^{N} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

The regression problem in matrix notation

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{j=1}^{N} \left(t(\mathbf{x}_j) - \sum_{i=1}^{K} w_i h_i(\mathbf{x}_j) \right)^2$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \underbrace{(\mathbf{H}\mathbf{w} - \mathbf{t})^T (\mathbf{H}\mathbf{w} - \mathbf{t})}_{\text{residual error}}$$

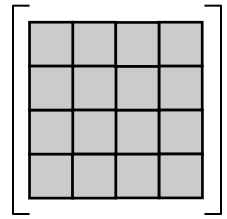
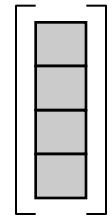


Regression solution = simple matrix operations

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \underbrace{(\mathbf{H}\mathbf{w} - \mathbf{t})^T(\mathbf{H}\mathbf{w} - \mathbf{t})}_{\text{residual error}}$$

loss function

$$\text{solution: } \mathbf{w}^* = \underbrace{(\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}}_{\mathbf{A}^{-1}} \underbrace{\mathbf{H}^{\mathrm{T}}\mathbf{t}}_{\mathbf{b}} = \mathbf{A}^{-1}\mathbf{b}$$

where $\mathbf{A} = \mathbf{H}^{\mathrm{T}}\mathbf{H} =$  $\mathbf{b} = \mathbf{H}^{\mathrm{T}}\mathbf{t} =$ 

k×k matrix
for k basis functions

k×1 vector

Bias-Variance Tradeoff

e.g. degree of polynomial

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

h_D very diff from $h_{D'}$

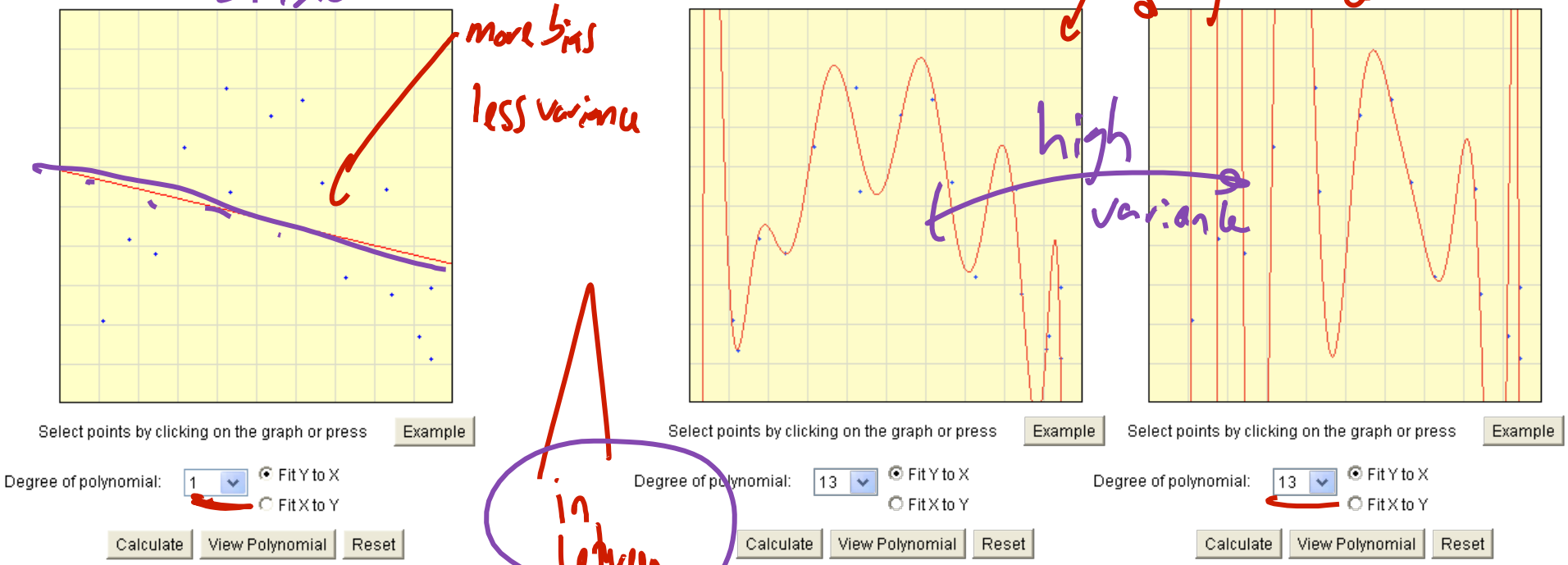
Biased

more bias

less variance

high variance

high variance



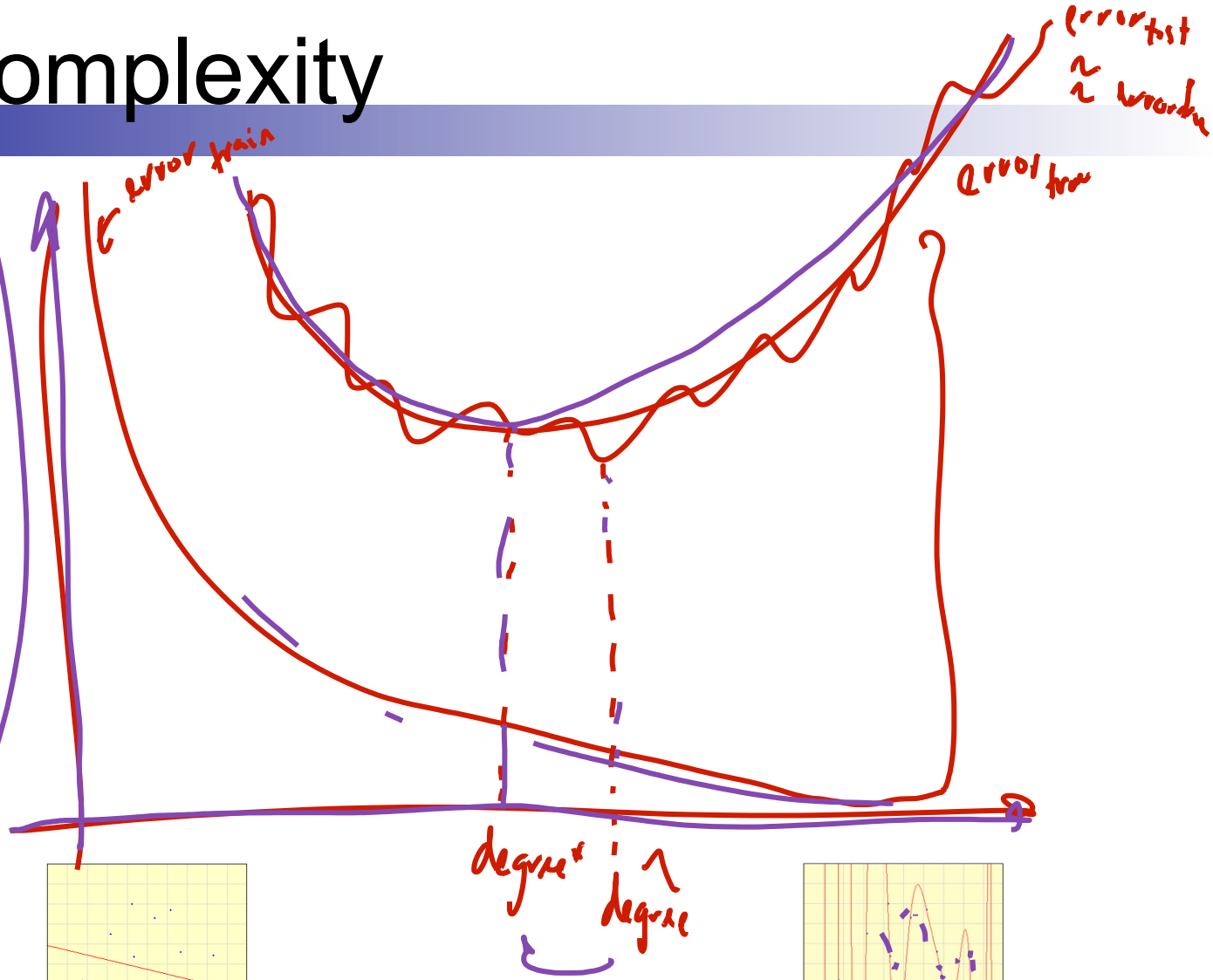
in between

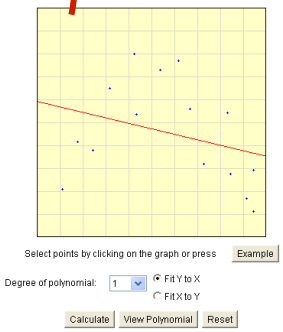
Test set error as a function of model complexity

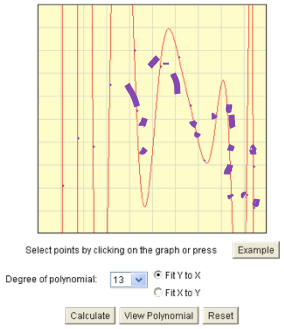
$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

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

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







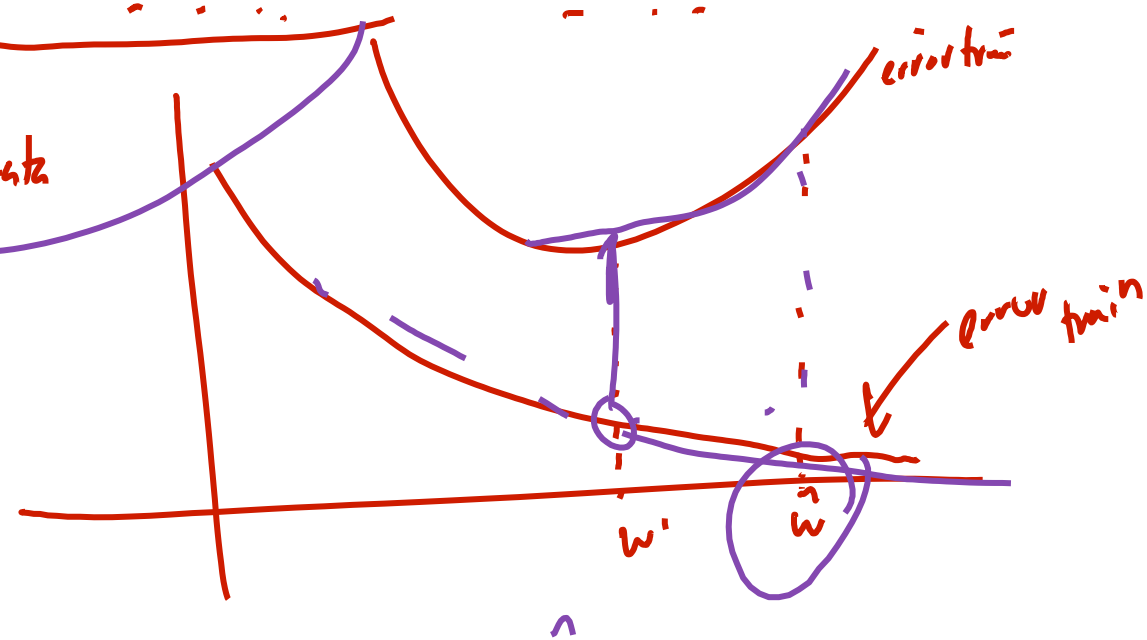
Overfitting

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

worst true error

$$[error_{train}(\hat{\mathbf{w}}) < error_{train}(\mathbf{w}')] \wedge [error_{true}(\mathbf{w}') < error_{true}(\hat{\mathbf{w}})]$$

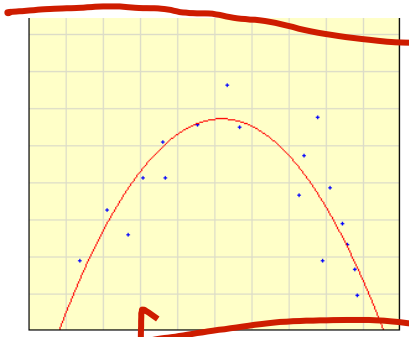
better on train data



Regularization in Linear Regression

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

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



sweet spot

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

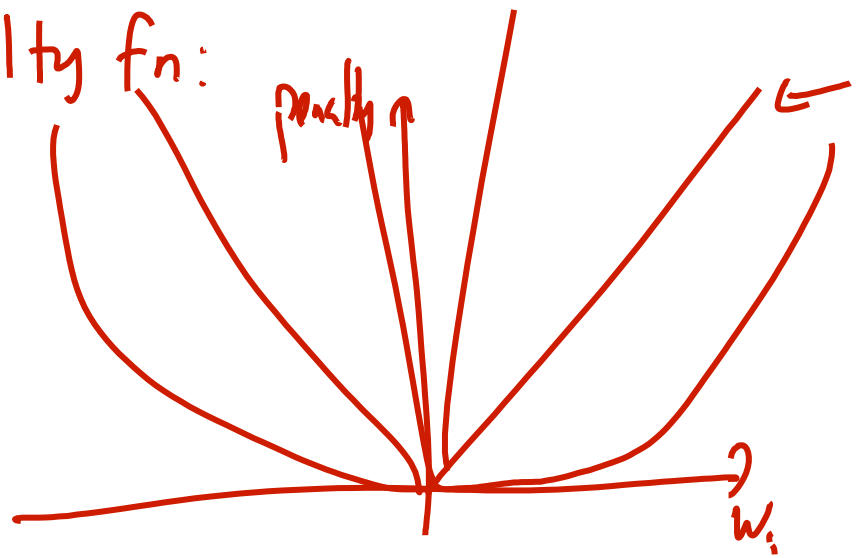
Quadratic Penalty (regularization)

- What we thought we wanted to minimize:

Squared error in prediction

- But weights got too big, penalize large weights:

penalty fn:



← increase as $|w_i|$ increases

Squared penalty

L_2 regularization

penalty w_i^2

Ridge Regression

$\|w\|_2^2 = \sum_{i=1}^{k} w_i^2$

$w = (w_1 \ldots w_k)$

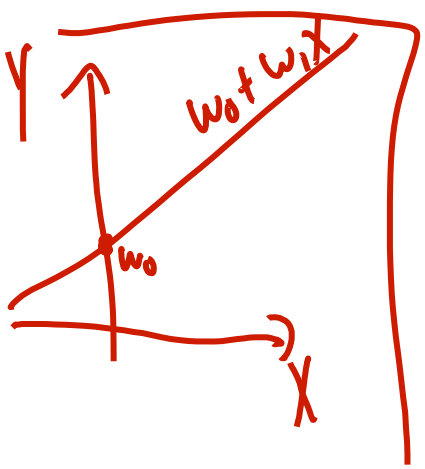
- Ameliorating issues with overfitting: penalizing large weights
- New objective:

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

fitting data

overfitting prevention penalty

$\lambda \geq 0$



in practice, never penalize constant w_0

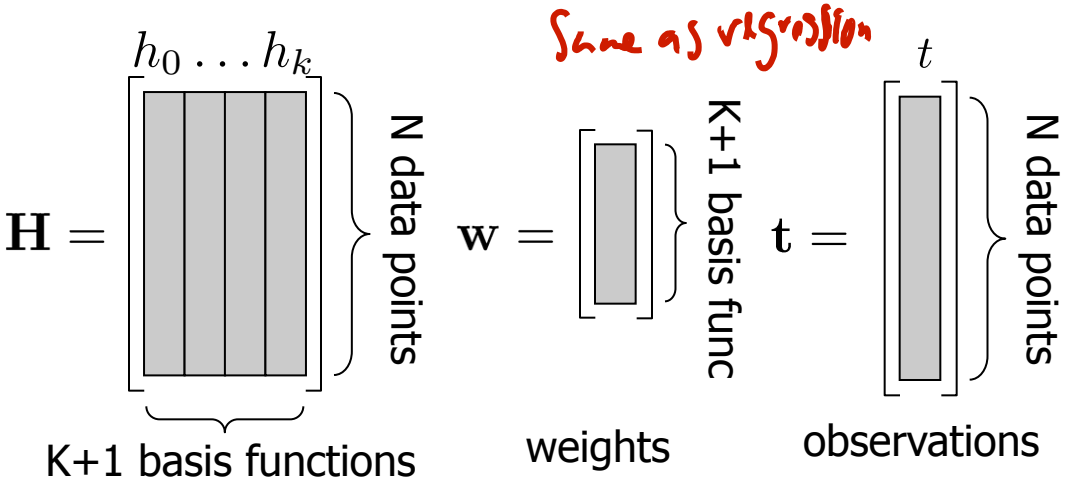
Ridge Regression in Matrix Notation

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

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

Same as regression

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



Minimizing the Ridge Regression Objective

$\frac{1}{a}$ as $a \to 0$

$$\hat{\mathbf{w}}^*_{\text{MLE}} = \underbrace{(\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}}_{\mathbf{A}^{-1}} \underbrace{\mathbf{H}^{\mathrm{T}}\mathbf{t}}_{\mathbf{b}} = \mathbf{A}^{-1}\mathbf{b}$$

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

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

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

Shrinkage Properties

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

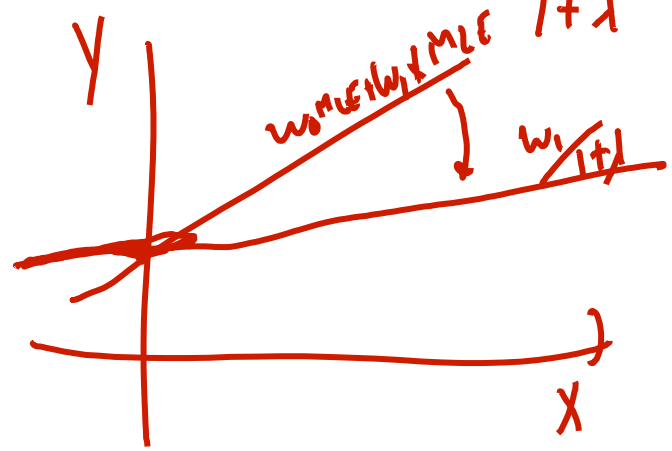
- If orthonormal features/basis: $H^T H = I = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$

$$\hat{w}_{ridge} = \underbrace{(I + \lambda I_{0+k})^{-1}}_{\begin{pmatrix} 1 & & 0 \\ & \frac{1}{1+\lambda} & \\ 0 & & \ddots \; \frac{1}{1+\lambda} \end{pmatrix}} H^T t$$

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

take w_i $i > 0$

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



Ridge Regression: Effect of Regularization

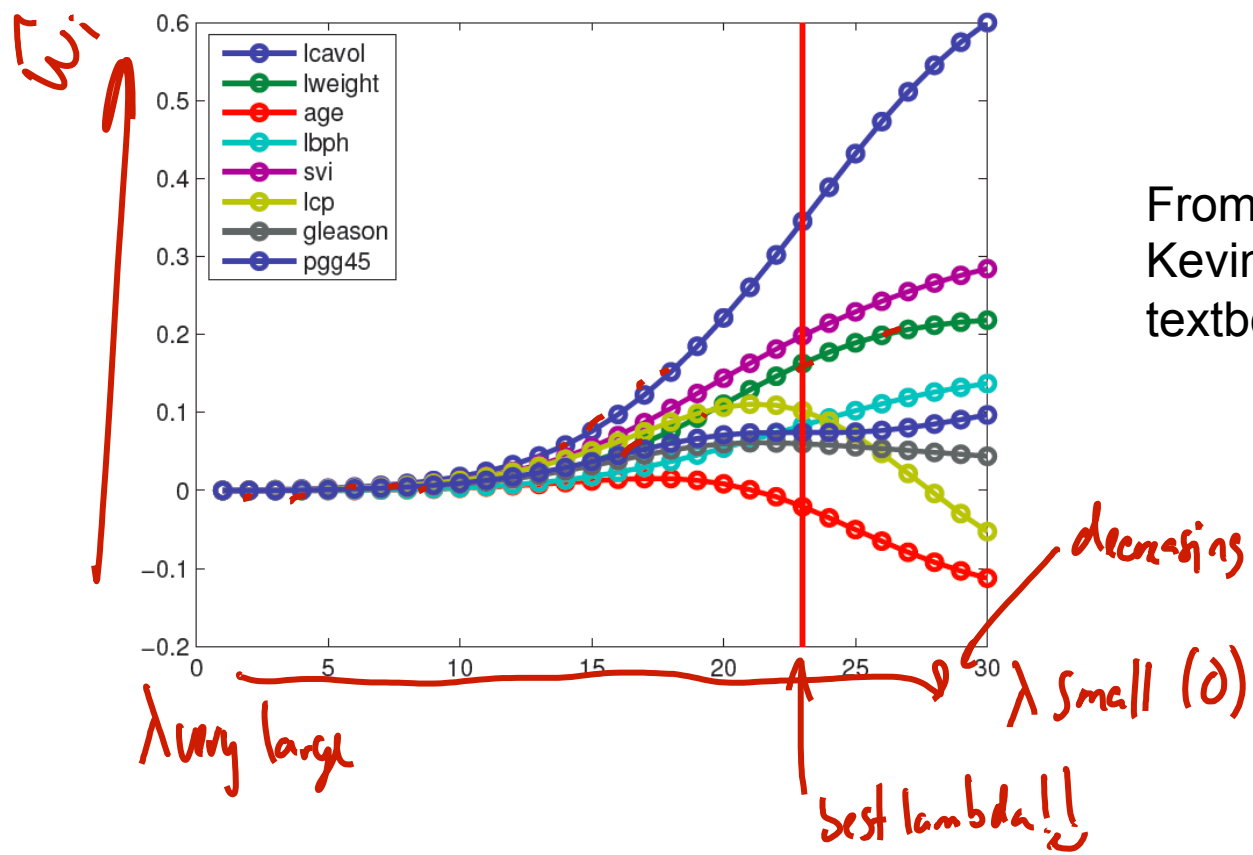
$$\hat{\mathbf{w}}_{ridge} = \arg\min_{w} \sum_{j=1}^{N} \left(t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j)) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

- Solution is indexed by the regularization parameter λ
- Larger λ more penalty → smoother, more bias
- Smaller λ less penalty → more flexible, more variance
- As λ → 0 $\hat{w}_{ridge} \to \hat{w}_{MLE}$
- As λ →∞ $w_1 \ldots w_k \to 0$

  get $w_0 \leftarrow$ average $t(x_j)$

Ridge Coefficient Path

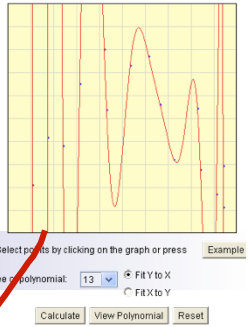
$\|w\|_2 = \sqrt{\sum_i w_i^2}$



From
Kevin Murphy
textbook

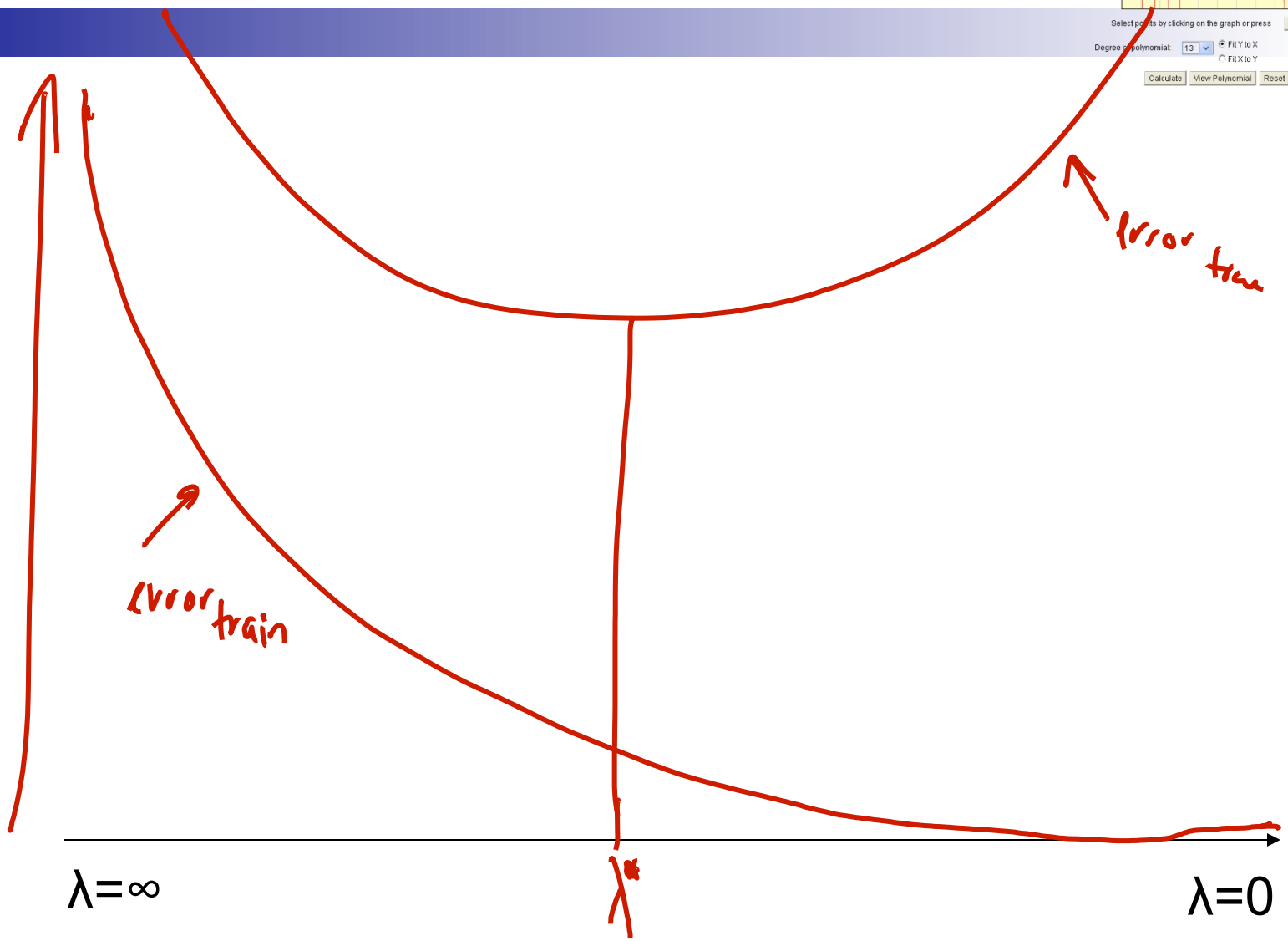
- Typical approach: select λ using cross validation, more on this later in the quarter

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



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

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



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 – CSEP546

Carlos Guestrin

University of Washington

January 13, 2014

Test set error as a function of model complexity

has extra variability

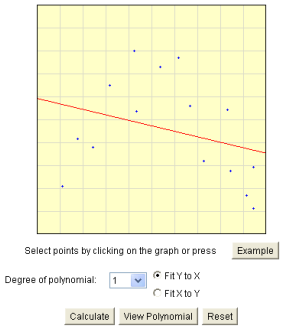
in expectation

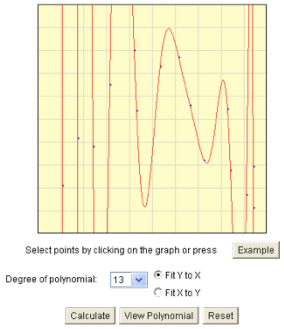
$error_{true} = error_{test}$

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

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

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





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. don't have a lot of data

(LOO) Leave-one-out cross validation

choose lambda based on 1 test example

- Consider a **validation set with 1 example**:
  - D – training data
  - $D\backslash j$ – training data with jth data point moved to validation set
- **Learn classifier $h_{D\backslash j}$ with $D\backslash j$ dataset** ← regressor/ do this for all j learn N functions
- **Estimate true error** as squared error on predicting $t(\mathbf{x}_j)$:
  - Unbiased estimate of $error_{true}(h_{D\backslash j})$! $error_{true}(h_{D/j}) = E_x[(t - h_{D/j})^2]$
  - 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 $h_{D\backslash j}$**
  - **Estimate error** as:

parameterized by λ

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

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!**

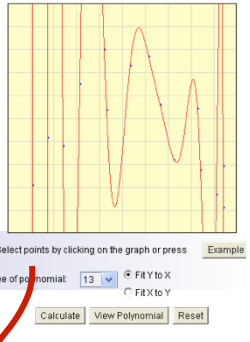
$\text{error}_{LOO} \approx \text{error}_{true}$

use LOO to pick $\hat{\lambda}$

use all N data points with $\hat{\lambda}$ to pick $\hat{w}$

- **Great news!**
  - **Use LOO error for model selection!!!**
  - **E.g., picking λ**

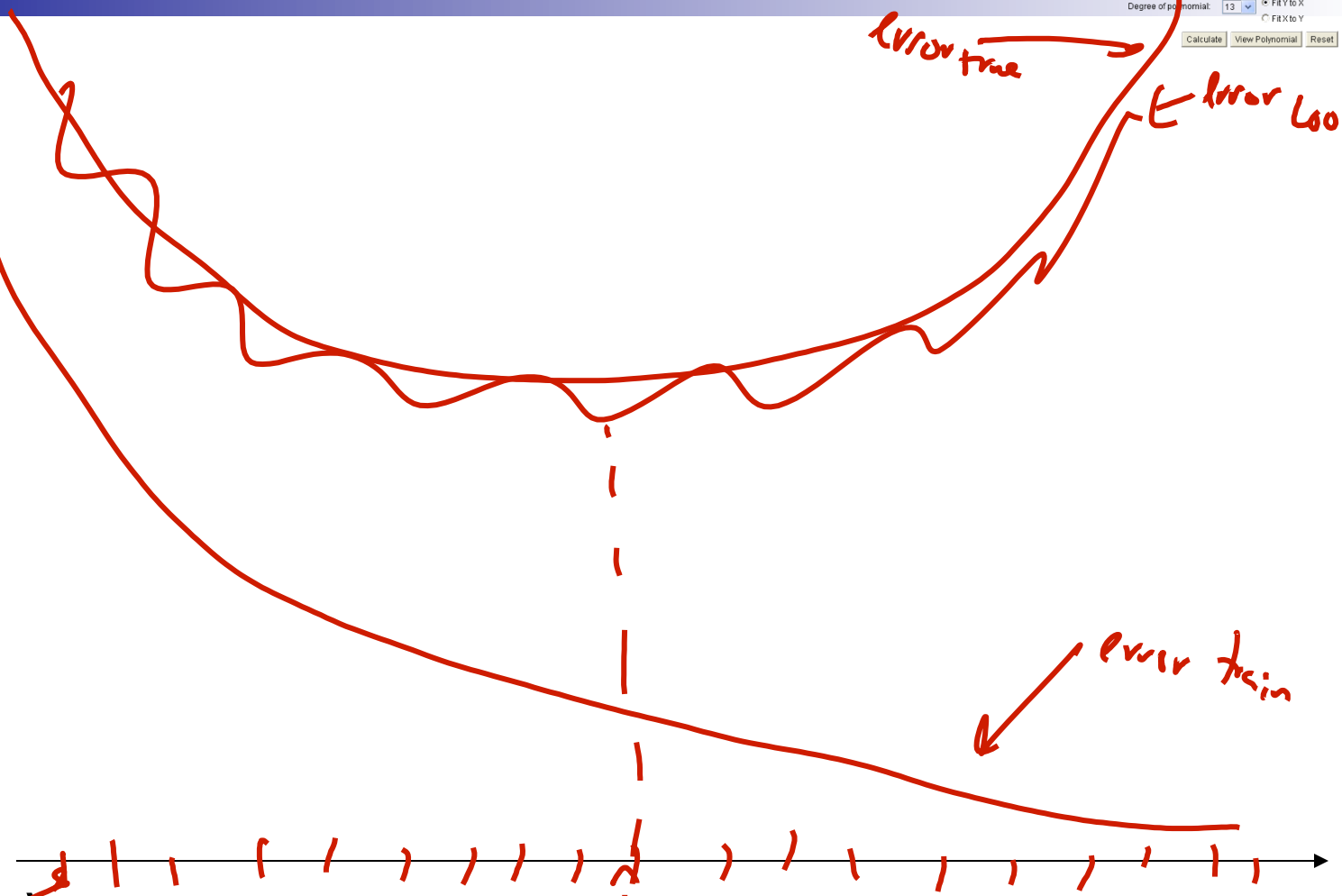
Using LOO to Pick λ



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

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

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



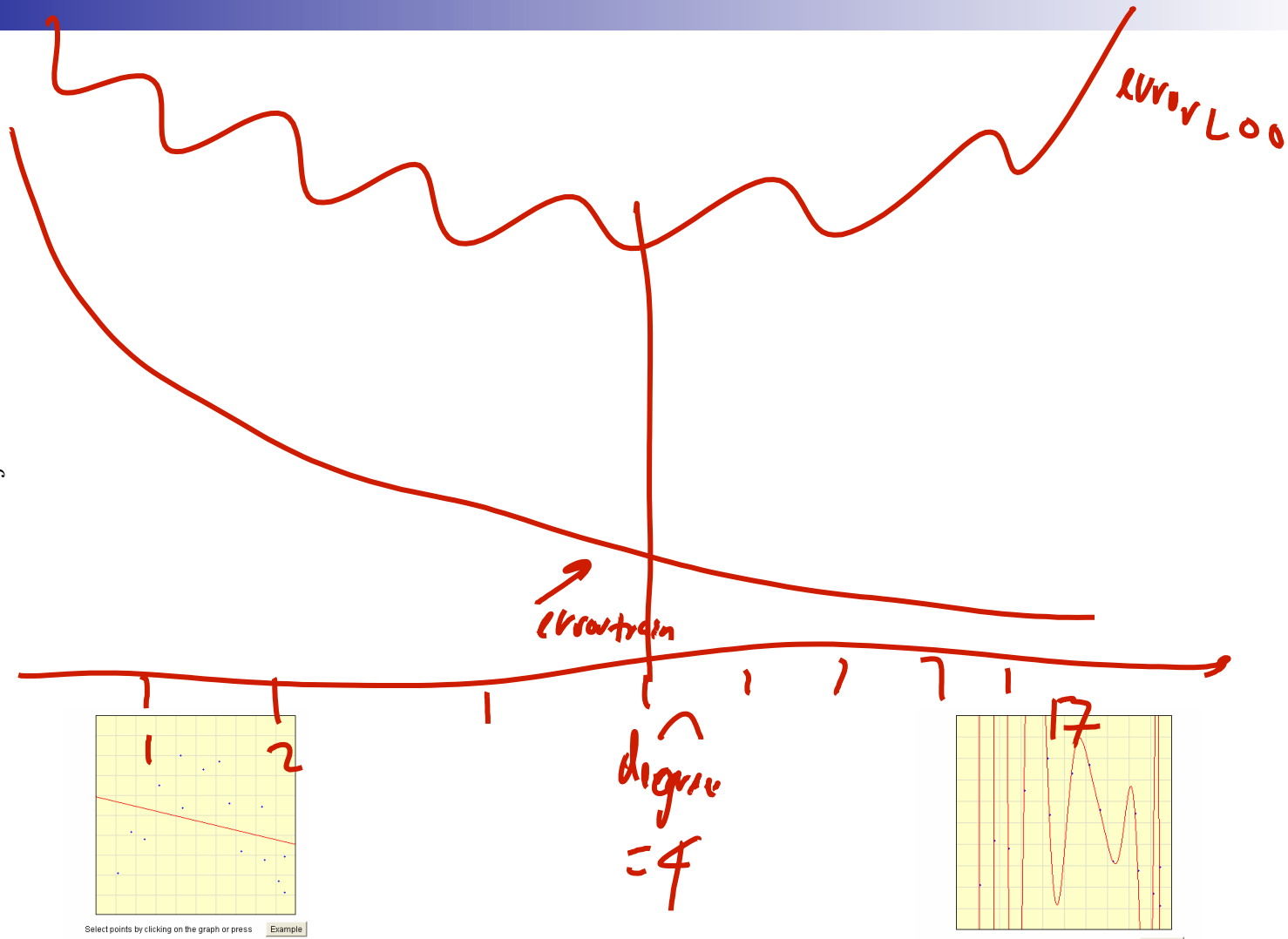
run regression N times

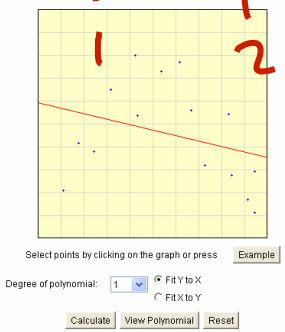
Using LOO error for model selection

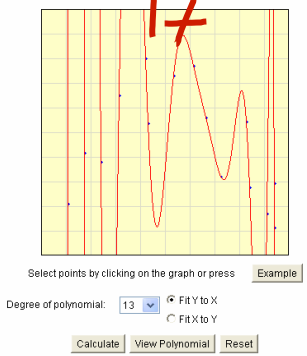
$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

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

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







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!!!
  - 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)

extra credit

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

- Randomly **divide training data into *k* equal parts**
  - $D_1,\ldots,D_k$

k-fold with k = N ≡ LOO

- For each *i*
  - **Learn classifier $h_{D\setminus Di}$ using data point not in D_i**
  - **Estimate error of $h_{D\setminus Di}$ on validation set D_i:**

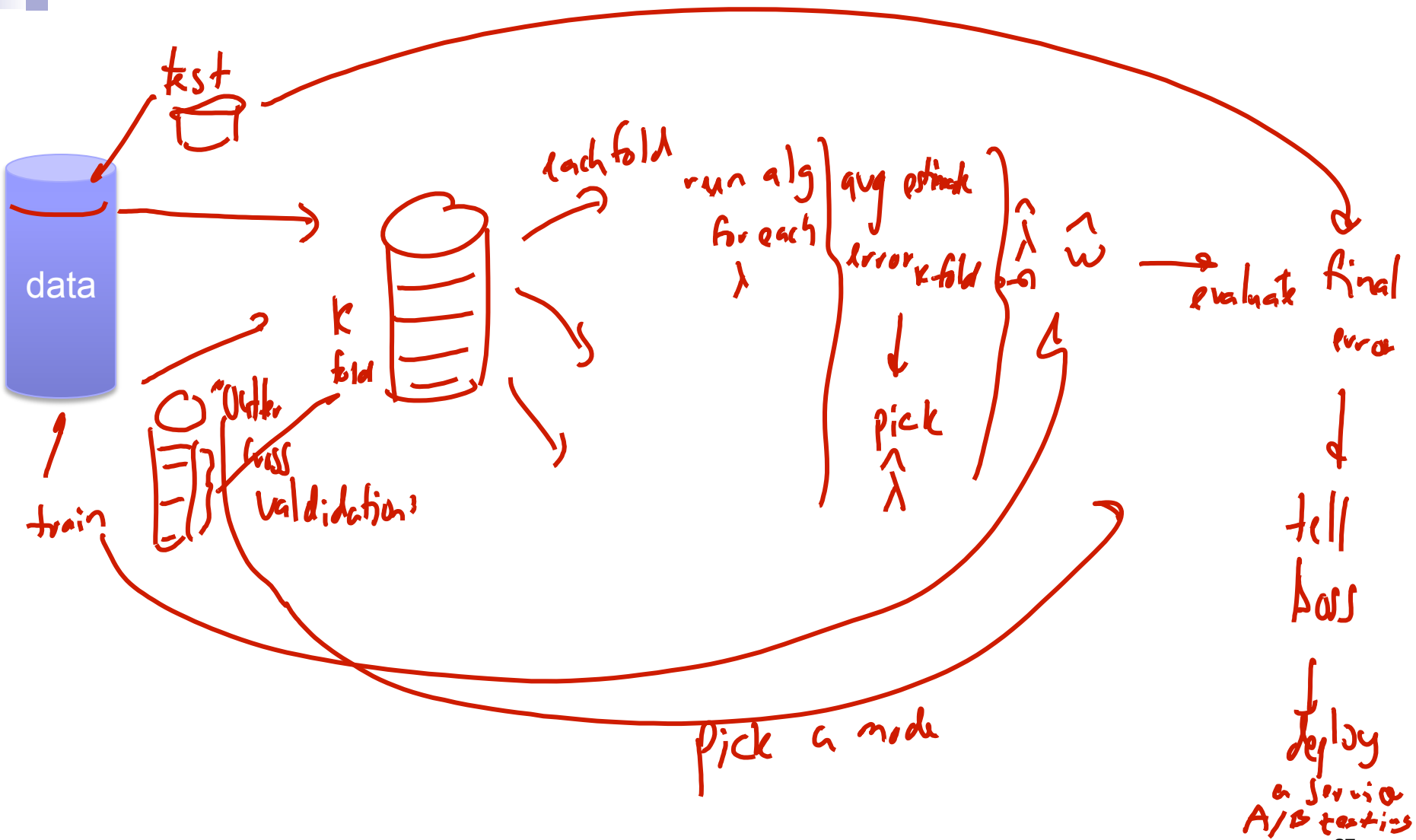
$$error_{\mathcal{D}_i} = \frac{k}{N} \sum_{\mathbf{x}_j \in \mathcal{D}_i} \left(t(\mathbf{x}_j) - h_{\mathcal{D}\setminus\mathcal{D}_i}(\mathbf{x}_j)\right)^2$$

- ***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 ~~*m*~~ N*(k-1)/k*
  - Usually, **k = 10** ☺

ML Pipeline



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 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

Variable Selection LASSO: Sparse Regression

Machine Learning – CSEP546

Carlos Guestrin

University of Washington

January 13, 2014

Sparsity

GPA age grade MC grade ..

- Vector **w** is sparse, if many entries are zero:

  $w = (0.8, 0, 0, 0, 0.7, 0, 0, 0, -2, \ldots)$

- Very useful for many tasks, e.g.,
  - **Efficiency**: If size(**w**) = 100B, each prediction is expensive:
    - If part of an online system, too slow
    - If **w** is sparse, prediction computation only depends on number of non-zeros
  - **Interpretability**: What are the relevant dimension to make a prediction?
    - E.g., what are the parts of the brain associated with particular words?

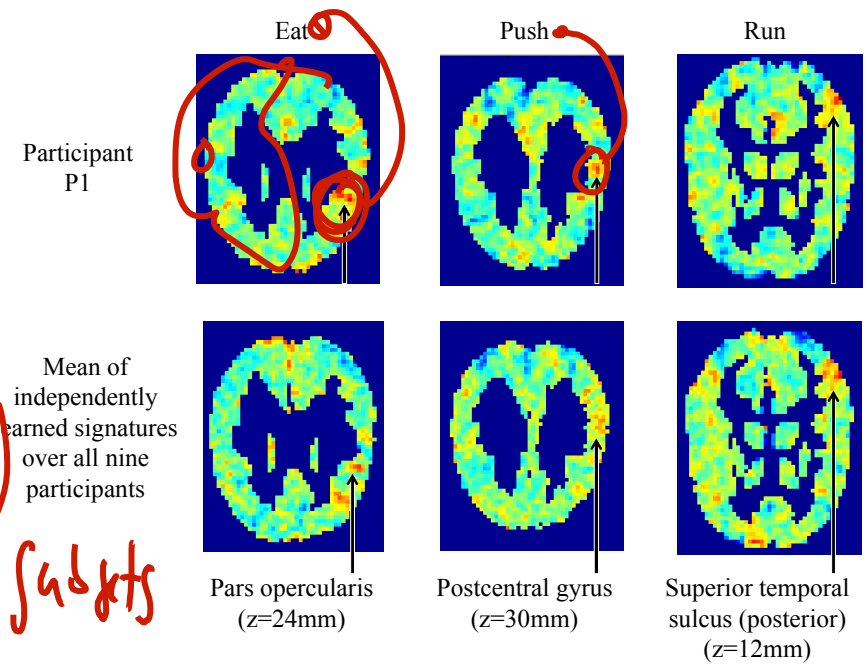


Figure from Tom Mitchell

- But computationally intractable to perform "all subsets" regression

$\binom{100B}{k}$ subsets

Simple greedy model selection algorithm

- Pick a dictionary of features
  - e.g., polynomials for linear regression
- Greedy heuristic:
  - Start from empty (or simple) set of features $F_0 = \emptyset$
  - Run learning algorithm for current set of features F_t
    - Obtain h_t
  - Select **next best feature X_i^***
    - e.g., X_j that results in lowest training error learner when learning with $F_t + \{X_j\}$
  - $F_{t+1} \leftarrow F_t + \{X_i^*\}$
  - Recurse

decreases error the most

Greedy model selection

- Applicable in many settings:
  - Linear regression: Selecting basis functions
  - Naïve Bayes: Selecting (independent) features $P(X_i|Y)$
  - Logistic regression: Selecting features (basis functions)
  - Decision trees: Selecting leaves to expand
- Only a heuristic!
  - But, sometimes you can prove something cool about it
    - e.g., [Krause & Guestrin '05]: Near-optimal in some settings that include Naïve Bayes
- There are many more elaborate methods out there

When do we stop???

- Greedy heuristic:
  - …
  - Select **next best feature X_i^***
    - e.g., X_j that results in lowest training error learner when learning with $F_t + \{X_j\}$
  - $F_{t+1} \leftarrow F_t + \{X_i^*\}$
  - Recurse

When do you stop???

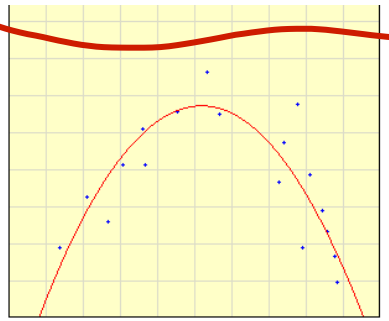
- When training error is low enough? no
- When test set error is low enough? NO
-

Cross validation

Regularization in Linear Regression

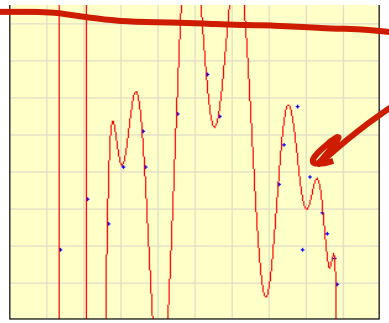
- Overfitting usually leads to very large parameter choices, e.g.:

$-2.2 + 3.1\,X - 0.30\,X^2$



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

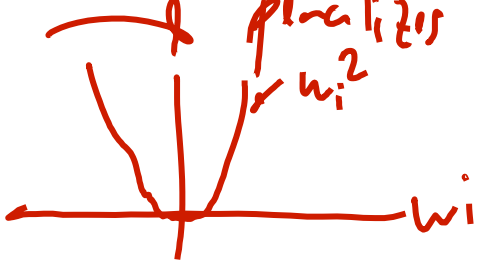
penalty for large weights



overfitting

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

L_2 regularization



towards smoother functions

Variable Selection by Regularization

- Ridge regression: Penalizes large weights

- What if we want to perform "feature selection"?
  - E.g., Which regions of the brain are important for word prediction?
  - Can't simply choose features with largest coefficients in ridge solution

thresholding → can lead to poor solution in practice

$\hat{w}_{ridge}$ (0.8 .. _ε_ + ε ε-e _)

- Try new penalty: Penalize non-zero weights
  - Regularization penalty:

LASSO → instead $\lambda \|w\|_2^2 = \sum_i w_i^2 \longrightarrow \lambda \|w\|_1$ ← L_1 norm

$\|w\|_1 = \sum_i |w_i|$

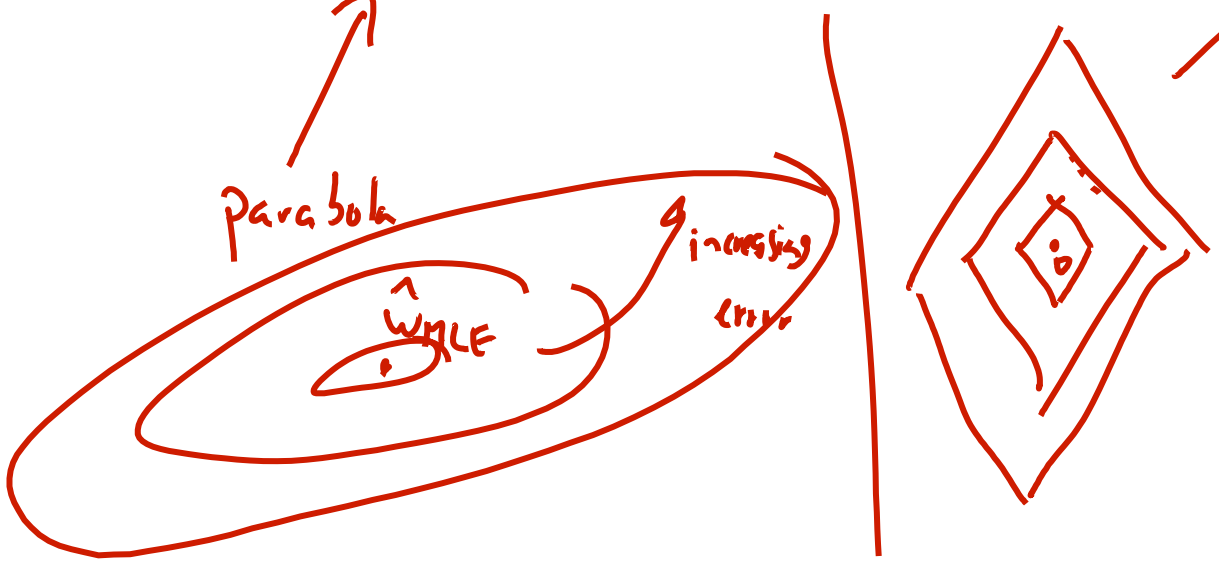
  - Leads to sparse solutions
  - Just like ridge regression, solution is indexed by a continuous param λ
  - This simple approach has changed statistics, machine learning & electrical engineering

LASSO Regression

- **LASSO:** least absolute shrinkage and selection operator

- New objective:

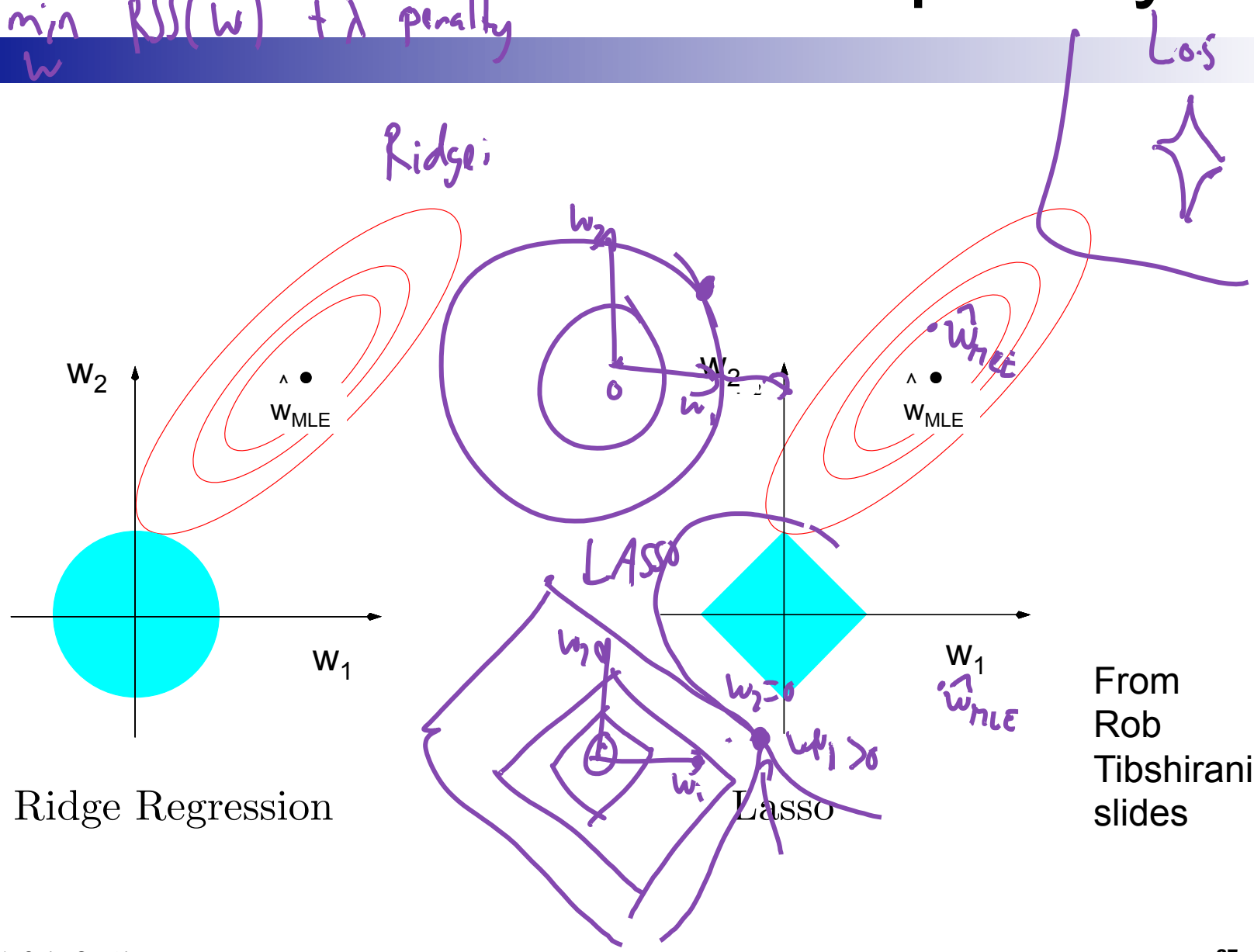
$$\min_{w} \sum_{j=1}^{N} \Big(\overbrace{f(x_j) - \big(w_0 + \sum_{i=1}^{k} w_i h_i(x_j)\big)}^{RSS} \Big)^2 + \overbrace{\lambda \sum_{i=1}^{K} |w_i|}^{penalty}$$



don't penalize w_0, it did nothing to you....

Geometric Intuition for Sparsity

$\min_w RSS(w) + \lambda$ penalty



Ridge Regression

Lasso

From Rob Tibshirani slides

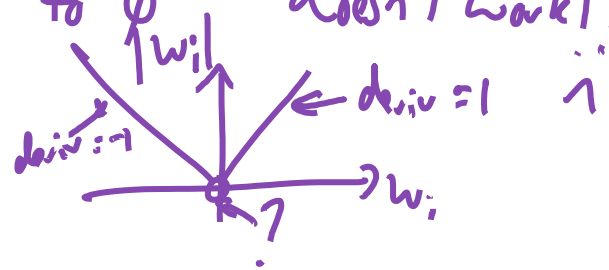
Optimizing the LASSO Objective

- LASSO solution:

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

Take derivative set to $\emptyset$ doesn't work!!

1. derivative $|w_i|$



2. Even if you could take derivative (subgradient) there is no closed-form solution!! for $\hat{w}_{LASSO}$

Coordinate Descent

testing convergence:
round robin: diff in error between passes over all parameter

$F(w^{(0)})$
diff (pass over all params
same $F(w^{(1)})$) → Stop
small ⇒ $< \varepsilon$

- Given a function F
  - Want to find minimum

$\hat{w} = \arg\min_{w} F(w_0, w_1, \ldots, w_k)$

- Often, hard to find minimum for all coordinates, but easy for one coordinate

1-d optimization problems

- Coordinate descent: initialize $w = 0$ or something else
  while not converged,
  pick coordinate i
  $\hat{w}_i \leftarrow \arg\min_{z} F(\hat{w}_0, \hat{w}_1, \ldots \hat{w}_{i-1}, z, \hat{w}_{i+1}, \ldots \hat{w}_k)$

fix all but w_i

↑ optimize w_i

- How do we pick next coordinate?

round robin, random, "smartly"

LASSO OPT !!
other times: local optima or worse

- Super useful approach for *many* problems
  - Converges to optimum in some cases, such as LASSO

How do we find the minimum over each coordinate?

- Key step in coordinate descent:
  - Find minimum over each coordinate

$$\underset{z}{\operatorname{argmin}} \; F(w_0, w_1 \ldots w_{i-1}, z, w_{i+1}, \ldots w_k)$$

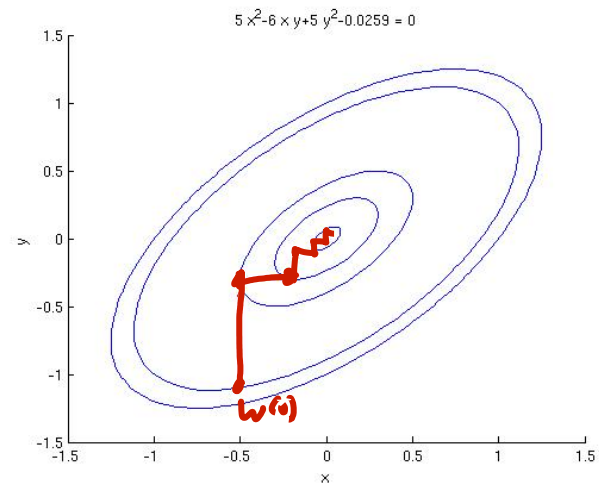


Illustration from Wikipedia

- Standard approach: partial
  take derivative & set to $\emptyset$
  wrt w_i

Optimizing LASSO Objective One Coordinate at a Time

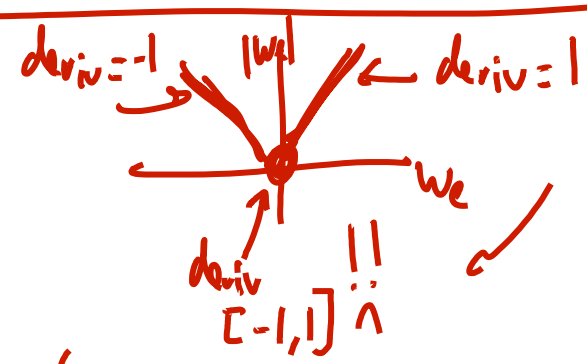
$$\overbrace{\sum_{j=1}^{N}\left(t(x_j)-(w_0+\sum_{i=1}^{k}w_ih_i(x_j))\right)^2}^{RSS(w)}+\overbrace{\lambda\sum_{i=1}^{k}|w_i|}^{penalty}$$

- Taking the derivative:
  - Residual sum of squares (RSS):

$$\frac{\partial}{\partial w_\ell}RSS(\mathbf{w})=\underbrace{-2\sum_{j=1}^{N}h_\ell(x_j)\left(t(x_j)-(w_0+\sum_{i=1}^{k}w_ih_i(x_j))\right)}$$

  - Penalty term:

$$\frac{d}{\partial w_\ell}\lambda\sum_{i=1}^{k}|w_i| = \lambda\frac{d}{\partial w_\ell}|w_\ell|$$



important because I want $w_\ell = 0$ for most ℓ

Need more advanced machinery called subgradient

Coordinate Descent for LASSO (aka Shooting Algorithm)

Outer loop explores decreasing values of λ using soln from one λ to init for next λ

stop old-error − new error $< \varepsilon$

initialize $w = 0$ ← for first λ

old error = $RSS(w) + \lambda\|w\|_1$

new error ...

- Repeat until convergence
  - Pick a coordinate ℓ at (random or sequentially) ← for $\ell = 0 \ldots k$
    - Set: $\forall \ell > 0$

$$\hat{w}_\ell = \begin{cases} (c_\ell + \lambda)/a_\ell & c_\ell < -\lambda \\ 0 & c_\ell \in [-\lambda, \lambda] \\ (c_\ell - \lambda)/a_\ell & c_\ell > \lambda \end{cases}$$

    - Where:

$$a_\ell = 2\sum_{j=1}^{N} (h_\ell(\mathbf{x}_j))^2$$

re compute

$$c_\ell = 2\sum_{j=1}^{N} h_\ell(\mathbf{x}_j)\left(t(\mathbf{x}_j) - (\hat{w}_0 + \sum_{i \neq \ell} \hat{w}_i h_i(\mathbf{x}_j))\right)$$

depends on all $\hat{w}_i$ except for $\hat{w}_\ell$

don't regularize w_0

$\hat{w}_0 = c_0 / a_0$

$$\hat{w}_0 = \frac{1}{N}\sum_{j=1}^{N}\left(t(x_j) - \sum_{i=1}^{k} \hat{w}_i h_i(x_j)\right)$$

  - For convergence rates, see Shalev-Shwartz and Tewari 2009
- Other common technique = LARS
  - Least angle regression and shrinkage, Efron et al. 2004

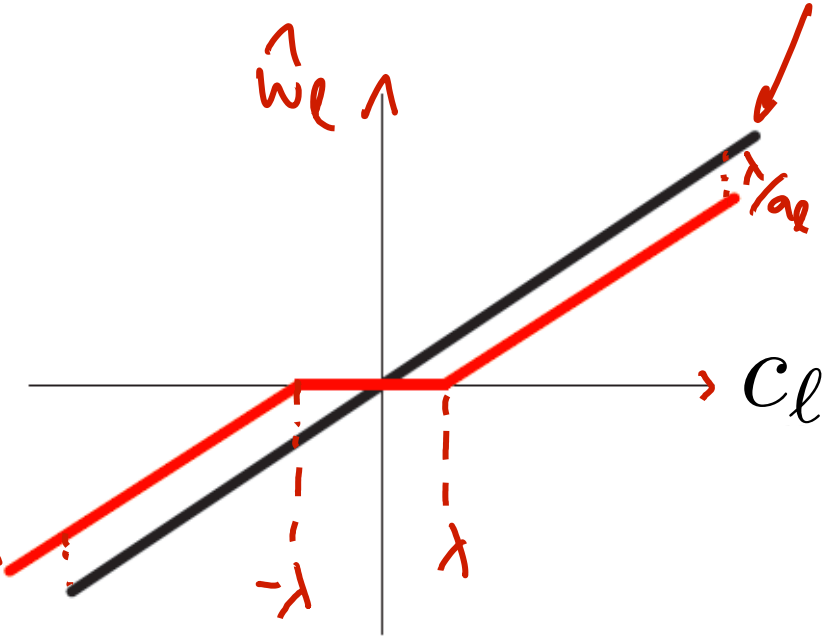
Soft Thresholding

$\lambda = 0 \rightarrow$ MLE solution is $\hat{w}_\ell^{MLE} = c_\ell / a_\ell$

$$\hat{w}_\ell^{LASSO} = \begin{cases} (c_\ell + \lambda)/a_\ell & c_\ell < -\lambda \\ 0 & c_\ell \in [-\lambda, \lambda] \\ (c_\ell - \lambda)/a_\ell & c_\ell > \lambda \end{cases}$$

inner loop of LASSO solver

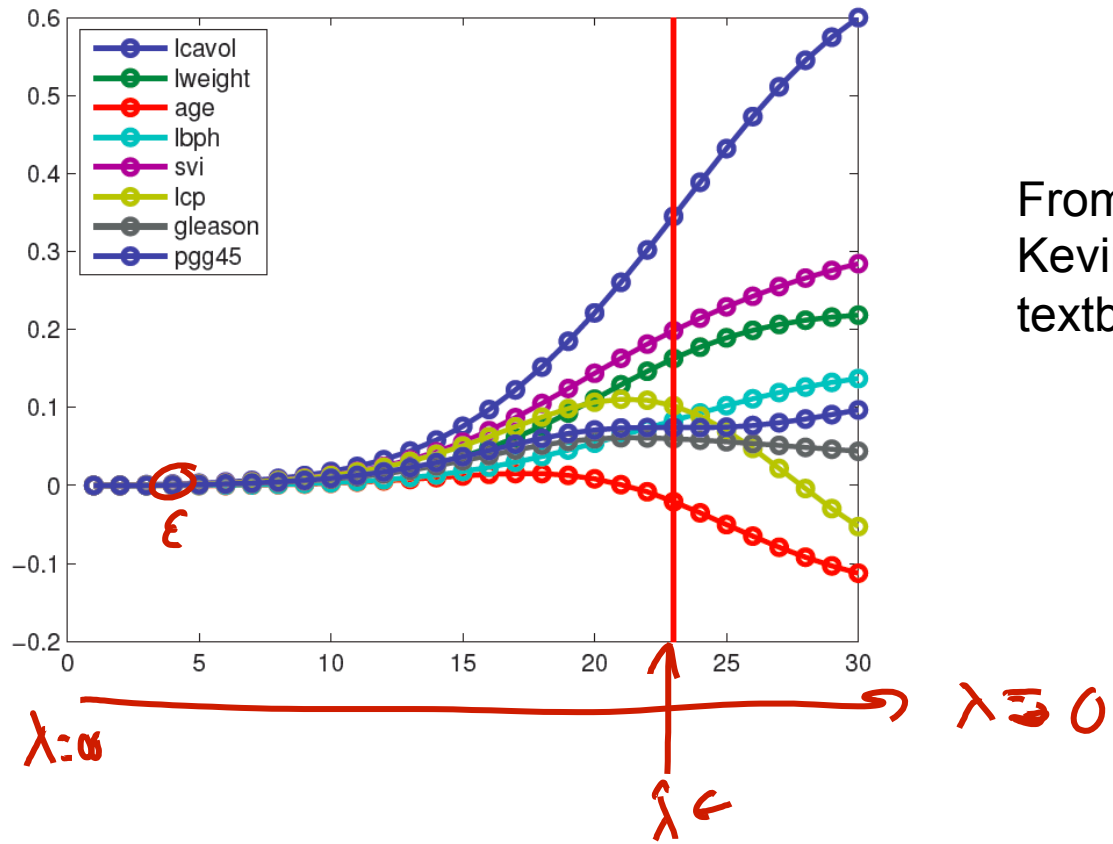
$\hat{w}_\ell^{MLE} = c_\ell / a_\ell$



$\hat{w}_\ell^{LASSO}$

From Kevin Murphy textbook

Recall: *Ridge Coefficient Path*

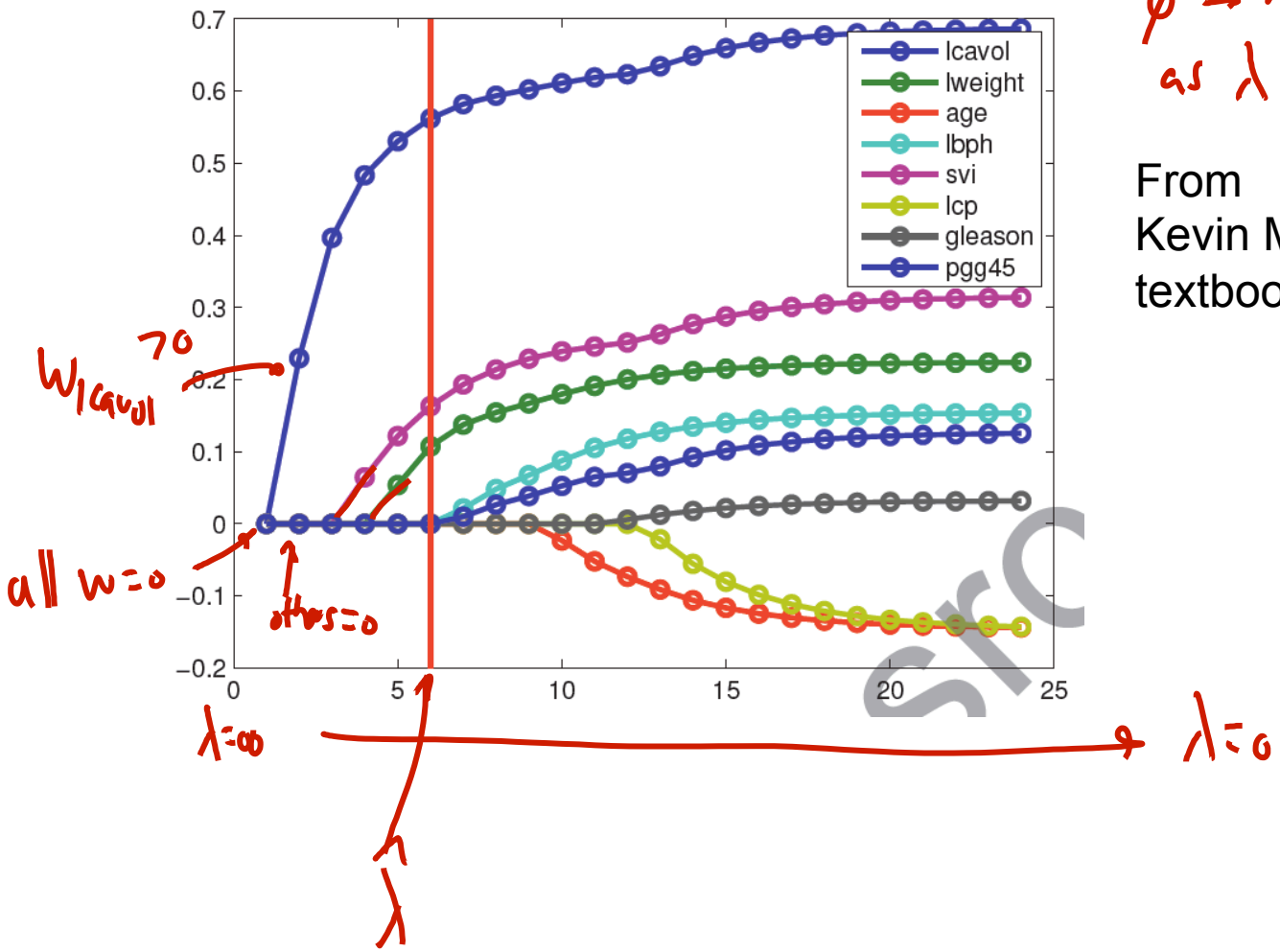


From
Kevin Murphy
textbook

- Typical approach: select λ using cross validation

Now: *LASSO Coefficient Path*

params can be
$\emptyset \to$ nonzero $\to \emptyset$
as λ decreases



From
Kevin Murphy
textbook

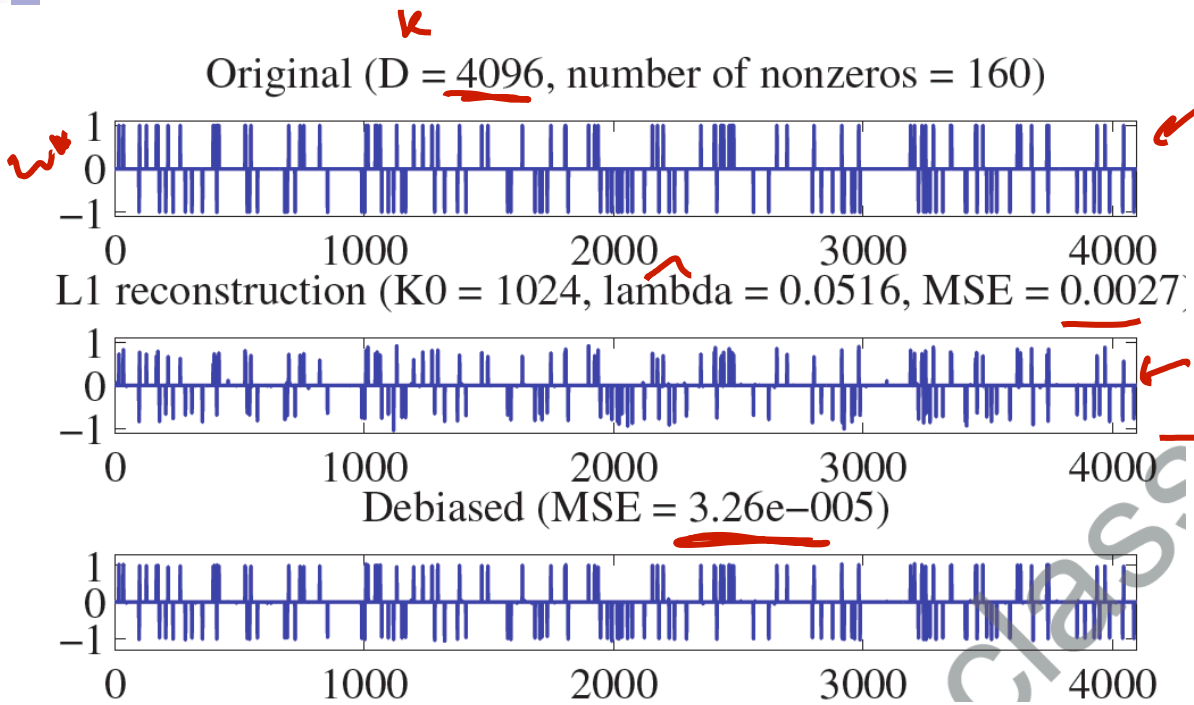
LASSO Example

Term	Least Squares	Ridge	Lasso
Intercept	2.465	2.452	2.468
lcavol	0.680	0.420	0.533
lweight	0.263	0.238	0.169
age	−0.141	−0.046	
lbph	0.210	0.162	0.002
svi	0.305	0.227	0.094
lcp	−0.288	0.000	
gleason	−0.021	0.040	
pgg45	0.267	0.133	

From Rob Tibshirani slides

Debiasing

MSE = RSS



K

w^*

true params: all -1, 0, or 1

LASSO

pushed to 0, by soft thresholding

debiasing step

solve unregularized regression using only selected features

$w_e \neq 0$

From Kevin Murphy textbook

What you need to know

- Variable Selection: find a sparse solution to learning problem
- L_1 regularization is one way to do variable selection
  - Applies beyond regressions
  - Hundreds of other approaches out there
- LASSO objective non-differentiable, but convex ➔ Use subgradient
- No closed-form solution for minimization ➔ Use coordinate descent
- Shooting algorithm is very simple approach for solving LASSO