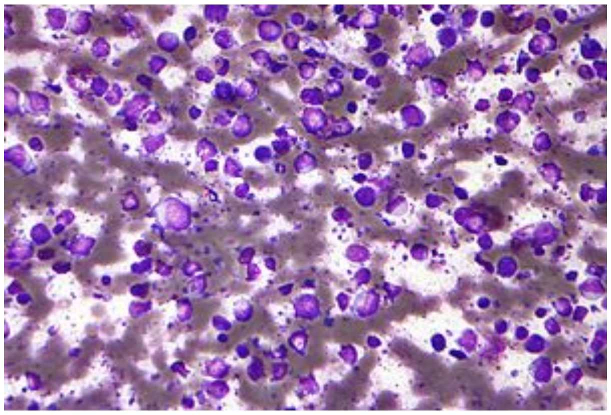
High Dimensional Analysis

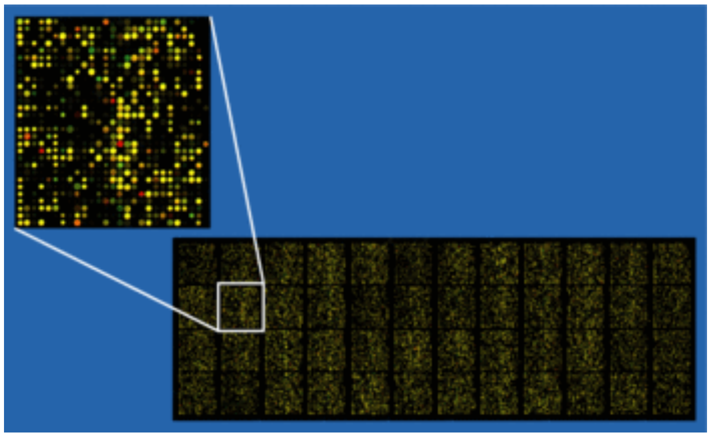
Octavio Mesner

Understanding Gene Expression and Lymphoma


Micrograph (Field stain) of a Diffuse Large B-Cell Lymphoma

From Wikipedia:

- Diffuse large B-cell lymphoma (DLBCL) is the most common lymphoid malignancy in adults
- Cancer of B cells, a type of lymphocyte
- Incidence: 7-8 cases per 100,000 people per year in the US
- occurs primarily in older individuals, with a median age of diagnosis at ~70 years
- it can occur in young adults and, in rare cases, children
- DLBCL can arise in virtually any part of the body and, depending on various factors, is often a very aggressive malignancy
- The causes of diffuse large B-cell lymphoma are not well understood
- Curable in < 50% of patients (in 2002)
- Here, we want to use gene expression or *microarray data* to predict DLBCL
- Microarrays (https://en.wikipedia.org/wiki/Microarray) simultaneously detect the expression of thousands of genes from a sample


Microarray Data

- This paper (https://idp.nature.com/authorize/casa?redirect_uri=https://www.nature.com/articles/nm0102-68&casa_token=vw6a-r_uXz8AAAAA:PBI84ZmryGuDsD_exYAi6aTypeES0fxnpFfASvUwbJ7o_Ihb5FsyW_IvxXGlrCj4UlIyydKIpRIbH4rx8A) use a supervised learning model on 6817 gene expressions (microarray) from 71 patients to distinguish between DLBCL and follicular lymphoma (FL)
- This data is on Github (https://github.com/ramhiser/datamicroarray/wiki/Shipp-%282002%29)
- In this data, each row corresponds to an individual
- Each column corresponds to gene expression using a microarray, one column gives the type of lymphoma
- **Goal**: Determine which genes are associated with with lymphoma type
- Start by looking at the data

```
load("./shipp.RData")
ls() # .RData files can include any R data structure.  The "ls()" command shows what it con
tains.
```

```
## [1] "shipp"
```

```
names(shipp)
```

```
## [1] "x" "y"
```

```
dim(shipp$x)
```

```
## [1]   77 7129
```

```
names(shipp$x)[1:10]  #first 10 var names
```

```
##  [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10"
```

```
shipp$x[1:10,1:6]
```

```
##      V1   V2   V3  V4   V5   V6
## 1  -104 -187  -26  59 -238 -258
## 2  -152 -328  -52 267 -300 -314
## 3  -158 -129   11  88 -239 -429
## 4  -124 -121   -3 -37 -210 -309
## 5   -93 -258  -36 109 -109 -272
## 6   -34 -257 -104  71 -196 -250
## 7  -251 -264  -99  31 -244 -110
## 8  -204 -293  -32 148 -327 -215
## 9  -144 -356 -194  84 -269 -235
## 10  -94 -204  -28  53 -166 -284
```

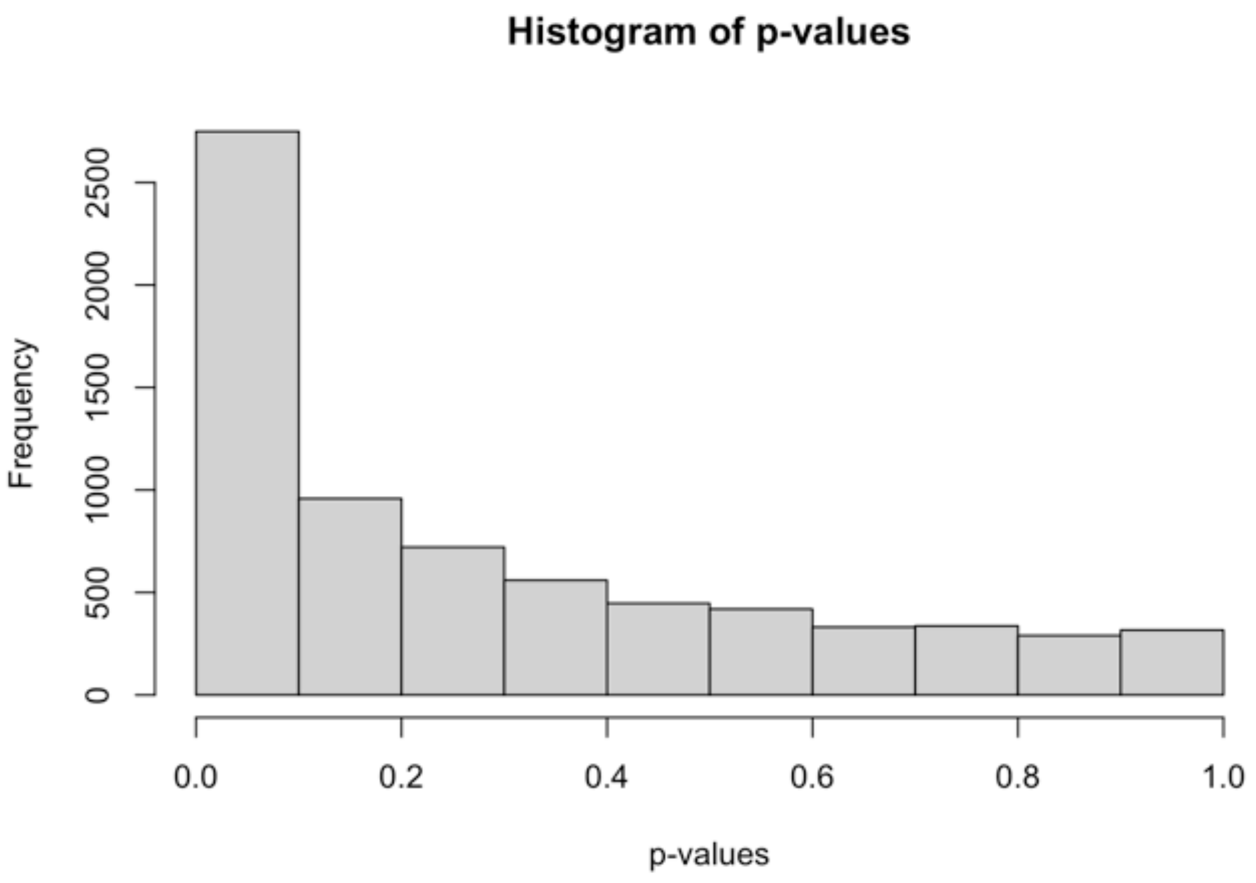
```
table(shipp$y)
```

```
## 
## DLBCL    FL 
##    58    19
```

- Note that there are far more columns than rows here
- For this, I would visually inspect data data
- It doesn’t make sense to try to look at >7k plots
- Question: Will standard regression methods will not work?
    - A: Yes
    - B: No, there would be more parameters in the model than rows of data
    - C: No, $X^T X$ is a singular matrix
    - D: Not Sure

- Question: Should we use a t-test to determine which genes are associated with DLBCL?
    - A: Yes
    - B: No
    - C: Not Sure

```
pvals <-c()
for(var in names(shipp$x)){
  pvals <- c(pvals, t.test(shipp$x[[var]] ~ shipp$y)$p.value)
}
hist(pvals, xlab = 'p-values', main='Histogram of p-values')
```



```
sum(pvals < 0.05)
```

```
## [1] 2065
```

```
sum(pvals < 0.05)/length(pvals) # pct significant at alpha = 0.05
```

```
## [1] 0.2896619
```

- Question: Do you believe that all tests with p < 0.05 are actually statistically significant?
    - A: Yes
    - B: No
    - C: Not Sure

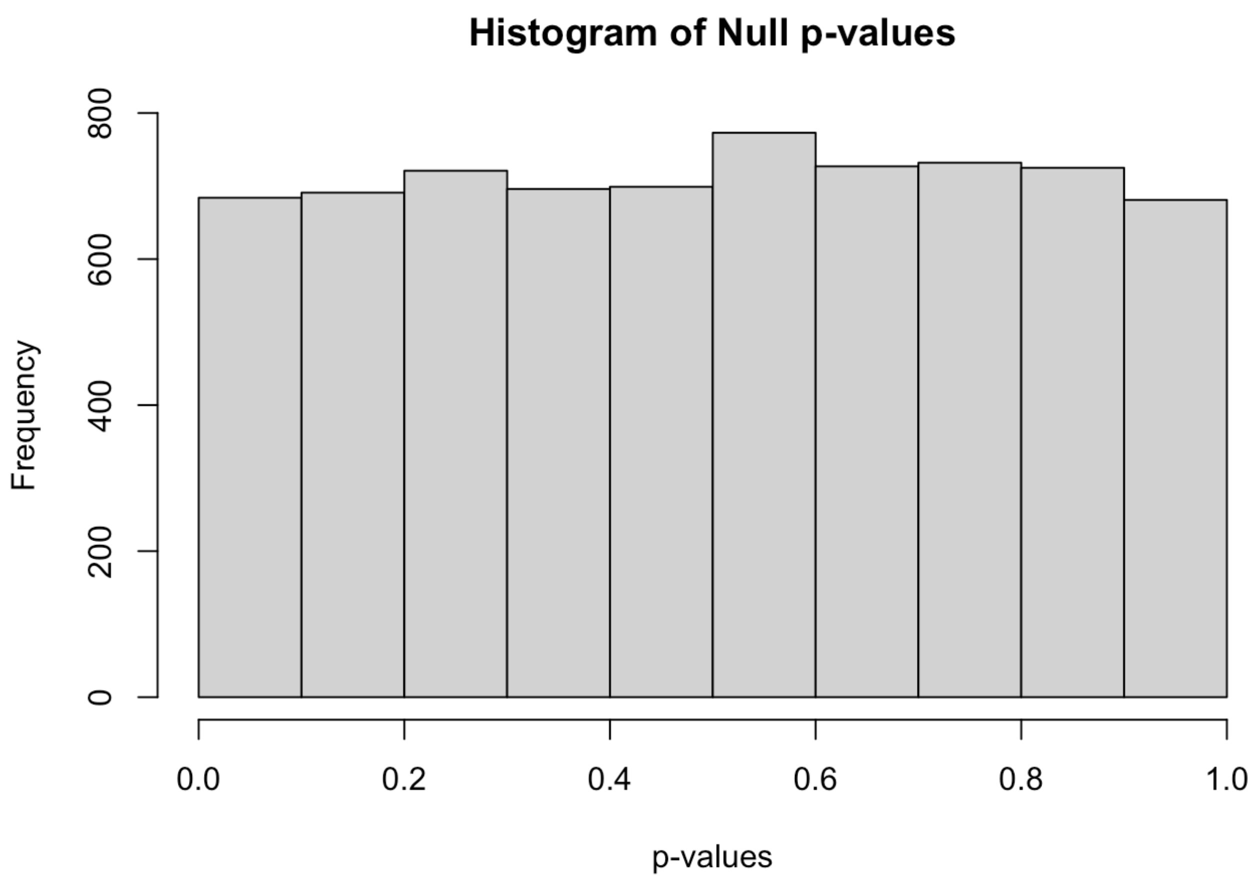
Multiple testing error

- What would happen if we ran many t-tests for columns where none are associated with the outcome?
- The data below are generated randomly and independent from the outcome
- Randomly generate each column independent of the outcome
- For each column, V in the generated data set, we know that $V \perp Y$

```
set.seed(1234)
null_dat <- data.frame(matrix(rnorm(dim(shipp$x)[1] * dim(shipp$x)[2]), ncol=dim(shipp$x)[2
]))
dim(null_dat)
```

```
## [1]   77 7129
```

```
null_pvals <-c()
for(var in names(null_dat)){
  null_pvals <- c(null_pvals, t.test(null_dat[[var]] ~ shipp$y)$p.value)
}
hist(null_pvals, xlab = 'p-values', main='Histogram of Null p-values')
```



- Question: What density does this look like?
  - A: Exponential
  - B: Beta
  - C: Negative Binomial
  - D: Uniform

- Question: What proportion is less than 0.05?
  - Enter response as decimal

```
sum(null_pvals < 0.05)/length(null_pvals) # pct significant at alpha = 0.05
```

```
## [1] 0.04586899
```

- Intuitively, what is the problem here? Why are so many p-values < 0.05 when none are associated with the outcome?

- Question: Did you expect the p-values to have a uniform distribution?

  - A: Yes
  - B: No
  - C: Not Sure
- Why do they look uniform?

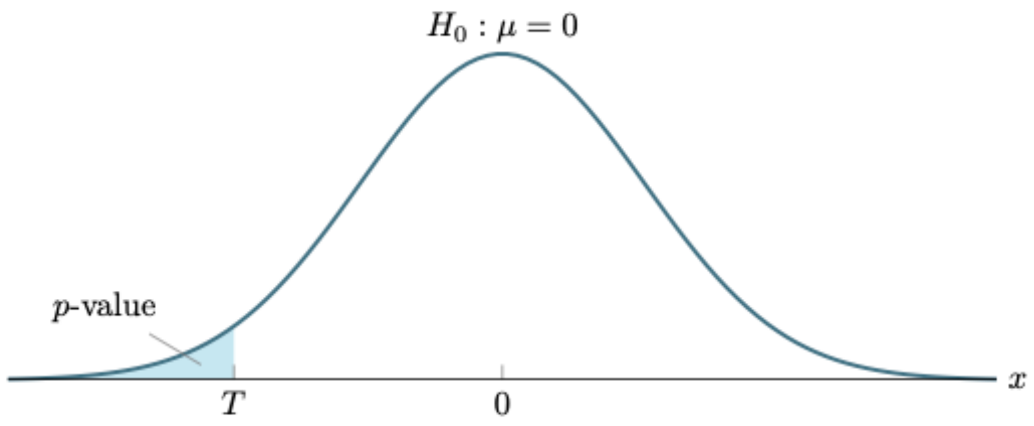
What is a hypothesis test/p-value?

- Example:

  - Let $X_1, X_2, \ldots, X_n$ be an independent, identically distributed sample

  - Let $E[X_i] = \mu$ (unknown) and $E[(X_i - \mu)^2] = \sigma^2 = 1$ (known) for each $i = 1, 2, \ldots, n$

- Is μ positive or negative?
- Let $H_0 : \mu \geq 0$ and $H_1 : \mu < 0$
- Let $\bar{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$
- Intuitively, if $\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} = \sqrt{n}\bar{X}$ is small (too negative), then we can reject H_0
- From the central limit theorem (https://en.wikipedia.org/wiki/Central_limit_theorem), we know that $\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \rightsquigarrow N(0, 1)$ (converges in distribution) as $n \to \infty$
- That is, our test statistic, $T = \sqrt{n}\bar{X}$ is approximately $N(0, 1)$ under H_0 (we're setting $\mu = 0$)
- p-value: $p = P[T < N(0, 1)] = \Phi(T) = \int_{-\infty}^{T} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$



p-value illustration

other illustration (http://blog.analytics-toolkit.com/2017/statistical-significance-ab-testing-complete-guide/)

Theorem: p-values are uniformly distributed under the null hypothesis.

Proof:

1. Let T be a test statistic of a hypothesis test and let F be the CDF of T under H_0, the null hypothesis
2. p-value: $p = F(T)$
3. Claim: A random variable, $Y \sim G$, is uniformly distributed on $[0, 1] \Leftrightarrow P[Y < y] = G(y) = y$ on $[0, 1]$
4. Goal: show that $P[p < y] = y$ under the null hypothesis
5. Using F^{-1}, the inverse function of F
6. Assume that F is montonically increasing, so that F^{-1} is also monontically increasing

$$\begin{aligned}
P[p < y] &= P[F(T) < y] && p = F(T) \\
&= P[F^{-1}(F(T)) < F^{-1}(y)] && F^{-1} \text{ monotonically increasing} \\
&= P[T < F^{-1}(y)] && F^{-1}(F(y)) = y \\
&= F(F^{-1}(y)) && F(t) = P(T < t) \\
&= y
\end{aligned}$$

7. So, under the null hypothesis, p-value $\sim$Unif(0,1) ■

- Question: Did this proof make sense?
  - 1 (not at all) to 5 (perfect sense)

Multiple Testing

- Want to run m tests
- Let $H_{0,i}$ be the null hypothesis for the ith test where $1 \leq i \leq m$
- Let p_i be the p-value of the ith test for $H_{0,i}$
- Let $R_i = 1$ if we reject $H_{0,i}$ and $R_i = 0$ if we fail to reject $H_{0,i}$ (sometimes called a discovery or positive result)
- $R = \sum_{i=1}^m R_i$ be the number of rejected tests
- Let $V_i = 1$ if we wrongly reject $H_{0,i}$ (false positive or type 1 error or false discovery)
- Let $V = \sum_{i=1}^m V_i$ be the false positive (discovery) count
- **Family-wise error rate (FWER)**

$$P[V > 0] \leq \alpha$$

  Same as $P[V = 0] \geq 1 - \alpha$

- **Per family error rate (PFER)**

$$E[V] \leq \alpha$$

- **False discovery rate (FDR)** controls

$$E\left[\frac{V}{R}\right]$$

  - If $R = 0$, use $R = 1$ instead
- **Global null**
  - Can we reject at least one of the m null hypotheses?
  - The global null hypothesis is

$$H_0 = \bigcap_{i=1}^m H_{0,i}$$

- Does not indicate which $H_{0,i}$ to reject, only that we can reject at least one $H_{0,i}$

Tests

Bonferroni

- Reject $H_{0,i}$ if $p_i \le p_{\text{Bon}} := \frac{\alpha}{m}$
- uses FWER, tells us which $H_{0,i}$ we can reject

proof

Let $I := \{i : H_{0,i} = 1\}$ be the set of true null hypotheses.

$$\begin{aligned}
P[V > 0] &= P\left[\bigcup_{i\in I}\left\{p_i \le \frac{\alpha}{m}\right\}\right] && V > 0 \Leftrightarrow \text{ at least one } p_i \text{ significant in nulls} \\
&\le \sum_{i\in I} P\left[p_i \le \frac{\alpha}{m}\right] && \text{Union bound} \\
&= \sum_{i\in I} \frac{\alpha}{m} && p_i \sim \text{Unif}(0,1) \text{ for } i \in I \\
&= \frac{\alpha|I|}{m} && |I| = \#\text{ true null hypotheses} \\
&\le \alpha && \frac{|I|}{m} \le 1
\end{aligned}$$

More info on union bound (https://en.wikipedia.org/wiki/Boole%27s_inequality)

- Think about playing cards: $P[\text{Ace OR club}] \le P[\text{Ace}] + P[\text{club}]$
- Because of union bound, even holds when p-values are statistically dependent
- Using Bonferroni, which gene's should I recommend are associated with lymphoma?

```
bcorrection <- 0.05/length(pvals)
bcorrection
```

```
## [1] 7.013606e-06
```

```
sum(pvals <= bcorrection)
```

```
## [1] 245
```

```
which(pvals <= bcorrection)
```

```
##   [1]  163  171  173  191  200  203  228  233  259  275  282  302  316  322  323
##  [16]  355  373  375  379  385  386  405  407  438  440  441  463  506  538  544
##  [31]  582  592  605  640  649  658  778  792  834  922  972 1016 1092 1110 1119
##  [46] 1127 1132 1157 1166 1173 1176 1188 1217 1245 1248 1346 1352 1369 1372 1373
##  [61] 1394 1430 1445 1479 1480 1490 1551 1621 1691 1704 1733 1778 1780 1790 1804
##  [76] 1818 1823 1825 1829 1832 1860 1903 1930 1934 1942 2030 2043 2094 2111 2121
##  [91] 2137 2145 2177 2185 2220 2229 2237 2282 2306 2308 2309 2319 2380 2383 2386
## [106] 2392 2412 2419 2421 2527 2645 2668 2669 2729 2761 2789 2883 2919 2929 2937
## [121] 2941 2947 2988 3005 3137 3163 3181 3248 3257 3258 3283 3310 3325 3351 3387
## [136] 3397 3461 3494 3499 3545 3639 3693 3736 3757 3818 3887 3926 3928 3943 3987
## [151] 4010 4024 4028 4078 4087 4110 4116 4130 4133 4153 4158 4183 4202 4242 4243
## [166] 4254 4262 4273 4292 4328 4330 4340 4372 4373 4387 4406 4418 4424 4453 4463
## [181] 4485 4502 4503 4518 4535 4546 4567 4580 4582 4594 4667 4754 4970 5077 5092
## [196] 5169 5187 5254 5280 5302 5324 5336 5409 5579 5594 5600 5621 5655 5666 5671
## [211] 5704 5743 5882 5935 5956 5959 5994 5998 6023 6050 6058 6110 6179 6191 6295
## [226] 6310 6322 6334 6352 6362 6377 6378 6474 6524 6549 6612 6659 6676 6815 6986
## [241] 7005 7091 7102 7119 7123
```

What about the independent simulated data?

```
which(null_pvals<=bcorrection)
```

```
## integer(0)
```

- Notice that test may be associated (knowing one p-value may give information about others)
- Bonferroni still works here
- How could we construct a global null hypothesis from Bonferroni?
    - If at least one $p_i \leq \frac{\alpha}{m}$, then we can reject global null

Bonferroni is conservative

- Types of error: false positives (type I error) and false negatives (type II error)
- Bonferroni is very good at limiting false positives but not limiting false negatives
- Power = 1-P(type II error), is the probability of rejecting the null hypothesis when the alternative hypothesis is true
- Bonferroni may not always be a powerful test

Fisher Combination

- Uses global null framework
- Assumes that each p_i is independent (why might this not be reasonable in most settings?)
- If $H_{0,i}$ is true for each i (and each test is independent), then

$$T = \sum_{i=1}^{m} -2\log(p_i) \sim \chi^2(2m)$$

- uses that fact that transforming independent, uniform random variables this way will have a χ^2 distribution
- This test does not indicate which $H_{0,i}$ to reject
- Why should we not use this test here?
- Could work for meta-analysis combining p-values from many different, independent studies

Simes Test

- Uses global null framework
- Order all p-values, $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(m)}$ for the m hypotheses
- Simes test statistic

$$p_{\text{Simes}} = \min_{i \in [m]} \left\{ \frac{mp_{(i)}}{i} \right\}$$

- Reject global null if $p_{\text{simes}} \leq \alpha$
  - Equivalent: Reject global null if any $p_{(i)} \leq \frac{i\alpha}{m}$
- Proof is a little more complicated
  - Need to show that $p_{\text{Simes}} \sim \text{Unif}(0, 1)$
  - Uses order statistics properties to show this
- Does not require all p_i are independent
- More powerful (type II error is smaller) than Bonferroni global null test
  - Bonferroni will reject global null when $mp_{(1)} \leq \alpha$

```
simes <- function(vec) length(vec)*min(sort(vec)/(1:length(vec)))

simes(pvals)
```

```
## [1] 1.47472e-12
```

```
simes(null_pvals)
```

```
## [1] 0.6610137
```

- Coding tip: DRY (don't repeat yourself)
- if you're tempted to copy your own code to reuse it, write a function for it instead

Kolmogorov-Smirnov Test

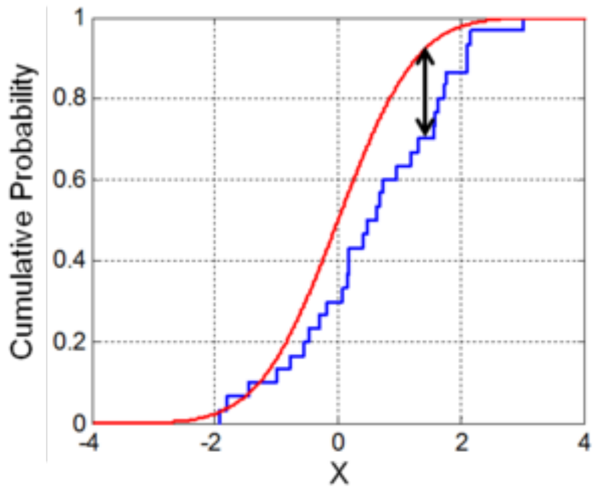


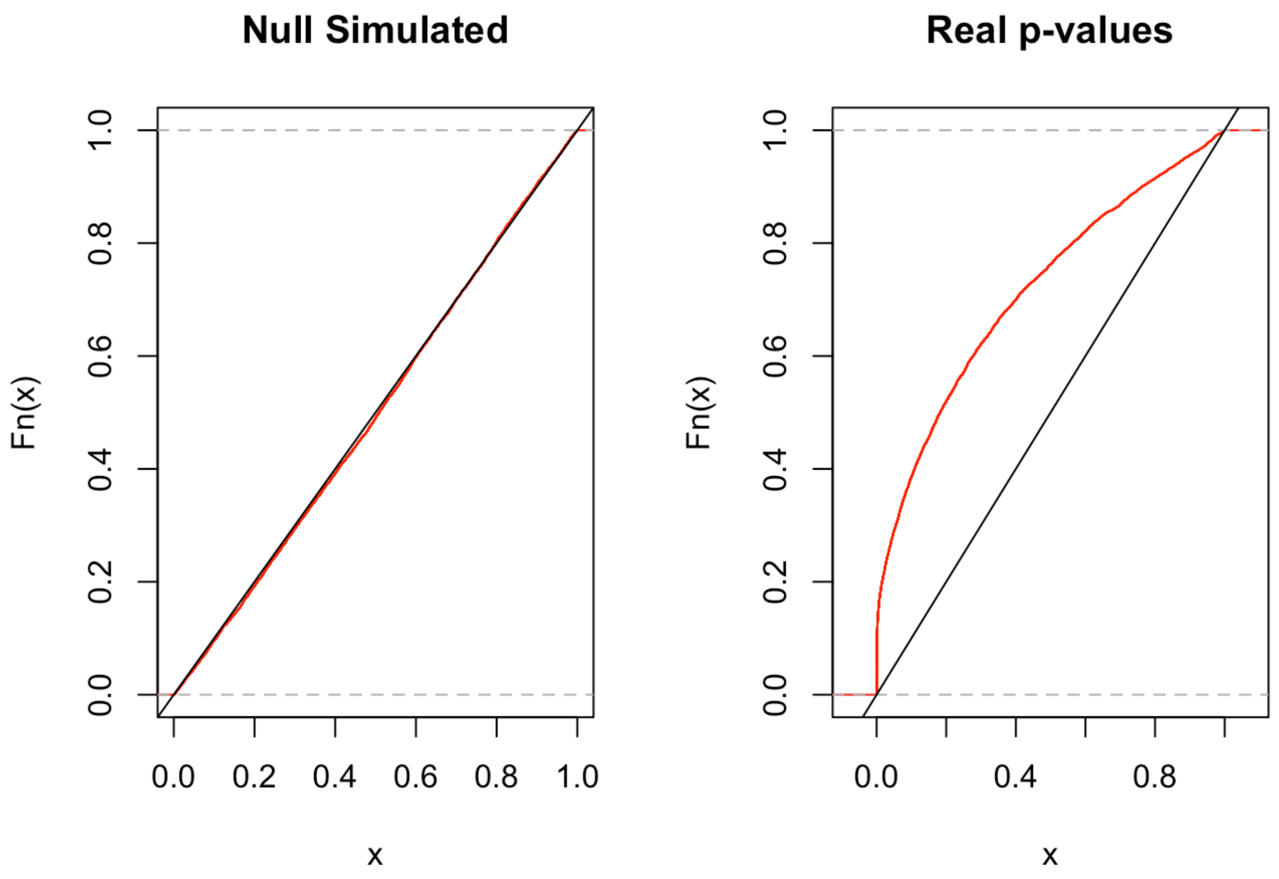
image from wikipedia

- compared empirical cdf to theoretical cdf
    - Here: Assess fit of empirical p-value cumulative distribution compared to uniform
- Uses global null framework
- Empirical CDF of p-values is

$$\hat{F}_m(t) = \frac{1}{m} \sum_{i=1}^{m} I(p_i \leq t)$$

- Uniform CDF $F(t) = t$ for $t \in [0, 1]$

```
par(mfrow=c(1,2))
plot(ecdf(null_pvals), col='red', xlim=c(0,1), ylim=c(0,1),
     main="Null Simulated")
abline(c(0,0), c(1,1))
plot(ecdf(pvals), col='red',
     main="Real p-values")
abline(c(0,0), c(1,1))
```



- Test statistic

$$T_m = \sup_{t \in [0,1]} \left| \hat{F}_m(t) - t \right|$$

- Using Hoeffding's inequality,

$$P\left[T_m > t\right] \leq 2\exp\left(-2t^2\right)$$

- Note: here so do not know the CDF of the test statistic, but we *consentration inequality* above allows us to bound the p-value
- Reject global null if

$$T_m > \sqrt{\frac{2\log(\frac{2}{\alpha})}{2}}$$

```
ks.test(pvals, runif(100000))
```

```
## Warning in ks.test(pvals, runif(1e+05)): p-value will be approximate in the
## presence of ties
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  pvals and runif(1e+05)
## D = 0.32639, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```
ks.test(null_pvals, runif(100000))
```

```
## Warning in ks.test(null_pvals, runif(1e+05)): p-value will be approximate in the
## presence of ties
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  null_pvals and runif(1e+05)
## D = 0.011325, p-value = 0.3607
## alternative hypothesis: two-sided
```

See ks.test documentation (https://stat.ethz.ch/R-manual/R-devel/library/stats/html/ks.test.html) for more info

Benjamini-Hochberg

- Method for controlling the false discovery rate (FDR)
- Interpretation: At an FDR of α, we would expect, at most, α of our significant tests to be false positives
- Method:
    1. Order all p-values, $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(m)}$ for the m hypotheses
    2. Let $j = \max\left\{i : p_{(i)} < \frac{i\alpha}{m}\right\}$, $T_{\text{BH}} = p_{(j)}$
    3. Reject $H_{0,i}$ for $p_i \leq T_{\text{BH}}$
- Alternative:
    - adjust p-values using

$$\frac{mp_{(i)}}{i}$$

Proof outline

Recall

$$\text{FDR} = E\left[\frac{V}{R}\right] = E\left[\frac{\text{\# Type 1 Error}}{\text{\# Rejections}}\right].$$

Let $W_i = 1$ if $H_{0,i}$ is true and $W_i = 0$ otherwise. Let $G(t)$ be the true CDF of the p-values and let $\hat{G}(t)$ be the empirical CDF as before.

$$\begin{aligned}
\text{FDR} &= E\left[\frac{V}{R}\right] \\
&= E\left[\frac{\frac{1}{m}\sum_{i=1}^{m} W_i I(p_i < t)}{\frac{1}{m}\sum_{i=1}^{m} I(p_i < t)}\right] \\
&\approx \frac{E\left[\frac{1}{m}\sum_{i=1}^{m} W_i I(p_i < t)\right]}{E\left[\frac{1}{m}\sum_{i=1}^{m} I(p_i < t)\right]} \\
&= \frac{t|I|}{G(t)} \leq \frac{t}{G(t)} \approx \frac{t}{\hat{G}(t)}
\end{aligned}$$

Let $t = p_{(i)}$ for some i. Notice that $\hat{G}(p_{(i)}) = \frac{i}{m}$. Then $\text{FDR} = \frac{p_{(i)} m}{i}$. Setting this value equal to α and solving for $p_{(i)}$, we get the BH test statistic.

```
bh_adj <- p.adjust(pvals, 'BH')
round(bh_adj, 3)[1:10]
```

```
##  [1] 0.150 0.649 0.989 0.501 0.855 0.678 0.436 0.984 0.591 0.936
```

```
round(pvals[1:10], 3)
```

```
##  [1] 0.040 0.492 0.982 0.318 0.773 0.529 0.248 0.975 0.424 0.892
```

```
which(bh_adj < 0.05)[1:10]
```

```
##  [1] 38 44 45 46 49 52 57 64 65 87
```

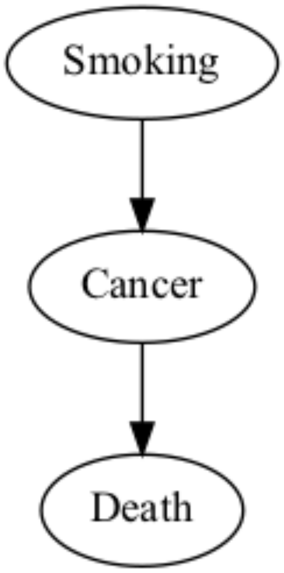
```
sum(bh_adj<0.05)
```

```
## [1] 1232
```

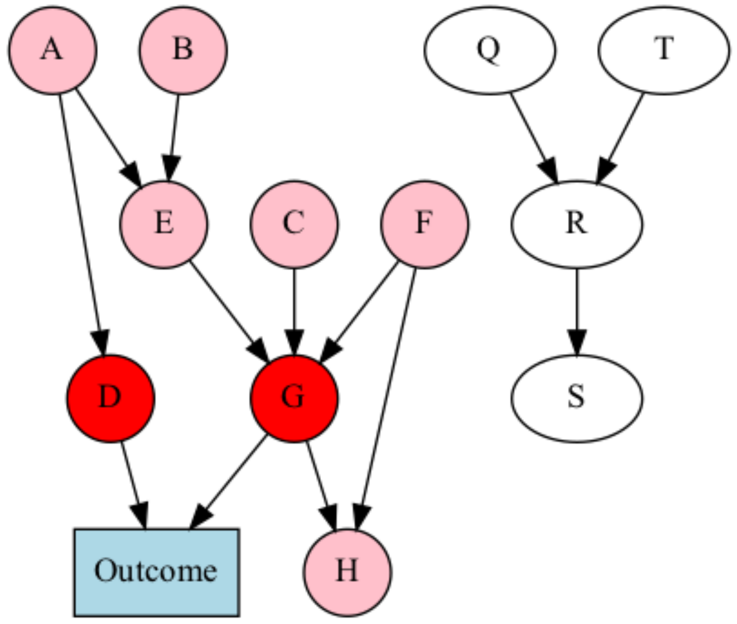
See p.adjust documentation (https://stat.ethz.ch/R-manual/R-devel/library/stats/html/p.adjust.html) for more info

Final thoughts on multiple testing

- Family-wise error and false discovery
- Only Bonferroni and Benjamini-Hochberg indicate which null hypotheses to reject, all others are global null
- Many applied papers do not use multiple testing procedures but should
- Prediction, causation, and dependence
    - Smoking and cancer can both be used to predict death, but should we use both?



- What about something more complex?
- Question: Should D in the model below associated with Outcome?
  - A: Yes
  - B: No
- Question: Should A in the model below associated with Outcome?
  - A: Yes
  - B: No
- Question: Should Q in the model below associated with Outcome?
  - A: Yes
  - B: No
- Question: In a regression model (Outcome ~ A+B+C+D+E+F+G+H+Q+R+S+T), which variables should be associated/significant?
  - A: Red Only
  - B: Pink Only
  - C: Red and Pink Only
  - D: All Variables
- Question: In a regression model (Outcome ~ A+B+C+E+F+G+H+Q+R+S+T, D is removed), which variables should be associated/significant?
  - A: G only
  - B: A and G only
  - C: A, E, and G only
  - D: A, B, E, and G only



- pink and red variables will be associated with Outcome
- D and G are sufficient for prediction
- if D and G are used, other variables provide no additional accuracy
- regression takes this into account
- $\beta = [\beta_A, \beta_B, \beta_C, \beta_D, \beta_E, \beta_F, \beta_G, \beta_H, \beta_Q, \beta_R, \beta_S, \beta_T] = [0, 0, 0, \beta_D, 0, 0, \beta_G, 0, 0, 0, 0, 0]$ where $\beta_D, \beta_G \neq 0$
- $\{\beta_D, \beta_G\}$ is the *active* set
- What should the active set be if G were missing?

Penalized Regression

Ordinary least squares

$$\text{Have } \mathbf{X} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}. \text{ Want to estimate } \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$

so that

$$\hat{\beta} = \arg\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2$$

Note: The Euclidean norm, $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_p^2} = \sqrt{x^T x}$ for $x \in \mathbb{R}^p$

Using calculus, it's easy to show that

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

Proof: Taking the gradient (derivative with respect to vector) and setting it equal to zero,

$$\begin{aligned}\frac{\partial}{\partial\beta}\|\mathbf{y}-\mathbf{X}\beta\|_2^2 &= \frac{\partial}{\partial\beta}(\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta)\\ &= \frac{\partial}{\partial\beta}\mathbf{y}^T\mathbf{y}-\frac{\partial}{\partial\beta}2\mathbf{y}^T\mathbf{X}\beta+\frac{\partial}{\partial\beta}\beta^TX^TX\beta\\ &= -2\mathbf{y}\mathbf{X}+2\mathbf{X}^T\mathbf{X}\beta=0\ .\end{aligned}$$

Solving for zero, we get the solution above.

OLS Pros:

- When true relationship between response and predictors is approximately linear, low bias
- When $n \gg p$, low variance

OLS Cons:

- When $n \not\gg p$, *overfitting* can be a problem
- $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$ only exists for $n >> p$.

Ridge Regression

$$\hat{\beta}=\arg\min_{\beta}\frac{1}{n}\sum_{i=1}^{n}\left\|y_i-X_i^T\beta\right\|^2+\lambda\|\beta\|_2^2$$

Similar to OLS, $\hat{\beta}$ also has a closed form

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

where I is the identity matrix

Proof: Taking the gradient (derivative with respect to vector) and setting it equal to zero,

$$\begin{aligned}\frac{\partial}{\partial\beta}\left[\|\mathbf{y}-\mathbf{X}\beta\|^2+\lambda\|\beta\|_2^2\right] &= \frac{\partial}{\partial\beta}\left[(\mathbf{y}-\mathbf{X}\beta)^T(\mathbf{y}-\mathbf{X}\beta)+\lambda\beta^T\beta\right]\\ &= \frac{\partial}{\partial\beta}\mathbf{y}^T\mathbf{y}-\frac{\partial}{\partial\beta}2\mathbf{y}^T\mathbf{X}\beta+\frac{\partial}{\partial\beta}\beta^TX^TX\beta+\frac{\partial}{\partial\beta}\lambda\beta^T\beta\\ &= -2\mathbf{y}\mathbf{X}+2\mathbf{X}^T\mathbf{X}\beta+2\lambda\beta=0\ .\end{aligned}$$

Solving for zero, we get the solution above.

Using `GLMNET`

- Below: create a dataset where 10 variable are associated with the outcome and 20 are not
- We are choosing the true β randomly from a uniform distribution

```
### Generating data
sample_size <- 50 # remember DRY
set.seed(1234)
num_active_vars <- 10
num_null_vars <- 20
true_beta <- 2*runif(num_active_vars) # randomly choosing true beta
true_beta
```

```
##  [1] 0.22740682 1.24459881 1.21854947 1.24675888 1.72183077 1.28062121
##  [7] 0.01899151 0.46510101 1.33216752 1.02850228
```

```
active_x <- matrix(rnorm(sample_size*num_active_vars), nrow=sample_size)
null_x <- apply(matrix(3*rnorm(sample_size*num_null_vars), nrow=sample_size), 2,
                function(x) x + 10*runif(1))
y <- active_x %*% true_beta + rnorm(sample_size)
dim(y) #sanity check
```

```
## [1] 50  1
```

```
dat <- data.frame(cbind(active_x, null_x, y))
dim(dat)
```

```
## [1] 50 31
```

```
names(dat)
```

```
##  [1] "X1"  "X2"  "X3"  "X4"  "X5"  "X6"  "X7"  "X8"  "X9"  "X10" "X11" "X12"
## [13] "X13" "X14" "X15" "X16" "X17" "X18" "X19" "X20" "X21" "X22" "X23" "X24"
## [25] "X25" "X26" "X27" "X28" "X29" "X30" "X31"
```

```
names(dat)[31] <- 'Y' # renaming response variable
head(dat)
```

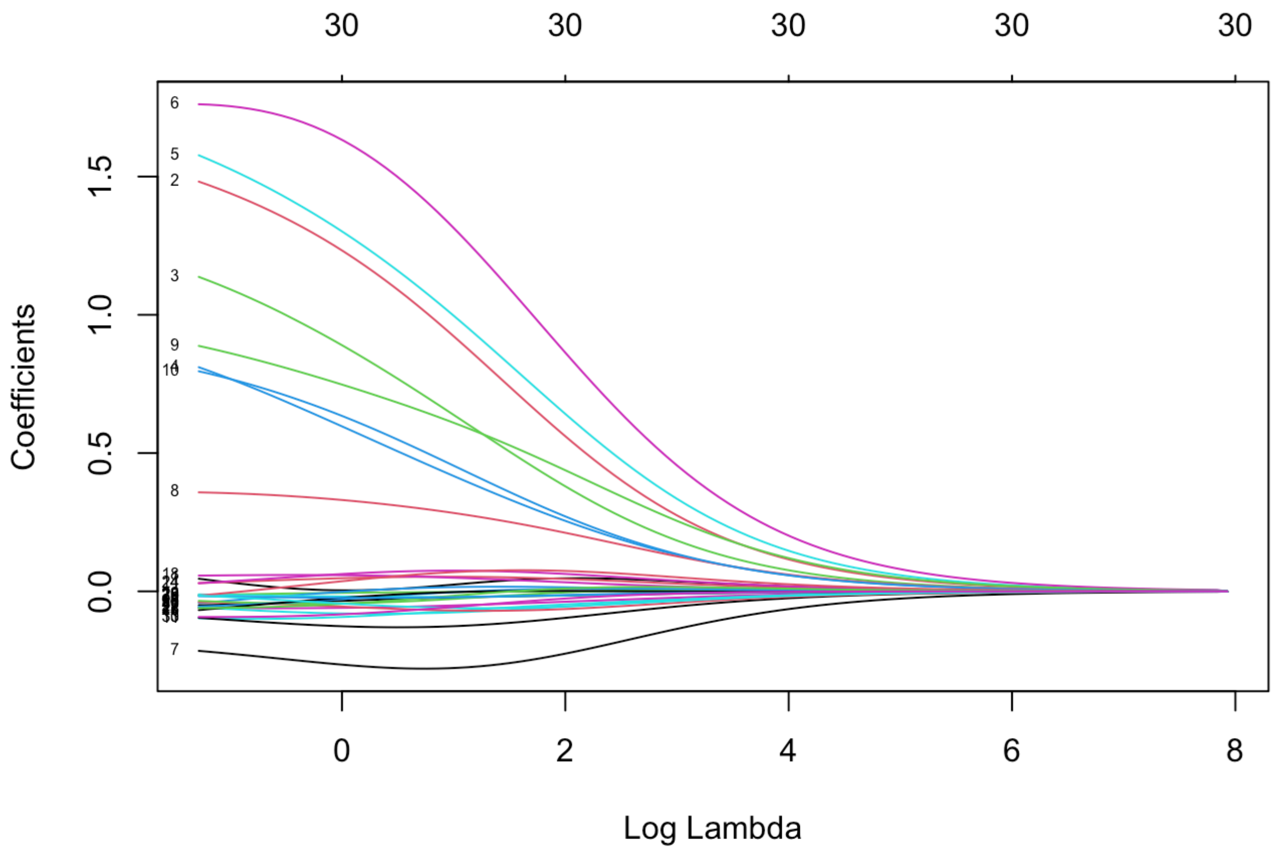
```
##           X1         X2          X3         X4          X5         X6
## 1  0.5060559  0.5630558  0.16698928  1.3621307  0.76046236 -0.3704975
## 2 -0.5747400  1.6478175 -0.89626463 -0.2346211  1.84246363  1.4769696
## 3 -0.5466319 -0.7733534  0.16818539 -1.0533828  1.11236284 -1.2239038
## 4 -0.5644520  1.6059096  0.35496826 -0.8697836  0.03266396  0.2580684
## 5 -0.8900378 -1.1578085 -0.05210512 -0.3901270 -1.11444896  0.4050028
## 6 -0.4771927  0.6565885 -0.19593462 -0.8473501  0.41805782  0.9758033
##           X7         X8          X9        X10        X11        X12       X13
## 1 -0.6490282  0.5475242  0.15344474 -0.6512008  2.8035838 11.0983215  6.223361
## 2 -0.5043742  1.8912270  1.46328305 -1.4736558 -1.0474032  6.8697732 10.087990
## 3  1.6143915 -0.8780771 -1.12150250 -1.2016658  5.0094210  0.7218307 11.205312
## 4 -0.4469598 -0.1125589 -0.51778808 -0.1487205  4.5974288 12.1763416  3.324929
## 5  0.7631768  1.9487131 -0.07494709  1.7970624  3.4810388 11.1607255  9.802090
## 6  1.4717187  0.9338163 -1.40779008  0.1048087  0.3642669  7.2122591  4.870920
##          X14        X15        X16       X17       X18      X19        X20
## 1  4.0752264 -0.6053643  4.7564041  5.321064  7.799665 5.592186  4.6772059
## 2  6.4132250  2.0202636  0.2970701 -0.318436  9.808652 4.291757 -0.1177785
## 3  2.6473692 -1.7636239 -0.8032459  2.324419  8.307224 2.954393  2.2330444
## 4  3.1922946  3.0787080  3.2034946  4.687638  6.210801 2.632948  3.8991051
## 5  5.0369160 -2.4553716  0.2688438  1.924475 10.546669 1.804470  2.5989062
## 6 -0.2713349  6.3026241  5.3144006  5.345141  7.745985 3.486791  7.8291542
##         X21       X22      X23       X24       X25       X26      X27       X28
## 1 -2.177259  9.976276 5.141210 14.573507 11.989885  7.121203 8.097692  5.660709
## 2  6.357154  7.971671 4.697821 12.306464 10.624856 10.030938 9.797545  5.769629
## 3  2.866913  7.527770 1.023797  7.109257 10.052675 11.150259 1.218631  2.688446
## 4  1.518939  9.099298 6.546246  9.615181  9.290212  5.690831 7.503465 12.380381
## 5  4.877752  6.373329 2.150257 13.436050 10.712848 10.109707 7.914734  4.128210
## 6  7.382241 12.917893 1.876893 12.719626  5.757507  7.645144 7.074355  2.815231
##         X29        X30          Y
## 1 10.888932  0.7104575  1.9144653
## 2  6.300410  0.3718349  9.8368493
## 3  5.952818 -0.5419693 -4.0199055
## 4  6.531286 -2.8194315  1.9355236
## 5  7.981174  3.4402196 -2.8156338
## 6  8.345330 -1.1286243 -0.1450641
```

```r
# install.packages("glmnet", repos = "http://cran.us.r-project.org") # only need 1st time
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.0-2
```

```
design_mat <- cbind(active_x, null_x) # glmnet only takes matrices, not dataframes
l2_fit <- glmnet(design_mat, y, family="gaussian", alpha=0) # alpha = 0 gives Ridge regress
ion
plot(l2_fit, xvar='lambda', label=TRUE)
```



```
names(l2_fit)
```

```
##  [1] "a0"         "beta"       "df"         "dim"        "lambda"     "dev.ratio"
##  [7] "nulldev"    "npasses"    "jerr"       "offset"     "call"       "nobs"
```

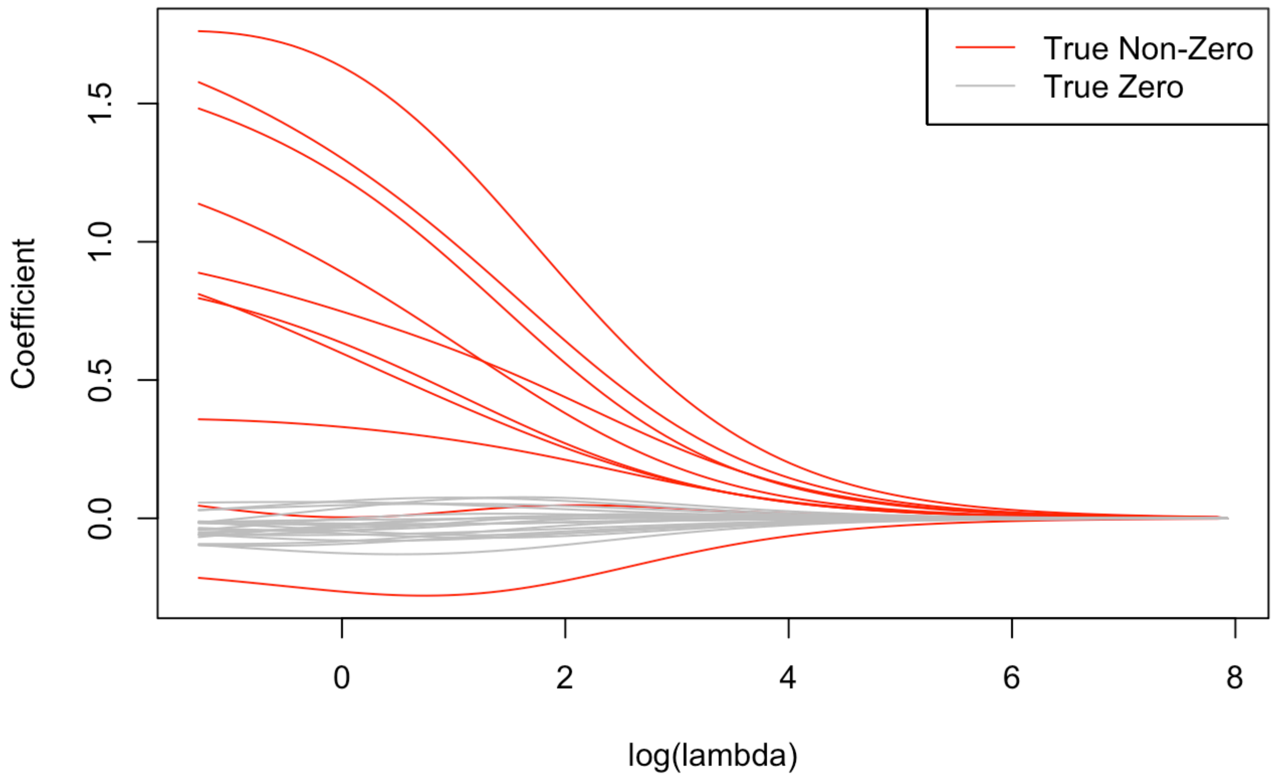
```
coef(l2_fit)[1:10,1:5]
```

```
## 10 x 5 sparse Matrix of class "dgCMatrix"
##                        s0            s1            s2           s3
## (Intercept)  7.001009e-01  0.6946076979  0.6940756757  0.693492506
## V1           2.683107e-37  0.0004376648  0.0004801991  0.000526852
## V2           1.665212e-36  0.0027187723  0.0029832648  0.003273426
## V3           1.082759e-36  0.0017686261  0.0019407721  0.002129642
## V4           8.842758e-37  0.0014418534  0.0015819206  0.001735540
## V5           2.190053e-36  0.0035726384  0.0039198744  0.004300743
## V6           3.029803e-36  0.0049414403  0.0054215992  0.005948243
## V7          -1.010926e-36 -0.0016470104 -0.0018068629 -0.001982153
## V8           9.657436e-37  0.0015728551  0.0017254526  0.001892775
## V9           1.899821e-36  0.0030947077  0.0033950155  0.003724314
##                        s4
## (Intercept)  0.6928533504
## V1           0.0005780196
## V2           0.0035917334
## V3           0.0023368538
## V4           0.0019040129
## V5           0.0047184792
## V6           0.0065258349
## V7          -0.0021743546
## V8           0.0020762267
## V9           0.0040853707
```

```
l2_fit$lambda
```

```
##   [1] 2774.4134622 2527.9422170 2303.3667979 2098.7420401 1912.2955818
##   [6] 1742.4125130 1587.6213878 1446.5814795 1318.0711679 1200.9773582
##  [11] 1094.2858398  997.0725019  908.4953289  827.7871079  754.2487828
##  [16]  687.2433998  626.1905903  570.5615442  519.8744292  473.6902178
##  [21]  431.6088844  393.2659407  358.3292784  326.4962930  297.4912622
##  [26]  271.0629584  246.9824723  225.0412302  205.0491875  186.8331829
##  [31]  170.2354379  155.1121908  141.3324513  128.7768658  117.3366840
##  [36]  106.9128164   97.4149765   88.7608984   80.8756249   73.6908573
##  [41]   67.1443646   61.1794443   55.7444311   50.7922495   46.2800060
##  [46]   42.1686179   38.4224741   35.0091274   31.8990130   29.0651926
##  [51]   26.4831210   24.1304335   21.9867523   20.0335098   18.2537880
##  [56]   16.6321718   15.1546155   13.8083212   12.5816280   11.4639109
##  [61]   10.4454886    9.5175402    8.6720282    7.9016291    7.1996701
##  [66]    6.5600713    5.9772926    5.4462864    4.9624534    4.5216027
##  [71]    4.1199160    3.7539140    3.4204266    3.1165653    2.8396983
##  [76]    2.5874273    2.3575674    2.1481276    1.9572939    1.7834133
##  [81]    1.6249798    1.4806210    1.3490867    1.2292376    1.1200355
##  [86]    1.0205346    0.9298732    0.8472658    0.7719970    0.7034149
##  [91]    0.6409255    0.5839874    0.5321076    0.4848366    0.4417651
##  [96]    0.4025199    0.3667611    0.3341791    0.3044915    0.2774413
```

```
glmnet_plot <- function(fit, num_active){ #assumes active vars are first
  plot(0, type='n', ylim = range(coef(fit)[-1,]), xlim = log(range(fit$lambda)),
     ylab = "Coefficient", xlab="log(lambda)")
  num_vars <- dim(coef(fit))[1]-1 # removing intercept
  for(itr in 1:num_vars){
    active = c(rep('red', num_active), rep('gray', num_vars-num_active))
    lines(log(fit$lambda), coef(fit)[itr+1,], col=active[itr])
    legend('topright', legend = c('True Non-Zero', 'True Zero'), col=c('red', 'gray'), lty
= 1)
  }
}
glmnet_plot(l2_fit, num_active_vars)
```

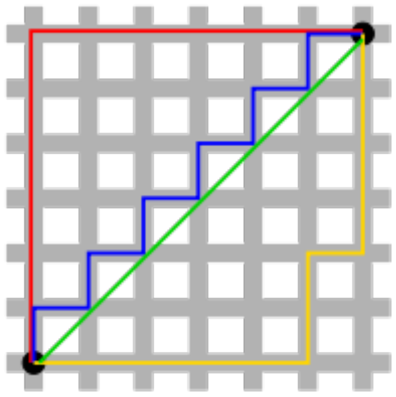


- As λ gets bigger, coefficients shrink
- Notice that the coefficients get closer to zero but are never exactly zero
- Because of closed form, fast to compute

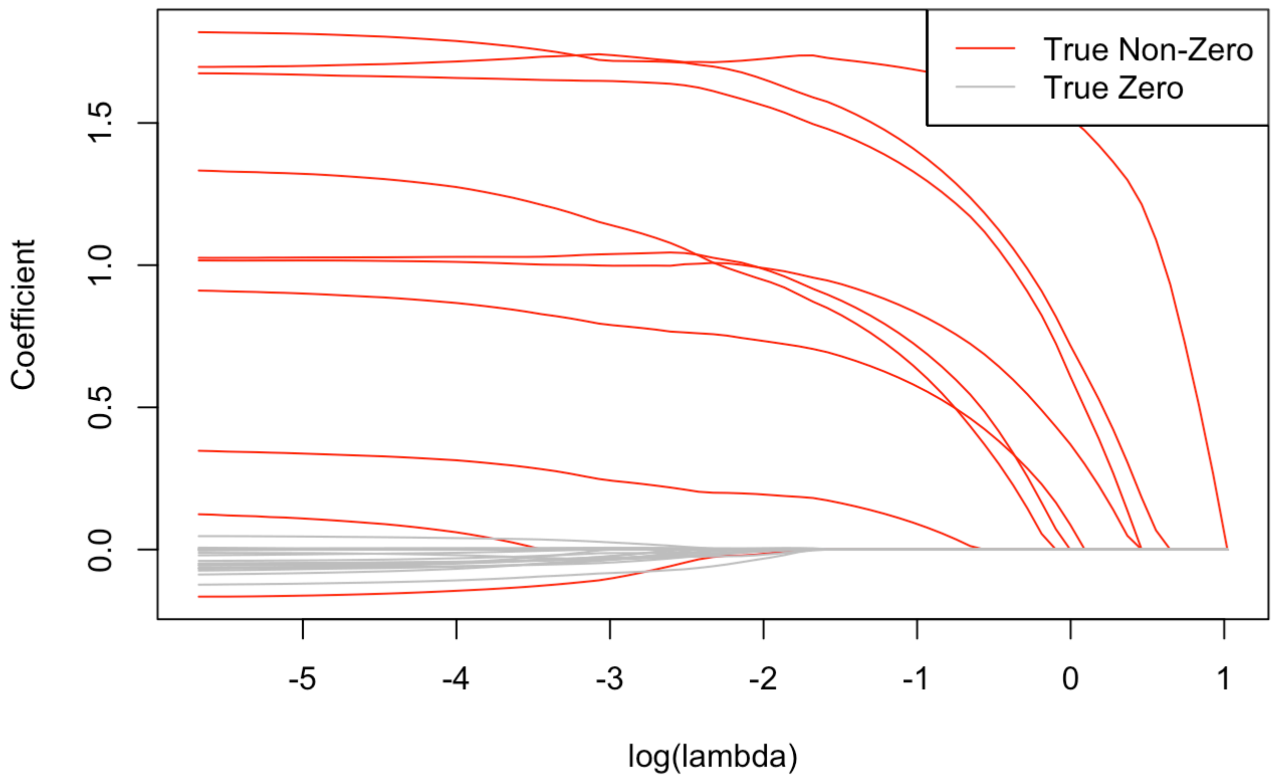
LASSO

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - X_i^T \beta\right)^2 + \lambda \|\beta\|_1$$

- The L_1 norm or taxicab norm, $\|x\|_1 = |x_1| + |x_2| + \cdots + |x_p|$ for $x \in \mathbb{R}^p$


from wikipedia

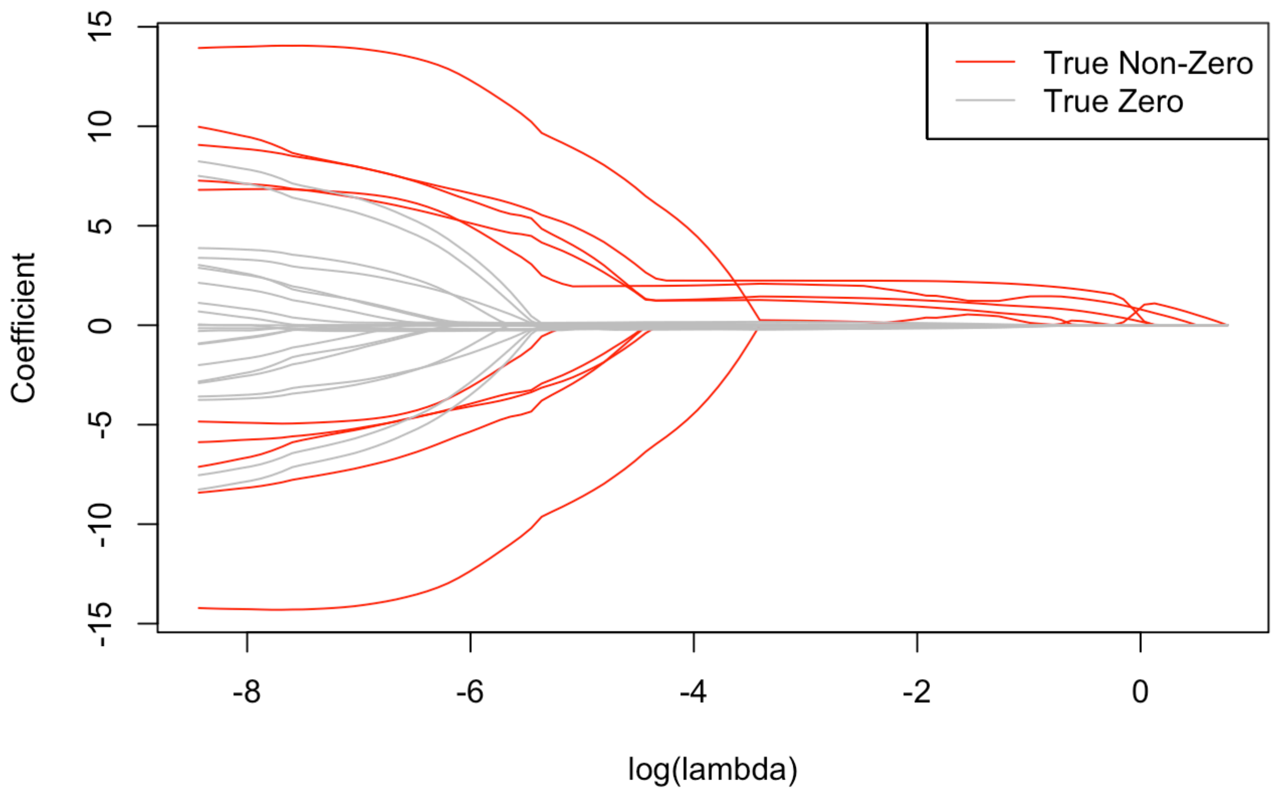
```
l1_fit <- glmnet(design_mat, y, family="gaussian", alpha=1) # alpha=1 gives lasso
glmnet_plot(l1_fit, num_active_vars)
```



- L_1 penalty generates sparse β (many zeros)
- As λ gets bigger, β shrinks and some go to zero
- Because of the absolute value, this function is not differentiable everywhere, standard optimization uses coordinate descent, also fast

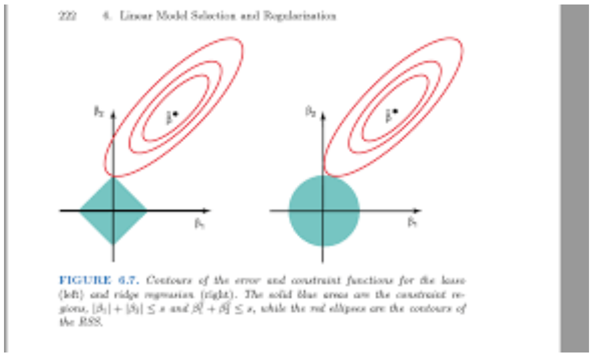
- if $p > n$, lasso selects at most n variables
- if using grouped (dummy) variables (like race or other categorical variables with more than 2 levles), lasso will ignore groups
- Can be erratic on collinear data

```
set.seed(1234)
dm_half <- design_mat[, c(1:5, 11:21)] # taking half of active and null columns
collin_vars <- dm_half+ 0.1*matrix(rnorm(dim(dm_half)[1]*dim(dm_half)[2]), nrow=sample_size
) #collinear vars
collin_dm <- cbind(dm_half[,1:5], collin_vars[,1:5], dm_half[,6:16], collin_vars[,6:16])
l1_fit_collin <- glmnet(collin_dm, y, family="gaussian", alpha=1)
glmnet_plot(l1_fit_collin, num_active_vars)
```



Difference between Ridge and LASSO

- Why does the L_1 -norm penalty in LASSO zero out some β values but not for the L_2 -norm penalty in Ridge?

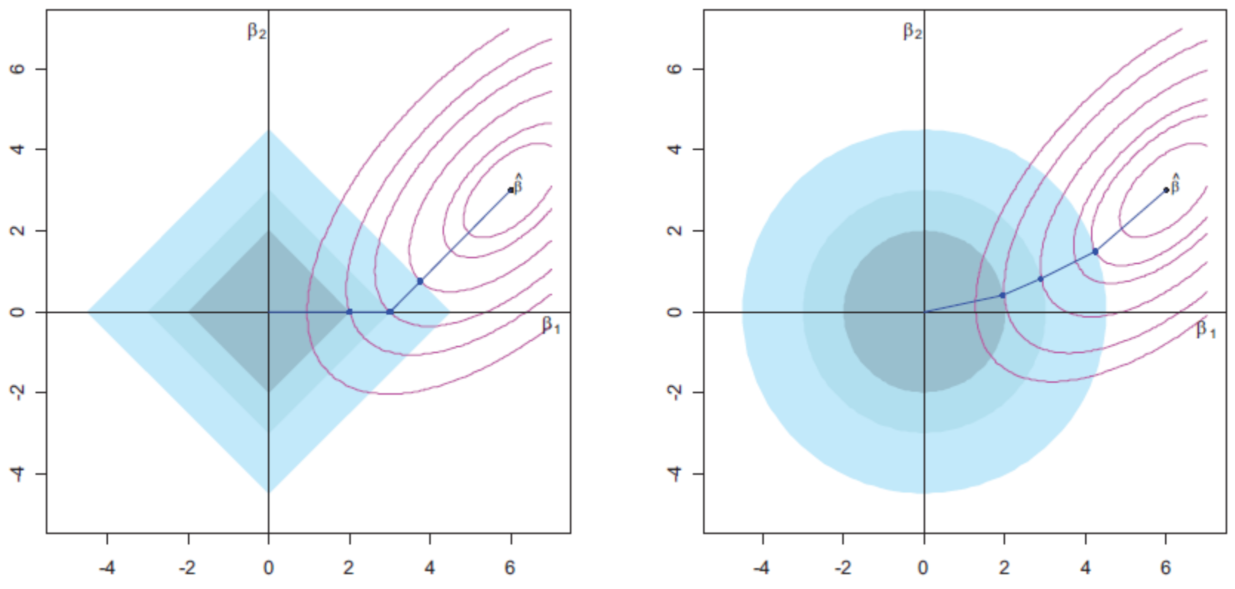


222 6. Linear Model Selection and Regularization

FIGURE 6.7. Contours of the error and constraint functions for the lasso (left) and ridge regression (right). The solid blue areas are the constraint regions, $|\beta_1| + |\beta_2| \le s$ and $\beta_1^2 + \beta_2^2 \le s$, while the red ellipses are the contours of the RSS.

from Elements of Statistical Learning

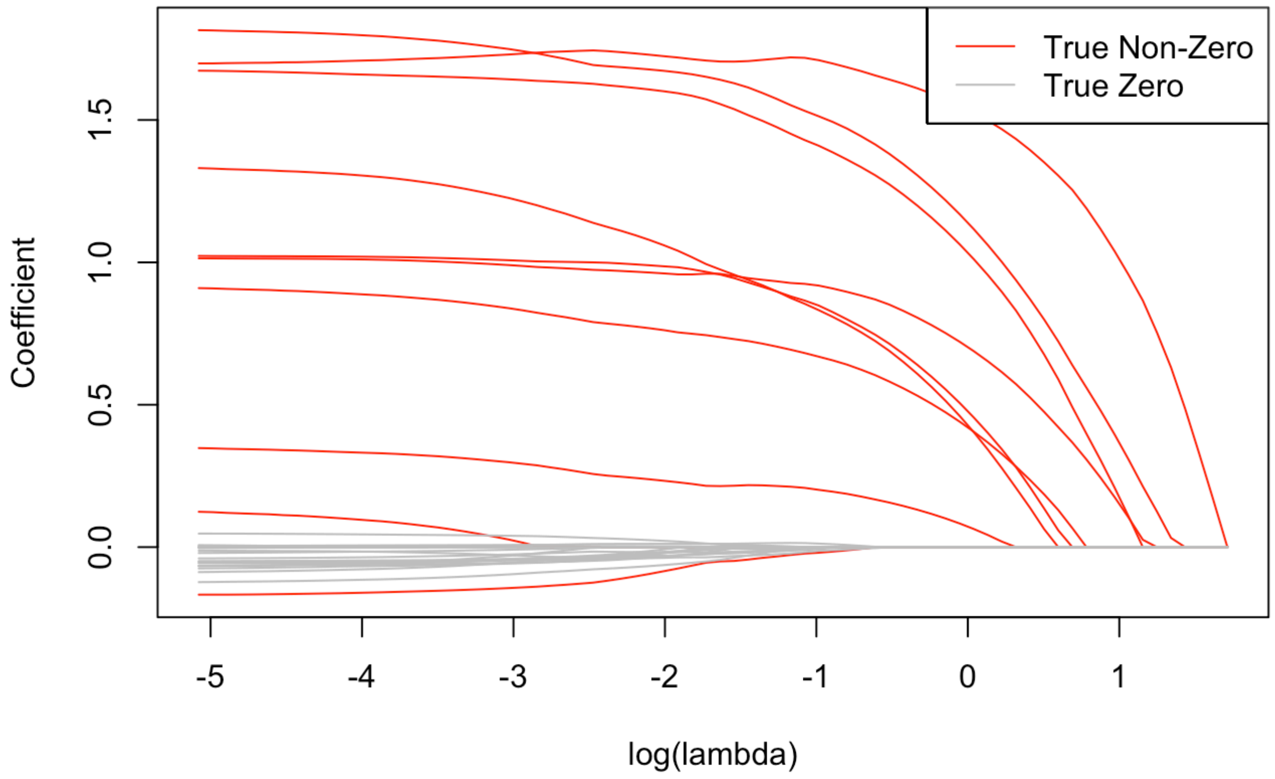
- Think of XY-plane as all possible values of $\beta = (\beta_1, \beta_2)$
- Without the penalty $h(\beta_1, \beta_2) = \frac{1}{n}\sum_{i=1}^{n}(y_i - X_i^T\beta)^2$
  - because h is quadratic, it forms a parabolid (https://en.wikipedia.org/wiki/Paraboloid)
  - Want to find values of (β_1, β_2) that minimize h
- $g(\beta_1, \beta_2) = \|\beta\|_1$ is an upside down pyramid with its bottom point at the origin
  - the level contours create empty diamond-shapes ($\{(x_1, x_2) : \|(x_1, x_2)\|_1 = z\}$ is a diamond-shape)
- By adding these two functions, the optimization must balance the contribution from each
  - h wants the paraboid min and g wants the origin
  - Because g has a sharp point, the lowest contour of h will likely hit there first



Elastic Net

$$\hat\beta = \arg\min_{\beta} \frac{1}{n}\sum_{i=1}^{n}\left(y_i - X_i^T\beta\right)^2 + \lambda\left[(1-\alpha)\|\beta\|_2^2/2 + \alpha|\beta\|_1\right]$$

```
l5_fit <- glmnet(design_mat, y, family="gaussian", alpha=0.5)
glmnet_plot(l5_fit, num_active_vars)
```



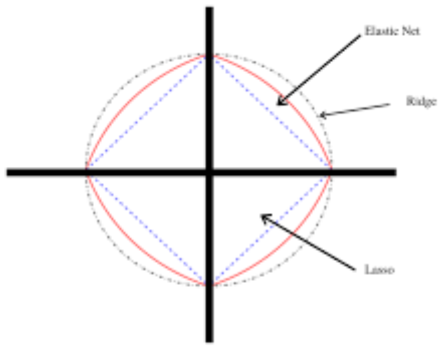


image from here (https://corporatefinanceinstitute.com/resources/knowledge/other/elastic-net/)

- L_1 penalty generates sparse model
- L_2 penalty
  - number of selected variables not bounded by n
  - stabilizes L_1 regularization path

Penalized Regression as Bayesian Method

- Bayes' Theorem

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

where $P(E) \neq 0$

- Note: Bayes' theorem is also frequently used outside of Bayesian methods
- In Bayesian inference, the goal is to estimate $P(H|E)$ by estimating $P(E|H)$ and $P(H)$
- $P(H|E)$, the *posterior probability*, is the probability H given evidence (data) E
- $P(H)$, the *prior probability*, is the probability of H before evidence (data) is observed
- $P(E|H)$ is the *likelihood*
- $P(E)$ is the marginal likelihood but is frequently used to make sure that the probability sums to one
- In Bayesian inference, we want to estimate the parameter, θ, to be a random variable, with a prior distribution, $\pi(\theta)$
- We use data, x, to improve our estimate of θ,

$$\pi(\theta|x) = \frac{f(x|\theta)\pi(\theta)}{m(x)}$$

where $m(x) = \int f(x|\theta)\pi(\theta)d\theta$

- Example: Let $X_1, \dots, X_n \sim \text{Bernoulli}(p)$ and $Y = \sum_i X_i$ (binomial random variable). Goal: Estimate p.
  - Assume $p \sim \text{Beta}(\alpha, \beta)$ is a prior distribution for p. That is,

$$\pi(p) = \frac{p^{\alpha-1}(1-p)^{\beta-1}}{B(\alpha,\beta)}$$

where B is the beta function. Then

$$\pi(p|Y) \propto \underbrace{\binom{n}{Y}p^Y(1-p)^{n-Y}}_{\text{likelihood}} \times \underbrace{\frac{p^{\alpha-1}(1-p)^{\beta-1}}{B(\alpha,\beta)}}_{\text{prior}} \propto p^{Y+\alpha-1}(1-p)^{n-Y+\beta-1}$$

Then

$$p|Y \sim \text{Beta}(Y+\alpha, n-Y+\beta)$$

so that

$$\hat{p} = \frac{Y+\alpha}{\alpha+\beta+n}$$

using the mean of a Beta distributed (https://en.wikipedia.org/wiki/Beta_distribution) random variable

- Penalized regression was introduced as a frequentist method in 1996
- Park and Casella (https://people.eecs.berkeley.edu/~jordan/courses/260-spring09/other-readings/park-casella.pdf) showed that it was equivalent to a Bayesian paradigm where parameters have a mean-zero Gaussian prior (ridge regression) or a meanzero-Laplace prior (lasso)
- Lasso as an optimization:

$$\beta_{\text{Lasso}} = \arg\min_{\beta} \|y - X\beta\|^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- Lasso as Bayesian: $y \sim \text{Gaussian}(X\beta, \sigma^2 I_n)$ with prior, $\beta_j \sim \frac{\lambda}{2\sigma} e^{-\lambda|\beta_j|/\sigma}$ (Laplace distribution)
- Ridge as optimization:

$$\beta_{\text{Ridge}} = \arg\min_{\beta} \|y - X\beta\|^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- Ridge as Bayesian: $y \sim \text{Gaussian}(X\beta, \sigma^2 I_n)$ with prior, $\beta_j \sim \text{Gaussian}(0, \lambda^{-1})$ (Gaussian distribution)

$$\begin{aligned}\text{Posterior Distribution} &= \underbrace{\prod_{i=1}^{n} \text{Gaussian}(y_n | X\beta, \sigma^2 I_n)}_{\text{likelihood}} \times \underbrace{\text{Gaussian}(\beta | 0, \lambda^{-1} I_p)}_{\text{prior}} \\ &\propto \exp\left(-\sum_{i=1}^{n} \frac{(y_i - X_i\beta)^2}{\sigma}\right) \times \exp\left(-\lambda \sum_{j=1}^{p} \beta_j^2\right) \\ &= \exp\left(-\sum_{i=1}^{n} \frac{(y_i - X_i\beta)^2}{\sigma} - \lambda \sum_{j=1}^{p} \beta_j^2\right) \sim \text{Gaussian}\end{aligned}$$

- Using standard methods to optimize this over β, we get a similar result to β_{Ridge} (λ must be scaled to accomodate σ)

Overfitting and Model Validation

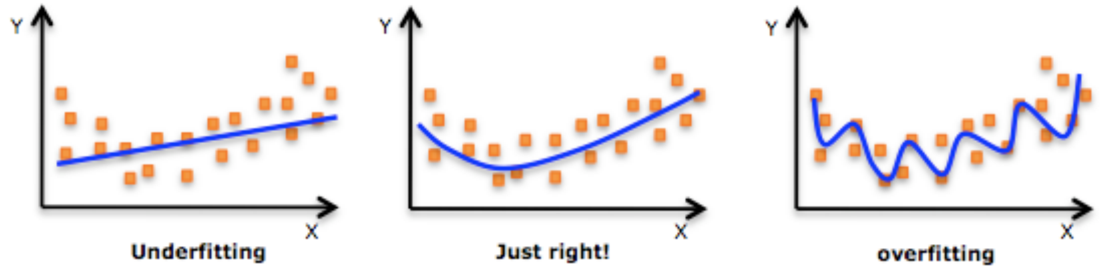


image from here (https://www.quora.com/What-are-the-key-trade-offs-between-overfitting-and-underfitting)

- Data generation paradigm: $Y = f(X) + \epsilon$ = signal + random noise
- prediction goal: estimate f with a model $\hat{f}$
- Overfitting is including random noise into a prediction model
- Complex models are more likely to overfit
- Simple models are more likely to underfit (miss signal)
- Issue: how much complexity is too much?
- Question: Does a large λ correspond more or less complexity?
- How do we assess fit?

Validation

- If a model has good fit, it should be able to make accurate predictions on new data
- Held out validation:
    1. Randomly split data into training set and test set (why random?)
    2. Fit many different models on the training set
    3. Use each model to make predictions on test set
    4. Evaluate predictions and choose simplest model with most accurate predictions

```
# this code is for learning
# don't do it this way in practice
set.seed(1234)
# step 1
test_ind <- sample(1:sample_size, 10, replace=FALSE)
test_ind
```

```
##  [1] 28 16 22 37 44  9  5 38 49  4
```

```
train_ind <- (1:sample_size)[! 1:sample_size %in% test_ind]
train_ind
```

```
##  [1]  1  2  3  6  7  8 10 11 12 13 14 15 17 18 19 20 21 23 24 25 26 27 29 30 31
## [26] 32 33 34 35 36 39 40 41 42 43 45 46 47 48 50
```

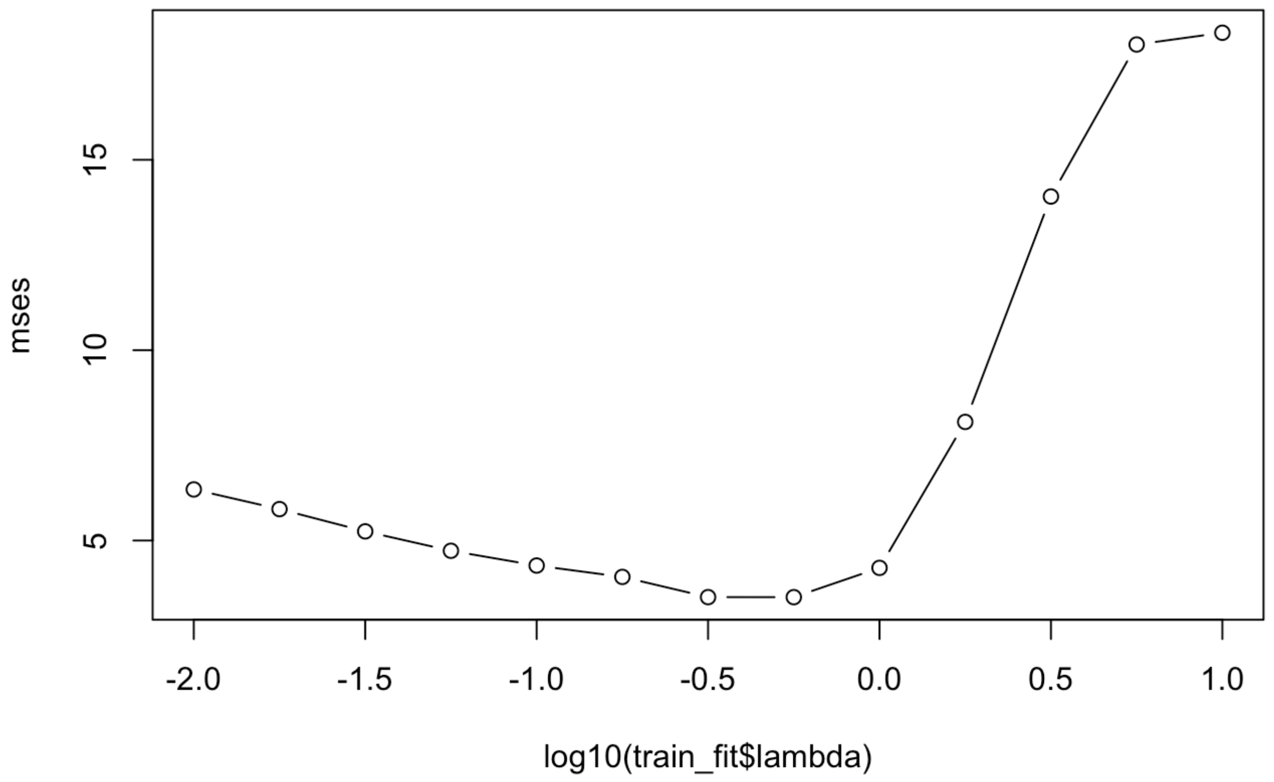
```
# step 2
lambdas <- 10^(seq(-2, 1, by=0.25))
train_fit <- glmnet(design_mat[train_ind,], y[train_ind], family="gaussian", alpha=0.5, lam
bda=lambdas)

# step 3
test_preds <- predict(train_fit, newx=design_mat[test_ind,], type='response')
test_preds
```

```
##            s0        s1          s2          s3         s4          s5
##  [1,] 1.03722 0.9799444  0.04124718 -2.13321088 -4.6132609 -5.89159016
##  [2,] 1.03722 1.0080332  0.70093102  0.09122676  0.2152240  0.07747343
##  [3,] 1.03722 1.0869251  2.13017022  4.12973971  4.6224140  4.67001125
##  [4,] 1.03722 1.0426066  1.14097945  1.28362668  0.2544462 -0.46344845
##  [5,] 1.03722 1.0546809  1.29727505  2.01094871  2.8265896  3.20813284
##  [6,] 1.03722 0.9046631 -1.15138925 -4.52055477 -6.8662968 -8.35131462
##  [7,] 1.03722 1.0583218  1.16285464 -0.20734041 -1.1782192 -1.71675234
##  [8,] 1.03722 0.9044524 -1.08515044 -2.23548545 -2.4349246 -2.67741218
##  [9,] 1.03722 1.1011828  2.03163064  2.10824104  1.8438863  1.62696710
## [10,] 1.03722 1.0479090  1.16670290  1.48476698  1.3522061  1.40061480
##                s6         s7          s8          s9         s10         s11
##  [1,] -6.49116681 -6.9638938  -7.2961855  -7.3165097  -7.2382415  -6.9887669
##  [2,]  0.01922609 -0.1142625  -0.4268567  -0.9615482  -1.4507142  -1.8457500
##  [3,]  4.57018627  4.3095933   3.9242865   3.3545116   2.8950431   2.5890005
##  [4,] -0.63554424 -0.4637328  -0.7591467  -1.0338734  -1.0972837  -1.1121568
##  [5,]  3.48088787  3.7975922   4.0542326   3.8981791   3.7690801   3.6036003
##  [6,] -9.12185418 -9.7381413 -10.1625744 -10.2693985 -10.2798794 -10.2292054
##  [7,] -2.10413303 -2.1243873  -1.9059435  -1.7056224  -1.4587594  -1.3625702
##  [8,] -2.78641409 -3.0023223  -3.1662041  -3.3177372  -3.4355102  -3.7107626
##  [9,]  1.49403911  1.5290357   1.2380084   0.8888110   0.6175895   0.4505577
## [10,]  1.46239504  1.2137308   0.9433404   0.8284501   0.7543063   0.6846772
##               s12
##  [1,]  -6.6470625
##  [2,]  -2.1718833
##  [3,]   2.3138798
##  [4,]  -1.2035047
##  [5,]   3.4399662
##  [6,] -10.1478933
##  [7,]  -1.3544433
##  [8,]  -3.9677990
##  [9,]   0.3609254
## [10,]   0.6304905
```

```
# step 4
# calculating mean squared error for each model
mses <- apply(test_preds, 2, function(x) mean((x-y[test_ind])^2))
cbind(mses, train_fit$lambda) # showing MSE with lambda value
```

```
##          mses
## s0  18.337272 10.00000000
## s1  18.028366  5.62341325
## s2  14.035926  3.16227766
## s3   8.115448  1.77827941
## s4   4.281505  1.00000000
## s5   3.512273  0.56234133
## s6   3.513439  0.31622777
## s7   4.044139  0.17782794
## s8   4.342859  0.10000000
## s9   4.730743  0.05623413
## s10  5.240411  0.03162278
## s11  5.825288  0.01778279
## s12  6.341917  0.01000000
```

```
plot(log10(train_fit$lambda), mses, type='b')
```



```
coef(train_fit, s=0.56234133) # coefficients for chosen model
```

```
## 31 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -0.84449896
## V1           .
## V2           1.12811732
## V3           0.67991453
## V4           0.64046300
## V5           1.40340133
## V6           1.98831224
## V7          -0.04047174
## V8           0.09139017
## V9           0.87757184
## V10          0.39330737
## V11          .
## V12          .
## V13         -0.04207614
## V14          0.02194223
## V15          .
## V16          .
## V17          .
## V18          .
## V19          .
## V20          .
## V21          .
## V22          .
## V23          0.03878893
## V24          .
## V25          .
## V26          0.08983453
## V27          .
## V28          .
## V29          .
## V30          .
```

```
# Comparing to the real betas
true_beta
```

```
##  [1] 0.22740682 1.24459881 1.21854947 1.24675888 1.72183077 1.28062121
##  [7] 0.01899151 0.46510101 1.33216752 1.02850228
```

Cross Validation

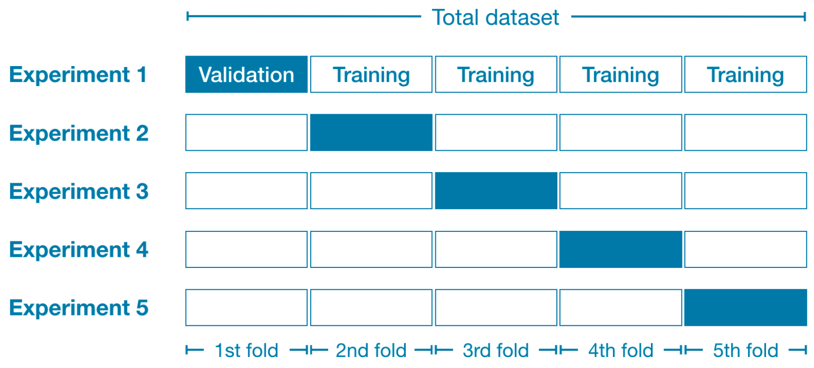


image from here (https://www.kaggle.com/alexisbcook/cross-validation)

- Normally used over held out validation
- Runs held out validation k-fold times Steps:

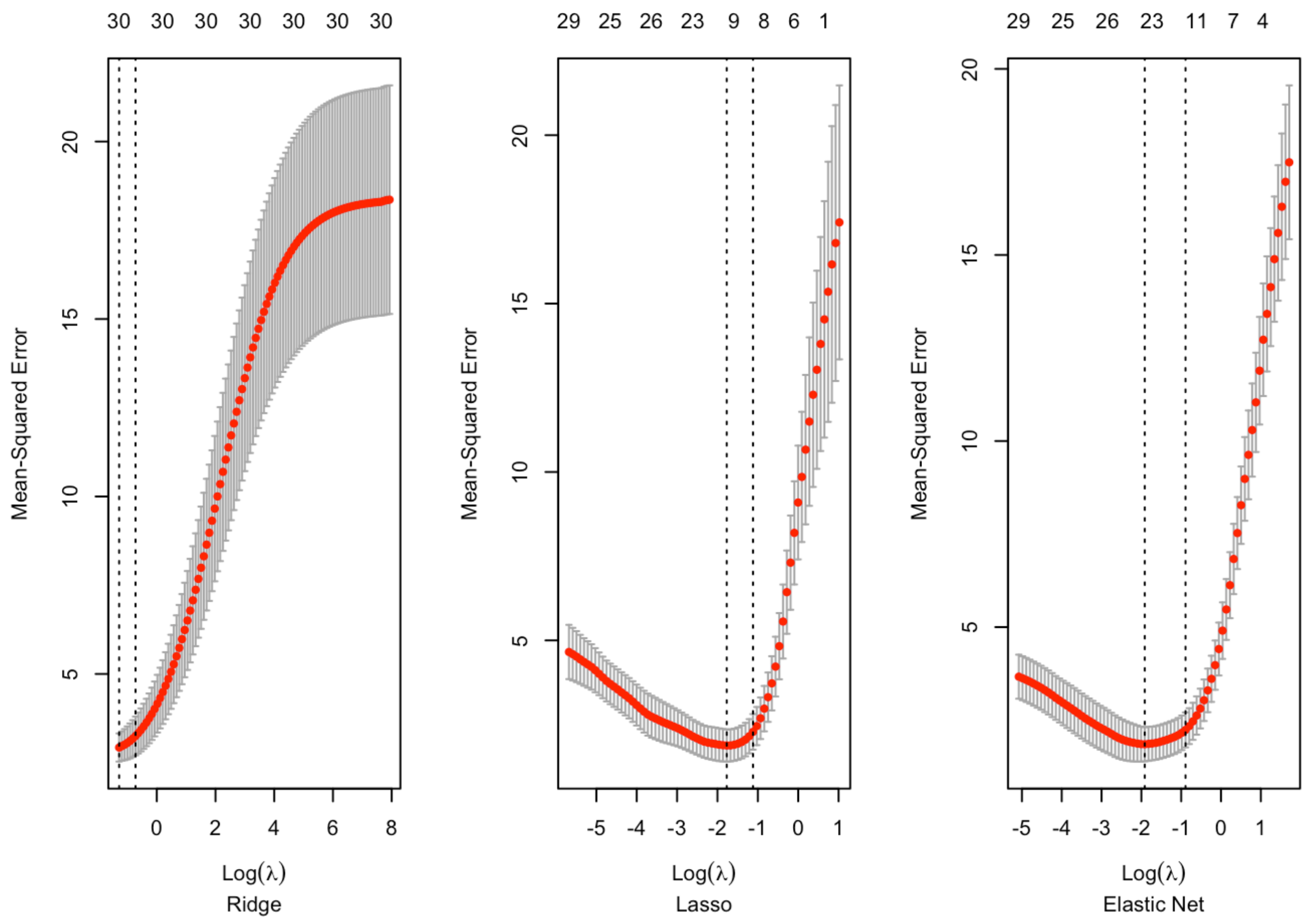
1. Partition data into k folds
2. Run held out validation k times:

i. On step j, train model on all folds with the jth fold removed
ii. Run predictions with trained model on jth fold
iii. Save predictions

3. Evaluate model using loss function

- glmnet does all of this for you

```
cv_fit0 <- cv.glmnet(design_mat, y, family="gaussian", alpha=0, nfolds=5)
cv_fit1 <- cv.glmnet(design_mat, y, family="gaussian", alpha=1, nfolds=5)
cv_fit5 <- cv.glmnet(design_mat, y, family="gaussian", alpha=0.5, nfolds=5)

par(mfrow=c(1,3))
plot(cv_fit0, sub='Ridge')
plot(cv_fit1, sub='Lasso')
plot(cv_fit5, sub='Elastic Net')
```

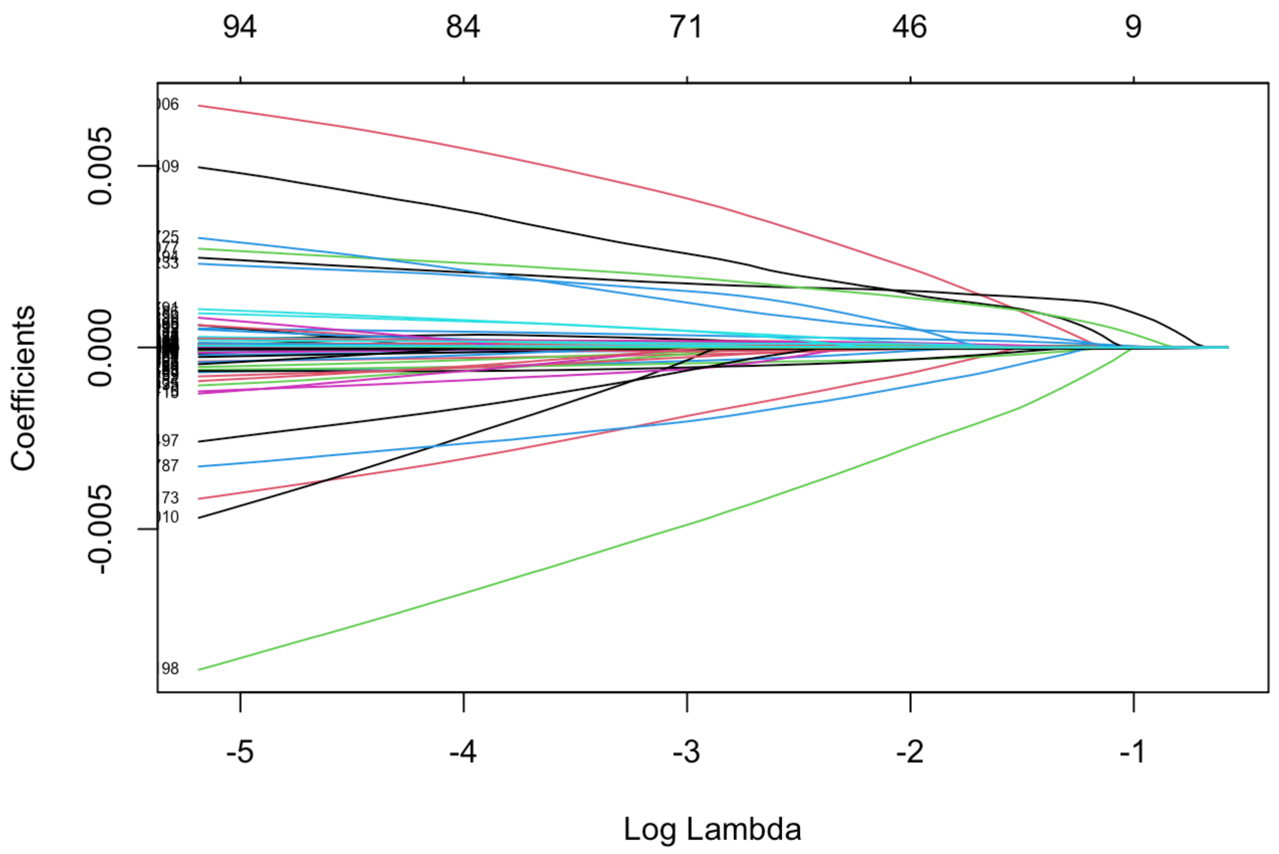


- Number above plot indicates number of nonzero coefficients at current λ

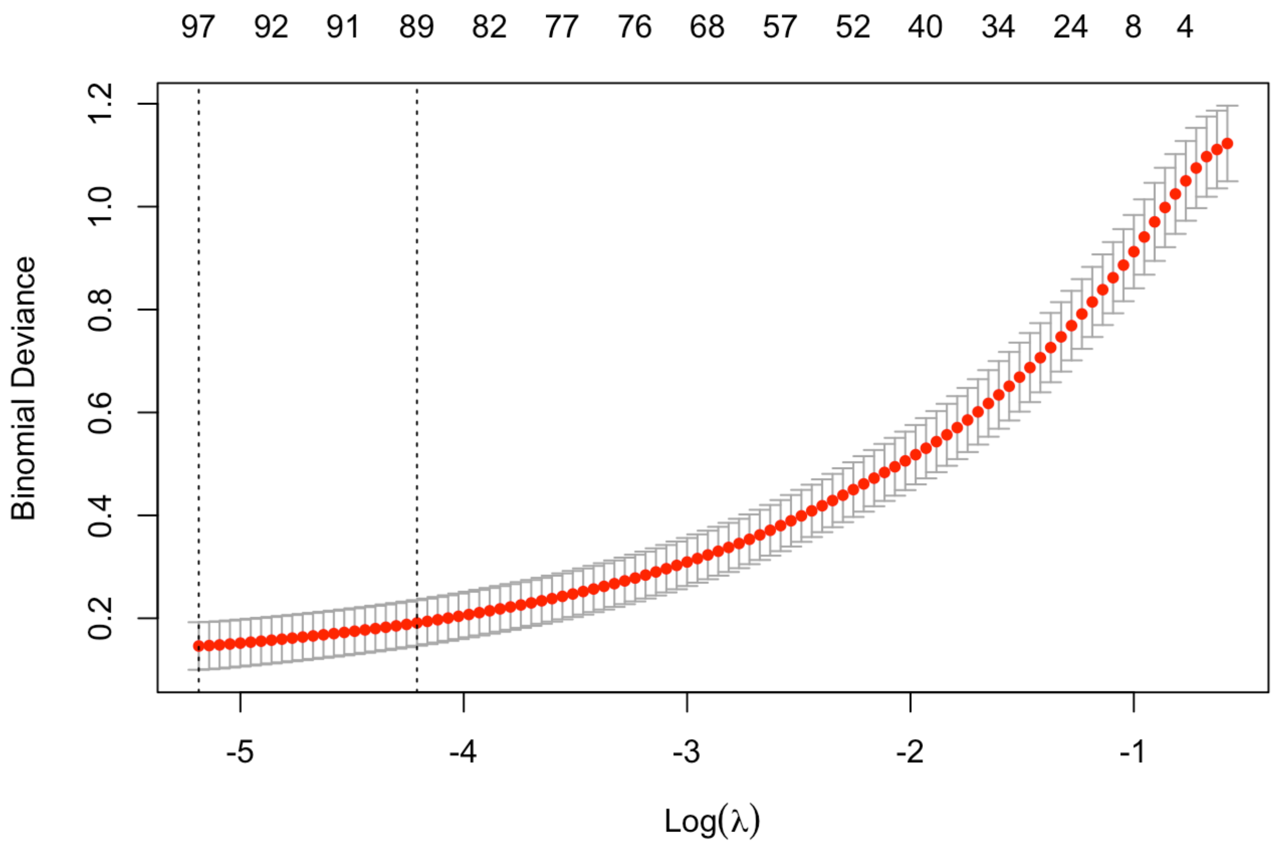
Gene data with penalized regression

Which penalized regression model should we use?

```
gene_mat <- as.matrix(shipp$x)
gene_fit <- glmnet(gene_mat, shipp$y, family="binomial", alpha=0.5)
plot(gene_fit, xvar='lambda', label=TRUE)
```



```
gene_fit_cv <- cv.glmnet(gene_mat, shipp$y, family="binomial", alpha=0.5, nfolds=5)
plot(gene_fit_cv)
```



97 92 91 89 82 77 76 68 57 52 40 34 24 8 4
Binomial Deviance
0.2
0.4
0.6
0.8
1.0
1.2
-5
-4
-3
-2
-1
Log(λ)