Regularization

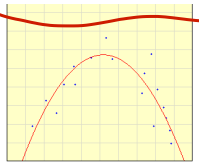
Machine Learning – CSE446
Carlos Guestrin
University of Washington

April 10, 2013

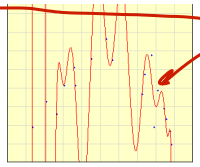
Regularization in Linear Regression

- 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 + …



penalty for large weights

overfitting

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

L_2 regularization → penalizes towards smoother functions

Ridge Regression

- Ameliorating issues with overfitting:
- New objective:

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

$$= \|w_{1..k}\|_2^2$$

error train

penalizing large weights "regularization"

$\lambda > 0$ $\|\|_2^2$ =

trade off error train with magnitude of coeffs.

L2 regularization

importantly, I don't penalize w_0

Let me fix

Ridge Regression in Matrix Notation

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

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

$\mathbf{H} =$ ($h_0 \ldots h_k$; N data points; K+1 basis functions)

$\mathbf{w} =$ (K+1 basis func; weights)

$\mathbf{t} =$ (t; N data points; observations)

Minimizing the Ridge Regression Objective

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

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

Shrinkage Properties

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

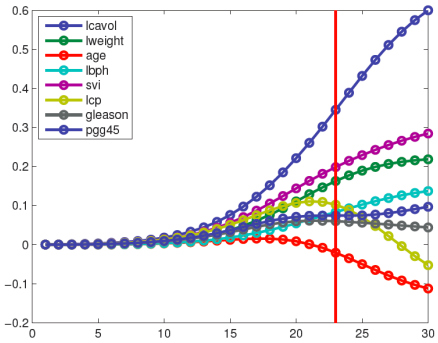
- If orthonormal features/basis: $H^T H = I$

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
- Smaller λ
- As λ → 0
- As λ →∞

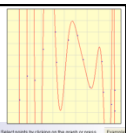
Ridge Coefficient Path



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



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

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

λ=∞ λ=0

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 – CSE446
Carlos Guestrin
University of Washington

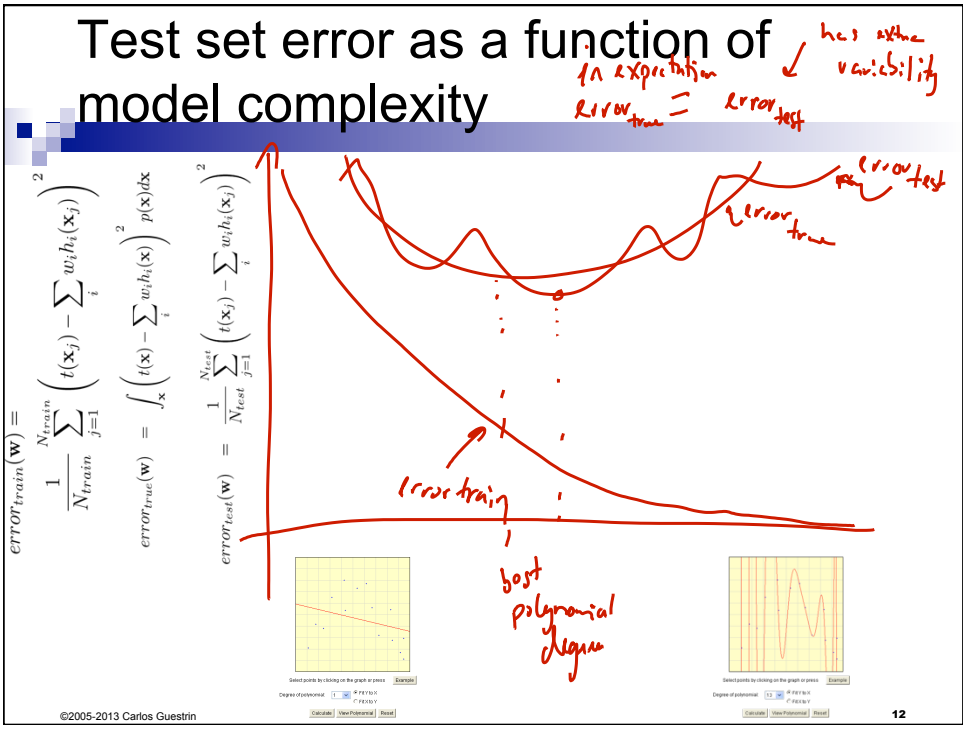
April 10, 2013

Test set error as a function of model complexity

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

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

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



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…

(LOO) Leave-one-out cross validation

- Consider a **validation set with 1 example**:
  - *D* – training data
  - *D*\j – training data with *j*th data point moved to validation set
- **Learn classifier $h_{D\setminus j}$ with *D*\j dataset**
- **Estimate true error** as squared error on predicting t($\mathbf{x}_j$):
  - Unbiased estimate of $error_{true}(h_{D\setminus j})$!
  - 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 $h_{D\setminus j}$**
  - **Estimate error** as:

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

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

Using LOO to Pick λ

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

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

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

λ=∞ ⟶ λ=0

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)

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

- Randomly **divide training data into *k* equal parts**
  - $D_1,\ldots,D_k$
- For each *i*
  - **Learn 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 *m(k-1)/k*
  - Usually, **k = 10** ☺

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 – CSE446

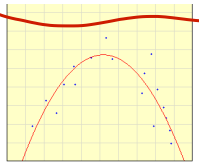
Carlos Guestrin
University of Washington

April 10, 2013

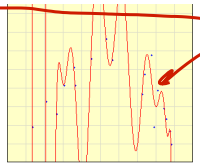
Regularization in Linear Regression

- 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 → penalizes towards smoother functions

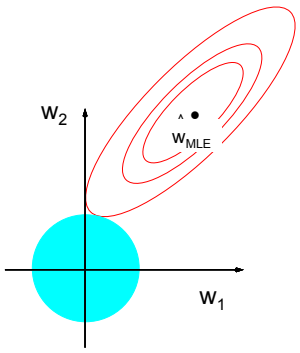
Variable Selection

- 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
  - Computationally intractable to perform "all subsets" regression
- Try new penalty: Penalize non-zero weights
  - Regularization penalty:
  - 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

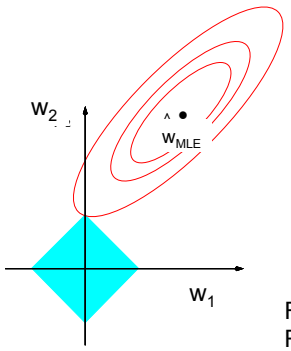
LASSO Regression

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

Geometric Intuition for Sparsity



Ridge Regression



Lasso

From Rob Tibshirani slides

Optimizing the LASSO Objective

- LASSO solution:

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

Coordinate Descent

- Given a function F
  - Want to find minimum
- Often, hard to find minimum for all coordinates, but easy for one coordinate
- Coordinate descent:
- How do we pick next coordinate?
- Super useful approach for *many* problems
  - Converges to optimum in some cases, such as LASSO

Optimizing LASSO Objective One Coordinate at a Time

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

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

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

  - Penalty term:

Subgradients of Convex Functions

- Gradients lower bound convex functions:

- Gradients are unique at **w** iff function differentiable at **w**

- Subgradients: Generalize gradients to non-differentiable points:
  - Any plane that lower bounds function:

Taking the Subgradient

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

- Gradient of RSS term:

$$\frac{\partial}{\partial w_\ell}RSS(\mathbf{w})=a_\ell w_\ell-c_\ell$$

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

$$c_\ell=2\sum_{j=1}^{N}h_\ell(\mathbf{x}_j)\left(t(\mathbf{x}_j)-(w_0+\sum_{i\neq\ell}w_i h_i(\mathbf{x}_j))\right)$$

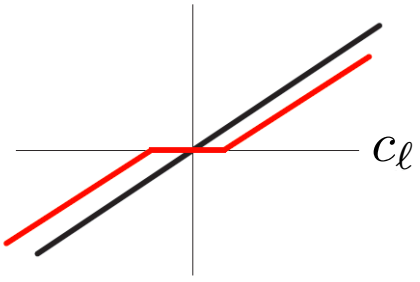
  - If no penalty:
- Subgradient of full objective:

Setting Subgradient to 0

$$\partial_{w_\ell} F(\mathbf{w}) = \begin{cases} a_\ell w_\ell - c_\ell - \lambda & w_\ell < 0 \\ [-c_\ell - \lambda, -c_\ell + \lambda] & w_\ell = 0 \\ a_\ell w_\ell - c_\ell + \lambda & w_\ell > 0 \end{cases}$$

Soft Thresholding

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

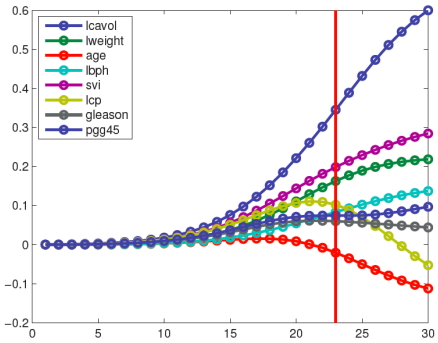


From
Kevin Murphy
textbook

Coordinate Descent for LASSO (aka Shooting Algorithm)

- Repeat until convergence
  - Pick a coordinate *l* at (random or sequentially)
    - Set:
$$\hat{w}_\ell = \begin{cases} (c_\ell + \lambda)/a_\ell & c_\ell < -\lambda \\ 0 & c_\ell \in [-\lambda, \lambda] \\ (c_\ell - \lambda)/a_\ell & c_\ell > \lambda \end{cases}$$
    - Where:
$$a_\ell = 2\sum_{j=1}^{N} (h_\ell(\mathbf{x}_j))^2$$
$$c_\ell = 2\sum_{j=1}^{N} h_\ell(\mathbf{x}_j) \left(t(\mathbf{x}_j) - (w_0 + \sum_{i \neq \ell} w_i h_i(\mathbf{x}_j)) \right)$$
  - For convergence rates, see Shalev-Shwartz and Tewari 2009
- Other common technique = LARS
  - Least angle regression and shrinkage, Efron et al. 2004

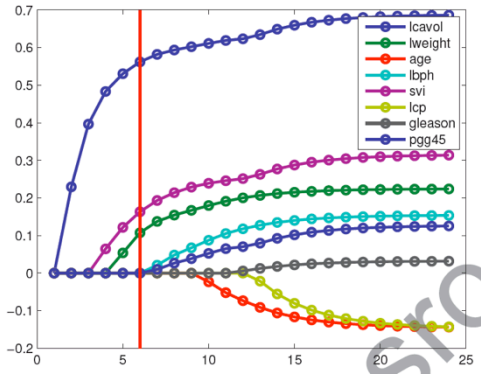
Recall: *Ridge Coefficient Path*



From Kevin Murphy textbook

- Typical approach: select λ using cross validation

Now: *LASSO Coefficient Path*



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

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