Data Mining
W4240 Sections 001, 003/004

Lauren A. Hannah

Columbia University, Department of Statistics

October 27, 2014

Outline

Outline

Subset Selection

Pick the best k ($\leq p$) covariates to use in linear regression

Subset Selection

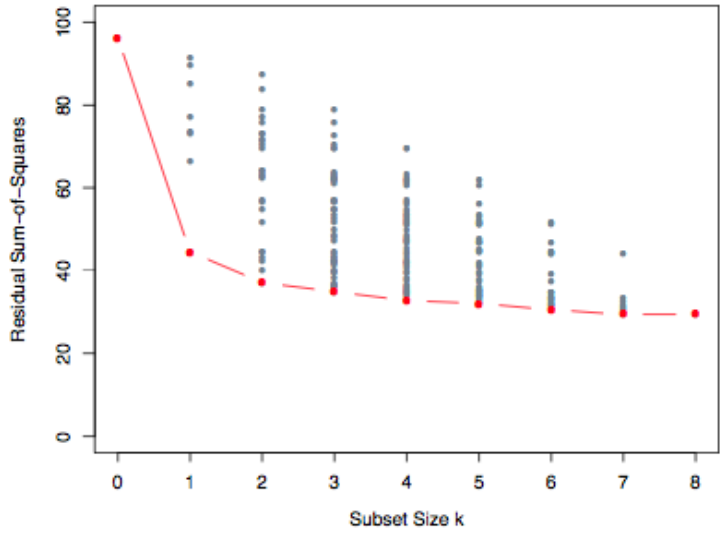
Pick the best k ($\leq p$) covariates to use in linear regression

Why?

- *Predictive Accuracy:* Linear least squares estimator has low bias, high variance. Reduce number of covariates, get a bit more bias but much less variance.
- *Interpretability:* Which variables matter? Which do not? Interpretability allows your model to say something about the data vs. just giving a prediction.

Subset Selection

How to pick the best k ($\leq p$) covariates for linear regression?



Best Subset Selection:

- enumerate possible subsets in a smart way for each k
- for each k, select subset that minimizes RSS
- pick best k: cross-validation or other model selection methods
- good method for $p < 30$ or 40

Model Selection

Rather than enumerating all possible subsets, model selection can be done in a few ways

- Cross-validation:
  - possibly more accurate
  - no need for asymptotic approximations (is n large enough to justify asymptotics?)
  - more flexible (can be used for things other than MLE)
- Model selection criteria (AIC, BIC, etc.):
  - often easy to compute
  - theoretically justifiable

Model Selection

Rather than enumerating all possible subsets, model selection can be done in a few ways

- Cross-validation:
  - possibly more accurate
  - no need for asymptotic approximations (is n large enough to justify asymptotics?)
  - more flexible (can be used for things other than MLE)
- Model selection criteria (AIC, BIC, etc.):
  - often easy to compute
  - theoretically justifiable

- **Today we will discuss shrinkage, a similar approach:**
  - can sometimes remove covariates (perform subset selection)
  - more generally, reduces estimator variance and complexity ($\Rightarrow$ better predictive performance)

Outline

Multivariate Linear Regression

Recall:

The *design matrix* is an $n \times p$ matrix:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & \dots & x_{1,p} \\ x_{2,1} & \dots & x_{2,p} \\ \vdots & & \vdots \\ x_{n,1} & \dots & x_{n,p} \end{bmatrix}$$

The *response vector* is an $n \times 1$ column vector:

$$\mathbf{y} = [y_1 \ y_2 \ \dots \ y_n]^T$$

The *parameter vector* is a $p \times 1$ column vector (as before):

$$\beta = [\beta_1 \ \dots \ \beta_p]^T$$

Today, we will center and scale the data; not use intercept

Reminder: Centering/Scaling for Linear Regression

Scaling: we do not *need* an intercept

- rescale data:

$$\tilde{X}^T = \left(\frac{X_1 - \bar{X}_1}{\hat{\sigma}_1}, \frac{X_2 - \bar{X}_2}{\hat{\sigma}_2}, \ldots, \frac{X_p - \bar{X}_p}{\hat{\sigma}_p} \right),$$

$$\tilde{Y} = \frac{Y - \bar{Y}}{\hat{\sigma}_Y}$$

- all elements of rescaled data have mean 0... so no need for intercept
- in `R`, use the function `scale()` (ex: `> x.bar <- scale(x)`)
- then fit linear function

Note: we need to rescale both X and Y! (Why is this?)

Multivariate Linear Regression

Recall least squares regression:

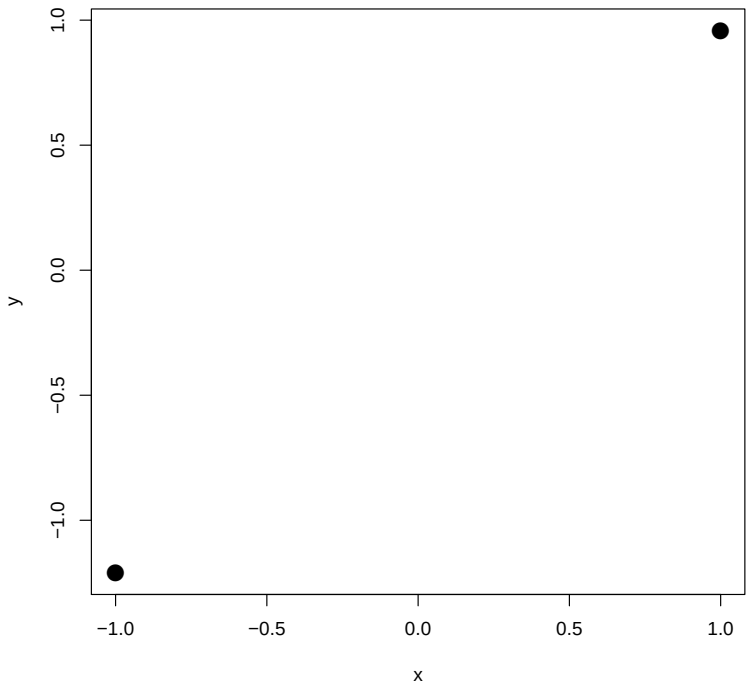
$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2$$
$$= \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}$$

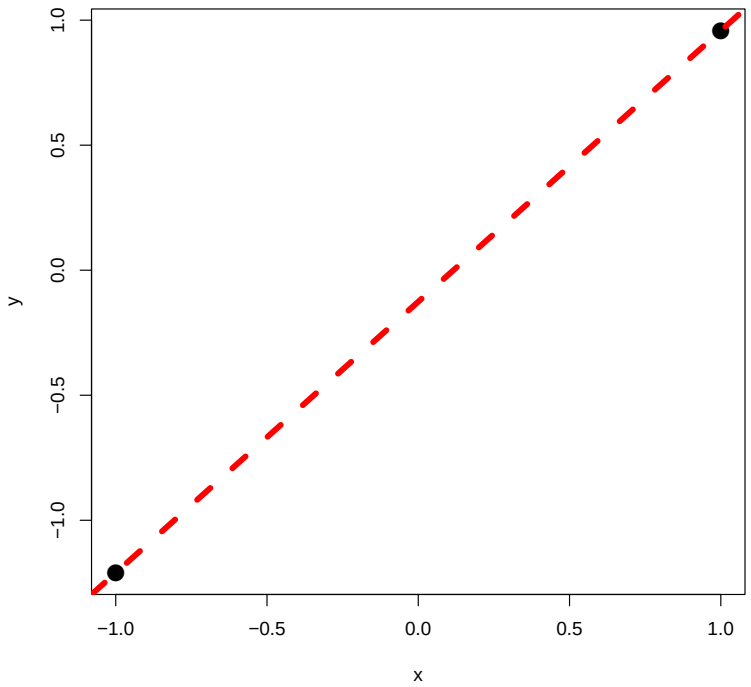
Problem: $(\mathbf{X}^T\mathbf{X})^{-1}$ does not always exist

- need at least p observations
- (may even need more if points are on a line)

Multivariate Linear Regression

Example: $p = 1$, have 2 points



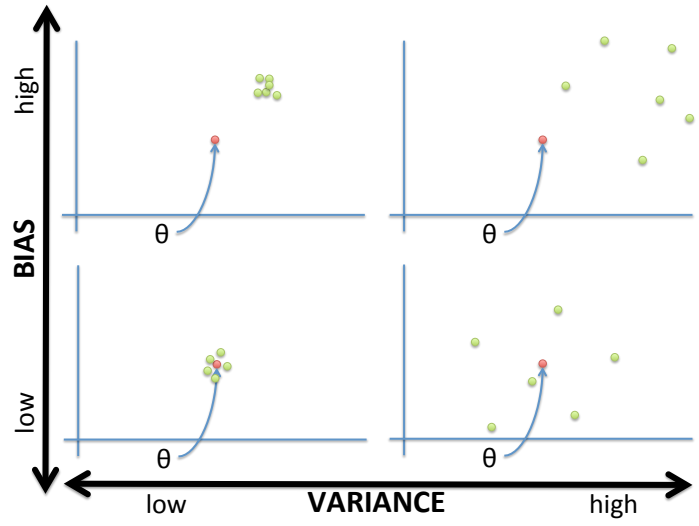


- Have $p + 1$ or fewer points, line goes through all (or p with mean 0 data)
- Have more than $p + 1$ (but still close to that number), line goes *close* to all points

Multivariate Linear Regression

Why linear regression:

- has few parameters to estimate (p)
- really restrictive model–low variance, higher bias



- *should be good for data with few observations, large number of covariates...*
- **... but we can't use it in this situation**

Multivariate Linear Regression and subset selection

Idea: if we have a large number of covariates compared to observations, say $n < 2p$, **best to estimate most coefficients as 0!**

- not enough info to determine all coefficients
- try to estimate ones with strong signal
- set everything else to 0 (or close)

Coefficients of 0 may not be a bad assumption...

If we have 1,000s of coefficients, are they all equally important?

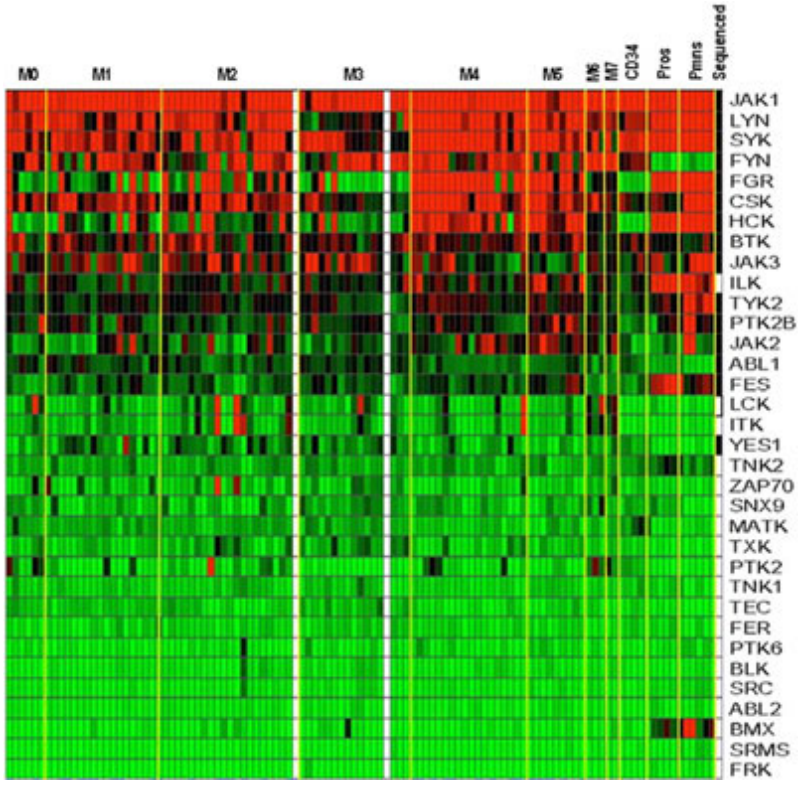
Gene Expression

Example: microarray gene expression data

- gene expression: want to measure the level at which information in a gene is used in the synthesis of a functional gene product (usually protein)
- can use gene expression data to determine subtype of cancer (e.g. which *type* of Lymphoma B?) or predict recurrence, survival time, etc
- problem: thousands of genes, hundreds of patients, $p > n$!

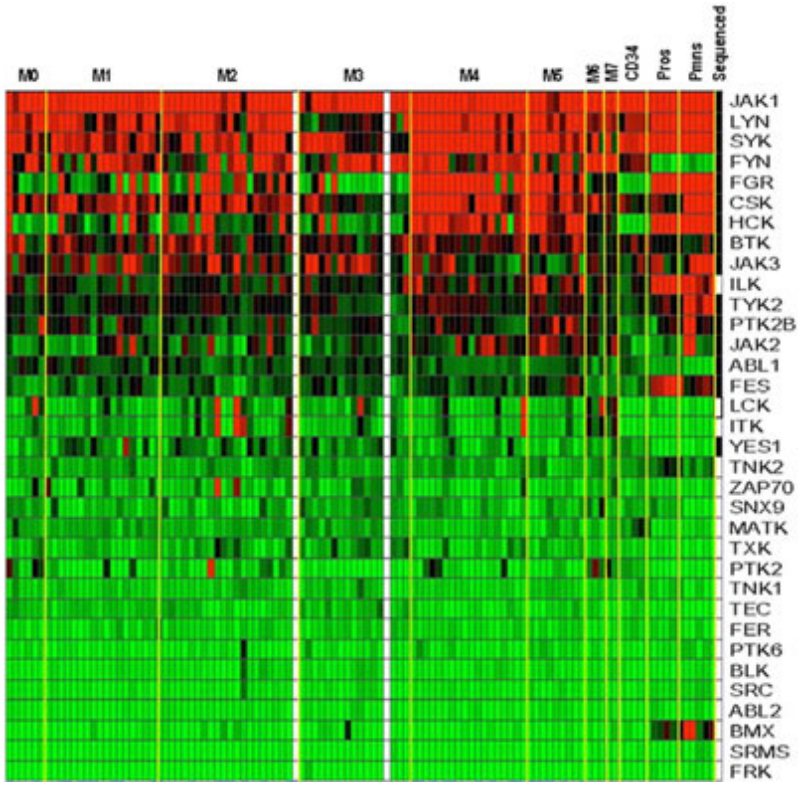
Intuition: only a handful of genes should affect outcomes

Gene Expression



- gene expression levels are continuous values
- data: observation i is gene expression levels from patient i, attached to outcome for patient (survival time)
- covariates: expression levels for p genes

Gene Expression



- collinearity: does it matter *which* gene is selected for *prediction*? No!
- overfitting: now fitting p' non-0 coefficients to n observations with $p' << n$ means less fitting of noise

Forward Stepwise Linear Regression

Forward stepwise linear regression:

- sequentially adds in predictors based on $F-$statistics (or AIC or BIC or adjusted R^2)
- can handle data with $p > n$
- ...but problems with multiple testing ($F-$statistic computed on same data again and again)
- ...and a lot of parameter bias (either 0 or much greater magnitude than it should be)
- ...and model selection can be unstable

Is there a more principled way to force (**shrink**) values to 0?

Outline

Regularized Linear Regression

Regularization (a hugely important concept):

- place a *penalty* on large values for β_1, ..., β_p (why not β_0? can always easily estimate mean)
- add this penalty to the objective function
- solve for $\hat{\beta}$!

New objective function:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \text{penalty}(\beta_j)$$

λ acts as a weight on penalty: low values mean few coefficients near 0, high values mean many coefficients near 0

Regularized Linear Regression

New objective function:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \text{penalty}(\beta_j)$$

When and why can this be a better predictor?

- It adds bias (we are not fitting the best β to the data)
- ...but it *greatly* reduces variance

Note 1: the data *need* to be centered and scaled. Why?
Note 2: will this always be a better predictor? Why not?

Regularized Linear Regression

Regularization: what is a good penalty function?

Same as penalties used to fit errors:

- Ridge regression (squared penalty):

$$\hat{\beta}^{Ridge} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- Lasso regression (absolute value penalty):

$$\hat{\beta}^{Lasso} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Regularized Linear Regression

Ridge and Lasso regression,

$$\hat{\beta}^{Ridge} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

$$\hat{\beta}^{Lasso} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

produce estimators with different properties

Outline

Ridge Regression

Geometrically, what does a squared penalty do?

$$\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

is equivalent to

$$\min_{\beta,\gamma} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \gamma$$

$$\text{subject to} : \lambda \sum_{j=1}^{p} \beta_j^2 \leq \gamma$$

Ridge Regression

To find $\hat{\beta}^{Ridge}$, solve:

$$\min_{\beta,\gamma} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \gamma$$

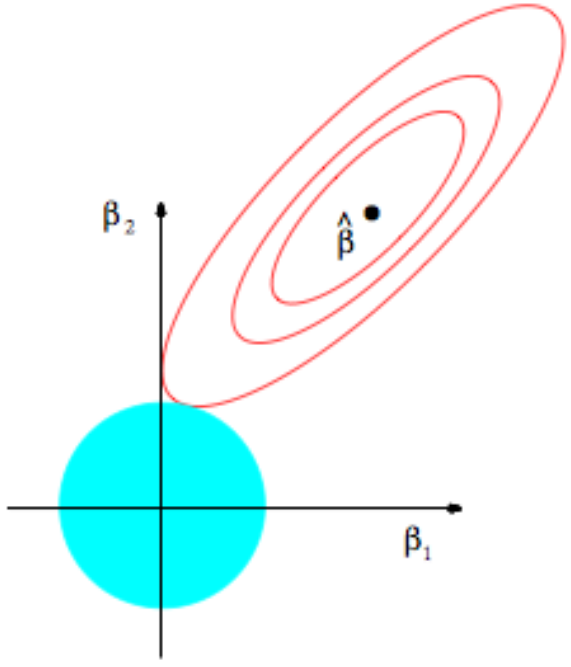
$$\text{subject to : } \lambda \sum_{j=1}^{p} \beta_j^2 \leq \gamma$$

$\sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2$: residual sum of squares

$\lambda \sum_{j=1}^{p} \beta_j^2 \leq \gamma$: coefficients restricted to sphere with radius $\sqrt{\frac{\gamma}{\lambda}}$

Ridge Regression

To find $\hat{\beta}^{Ridge}$,



$\sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2$: residual sum of squares

$\lambda \sum_{j=1}^{p} \beta_j^2 \leq \gamma$: coefficients restricted sphere with radius $\sqrt{\frac{\gamma}{\lambda}}$

Ridge Regression

To find $\hat{\beta}^{Ridge}$, solve:

$$\min_{\beta,\gamma} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \gamma$$

$$\text{subject to : } \lambda \sum_{j=1}^{p} \beta_j^2 \leq \gamma$$

Result:

- $\hat{\beta}$ has may values *near* 0, but not exactly 0

Ridge Regression

It turns out that we have a closed form solution for $\hat{\beta}^{Ridge}$:

$$0 = \nabla_\beta \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2,$$

$$\hat{\beta}^{Ridge} = \left(\mathbf{X}^T\mathbf{X} + \lambda I_p\right)^{-1} \mathbf{X}^T\mathbf{y}$$

and the matrix $\left(\mathbf{X}^T\mathbf{X} + \lambda I_p\right)^{-1}$ *always* exists

Outline

Lasso Regression

Geometrically, what does an absolute value penalty do?

$$\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

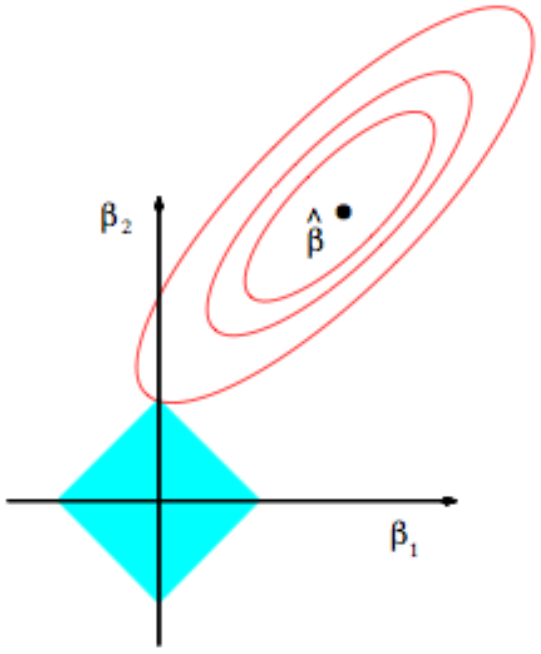
is equivalent to

$$\min_{\beta,\gamma} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \gamma$$

$$\text{subject to} : \lambda \sum_{j=1}^{p} |\beta_j| \leq \gamma$$

Lasso Regression

To find $\hat{\beta}^{Lasso}$,

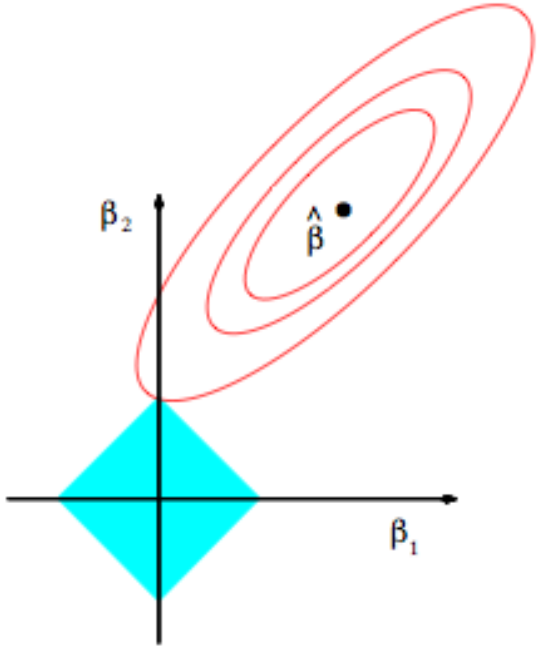


$\sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2$: residual sum of squares

$\lambda \sum_{j=1}^{p} |\beta_j| \leq \gamma$: coefficients restricted square with radius $\frac{\gamma}{\lambda}$

Lasso Regression

To find $\hat{\beta}^{Lasso}$,



Most of the time, the residual sum of squares is projected onto a vertex

- forces many coefficient values to *exactly* 0

Lasso Regression

It turns out that we don't have a closed form solution for $\hat{\beta}^{Lasso}$:

$$0 = \nabla_\beta \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=0}^{p} |\beta_j|$$

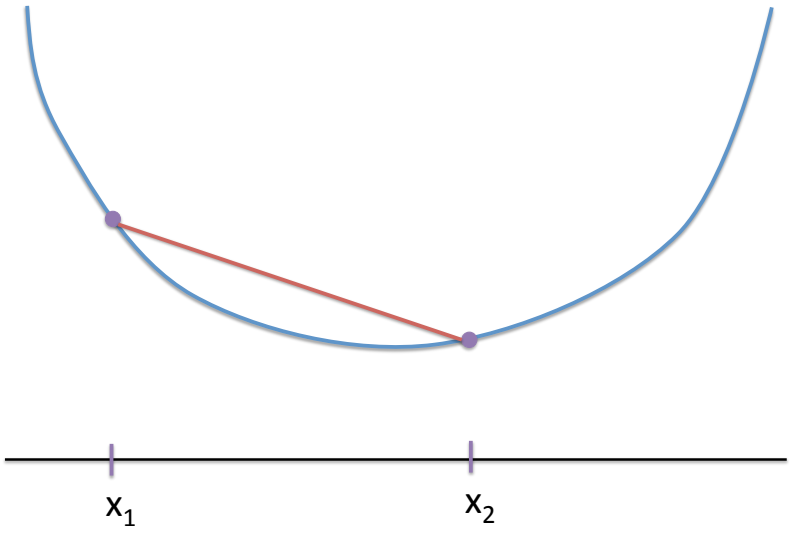
However, there are many `R` functions that will solve this problem efficiently even for large n and p through *convex optimization*

Outline

Convex Optimization

A function is *convex* if

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$



Can minimize with hill-climbing algorithms and you are *guaranteed* to get optimal decision

Convex Optimization

Convex optimization problem:

$$\begin{aligned} &\min_x f_0(x) \\ \text{subject to } &f_i(x) \leq 0 \\ &Ax = b \end{aligned}$$

Objective function: $f_0(x)$ is convex

Constraints: $f_i(x)$ is convex, $Ax = b$ is affine (linear)

- $f_i(x) = x^2$ is convex
- $f_i(x) = |x|$ is convex
- $f_i(x) = \mathbf{1}_{\{x \notin \{\ldots,-1,0,1,2,\ldots\}\}}$ is not convex
- $f_i(x) = \mathrm{card}(x)$ (number of non-zero elements) is not convex

Convex Optimization

Subset selection:

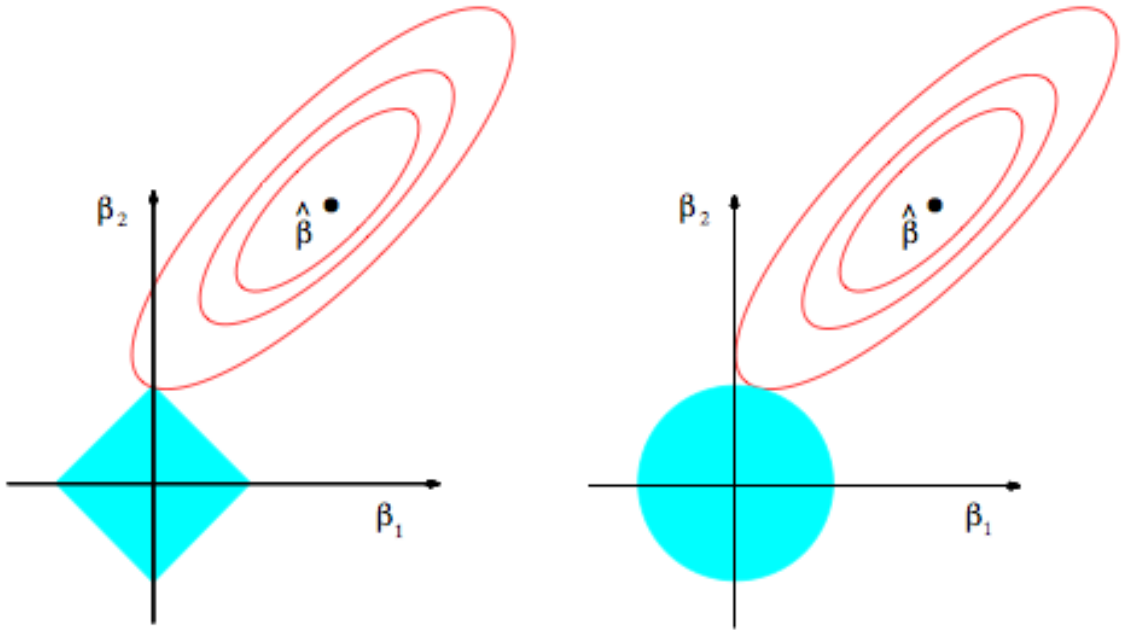
$$\min_{\beta} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \, \mathrm{card}(\beta)$$

In general,

- if penalty is norm $||\beta||_p = \left(\sum_j \beta_j^p\right)^{\frac{1}{p}}$ with $p \geq 1$, then problem is convex
- if penalty is norm $||\beta||_p$ with $p < 1$, then problem is not convex
- subset selection is not convex

Outline

Comparing Ridge and Lasso



Ridge is *stable* to small changes in $\mathbf{X}$ and $\mathbf{y}$; Lasso is not (might be projected onto different vertex)

Comparing Ridge and Lasso

	Ridge	**Lasso**
Objective	$\sum_{i=1}^{n}\left(y_i - \mathbf{x}_i^T\beta\right)^2 + \lambda\sum_{j=0}^{p}\beta_j^2$	$\sum_{i=1}^{n}\left(y_i - \mathbf{x}_i^T\beta\right)^2 + \lambda\sum_{j=0}^{p}\lvert\beta_j\rvert$
Estimator	$\left(\mathbf{X}^T\mathbf{X} + \lambda I\right)^{-1}\mathbf{X}^T\mathbf{y}$	not closed form
Coefs	most close to 0	most exactly 0
Stability	robust to changes in $\mathbf{X}$, $\mathbf{y}$	not robust to changes in $\mathbf{X}$, $\mathbf{y}$

Regularized linear regression is fantastic for low signal datasets or those with $p >> n$

- Ridge: good when many coefficients affect value but not large (gene expression)
- Lasso: good when you want an *interpretable* estimator

Choosing λ

Both Ridge and Lasso have a tunable parameter, λ

- use leave-one-out cross validation to find best λ

$$\hat{\lambda} = \arg\min_{\lambda} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \hat{\beta}_{-i,\lambda} \right)^2$$

- this is really slow for large datasets
- have closed form approximation called *generalized cross validation*
- `R` functions implement this to automatically choose λ for you

Outline

Regularized Linear Regression in R

Read in data about prostate cancer:

```
> prostate <- read.csv("Prostate.csv")
> names(prostate)
 [1] "lcavol"  "lweight" "age"     "lbph"
 [5] "svi"     "lcp"     "gleason" "pgg45"
 [9] "lpsa"    "train"
```

Predictors (columns 1–8):
lcavol lweight age lbph svi lcp gleason pgg45

Response (column 9):
lpsa

Training/testing indicator (column 10):
train

Regularized Linear Regression in R

There are 96 observations and 8 covariates

First, we center and scale the data (mean 0, var $= n$)

```
> xp <- scale(prostate[,1:9])
> prostate[1:2,]
      lcavol  lweight age      lbph svi       lcp gleason
1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6
2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6
  pgg45       lpsa train
1     0 -0.4307829  TRUE
2     0 -0.1625189  TRUE
> xp[1:2,]
        lcavol    lweight        age      lbph        svi
[1,] -1.637356 -2.0062118 -1.8624260 -1.024706 -0.5229409
[2,] -1.988980 -0.7220088 -0.7878962 -1.024706 -0.5229409
            lcp   gleason      pgg45      lpsa
[1,] -0.8631712 -1.042157 -0.8644665 -2.520226
[2,] -0.8631712 -1.042157 -0.8644665 -2.287827
```

Regularized Linear Regression in R

Now break the data into training and testing sets:

```
> xp.train <- xp[(prostate$train==TRUE),]
> xp.test <- xp[(prostate$train==FALSE),]
> dim(xp.train)
[1] 67  9
> dim(xp.test)
[1] 30  9
```

We have 67 training observations and 30 testing observations

Regularized Linear Regression in R

Let's begin by fitting ordinary least squares and least absolute deviation regression:

```
> xp.train.df <- data.frame(xp.train)
> names(xp.train.df)
[1] "lcavol"  "lweight" "age"     "lbph"    "svi"
[6] "lcp"     "gleason" "pgg45"   "lpsa"
> attach(xp.train.df)
> fit.ols <- lm(lpsa ~ lcavol + lweight + age + lbph + svi + lcp
+ gleason + pgg45 - 1)
```

Regularized Linear Regression in R

```
> fit.ols

Call:
lm(formula = lpsa ~ lcavol + lweight + age + lbph + svi + lcp +
    gleason + pgg45 - 1)

Coefficients:
  lcavol   lweight       age       lbph        svi        lcp
 0.58905   0.22825  -0.12455   0.18252   0.26395  -0.24848
 gleason     pgg45
-0.01566   0.22819

> y.pred.ols <- predict(fit.ols,data.frame(xp.test[,1:8]))
```

Regularized Linear Regression in R

To fit a regularized linear model, we use the package `glmnet`

- `glmnet` regresses on matrices, not data frames
- has parameter `alpha`, where `alpha` $=0$ means Ridge, `alpha` $=1$ means Lasso

$$\min_{\beta} \sum_{i=1}^{n} \left(y_i - x_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \left[(1-\alpha)\beta_j^2 + \alpha|\beta_j|\right]$$

- use the function `cv.glmnet( )` to find the right parameter for λ
- use the function `predict( )` to get a prediction

```
> library(glmnet)
> cv.fit.ridge <- cv.glmnet(xp.train[,1:8],lpsa,alpha=0)
> y.pred.ridge <- predict(cv.fit.ridge,xp.test[,1:8])
```

Regularized Linear Regression in R

```
> cv.fit.lasso <- cv.glmnet(xp.train[,1:8],lpsa,alpha=1)
> y.pred.lasso <- predict(cv.fit.lasso,xp.test[,1:8])
> # Compute MSE for test set
> c(mean((y.pred.ols-xp.test[,9])^2),
mean((y.pred.ridge-xp.test[,9])^2),
mean((y.pred.lasso-xp.test[,9])^2))
[1] 0.3891581 0.3982256 0.3726441
```

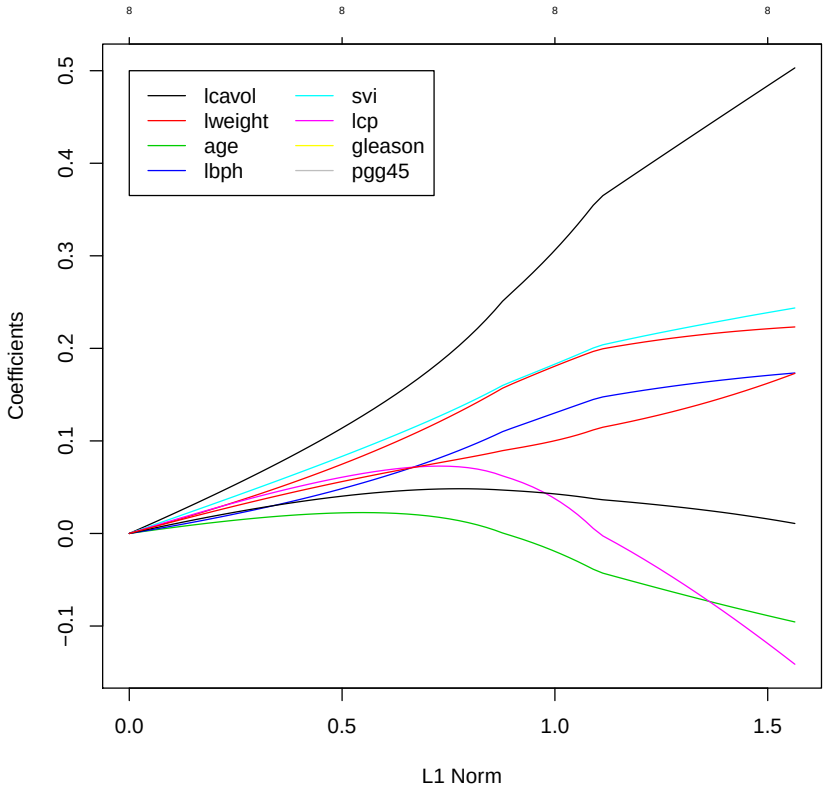
Regularized Linear Regression in R

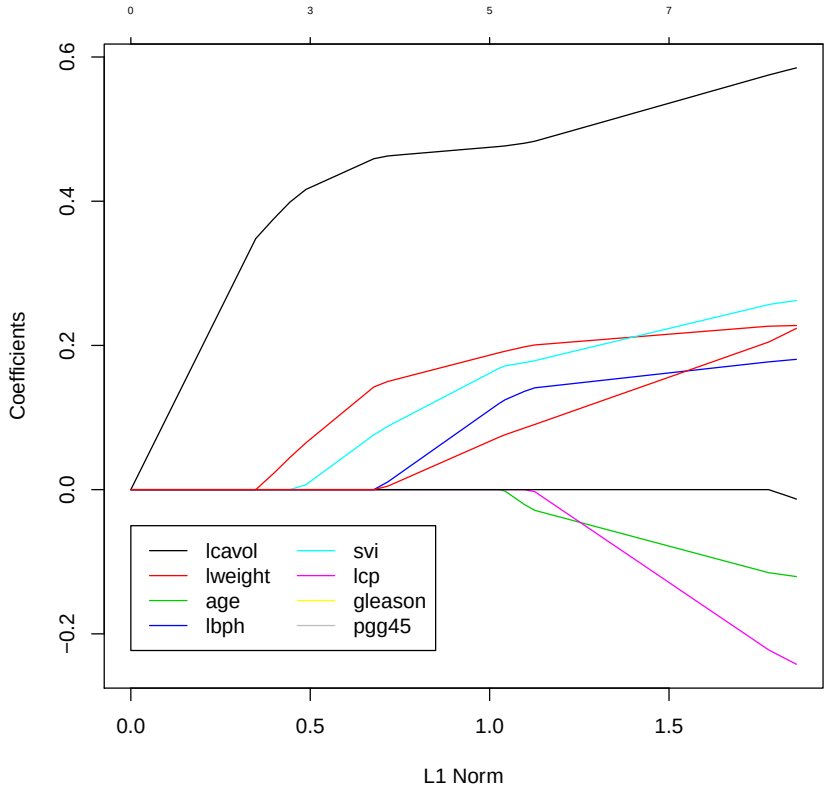
So how do the coefficients change with λ?

- can use glmnet to show these

```
 > fit.ridge <- glmnet(xp.train[,1:8],lpsa,alpha=0)
> plot(fit.ridge)
> legend(0,0.5,c("lcavol","lweight","age","lbph","svi","lcp",
"gleason", "pgg45"),col=1:8,lty=1,ncol=2)
> fit.lasso <- glmnet(xp.train[,1:8],lpsa,alpha=1)
> plot(fit.lasso)
> legend(0,-0.05,c("lcavol","lweight","age","lbph","svi","lcp",
"gleason", "pgg45"),col=1:8,lty=1,ncol=2)
```

Regularized Linear Regression in R





Regularized Linear Regression in R

What was that plot?

- L_1 norm on x-axis ($\sum |\beta_j|$)
- $\beta_{1:p}$ on y-axis

So, let's plot one coefficient...

```
> L1.norm <- function(x) sum(abs(x))
> plot(apply(fit.ridge$beta,2,L1.norm),fit.ridge$beta[1,],type="l")
> lines(apply(fit.ridge$beta,2,L1.norm),fit.ridge$beta[2,],col=2)
> lines(apply(fit.ridge$beta,2,L1.norm),fit.ridge$beta[3,],col=3)
> beta.min <- min(fit.ridge$beta)
> beta.max <- max(fit.ridge$beta)
> plot(apply(fit.ridge$beta,2,L1.norm),fit.ridge$beta[1,],type="l",
  ylim=c(beta.min,beta.max))
```

Regularized Linear Regression in R

Figuring out the coefficients for the optimal models:

```
> cv.fit.ridge$lambda.min
[1] 0.1006497
> which(cv.fit.ridge$lambda == cv.fit.ridge$lambda.min)
[1] 97
> fit.ridge$beta[,97]
     lcavol     lweight         age        lbph
 0.48281519  0.22108049 -0.08846349  0.17067165
        svi         lcp     gleason       pgg45
 0.23836712 -0.11783893  0.01591430  0.16183770
> cv.fit.lasso$lambda.min
[1] 0.009607497
> which(cv.fit.lasso$lambda == cv.fit.lasso$lambda.min)
[1] 48
> fit.lasso$beta[,48]
    lcavol    lweight        age       lbph
 0.5605983  0.2225770 -0.1016259  0.1715339
       svi        lcp    gleason      pgg45
 0.2444484 -0.1878424  0.0000000  0.1867427
```