Is the test error unbiased for these programs?

```python
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector
mu = np.mean(X, axis=0)
X = X - mu

idx = np.random.permutation(1000)
TRAIN = idx[0:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN,:]

# Solve for argmin_w ||Xtrain*w - ytrain||_2
w = np.linalg.solve( np.dot(Xtrain.T, Xtrain),
                     np.dot(Xtrain.T, ytrain) )
b = np.mean(ytrain)

ytest = y[TEST]
Xtest = X[TEST,:]

train_error = np.dot( np.dot(Xtrain, w)+b - ytrain,
                np.dot(Xtrain, w)+b - ytrain )/len(TRAIN)
test_error = np.dot( np.dot(Xtest, w)+b - ytest,
                np.dot(Xtest, w)+b - ytest )/len(TEST)

print('Train error = ',train_error)
print('Test error = ',test_error)
```

```python
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector
idx = np.random.permutation(1000)
TRAIN = idx[0:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN,:]
Xtrain_avg = np.mean(Xtrain, axis=0)
Xtrain = Xtrain - Xtrain_avg

# Solve for argmin_w ||Xtrain*w - ytrain||_2
w = np.linalg.solve( np.dot(Xtrain.T, Xtrain),
                     np.dot(Xtrain.T, ytrain) )
b = np.mean(ytrain)

ytest = y[TEST]
Xtest = X[TEST,:]
Xtest_avg = np.mean(Xtest, axis=0)
Xtest = Xtest - Xtest_avg

train_error = np.dot( np.dot(Xtrain, w)+b - ytrain,
                np.dot(Xtrain, w)+b - ytrain )/len(TRAIN)
test_error = np.dot( np.dot(Xtest, w)+b - ytest,
                np.dot(Xtest, w)+b - ytest )/len(TEST)

print('Train error = ',train_error)
print('Test error = ',test_error)
```

Is the test error unbiased for this program?

```
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector
idx = np.random.permutation(1000)
TRAIN = idx[0:800]
VAL = idx[800:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN,:]
yval = y[VAL]
Xval = X[VAL,:]

err = np.zeros(50)
for d in range(1,51):
    w, b = fit(Xtrain[:,0:d], ytrain)
    yval_hat = predict(w, b, Xval[:,0:d])
    err[d-1] = np.mean((yval_hat-yval)**2)
d_best = np.argmin(err)+1

Xtot = np.concatenate((Xtrain, Xval), axis=0)
ytot = np.concatenate((ytrain, yval), axis=0)
w, b = fit(Xtot[:,0:d_best], ytot)

ytest = y[TEST]
Xtest = X[TEST,:]

ytot_hat = predict(w, b, Xtot[:,0:d_best])
tot_train_error = np.mean((ytot_hat-ytot)**2)
ytest_hat = predict(w, b, Xtest[:,0:d_best])
test_error = np.mean((ytest_hat-ytest)**2)

print('Train error = ',train_error)
print('Test error = ',test_error)
```

```
def fit(Xin, Yin):
    mu = np.mean(Xin, axis=0)
    Xin = Xin - mu
    w = np.linalg.solve( np.dot(Xin.T, Xin),
                         np.dot(Xin.T, Yin) )
    b = np.mean(Yin) - np.dot(w, mu)
    return w, b

def predict(w, b, Xin):
    return np.dot(Xin, w)+b
```

Simple Variable Selection
LASSO: Sparse Regression

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 9, 2016

Sparsity

$$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2$$

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

- Very useful for many tasks, e.g.,
  - **Efficiency**: If size(**w**) = 100 Billion, 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

-

Sparsity

$$\widehat{w}_{LS} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

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

- Very useful for many tasks, e.g.,
  - **Efficiency**: If size(**w**) = 100 Billion, 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?

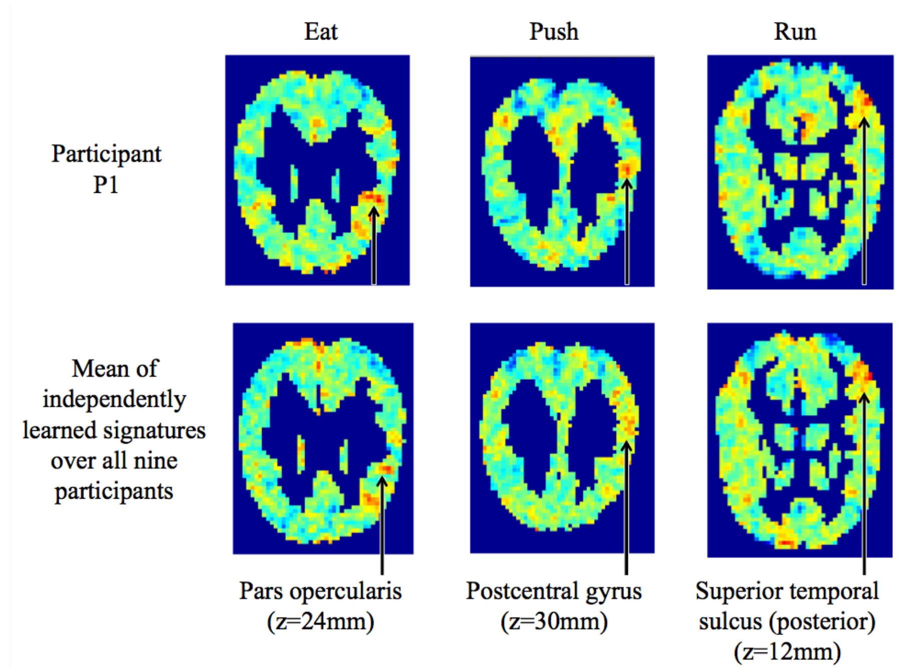


Figure from Tom Mitchell

Sparsity

$$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

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

- Very useful for many tasks, e.g.,
  - **Efficiency**: If size(**w**) = 100 Billion, 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?

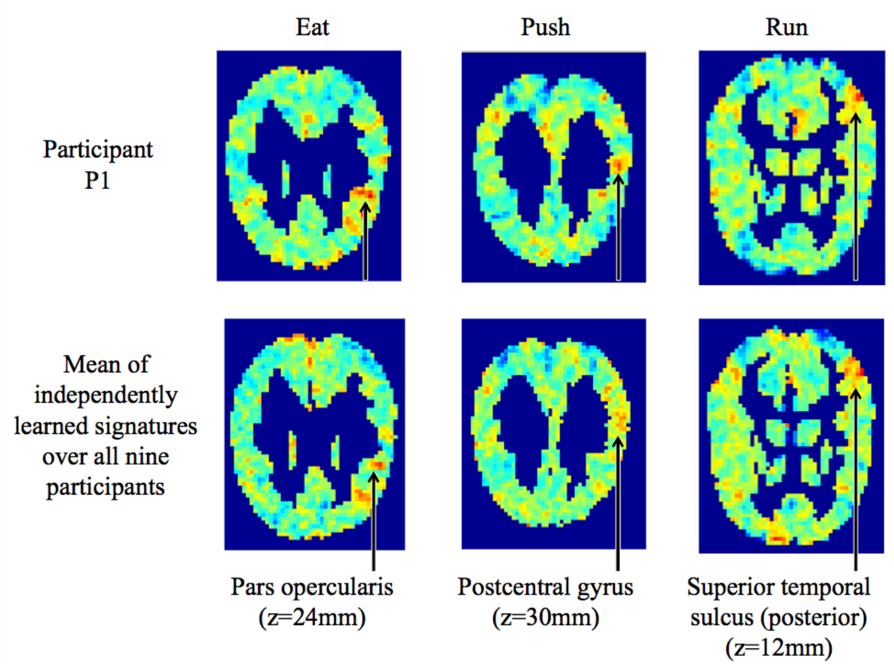


Figure from Tom Mitchell

- How do we find "best" subset among all possible?

Greedy model selection algorithm

- Pick a dictionary of features
  - e.g., cosines of random inner products
- Greedy heuristic:
  - Start from empty (or simple) set of features $F_0 = \emptyset$
  - Run learning algorithm for current set of features F_t
    - Obtain weights for these features
  - Select **next best feature** $\mathbf{h_i(x)^*}$
    - e.g., $h_j(x)$ that results in lowest training error learner when using $F_t + \{h_j(x)^*\}$
  - $F_{t+1} \leftarrow F_t + \{h_i(x)^*\}$
  - Recurse

Greedy model selection

- Applicable in many other settings:
  - Considered later in the course:
    - Logistic regression: Selecting features (basis functions)
    - Naïve Bayes: Selecting (independent) features $P(X_i|Y)$
    - Decision trees: Selecting leaves to expand
- Only a heuristic!
  - **Finding the best set of k features is computationally intractable!**
  - Sometimes you can prove something strong about it…

When do we stop???

- Greedy heuristic:
  - …
  - Select **next best feature $\mathbf{X_i^*}$**
    - E.g. $h_j(x)$ that results in lowest training error learner when using $F_t + \{h_j(x)^*\}$
  - Recurse

When do you stop???

- When training error is low enough?
- When test set error is low enough?
- Using cross validation?

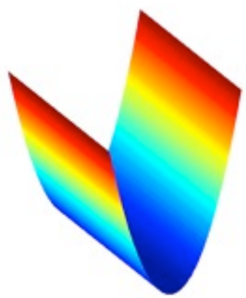
Is there a more principled approach?

Recall Ridge Regression

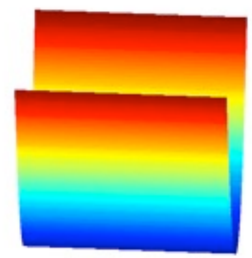
- Ridge Regression objective:

$$\widehat{w}_{ridge} = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

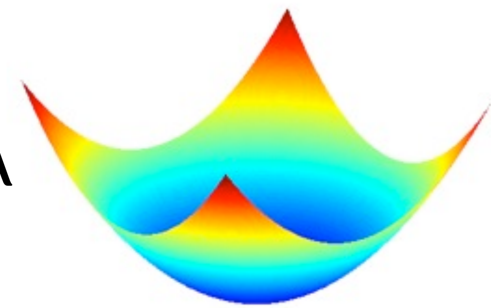
+



+ ... +



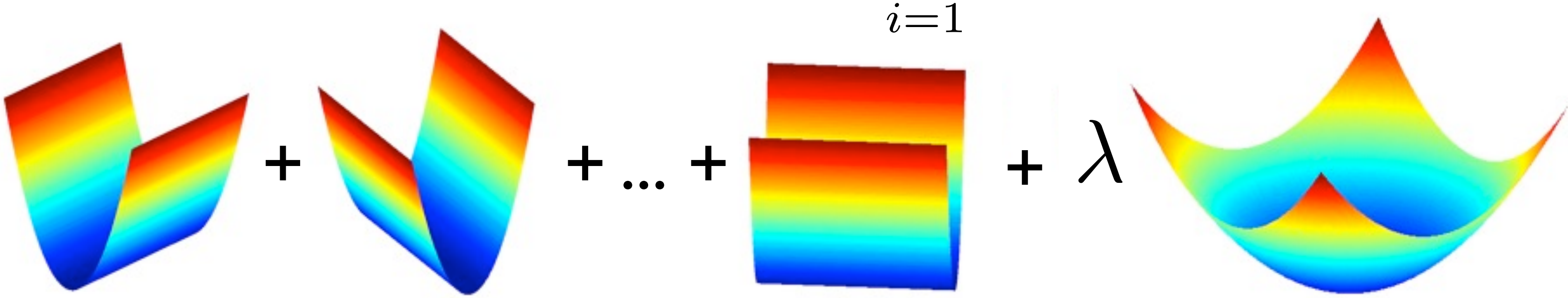
+ λ



Ridge vs. Lasso Regression

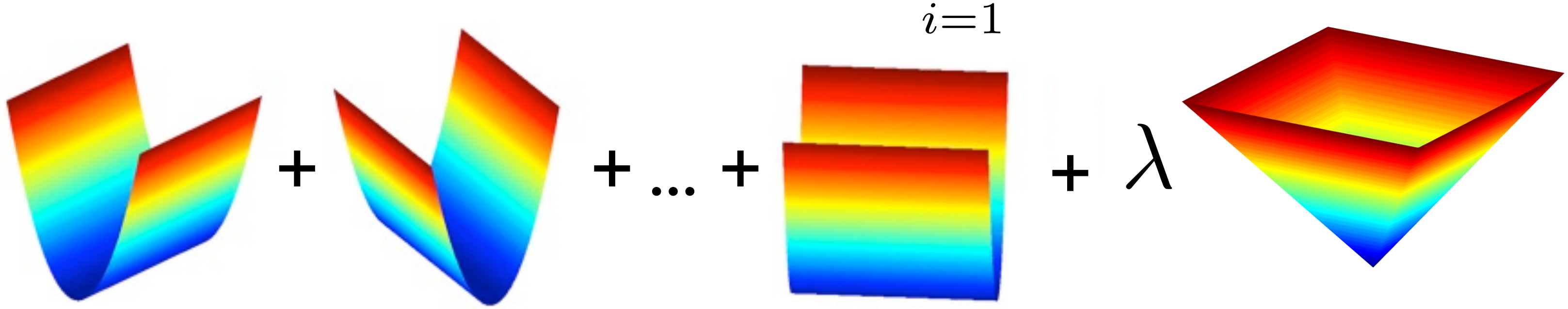
- Ridge Regression objective:

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$



- Lasso objective:

$$\widehat{w}_{lasso} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_1$$



Penalized Least Squares

$$\text{Ridge} : r(w) = ||w||_2^2 \qquad \text{Lasso} : r(w) = ||w||_1$$

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 + \lambda r(w)$$

Penalized Least Squares

$$\text{Ridge} : r(w) = ||w||_2^2 \qquad \text{Lasso} : r(w) = ||w||_1$$

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 + \lambda r(w)$$

For any $\lambda \geq 0$ for which $\widehat{w}_r$ achieves the minimum, there exists a $\nu \geq 0$ such that

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 \qquad \text{subject to } r(w) \leq \nu$$

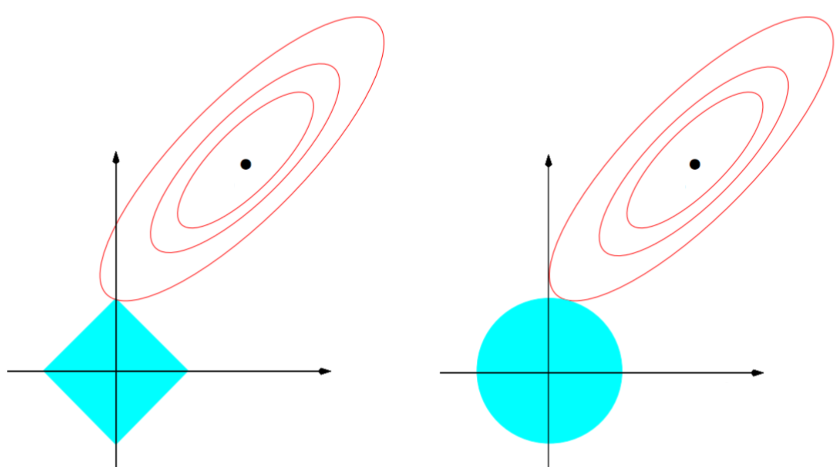
Penalized Least Squares

$$\text{Ridge} : r(w) = ||w||_2^2 \qquad \text{Lasso} : r(w) = ||w||_1$$

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 + \lambda r(w)$$

For any $\lambda \geq 0$ for which $\widehat{w}_r$ achieves the minimum, there exists a $\nu \geq 0$ such that

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 \qquad \text{subject to } r(w) \leq \nu$$



Optimizing the LASSO Objective

- LASSO solution:

$$\widehat{w}_{lasso}, \widehat{b}_{lasso} = \arg\min_{w,b} \sum_{i=1}^{n} \left(y_i - (x_i^T w + b)\right)^2 + \lambda ||w||_1$$

$$\widehat{b}_{lasso} = \arg\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - x_i^T \widehat{w}_{lasso})\right)$$

Optimizing the LASSO Objective

- LASSO solution:

$$\widehat{w}_{lasso}, \widehat{b}_{lasso} = \arg\min_{w,b} \sum_{i=1}^{n} \left(y_i - (x_i^T w + b)\right)^2 + \lambda ||w||_1$$

$$\widehat{b}_{lasso} = \arg\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - x_i^T \widehat{w}_{lasso})\right)$$

So as usual, preprocess to make sure that $\frac{1}{n}\sum_{i=1}^{n} y_i = 0, \frac{1}{n}\sum_{i=1}^{n} x_i = \mathbf{0}$

so we don't have to worry about an offset.

Optimizing the LASSO Objective

- LASSO solution:

$$\widehat{w}_{lasso}, \widehat{b}_{lasso} = \arg\min_{w,b} \sum_{i=1}^{n} \left(y_i - (x_i^T w + b)\right)^2 + \lambda ||w||_1$$

$$\widehat{b}_{lasso} = \arg\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - x_i^T \widehat{w}_{lasso})\right)$$

So as usual, preprocess to make sure that $\frac{1}{n}\sum_{i=1}^{n} y_i = 0, \frac{1}{n}\sum_{i=1}^{n} x_i = \mathbf{0}$

so we don't have to worry about an offset.

$$\widehat{w}_{lasso} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_1$$

How do we solve this?

Coordinate Descent

- Given a function, we want to find minimum

- Often, it is easy to find minimum along a single coordinate:

- 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

Fix any $j \in \{1, \ldots, d\}$

$$\sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_1 = \sum_{i=1}^{n} \left(y_i - \sum_{k=1}^{d} x_{i,k}\, w_k \right)^2 + \lambda \sum_{k=1}^{d} |w_k|$$

$$= \sum_{i=1}^{n} \left(\left(y_i - \sum_{k \neq j} x_{i,k}\, w_k \right) - x_{i,j}\, w_j \right)^2 + \lambda \sum_{k \neq j} |w_k| + \lambda |w_j|$$

Optimizing LASSO Objective One Coordinate at a Time

Fix any $j \in \{1,\dots,d\}$

$$\sum_{i=1}^n \left(y_i - x_i^T w\right)^2 + \lambda ||w||_1 = \sum_{i=1}^n \left(y_i - \sum_{k=1}^d x_{i,k}\, w_k \right)^2 + \lambda \sum_{k=1}^d |w_k|$$

$$= \sum_{i=1}^n \left(\left(y_i - \sum_{k \neq j} x_{i,k}\, w_k \right) - x_{i,j}\, w_j \right)^2 + \lambda \sum_{k \neq j} |w_k| + \lambda |w_j|$$

Initialize $\widehat{w}_k = 0$ for all $k \in \{1,\dots,d\}$

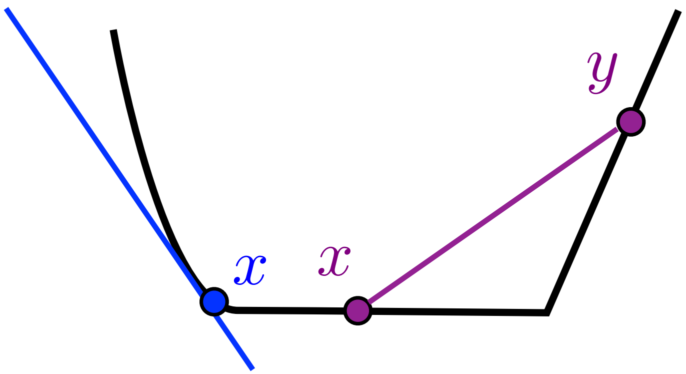
Loop over $j \in \{1,\dots,n\}$:

$$r_i^{(j)} = y_i - \sum_{k \neq j} x_{i,j} \widehat{w}_k$$

$$\widehat{w}_j = \arg\min_{w_j} \sum_{i=1}^n \left(r_i^{(j)} - x_{i,j}\, w_j \right)^2 + \lambda |w_j|$$

Convex Functions

- Equivalent definitions of convexity:



f convex:

$$f(\lambda x + (1-\lambda) y) \leq \lambda f(x) + (1-\lambda) f(y) \qquad \forall x, y, \lambda \in [0,1]$$

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \qquad \forall x, y$$

- **Gradients** lower bound convex functions and are unique at **x** iff function differentiable at **x**
- **Subgradients** generalize gradients to non-differentiable points:
  - Any supporting hyperplane at **x** that lower bounds entire function

$$g \text{ is a subgradient at } x \text{ if } f(y) \geq f(x) + g^T (y - x)$$

Taking the Subgradient

$$\widehat{w}_j = \arg\min_{w_j} \sum_{i=1}^n \left(r_i^{(j)} - x_{i,j}\, w_j \right)^2 + \lambda |w_j|$$

$$g \text{ is a subgradient at } x \text{ if } f(y) \geq f(x) + g^T(y - x)$$

- Convex function is minimized at w if 0 is a sub-gradient at w.

$$\partial_{w_j} |w_j| =$$

$$\partial_{w_j} \sum_{i=1}^n \left(r_i^{(j)} - x_{i,j}\, w_j \right)^2 =$$

Setting Subgradient to 0

$$\partial_{w_j}\left(\sum_{i=1}^{n}\left(r_i^{(j)} - x_{i,j}\, w_j\right)^2 + \lambda|w_j|\right) = \begin{cases} a_j w_j - c_j - \lambda & \text{if } w_j < 0 \\ [-c_j - \lambda, -c_j + \lambda] & \text{if } w_j = 0 \\ a_j w_j - c_j + \lambda & \text{if } w_j > 0 \end{cases}$$

$$a_j = (\sum_{i=1}^{n} x_{i,j}^2) \qquad c_j = 2(\sum_{i=1}^{n} r_i^{(j)} x_{i,j})$$

Setting Subgradient to 0

$$\partial_{w_j}\left(\sum_{i=1}^{n}\left(r_i^{(j)} - x_{i,j}\,w_j\right)^2 + \lambda|w_j|\right) = \begin{cases} a_j w_j - c_j - \lambda & \text{if } w_j < 0 \\ [-c_j - \lambda, -c_j + \lambda] & \text{if } w_j = 0 \\ a_j w_j - c_j + \lambda & \text{if } w_j > 0 \end{cases}$$

$$a_j = (\sum_{i=1}^{n} x_{i,j}^2) \qquad c_j = 2(\sum_{i=1}^{n} r_i^{(j)} x_{i,j})$$

$$\widehat{w}_j = \arg\min_{w_j} \sum_{i=1}^{n}\left(r_i^{(j)} - x_{i,j}\,w_j\right)^2 + \lambda|w_j|$$

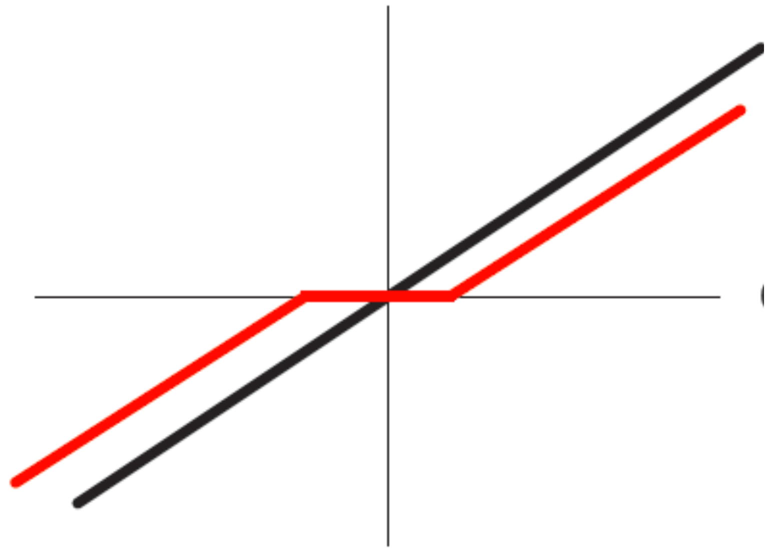
w is a minimum if
0 is a sub-gradient at w

$$\widehat{w}_j = \begin{cases} (c_j + \lambda)/a_j & \text{if } c_j < -\lambda \\ 0 & \text{if } |c_j| \leq \lambda \\ (c_j - \lambda)/a_j & \text{if } c_j > \lambda \end{cases}$$

Soft Thresholding

$$\widehat{w}_j = \begin{cases} (c_j + \lambda)/a_j & \text{if } c_j < -\lambda \\ 0 & \text{if } |c_j| \leq \lambda \\ (c_j - \lambda)/a_j & \text{if } c_j > \lambda \end{cases}$$

$$a_j = \sum_{i=1}^{n} x_{i,j}^2 \qquad\qquad c_j = 2\sum_{i=1}^{n} \Big(y_i - \sum_{k \neq j} x_{i,k}\, w_k \Big) x_{i,j}$$



Coordinate Descent for LASSO (aka Shooting Algorithm)

- Repeat until convergence (initialize w=0)
  - Pick a coordinate *l* at (random or sequentially)
    - Set:

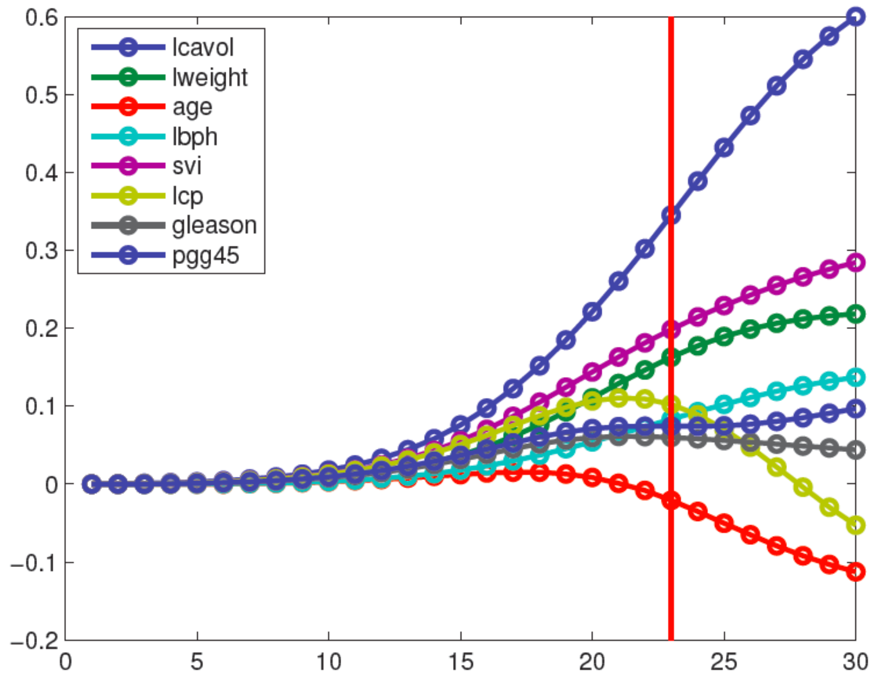
$$\widehat{w}_j = \begin{cases} (c_j + \lambda)/a_j & \text{if } c_j < -\lambda \\ 0 & \text{if } |c_j| \leq \lambda \\ (c_j - \lambda)/a_j & \text{if } c_j > \lambda \end{cases}$$

    - Where:

$$a_j = \sum_{i=1}^{n} x_{i,j}^2 \qquad c_j = 2\sum_{i=1}^{n}\Big(y_i - \sum_{k \neq j} x_{i,k}\,\widehat{w}_k\Big)x_{i,j}$$

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

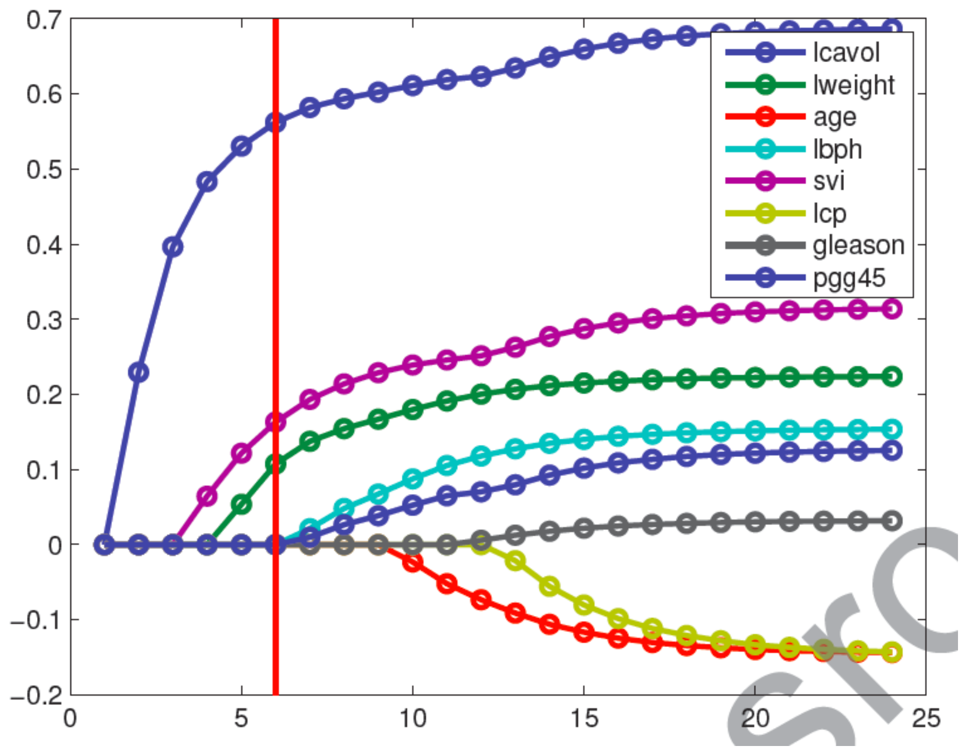
Recall: *Ridge Coefficient Path*



From
Kevin Murphy
textbook

- Typical approach: select λ using cross validation

Now: *LASSO Coefficient Path*



From
Kevin Murphy
textbook

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 regression
  - 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 simple approach for solving LASSO