Data Science and Predictive Analytics (UMich HS650)

Regularized Linear Modeling and Controlled Variable Selection

SOCR/MIDAS (Ivo Dinov)

July 2017

Many biomedical and biosocial studies involve large amounts of complex data, including cases where the number of features (k) is large and may exceed the number of cases (n). In such situations, parameter estimates are difficult to compute or may be unreliable as the system is underdetermined (https://en.wikipedia.org/wiki/Underdetermined_system). Regularization provides one approach to improve model reliability, prediction accuracy, and result interpretability. It is based on augmenting the fidelity term of the objective function used in the model fitting process with a regularization term that provides restrictions on the parameter space.

Classical techniques for choosing *important* covariates to include in a model of complex multivariate data relied on various types of stepwise variable selection processes, see Chapter 16 (http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650/notes/16_FeatureSelection.html). These tend to improve prediction accuracy in certain situations, e.g., when a small number of features are strongly predictive, or associated, with the clinical outcome or biosocial trait. However, the prediction error may be large when the model relies purely on a fidelity term. Including a regularization term in the optimization of the cost function improves the prediction accuracy. For example, below we show that by shrinking large regression coefficients,

ridge regularization reduces overfitting and improves prediction error. Similarly, the least absolute shrinkage and selection operator (LASSO) employs regularization to perform simultaneous parameter estimation and variable selection. LASSO enhances the prediction accuracy and provides a natural interpretation of the resulting model. *Regularization* refers to forcing certain characteristics of model-based scientific inference, e.g., discouraging complex models or extreme explanations, even if they fit the data, enforcing model generalizability to prospective data, or restricting model overfitting of accidental samples.

In this chapter, we extend the mathematical foundation we presented in Chapter 4 (http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650/notes/04_LinearAlgebraMatrixComputing.html) and (1) discuss computational protocols for handling complex high-dimensional data, (2) illustrate model estimation by controlling the false-positive rate of selection of salient features, and (3) derive effective forecasting models.

1 Questions

- How to deal with extremely high-dimensional data (hundreds or thousands of features)?
- Why mix fidelity (model fit) and regularization (model interpretability) terms in objective function optimization?
- How to reduce the false-positive rate, increase scientific validation, and improve result reproducibility (e.g., Knockoff filtering)?

2 Matrix notation

We should review the basics of matrix notation, linear algebra, and matrix computing we covered in Chapter 4 (http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650/notes/04_LinearAlgebraMatrixComputing.html). At the core of matrix manipulations are scalars, vectors and matrices.

- y_i: output or response variable, $i = 1, \ldots, n$ (cases/subjects).
- x_{ij}: input, predictor, or feature variable, $1 \leq j \leq k,\ 1 \leq i \leq n$.

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad ,$$

and

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,k} \end{pmatrix}.$$

3 Regularized Linear Modeling

If we assume that the covariates are orthonormal, i.e., we have a special kind of a *design matrix* $X^T X = I$, then:

- The ordinary least squares (`OLS`) estimates minimize

$$\min_{\beta \in \mathbb{R}^k} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 \right\},$$

and are defined by

$$\hat{\beta}^{OLS} = (X^T X)^{-1} X^T y = X^T y,$$

- `LASSO` estimates minimize

$$\min_{\beta \in \mathbb{R}^k} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right\},$$

and are defined as a soft-threshold function of the OLS estimates:

$$\hat{\beta}_j = S_{N\lambda}(\hat{\beta}_j^{\text{OLS}}) = \hat{\beta}_j^{\text{OLS}} \max\left(0, 1 - \frac{N\lambda}{|\hat{\beta}_j^{OLS}|}\right),$$

where $S_{N\lambda}$ is a soft thresholding operator translating values towards zero, instead of setting smaller values to zero and leaving larger ones untouched as the hard thresholding operator would.

- `Ridge` regression, where the objective is to minimize:

$$\min_{\beta \in \mathbb{R}^k} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \right\},$$

which yields estimates $\hat{\beta}_j = (1 + N\lambda)^{-1} \hat{\beta}_j^{OLS}$. Thus, ridge regression shrinks all coefficients by a uniform factor, $(1 + N\lambda)^{-1}$, and does not set any coefficients to zero.

- `Best subset` selection regression, orthogonal matching pursuit (OMP), minimizing:

$$\min_{\beta \in \mathbb{R}^k} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 + \lambda \|\beta\|_0 \right\},$$

where $\|.\|_0$ is the "ℓ^0 norm", defined as $|z| = m$ if exactly m components of z are nonzero. In this case, the estimates are:

$$\hat{\beta}_j = H_{\sqrt{N\lambda}}\left(\hat{\beta}_j^{OLS}\right) = \hat{\beta}_j^{OLS} I\left(\left|\hat{\beta}_j^{OLS}\right| \geq \sqrt{N\lambda}\right),$$

where H_α is a `hard-thresholding` function and I is an indicator function (it is 1 if its argument is true, and 0 otherwise).

The LASSO estimates share features of the estimates from both `ridge` and `best` subset selection regression since they both shrink the magnitude of all the coefficients, like ridge regression, but also set some of them to zero, as in the best subset selection case. `Ridge` regression scales all of the coefficients by a constant factor, whereas LASSO translates the coefficients towards zero by a constant value and sets them to zero if they reach it.

3.1 Ridge Regression

Ridge regression (https://en.wikipedia.org/wiki/Tikhonov_regularization) relies on L^2 regularization to improve the model prediction accuracy. It improves prediction error by shrinking large regression coefficients and reduce overfitting. By itself, ridge regularization does not perform variable selection and does not really help with model interpretation.

Let's look at one example using one of our datasets 01a_data.txt (https://umich.instructure.com/courses/38100/files/folder/data).

```
# Data: https://umich.instructure.com/courses/38100/files/folder/data   (01a_data.txt)
data <- read.table('https://umich.instructure.com/files/330381/download?download_frd=1', as.is=T, header=T)
attach(data); str(data)
```

```
## 'data.frame':    1034 obs. of  6 variables:
##  $ Name    : chr  "Adam_Donachie" "Paul_Bako" "Ramon_Hernandez" "Kevin_Millar" ...
##  $ Team    : chr  "BAL" "BAL" "BAL" "BAL" ...
##  $ Position: chr  "Catcher" "Catcher" "Catcher" "First_Baseman" ...
##  $ Height  : int  74 74 72 72 73 69 69 71 76 71 ...
##  $ Weight  : int  180 215 210 210 188 176 209 200 231 180 ...
##  $ Age     : num  23 34.7 30.8 35.4 35.7 ...
```

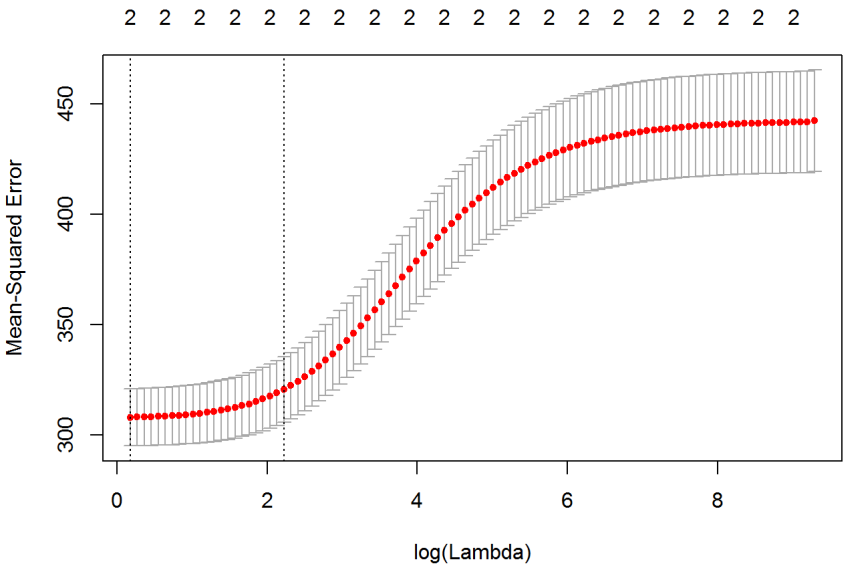
```
# Training Data
# Full Model: x <- model.matrix(Weight ~ ., data = data[1:900, ])
# Reduced Model
x <- model.matrix(Weight ~ Age + Height, data = data[1:900, ])
# creates a design (or model) matrix, and adds 1 column for outcome according to the formula.
y <- data[1:900, ]$Weight

# Testing Data
x.test <- model.matrix(Weight ~ Age + Height, data = data[901:1034, ])
y.test <- data[901:1034, ]$Weight

# install.packages("glmnet")
library("glmnet")
```

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

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-5
```

```
cv.ridge <-  cv.glmnet(x, y, type.measure = "mse", alpha = 0)
## alpha =1 for lasso only, alpha = 0 for ridge only, and 0<alpha<1 to blend ridge & lasso penalty !!!!
plot(cv.ridge)
```



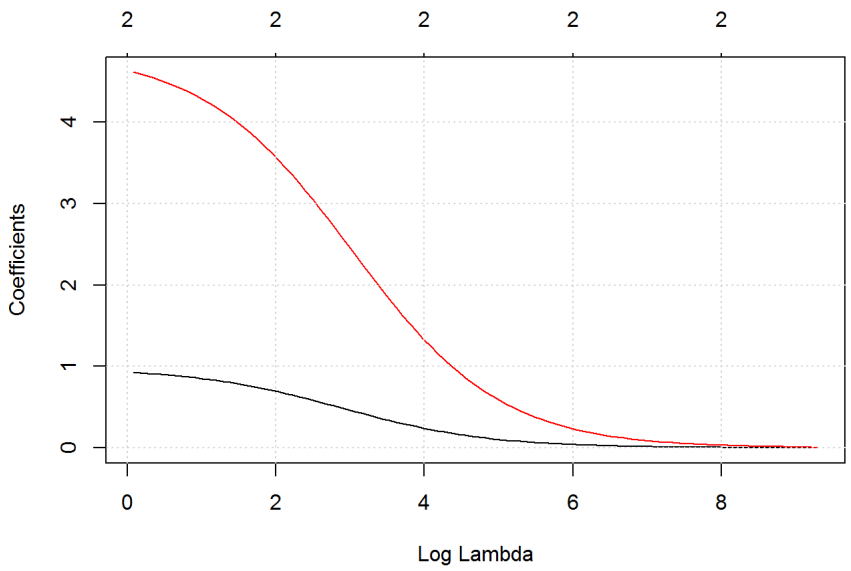
```
coef(cv.ridge)
```

```
## 4 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -63.5965544
## (Intercept)   .
## Age           0.6472381
## Height        3.3469182
```

```
sqrt(cv.ridge$cvm[cv.ridge$lambda == cv.ridge$lambda.1se])
```

```
## [1] 17.90412
```

```
#plot variable feature coefficients against the shrinkage parameter lambda.
glmmod <-glmnet(x, y, alpha = 0)
plot(glmmod, xvar="lambda")
grid()
```

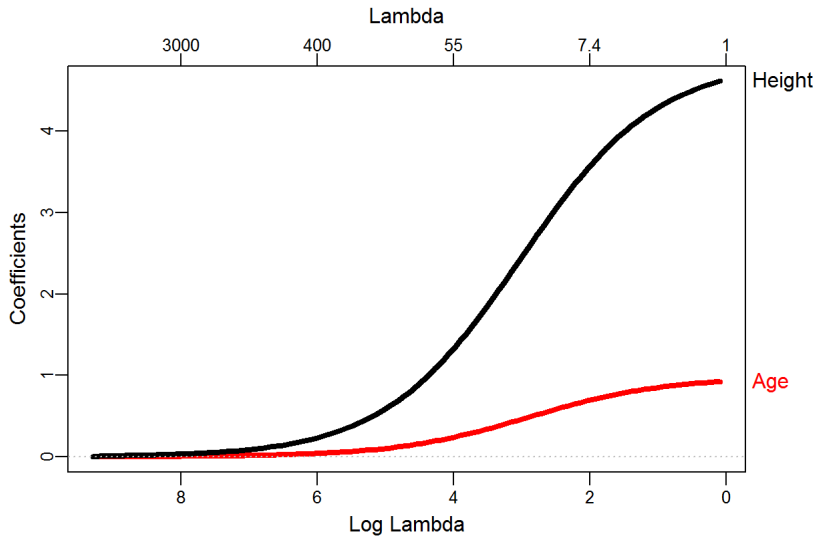



```
# for plot_glmnet with ridge/lasso coefficient path labels
# install.packages("plotmo")
library(plotmo)
```

```
## Loading required package: plotrix
```

```
## Loading required package: TeachingDemos
```

```
plot_glmnet(glmmod, lwd=4)        #default colors
```




```
# More elaborate plots can be generated using:
# plot_glmnet(glmmod,label=2,lwd=4) #Label the 2 biggest final coefs
# specify color of each line
# g <- "blue"
# plot_glmnet(glmmod, lwd=4, col=c(2,g))


# report the model coefficient estimates
coef(glmmod)[, 1]
```

```
##  (Intercept)  (Intercept)          Age       Height
## 2.016556e+02 0.000000e+00 8.327372e-37 4.789383e-36
```

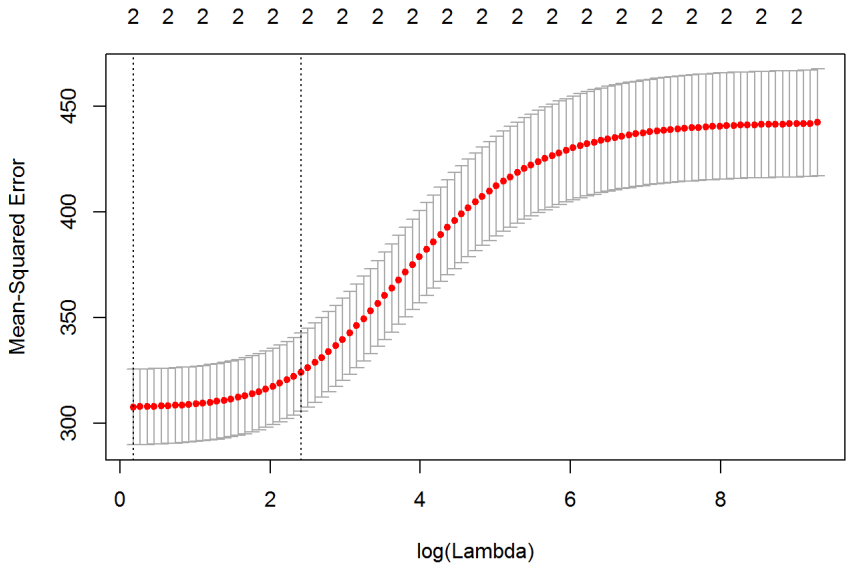
```
cv.glmmod <- cv.glmnet(x, y, alpha=0)

mod.ridge <-  cv.glmnet(x, y, alpha = 0, thresh = 1e-12)
lambda.best <-  mod.ridge$lambda.min
lambda.best
```

```
## [1] 1.192177
```

```
ridge.pred <-  predict(mod.ridge, newx = x.test, s = lambda.best)
ridge.RMS <- mean((y.test - ridge.pred)^2); ridge.RMS
```

```
## [1] 264.083
```

```
ridge.test.r2 <-  1 - mean((y.test - ridge.pred)^2)/mean((y.test - mean(y.test))^2)

plot(cv.glmmod)
```




```
best_lambda <- cv.glmmod$lambda.min
best_lambda
```

```
## [1] 1.192177
```

In the plots above, different colors represents the vector of features, and the corresponding coefficients, displayed as a function of the regularization parameter, λ. The top axis indicates the number of nonzero coefficients at the current value of λ. For LASSO regularization, this top-axis corresponds to the effective degrees of freedom (df) for the model.

Notice the usefulness of Ridge regularization for model estimation in highly ill-conditioned problems ($n << k$) where slight feature perturbations may cause disproportionate alterations of the corresponding weight calculations. When λ is very large, the regularization effect dominates the optimization of the objective function and the coefficients tend to zero. At the other extreme, as $\lambda \longrightarrow 0$, the resulting model solution tends towards the ordinary least squares (OLS) and the coefficients exhibit large oscillations. Often in practice we need to tune λ to balance this tradeoff.

Also note that in the `cv.glmnet` call, `alpha = 0` (ridge) and `alpha = 1` (LASSO) correspond to different types of regularization, and $0 < alpha < 1$ corresponds to *elastic net* blended regularization.

3.2 Least Absolute Shrinkage and Selection Operator (LASSO) Regression

Estimating the linear regression coefficients in a linear regression model using LASSO involves minimizing an objective function that includes an L^1 regularization term which tends to shrink the number of features. A descriptive representation of the fidelity (left) and regularization (right) terms of the objective function are shown below:

$$\underbrace{\sum_{i=1}^{n}\left[y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right]^2}_{\text{fidelity term}} + \underbrace{\lambda\sum_{j=1}^{p}|\beta_j|}_{\text{regilarization term}} \quad .$$

LASSO jointly achieves model quality, reliability and variable selection by penalizing the sum of the absolute values of the regression coefficients. This forces the shrinkage of certain coefficients effectively acting as a variable selection process. This is similar to ridge regression's penalty on the sum of the squares of the regression coefficients, although ridge regression only shrinks the magnitude of the coefficients without truncating them to 0.

Let's show how to select the regularization weight parameter λ using `training` data and report the error using `testing` data.

```
mod.lasso <-  cv.glmnet(x, y, alpha = 1, thresh = 1e-12)
## alpha =1 for lasso only, alpha = 0 for ridge only, and 0<alpha<1 for elastic net, a blend ridge & lasso penalty !!!!
lambda.best <- mod.lasso$lambda.min
lambda.best
```

```
## [1] 0.05933494
```

```
lasso.pred <- predict(mod.lasso, newx = x.test, s = lambda.best)
LASSO.RMS <- mean((y.test - lasso.pred)^2); LASSO.RMS
```

```
## [1] 261.8194
```

Let's retrieve the estimates of the model coefficients.

```
mod.lasso <-  glmnet(x, y, alpha = 1)
predict(mod.lasso, s = lambda.best, type = "coefficients")
```

```
## 4 x 1 sparse Matrix of class "dgCMatrix"
##                      1
## (Intercept) -181.9254079
## (Intercept)    .
## Age            0.9654354
## Height         4.8284803
```

```
lasso.test.r2 <-  1 - mean((y.test - lasso.pred)^2)/mean((y.test - mean(y.test))^2)
```

Perhaps obtain a classical OLS linear model, as well.

```
lm.fit <-  lm(Weight ~ Age + Height, data = data[1:900, ])
summary(lm.fit)
```

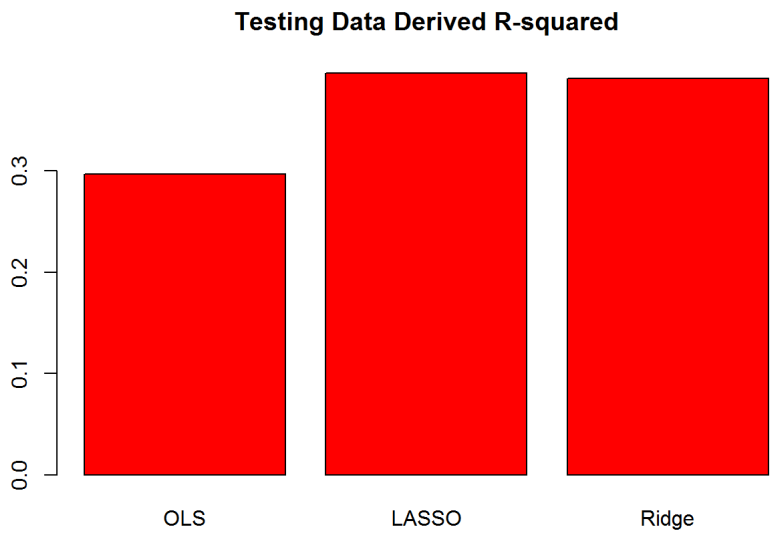
```
##
## Call:
## lm(formula = Weight ~ Age + Height, data = data[1:900, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -50.602 -12.399  -0.718  10.913  74.446
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -184.3736    19.4232  -9.492  < 2e-16 ***
## Age            0.9799     0.1335   7.341 4.74e-13 ***
## Height         4.8561     0.2551  19.037  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17.5 on 897 degrees of freedom
## Multiple R-squared:  0.3088, Adjusted R-squared:  0.3072
## F-statistic: 200.3 on 2 and 897 DF,  p-value: < 2.2e-16
```

The OLS linear (unregularized) model has slightly larger coefficients and greater MSE than LASSO, which attests to the shrinkage of LASSO.

```
lm.pred <-  predict(lm.fit, newx = x.test)
LM.RMS <- mean((y - lm.pred)^2); LM.RMS
```

```
## [1] 305.1995
```

```
lm.test.r2 <-  1 - mean((y - lm.pred)^2) / mean((y.test - mean(y.test))^2)

barplot(c(lm.test.r2, lasso.test.r2, ridge.test.r2), col = "red", names.arg = c("OLS", "LASSO", "Ridge"), main = "Testing Data Derive
d R-squared")
```



Compare the results of the three alternative models (LM, LASSO and Ridge) for these data and contrast the derived RSS results.

```
library(knitr) #  kable function to convert tabular R-results into Rmd tables
# create table as data frame
RMS_Table = data.frame(LM=LM.RMS, LASSO=LASSO.RMS, Ridge=ridge.RMS)

# convert to markdown
kable(RMS_Table, format="pandoc", caption="Test Dataset RSS Results", align=c("c", "c", "c"))
```

Test Dataset RSS Results

LM	LASSO	Ridge
305.1995	261.8194	264.083

As both the *inputs* (features or predictors) and the *output* (response) are observed for the testing data, we can build a learner examining the relationship between the two types of features (controlled covariates and observable responses). Most often, we are interested in forecasting s or predicting of responses using prospective (new, testing, or validation) data.

4 Predictor Standardization

Prior to fitting regularized linear modeling and estimating the effects, covariates may be standardized. This can be accomplished by using the classic "z-score" formula (http://wiki.socr.umich.edu/index.php/AP_Statistics_Curriculum_2007_Normal_Prob#General_Normal_Distribution%5D). This puts each predictor on the same scale (unitless quantities) - the mean is 0 and the variance is 1. We use $\hat{\beta}_0 = \bar{y}$, for the mean intercept parameter, and estimate the coefficients or the remaining predictors. To facilitate interpretation of the model or results in the context of the specific case-study, we can transform the results back to the original scale/units after the model is estimated.

5 Estimation Goals

The basic setting here is: given a set of predictors X, find a function, $f(X)$, to model or predict the outcome Y.

Let's denote the objective (loss or cost) function by $L(y, f(X))$. It determines adequacy of the fit and allows us to estimate the squared error loss:

$$L(y, f(X)) = (y - f(X))^2.$$

We are looking to find f that minimizes the expected loss:

$$E[(Y - f(X))^2] \Rightarrow f = E[Y|X = x].$$

6 Linear Regression

Let's assume that:

- $$Y_i = \beta_0 + x_{i1}\beta_1 + x_{i2}\beta_2 + \cdots + x_{ip}\beta_p + \epsilon,$$
- In shorthand matrix notation, that is: $Y = X\beta$.
- And the expectation of the observed outcome given the data, $E[Y|X = x]$, is a linear function, which in certain situations can be expressed as:

$$\arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - x_i^T\beta).$$

Multiplying both hands sides by $X^T = X'$, the transpose of the design matrix X (on the left, as matrix multiplication is not always commutative), yields:

$$X^T Y = X^T(X\beta) = (X^T X)\beta.$$

To solve for the effect-sizes β, we can multiply both sides of the equation by the inverse of its (right hand side) multiplier:

$$(X^T X)^{-1}(X^T Y) = (X^T X)^{-1}(X^T X)\beta = \beta.$$

The *ordinary least squares (OLS)* estimate of β is given by:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 = \arg\min_{\beta} \|y - X\beta\|_2^2 \Rightarrow$$

$$\hat{\beta}^{OLS} = (X'X)^{-1}X'y \Rightarrow \hat{f}(x_i) = x_i'\hat{\beta}.$$

6.1 Drawbacks of Linear Regression

Despite its wide use and elegant theory, linear regression has some shortcomings.

- Prediction accuracy - Often can be improved upon;
- Model interpretability - Linear model does not automatically do variable selection.

6.2 Assessing Prediction Accuracy

Given a new input, x_0, how do we assess our prediction $\hat{f}(x_0)$?

The *Expected Prediction Error (EPE)* is:

$$\begin{aligned} EPE(x_0) &= E[(Y_0 - \hat{f}(x_0))^2] \\ &= \text{Var}(\epsilon) + \text{Var}(\hat{f}(x_0)) + \text{Bias}(\hat{f}(x_0))^2 \ . \\ &= \text{Var}(\epsilon) + MSE(\hat{f}(x_0)) \end{aligned}$$

where

- $\text{Var}(\epsilon)$: irreducible error variance

- $\text{Var}(\hat{f}(x_0))$: sample-to-sample variability of $\hat{f}(x_0)$, and
- $\text{Bias}(\hat{f}(x_0))$: average difference of $\hat{f}(x_0)$ & $f(x_0)$.

6.3 Estimating the Prediction Error

Common approach to estimating prediction error include:

- Randomly splitting the data into "training" and "testing" sets, where the testing data has m observations that will be used to independently validate the model quality. We estimate/calculate $\hat{f}$ using training data;
- Estimating prediction error using the *testing set MSE*

$$\hat{MSE}(\hat{f}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{f}(x_i))^2.$$

Ideally, we want our model/predictions to perform well with new or prospective data.

6.4 Improving the Prediction Accuracy

If $f(x) \approx \text{linear}$, $\hat{f}$ will have low bias but possibly high variance, e.g., in high-dimensional setting due to correlated predictors, over ($k \text{ features} << n \text{ cases}$), or underdetermination ($k > n$). The goal is to minimize total error by trading off bias and precision:

$$MSE(\hat{f}(x)) = \text{Var}(\hat{f}(x)) + \text{Bias}(\hat{f}(x))^2.$$

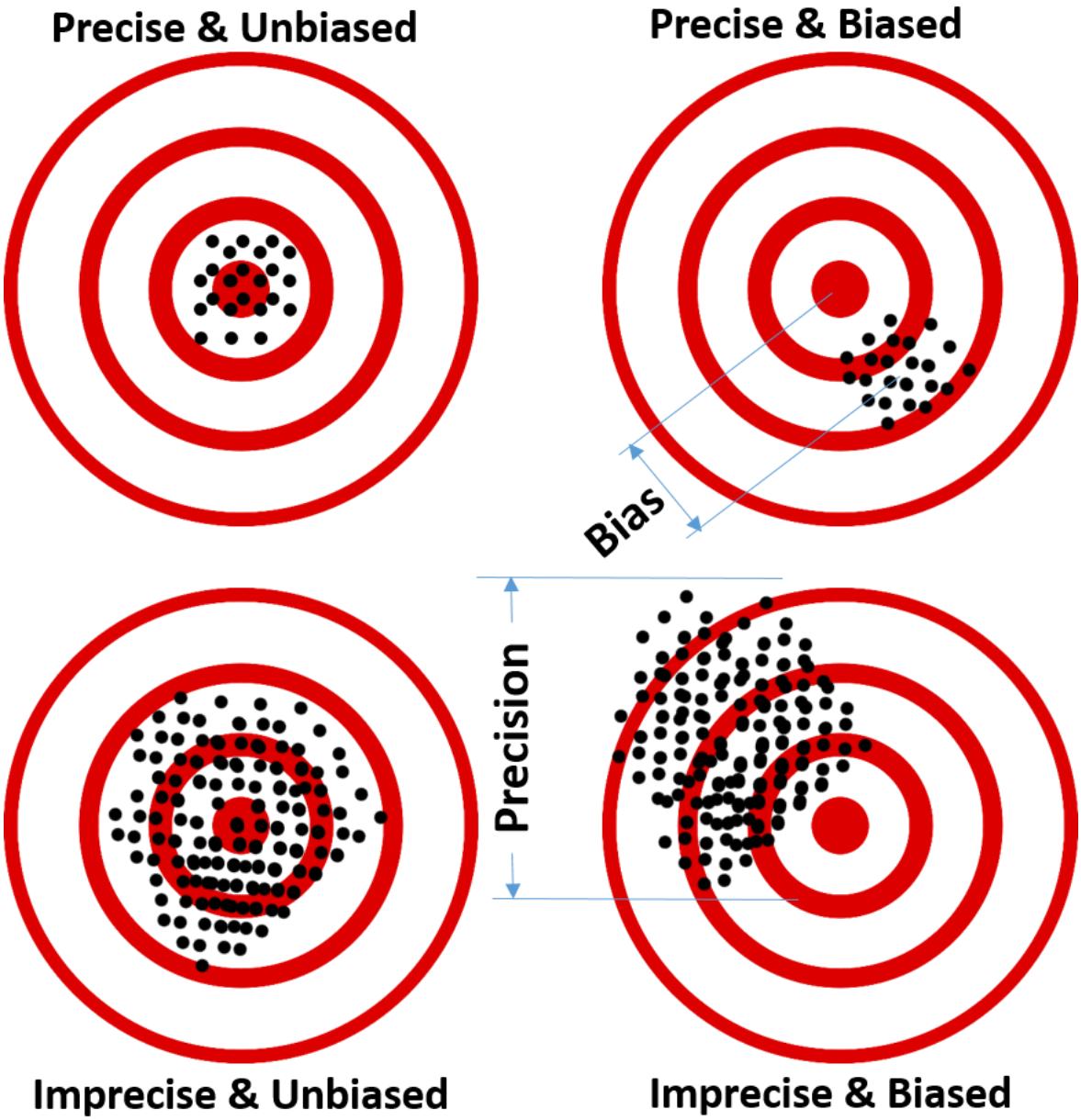
We can sacrifice bias to reduce variance, which may lead to decrease in MSE. So, regularization allows us to tune this tradeoff.

We aim to predict the outcome variable, $Y_{n\times 1}$, in terms of other features $X_{n,k}$. Assume a first-order relationship relating Y and X is of the form $Y = f(X) + \epsilon$, where the error term $\epsilon \sim N(0, \sigma)$. An estimate model $\hat{f}(X)$ can be computed in many different ways (e.g., using least squares calculations for linear regressions, Newton-Raphson, steepest decent and other methods). Then, we can decompose the expected squared prediction error at x as:

$$E(x) = E[(Y - \hat{f}(x))^2] = \underbrace{\left(E[\hat{f}(x)] - f(x)\right)^2}_{Bias^2} + \underbrace{E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right]}_{\text{precision (variance)}} + \underbrace{\sigma^2}_{\text{irreducible error (noise)}}.$$

When the true Y vs. X relation is not known, infinite data may be necessary to calibrate the model $\hat{f}$ and it may be impractical to jointly reduce both the model *bias* and *variance*. In general, minimizing the *bias* at the same time as minimizing the *variance* may not be possible.

The figure below illustrates diagrammatically the dichotomy between *bias* and *precision* (variance), additional information is available in the SOCR SMHS EBook (http://wiki.socr.umich.edu/index.php/SMHS_BiasPrecision).



6.5 Variable Selection

Oftentimes, we are only interested in using a subset of the original features as model predictors. Thus, we need to identify the most relevant predictors, which usually capture the big picture of the process. This helps us avoid overly complex models that may be difficult to interpret. Typically, when considering several models achieve similar results, it's natural to select the simplest of them.

Linear regression does not directly determine the importance of features to predict a specific outcome. The problem of selecting critical predictors is therefore very important.

Automatic feature subset selection methods should directly determine an optimal subset of variables. Forward or backward stepwise variable selection and forward stagewise are examples of classical methods for choosing the best subset by assessing various metrics like MSE, C_p, AIC, or BIC, see Chapter 16 (http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650/notes/16_FeatureSelection.html).

7 Regularization Framework

As before, we start with a given X and look for a (linear) function, $f(X)$, to model or predict y subject to certain objective cost function, e.g., squared error loss. Adding a second term to the cost function minimization process yields (model parameter) estimates expressed as:

$$\hat{\beta}(\lambda) = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda J(\beta) \right\}$$

In the above expression, $\lambda \geq 0$ is the regularization (tuning or penalty) parameter, $J(\beta)$ is a `user-defined penalty function` - typically, the intercept is not penalized.

7.0.1 Role of the *Penalty Term*

Consider $\arg\min J(\beta) = \sum_{j=1}^{k} \beta_j^2 = \|\beta\|_2^2$ (Ridge Regression, *RR*).

Then, the formulation of the regularization framework is:

$$\hat{\beta}(\lambda)^{RR} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{k} \beta_j^2 \right\}.$$

Or, alternatively:

$$\hat{\beta}(t)^{RR} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2,$$

subject to

$$\sum_{j=1}^{k} \beta_j^2 \leq t.$$

7.0.2 Role of the *Regularization Parameter*

The regularization parameter $\lambda \geq 0$ directly controls the bias-variance trade-off:

- $\lambda = 0$ corresponds to OLS, and
- $\lambda \to \infty$ puts more weight on the penalty function and results in more shrinkage of the coefficients, i.e., we introduce bias at the sake of reducing the variance.

The choice of λ is crucial and will be discussed below as each λ results in a different solution $\hat{\beta}(\lambda)$.

7.1 LASSO

The LASSO (Least Absolute Shrinkage and Selection Operator) regularization relies on:

$$\arg\min J(\beta) = \sum_{j=1}^{k} |\beta_j| = \|\beta\|_1,$$

which leads to the following objective function:

$$\hat{\beta}(\lambda)^{L} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \sum_{j=1}^{k} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{k} |\beta_j| \right\}.$$

In practice, subtle changes in the penalty terms frequently lead to big differences of the results. Not only does the regularization term shrink coefficients towards zero, but it sets some of them to be exactly zero. Thus, it performs continuous variable selection, hence the name, Least Absolute Shrinkage and Selection Operator (LASSO).

For further details, see the "Tibshirani's LASSO Page" (http://statweb.stanford.edu/~tibs/LASSO.html).

7.2 General Regularization Framework

The general regularization framework involves optimization of a more general objective function:

$$\min_{f \in H} \sum_{i=1}^{n} \left\{ L(y_i, f(x_i)) + \lambda J(f) \right\},$$

where $\mathcal{H}$ is a space of possible functions, L is the *fidelity term*, e.g., squared error, absolute error, zero-one, negative log-likelihood (GLM), hinge loss (support vector machines), and J is the *regularizer*, e.g., ridge regression, LASSO, adaptive LASSO, group LASSO, fused LASSO, thresholded LASSO, generalized LASSO, constrained LASSO, elastic-net, Dantzig selector, SCAD, MCP, smoothing splines (http://www.stat.cmu.edu/~ryantibs/papers/genlasso.pdf), etc.

This represents a very general and flexible framework that allows us to incorporate prior knowledge (sparsity, structure, etc.) into the model estimation.

8 Implementation of Regularization

8.1 Example: Neuroimaging-genetics study of Parkinson's Disease Dataset

More information about this specific study and the included derived neuroimaging biomarkers is available here (http://wiki.socr.umich.edu/index.php/SOCR_Data_PD_BiomedBigMetadata). A link to the data and a brief summary of the features are included below:

- 05_PPMI_top_UPDRS_Integrated_LongFormat1.csv (https://umich.instructure.com/files/330397/download?download_frd=1)
- Data elements include: FID_IID, L_insular_cortex_ComputeArea, L_insular_cortex_Volume, R_insular_cortex_ComputeArea, R_insular_cortex_Volume, L_cingulate_gyrus_ComputeArea, L_cingulate_gyrus_Volume, R_cingulate_gyrus_ComputeArea, R_cingulate_gyrus_Volume, L_caudate_ComputeArea, L_caudate_Volume, R_caudate_ComputeArea, R_caudate_Volume, L_putamen_ComputeArea, L_putamen_Volume, R_putamen_ComputeArea, R_putamen_Volume, Sex, Weight, ResearchGroup, Age, chr12_rs34637584_GT, chr17_rs11868035_GT, chr17_rs11012_GT, chr17_rs393152_GT, chr17_rs12185268_GT, chr17_rs199533_GT, UPDRS_part_I, UPDRS_part_II, UPDRS_part_III, time_visit

Note that the dataset includes missing values and repeated measures.

The *goal* of this demonstration is to use `OLS`, `ridge regression`, and the `LASSO` to **find the best predictive model for the clinical outcomes** – UPRDR score (vector) and Research Group (factor variable), in terms of demographic, genetics, and neuroimaging biomarkers.

We can utilize the `glmnet` package in R for most calculations.

```
#### Initial Stuff ####
# clean up
rm(list=ls())
# load required packages
# install.packages("arm")
library(glmnet)
library(arm)
library(knitr) #  kable function to convert tabular R-results into Rmd tables
# pick a random seed, but set.seed(seed) only effects next block of code!
seed = 1234

#### Organize Data ####
# load dataset
# Data: https://umich.instructure.com/courses/38100/files/folder/data
# (05_PPMI_top_UPDRS_Integrated_LongFormat1.csv)
data1 <- read.table('https://umich.instructure.com/files/330397/download?download_frd=1', sep=",", header=T)
# we will deal with missing values using multiple imputation later. For now, let's just ignore incomplete cases
data1.completeRowIndexes <- complete.cases(data1); table(data1.completeRowIndexes)
```

```
## data1.completeRowIndexes
## FALSE  TRUE
##   609  1155
```

```
prop.table(table(data1.completeRowIndexes))
```

```
## data1.completeRowIndexes
##     FALSE      TRUE
## 0.3452381 0.6547619
```

```
attach(data1)
# View(data1[data1.completeRowIndexes, ])

# define response and predictors
y <- data1$UPDRS_part_I + data1$UPDRS_part_II + data1$UPDRS_part_III
table(y)   # Show Clinically relevant classification
```

```
## y
##  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
## 54 20 25 12  8  7 11 16 16  9 21 16 13 13 22 25 21 31 25 29 29 28 20 25 28
## 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
## 26 35 41 23 34 32 31 37 34 28 36 29 27 22 19 17 18 18 19 16  9 10 12  9 11
## 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 66 68 69 71 75 80 81 82
##  7 10 11  5  7  4  1  5  9  4  3  2  1  6  1  2  1  2  1  1  2  3  1
```

```
y <- y[data1.completeRowIndexes]

# X = scale(data1[,])  # Explicit Scaling is not needed, as glmnet auto standardizes predictors
# X = as.matrix(data1[, c("R_caudate_Volume", "R_putamen_Volume", "Weight", "Age", "chr17_rs12185268_GT")])  # X needs to be a matri
x, not a data frame
drop_features <- c("FID_IID", "ResearchGroup", "PDRS_part_I", "UPDRS_part_II", "UPDRS_part_III")
X <- data1[ , !(names(data1) %in% drop_features)]
X = as.matrix(X)   # remove columns: index, ResearchGroup, and y=(PDRS_part_I + UPDRS_part_II + UPDRS_part_III)
X <- X[data1.completeRowIndexes, ]
summary(X)
```

```
##  L_insular_cortex_ComputeArea L_insular_cortex_Volume
##  Min.   :  50.03              Min.   :   22.63
##  1st Qu.:2174.57              1st Qu.: 5867.23
##  Median :2522.52              Median : 7362.90
##  Mean   :2306.89              Mean   : 6710.18
##  3rd Qu.:2752.17              3rd Qu.: 8483.80
##  Max.   :3650.81              Max.   :13499.92
##  R_insular_cortex_ComputeArea R_insular_cortex_Volume
##  Min.   :  40.92              Min.   :  11.84
##  1st Qu.:1647.69              1st Qu.:3559.74
##  Median :1931.21              Median :4465.12
##  Mean   :1758.64              Mean   :4127.87
##  3rd Qu.:2135.57              3rd Qu.:5319.13
##  Max.   :2791.92              Max.   :8179.40
##  L_cingulate_gyrus_ComputeArea L_cingulate_gyrus_Volume
##  Min.   : 127.8                Min.   :   57.33
##  1st Qu.:2847.4                1st Qu.: 6587.07
##  Median :3737.7                Median : 8965.03
##  Mean   :3411.3                Mean   : 8265.03
##  3rd Qu.:4253.7                3rd Qu.:10815.06
##  Max.   :5944.2                Max.   :17153.19
##  R_cingulate_gyrus_ComputeArea R_cingulate_gyrus_Volume
##  Min.   : 104.1                Min.   :   47.67
##  1st Qu.:2829.4                1st Qu.: 6346.31
##  Median :3719.4                Median : 9094.15
##  Mean   :3368.4                Mean   : 8194.07
##  3rd Qu.:4261.8                3rd Qu.:10832.53
##  Max.   :6593.7                Max.   :19761.77
##  L_caudate_ComputeArea L_caudate_Volume    R_caudate_ComputeArea
##  Min.   :   1.782      Min.   :   0.1928   Min.   :   1.782
##  1st Qu.: 318.806      1st Qu.: 264.0013   1st Qu.: 660.696
##  Median : 710.779      Median : 998.2269   Median :1063.046
##  Mean   : 657.442      Mean   : 992.2892   Mean   : 894.806
##  3rd Qu.: 951.868      3rd Qu.:1568.3643   3rd Qu.:1183.659
##  Max.   :1453.506      Max.   :2746.6208   Max.   :1684.563
##  R_caudate_Volume   L_putamen_ComputeArea L_putamen_Volume
##  Min.   :   0.193   Min.   :   6.76       Min.   :   1.228
##  1st Qu.: 893.637   1st Qu.: 775.73       1st Qu.:1234.601
##  Median :1803.281   Median :1029.17       Median :1911.089
##  Mean   :1548.739   Mean   : 959.15       Mean   :1864.390
##  3rd Qu.:2152.509   3rd Qu.:1260.56       3rd Qu.:2623.722
##  Max.   :3579.373   Max.   :2129.67       Max.   :4712.661
##  R_putamen_ComputeArea R_putamen_Volume        Sex            Weight
##  Min.   :  13.93       Min.   :   3.207   Min.   :1.000   Min.   : 43.20
##  1st Qu.:1255.62       1st Qu.:2474.041   1st Qu.:1.000   1st Qu.: 69.90
##  Median :1490.05       Median :3510.249   Median :1.000   Median : 80.90
##  Mean   :1332.01       Mean   :3083.007   Mean   :1.347   Mean   : 82.06
##  3rd Qu.:1642.41       3rd Qu.:3994.733   3rd Qu.:2.000   3rd Qu.: 90.70
##  Max.   :2251.41       Max.   :7096.580   Max.   :2.000   Max.   :135.00
##       Age        chr12_rs34637584_GT chr17_rs11868035_GT chr17_rs11012_GT
##  Min.   :31.18   Min.   :0.00000     Min.   :0.0000      Min.   :0.0000
##  1st Qu.:53.87   1st Qu.:0.00000     1st Qu.:0.0000      1st Qu.:0.0000
##  Median :62.16   Median :0.00000     Median :1.0000      Median :0.0000
##  Mean   :61.25   Mean   :0.01212     Mean   :0.6364      Mean   :0.3654
##  3rd Qu.:68.83   3rd Qu.:0.00000     3rd Qu.:1.0000      3rd Qu.:1.0000
##  Max.   :83.03   Max.   :1.00000     Max.   :2.0000      Max.   :2.0000
##  chr17_rs393152_GT chr17_rs12185268_GT chr17_rs199533_GT  UPDRS_part_I
##  Min.   :0.0000    Min.   :0.0000      Min.   :0.0000    Min.   : 0.000
##  1st Qu.:0.0000    1st Qu.:0.0000      1st Qu.:0.0000    1st Qu.: 0.000
##  Median :0.0000    Median :0.0000      Median :0.0000    Median : 1.000
##  Mean   :0.4468    Mean   :0.4268      Mean   :0.4052    Mean   : 1.306
##  3rd Qu.:1.0000    3rd Qu.:1.0000      3rd Qu.:1.0000    3rd Qu.: 2.000
##  Max.   :2.0000    Max.   :2.0000      Max.   :2.0000    Max.   :13.000
##    time_visit
##  Min.   : 0.00
##  1st Qu.: 9.00
##  Median :24.00
##  Mean   :23.83
##  3rd Qu.:36.00
##  Max.   :54.00
```

```
# randomly split data into training (80%) and test (20%) sets
set.seed(seed)
train = sample(1 : nrow(X), round((4/5) * nrow(X)))
test = -train

# subset training data
yTrain = y[train]
XTrain = X[train, ]
XTrainOLS = cbind(rep(1, nrow(XTrain)), XTrain)

# subset test data
yTest = y[test]
XTest = X[test, ]

#### Model Estimation & Selection ####
# Estimate models
fitOLS = lm(yTrain ~ XTrain)  # Ordinary Least Squares
# glmnet automatically standardizes the predictors
fitRidge = glmnet(XTrain, yTrain, alpha = 0)  # Ridge Regression
fitLASSO = glmnet(XTrain, yTrain, alpha = 1)  # The LASSO
```

Readers are encouraged to compare the two models, *ridge* and *LASSO.*

8.2 Computational Complexity

Recall that the regularized regression estimates depend on the regularization parameter λ. Fortunately, efficient algorithms for choosing optimal λ parameters do exist. Examples of solution path algorithms include:

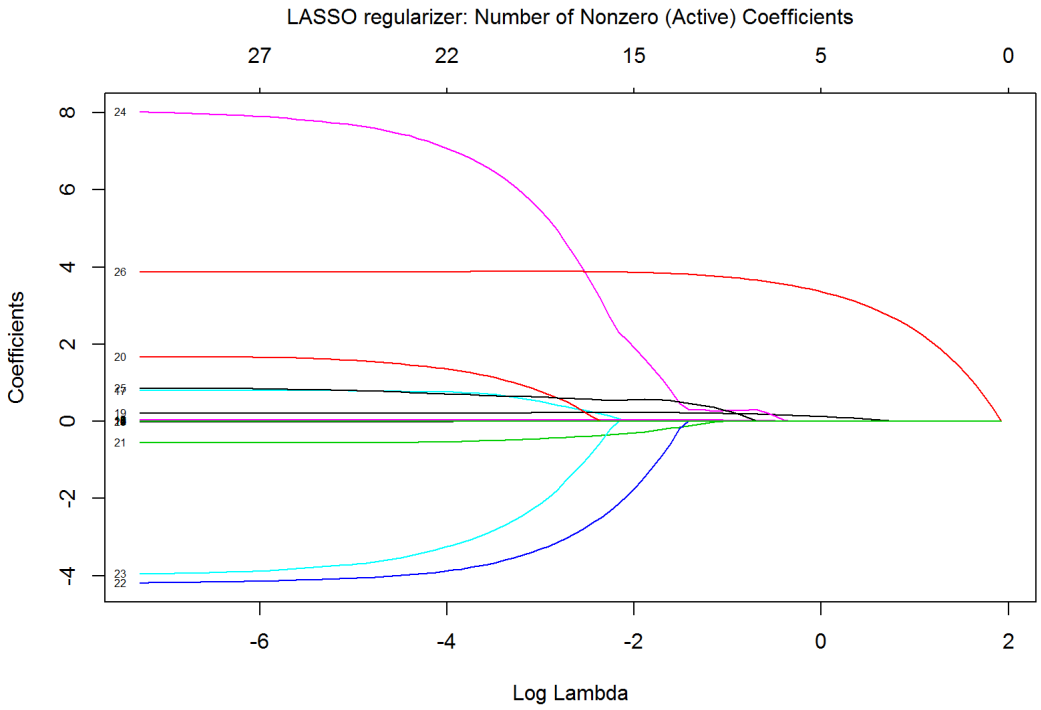
```
- [LARS Algorithm for the LASSO](http://projecteuclid.org/euclid.aos/1083178935) (Efron et al., 2004)
- [Piecewise linearity](http://www.jstor.org/stable/25463590?seq=1#page_scan_tab_contents) (Rosset & Zhu, 2007)
- [Generic path algorithm](http://dx.doi.org/10.1080/01621459.2013.864166) (Zhou & Wu, 2013)
- [Pathwise coordinate descent](http://projecteuclid.org/euclid.aoas/1196438020) (Friedman et al., 2007)
- [Alternating Direction Method of Multipliers (ADMM)](https://doi.org/10.1561/2200000016) (Boyd et al. 2011)
```

We will show how to visualize the relations between the regularization parameter ($\ln(\lambda)$) and the number and magnitude of the corresponding coefficients for each specific regularized regression method.

8.3 LASSO and Ridge Solution Paths

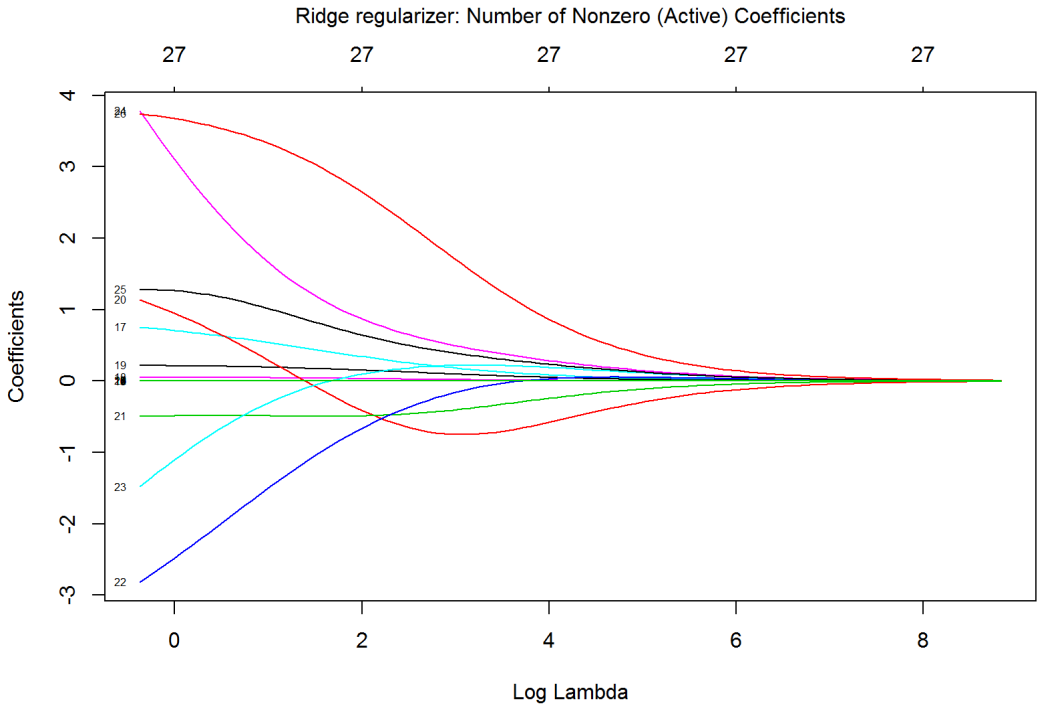
The plot for the *LASSO* results can be obtained via:

```
### Plot Solution Path ###
# LASSO
plot(fitLASSO, xvar="lambda", label="TRUE")
# add label to upper x-axis
mtext("LASSO regularizer: Number of Nonzero (Active) Coefficients", side=3, line=2.5)
```



Similarly, the plot for the *Ridge* regularization can be obtained by:

```
### Plot Solution Path ###
# Ridge
plot(fitRidge, xvar="lambda", label="TRUE")
# add Label to upper x-axis
mtext("Ridge regularizer: Number of Nonzero (Active) Coefficients", side=3, line=2.5)
```



8.4 Regression Solution Paths - Ridge vs. LASSO

Let's try to compare the paths of the *LASSO* and *Ridge* regression solutions. Below, you will see that the curves of LASSO are steeper and non-differentiable at some points, which is the result of using the L_1 norm. On the other hand, the Ridge path is smoother and asymptotically tends to 0 as λ increases.

Let's start by examining the joint objective function (including LASSO and Ridge terms):

$$\min_{\beta} \left(\sum_i (y_i - x_i\beta)^2 + \frac{1-\alpha}{2} ||\beta||_2^2 + \alpha ||\beta||_1 \right),$$

where $||\beta||_1 = \sum_{j=1}^{p} |\beta_j|$ and $||\beta||_2 = \sqrt{\sum_{j=1}^{p} ||\beta_j||^2}$ are the norms of $\boldsymbol{\beta}$ corresponding to the L_1 and L_2 distance measures, respectively. When $\alpha = 0$ and $\alpha = 1$ correspond to *Ridge* and *LASSO* regularization. The following two natural questions raise:

- What if $0 < \alpha < 1$?
- How does the regularization penalty term affect the optimal solution?

In Chapter 9 (http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650/notes/09_RegressionForecasting.html), we explored the minimal SSE (Sum of Square Error) for the OLS (without penalty) where the feasible parameter (β) spans the entire real solution space. In penalized optimization problems, the best solution may actually be unachievable. Therefore, we look for solutions that are "closest", within the feasible region, to the enigmatic best solution.

The effect of the penalty term on the objective function is separate from the *fidelity term* (OLS solution). Thus, the effect of $0 \leq \alpha \leq 1$ is limited to the **size and shape of the penalty region**. Let's try to visualize the feasible region as:

- centrosymmetric, when $\alpha = 0$, and
- super diamond, then $\alpha = 1$.

Here is a hands-on example:

```
require(needs)
```

```
## Loading required package: needs
```

```
# Constructing Quadratic Formula
result <- function(a,b,c){
  if(delta(a,b,c) > 0){ # first case D>0
    x_1 = (-b+sqrt(delta(a,b,c)))/(2*a)
    x_2 = (-b-sqrt(delta(a,b,c)))/(2*a)
    result = c(x_1,x_2)
  }
  else if(delta(a,b,c) == 0){ # second case D=0
    x = -b/(2*a)
  }
  else {"There are no real roots."} # third case D<0
}
# Constructing delta
delta<-function(a,b,c){
  b^2-4*a*c
}
```

To make this realistic, we will use the MLB dataset (https://umich.instructure.com/files/330381/download?download_frd=1) to first fit an OLS model. The dataset contains $1,034$ records of *heights and weights* for some current and recent Major League Baseball (MLB) Players.

- *Height*: Player height in inch,
- *Weight*: Player weight in pounds,
- *Age*: Player age at time of record.

Then, we can obtain the SSE for any $||\boldsymbol{\beta}||$:

$$SSE = ||Y - \hat{Y}||^2 = (Y - \hat{Y})^T(Y - \hat{Y}) = Y^TY - 2\beta^TX^TY + \beta^TX^TX\beta.$$

Next, we will compute the contours for SSE in several situations.

```
library("ggplot2")
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'data':
##
##     Position
```

```
# load data
mlb<- read.table('https://umich.instructure.com/files/330381/download?download_frd=1', as.is=T, header=T)
str(mlb)
```

```
## 'data.frame':    1034 obs. of  6 variables:
##  $ Name    : chr  "Adam_Donachie" "Paul_Bako" "Ramon_Hernandez" "Kevin_Millar" ...
##  $ Team    : chr  "BAL" "BAL" "BAL" "BAL" ...
##  $ Position: chr  "Catcher" "Catcher" "Catcher" "First_Baseman" ...
##  $ Height  : int  74 74 72 72 73 69 69 71 76 71 ...
##  $ Weight  : int  180 215 210 210 188 176 209 200 231 180 ...
##  $ Age     : num  23 34.7 30.8 35.4 35.7 ...
```

```
fit<-lm(Height~Weight+Age-1, data = as.data.frame(scale(mlb[,4:6])))
points = data.frame(x=c(0,fit$coefficients[1]),y=c(0,fit$coefficients[2]),z=c("(0,0)","OLS Coef"))

Y=scale(mlb$Height)
X = scale(mlb[,c(5,6)])
beta1=seq(-0.556, 1.556, length.out = 100)
beta2=seq(-0.661, 0.3386, length.out = 100)
df <- expand.grid(beta1 = beta1, beta2 = beta2)
b = as.matrix(df)
df$sse <- rep(t(Y)%*%Y,100*100) - 2*b%*%t(X)%*%Y + diag(b%*%t(X)%*%X%*%t(b))
```

```
base <- ggplot(df) +
  stat_contour(aes(beta1, beta2, z = sse),breaks = round(quantile(df$sse, seq(0, 0.2, 0.03)), 0),
               size = 0.5,color="darkorchid2",alpha=0.8)+
  scale_x_continuous(limits = c(-0.4,1))+
  scale_y_continuous(limits = c(-0.55,0.4))+
  coord_fixed(ratio=1)+
  geom_point(data = points,aes(x,y))+
  geom_text(data = points,aes(x,y,label=z),vjust = 2,size=3.5)+
  geom_segment(aes(x = -0.4, y = 0, xend = 1, yend = 0),colour = "grey46",
               arrow = arrow(length=unit(0.30,"cm")),size=0.5,alpha=0.8)+
  geom_segment(aes(x = 0, y = -0.55, xend = 0, yend = 0.4),colour = "grey46",
               arrow = arrow(length=unit(0.30,"cm")),size=0.5,alpha=0.8)
```

```
plot_alpha = function(alpha=0,restrict=0.2,beta1_range=0.2,annot=c(0.15,-0.25,0.205,-0.05)){
  a=alpha; t=restrict; k=beta1_range; pos=data.frame(V1=annot[1:4])
  tex=paste("(",as.character(annot[3]),",",as.character(annot[4]),")",sep = "")
  K = seq(0,k,length.out = 50)
  y = unlist(lapply((1-a)*K^2/2+a*K-t,result,a=(1-a)/2,b=a))[seq(1,99,by=2)]
  fills = data.frame(x=c(rev(-K),K),y1=c(rev(y),y),y2=c(-rev(y),-y))
  p<-base+geom_line(data=fills,aes(x = x,y = y1),colour = "salmon1",alpha=0.6,size=0.7)+
    geom_line(data=fills,aes(x = x,y = y2),colour = "salmon1",alpha=0.6,size=0.7)+
    geom_polygon(data = fills, aes(x, y1),fill = "red", alpha = 0.2)+
    geom_polygon(data = fills, aes(x, y2), fill = "red", alpha = 0.2)+
    geom_segment(data=pos,aes(x = V1[1] , y = V1[2], xend = V1[3], yend = V1[4]),
                 arrow = arrow(length=unit(0.30,"cm")),alpha=0.8,colour = "magenta")+
    ggplot2::annotate("text", x = pos$V1[1]-0.01, y = pos$V1[2]-0.11,
                      label = paste(tex,"\n","Point of Contact \n i.e., Coef of", "alpha=",fractions(a)),size=3)+
    xlab(expression(beta[1]))+
    ylab(expression(beta[2]))+
    ggtitle(paste("alpha =",as.character(fractions(a))))+
    theme(legend.position="none")
}
```

```
# $\alpha=0$ - Ridge
p1 <- plot_alpha(alpha=0,restrict=(0.21^2)/2,beta1_range=0.21,annot=c(0.15,-0.25,0.205,-0.05))
p1 <- p1 + ggtitle(expression(paste(alpha, "=0 (Ridge)")))
# $\alpha=1/9$
p2 <- plot_alpha(alpha=1/9,restrict=0.046,beta1_range=0.22,annot =c(0.15,-0.25,0.212,-0.02))
p2 <- p2 + ggtitle(expression(paste(alpha, "=1/9")))
# $\alpha=1/5$
p3 <- plot_alpha(alpha=1/5,restrict=0.063,beta1_range=0.22,annot=c(0.13,-0.25,0.22,0))
p3 <- p3 + ggtitle(expression(paste(alpha, "=1/5")))
# $\alpha=1/2$
p4 <- plot_alpha(alpha=1/2,restrict=0.123,beta1_range=0.22,annot=c(0.12,-0.25,0.22,0))
p4 <- p4 + ggtitle(expression(paste(alpha, "=1/2")))
# $\alpha=3/4$
p5 <- plot_alpha(alpha=3/4,restrict=0.17,beta1_range=0.22,annot=c(0.12,-0.25,0.22,0))
p5 <- p5 + ggtitle(expression(paste(alpha, "=3/4")))
```

```
# $\alpha=1$ - LASSO
t=0.22
K = seq(0,t,length.out = 50)
fills = data.frame(x=c(-rev(K),K),y1=c(rev(t-K),c(t-K)),y2=c(-rev(t-K),-c(t-K)))
p6 <- base +
  geom_segment(aes(x = 0, y = t, xend = t, yend = 0),colour = "salmon1",alpha=0.1,size=0.2)+
  geom_segment(aes(x = 0, y = t, xend = -t, yend = 0),colour = "salmon1",alpha=0.1,size=0.2)+
  geom_segment(aes(x = 0, y = -t, xend = t, yend = 0),colour = "salmon1",alpha=0.1,size=0.2)+
  geom_segment(aes(x = 0, y = -t, xend = -t, yend = 0),colour = "salmon1",alpha=0.1,size=0.2)+
  geom_polygon(data = fills, aes(x, y1),fill = "red", alpha = 0.2)+
  geom_polygon(data = fills, aes(x, y2), fill = "red", alpha = 0.2)+
  geom_segment(aes(x = 0.12 , y = -0.25, xend = 0.22, yend = 0),colour = "magenta",
               arrow = arrow(length=unit(0.30,"cm")),alpha=0.8)+
  ggplot2::annotate("text", x = 0.11, y = -0.36,
                    label = "(0.22,0)\n Point of Contact \n i.e Coef of LASSO",size=3)+
  xlab( expression(beta[1]))+
  ylab( expression(beta[2]))+
  theme(legend.position="none")+
  ggtitle(expression(paste(alpha, "=1 (LASSO)")))
```

Then, let's add the six feasible regions corresponding to $\alpha = 0$ (Ridge), $\alpha = \frac{1}{9}$, $\alpha = \frac{1}{5}$, $\alpha = \frac{1}{2}$, $\alpha = \frac{3}{4}$ and $\alpha = 1$ (LASSO).

This figure provides some intuition into the continuum from Ridge to LASSO regularization. The feasible regions are drawn as ellipse contours of the SSE in *red*. Curves around the corresponding feasible regions represent the *boundary of the constraint function* $\frac{1-\alpha}{2}||\beta||_2^2 + \alpha||\beta||_1 \leq t$.

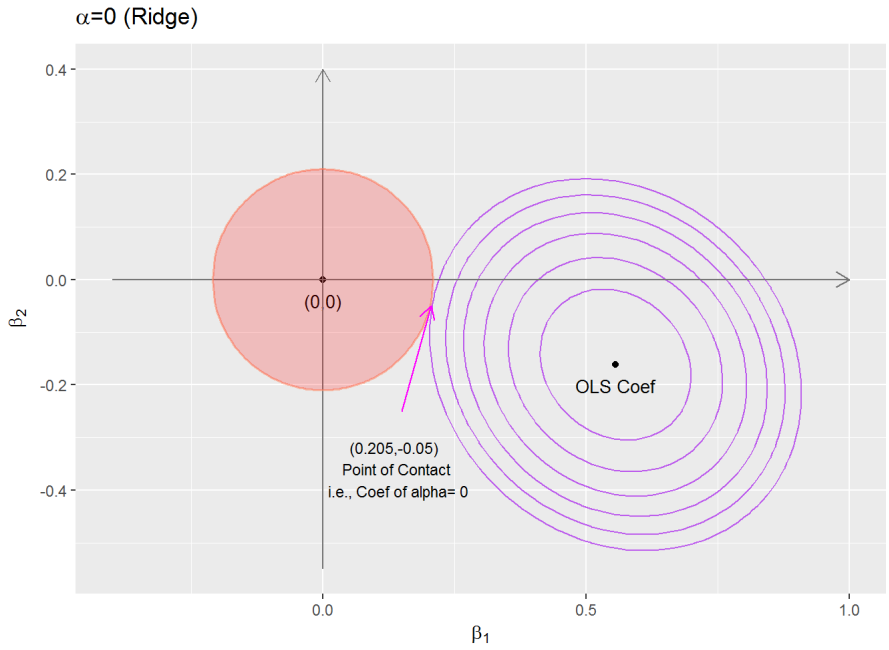
In this example, β_2 shrinks to 0 for $\alpha = \frac{1}{5}$, $\alpha = \frac{1}{2}$, $\alpha = \frac{3}{4}$ and $\alpha = 1$.

We observe that it is almost impossible for the contours of Ridge regression to touch the circle at any of the coordinate axes. This is also true in higher dimensions (nD), where the L_1 and L_2 metrics are unchanged and the 2D ellipse representations of the feasibility regions become hyper-ellipsoidal shapes.

Generally, as α goes from 0 to 1. The coefficients of more features tend to shrink towards 0. This specific property makes LASSO useful for variable selection.

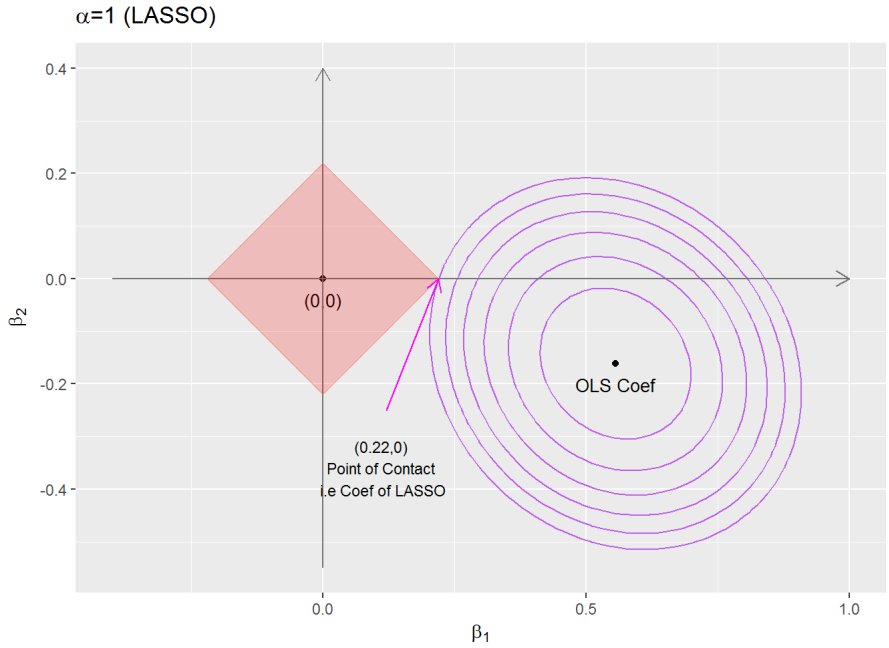
Let's compare the feasibility regions corresponding to *Ridge* (top, $p1$) and *LASSO* (bottom, $p6$) regularization.

```
plot(p1)
```



SSE Contour and Penalty Region (Ridge)

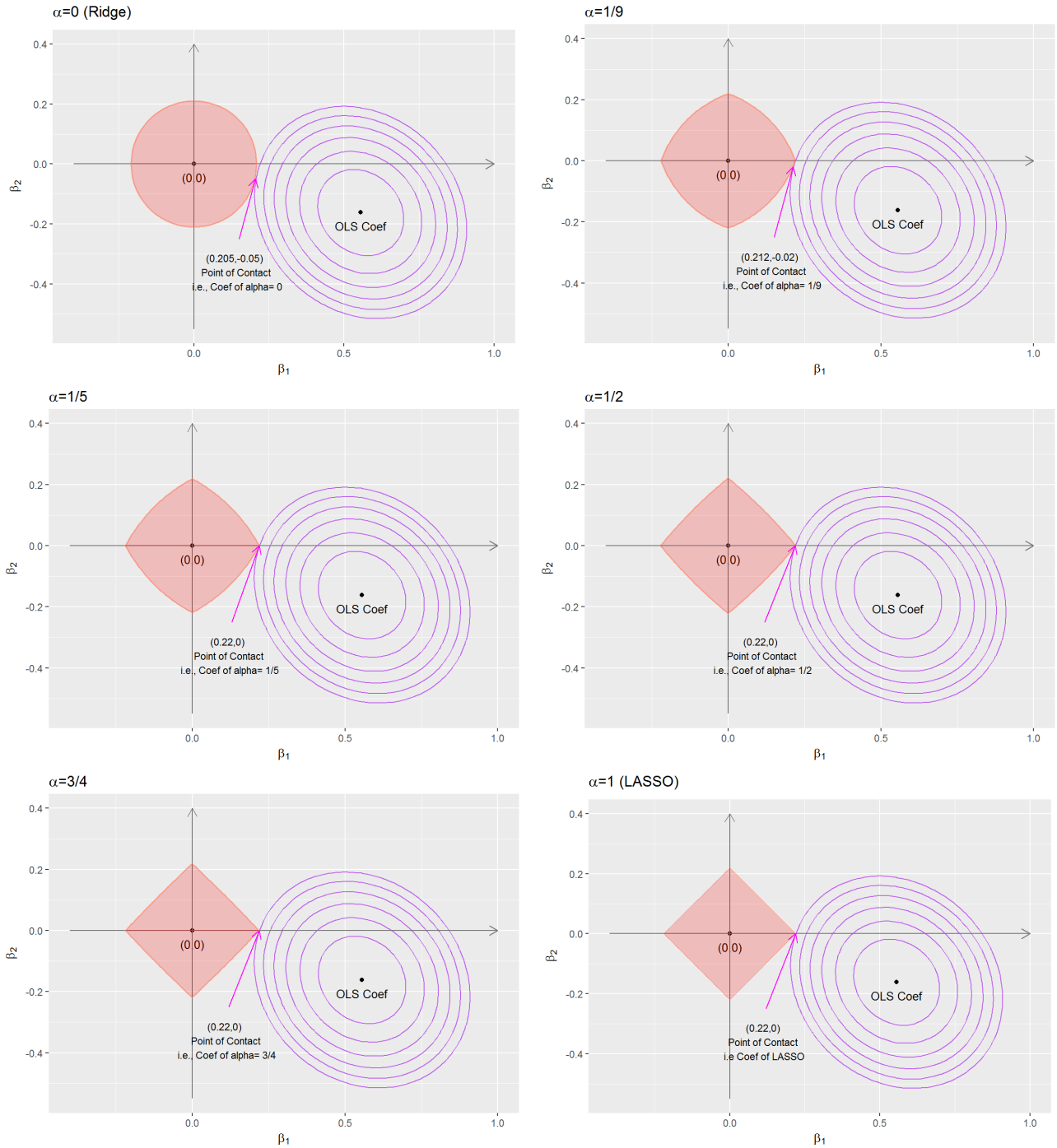
```
plot(p6)
```



SSE Contour and Penalty Region (LASSO)

Then, we can plot the *progression* from Ridge to LASSO. (This composite *plot is intense* and may take several minutes to render!)

```
library("gridExtra")
grid.arrange(p1,p2,p3,p4,p5,p6,nrow=3)
```



SSE Contour and Penalty Region for 6 values of Alpha

8.5 Choice of the Regularization Parameter

Efficiently obtaining the entire solution path is nice, but we still have to choose a specific λ regularization parameter. This is critical as λ `controls the bias-variance tradeoff`.

Traditional model selection methods rely on various metrics like Mallows' C_p (https://en.wikipedia.org/wiki/Mallows%27s_Cp), AIC (https://en.wikipedia.org/wiki/Akaike_information_criterion), BIC (https://en.wikipedia.org/wiki/Bayesian_information_criterion), and adjusted R^2.

Internal statistical validation (Cross validation) is a popular modern alternative, which offers some of these benefits:

- Choice is based on predictive performance,
- Makes fewer model assumptions,
- More widely applicable.

8.6 Cross Validation Motivation

Ideally, we would like a separate validation set for choosing λ for a given method. Reusing training sets may encourage overfitting and using testing data to pick λ may underestimates the true error rate. Often, when we do not have enough data for a separate validation set, cross validation provides an alternative strategy.

8.7 n-Fold Cross Validation

We have already seen examples of using cross-validation, e.g., Chapter 13 (http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650/notes/13_ModelEvaluation.html#4_estimating_future_performance_(internal_statistical_v and Chapter 20 (http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650/notes/20_PredictionCrossValidation.html) provides more details about this internal statistical assessment strategy.

We can use either automated or manual cross-validation. In either case, the protocol involves the following iterative steps:

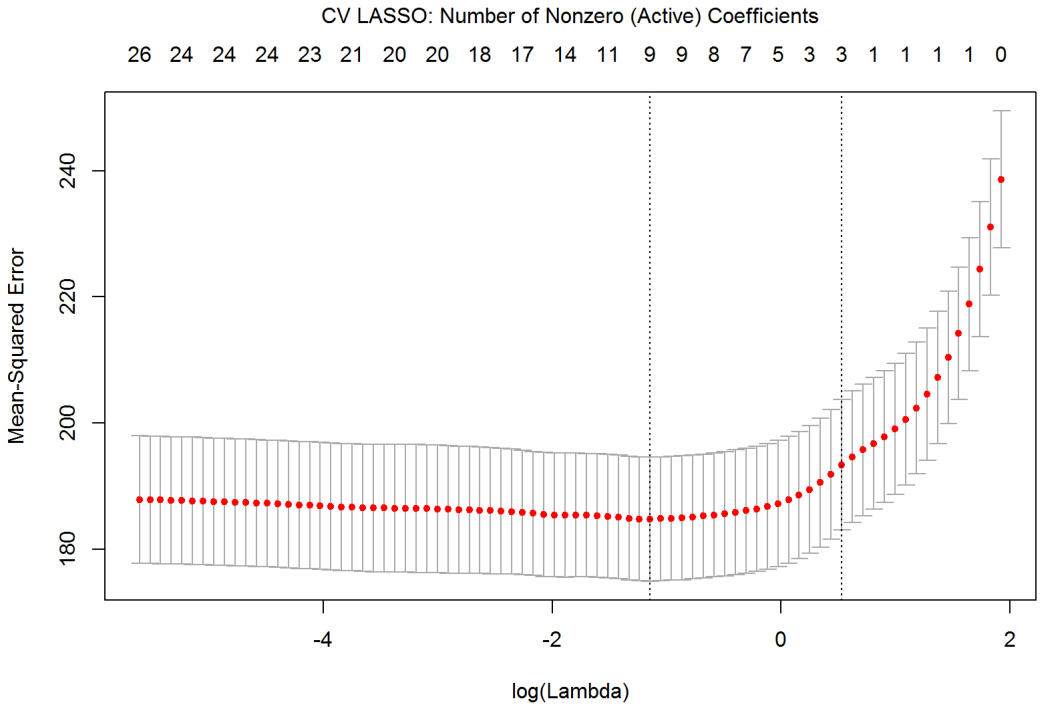
1. Randomly split the training data into n parts ("folds").
2. Fit a model using data in $n-1$ folds for multiple λs.
3. Calculate some prediction quality metrics (e.g., MSE, accuracy) on the last remaining fold, see Chapter 13 (http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650/notes/13_ModelEvaluation.html).
4. Repeat the process and average the prediction metrics across iterations.

Common choices of n are 5, 10, and n (which corresponds to `leave-one-out` CV). One standard error rule is to choose λ corresponding to smallest model with MSE within one standard error of the minimum MSE.

8.8 LASSO 10-Fold Cross Validation

Now, let's apply an internal statistical cross-validation to assess the quality of the LASSO and Ridge models, based on our Parkinson's disease case-study. Recall our split of the PD data into training (yTrain, XTrain) and testing (yTest, XTest) sets.

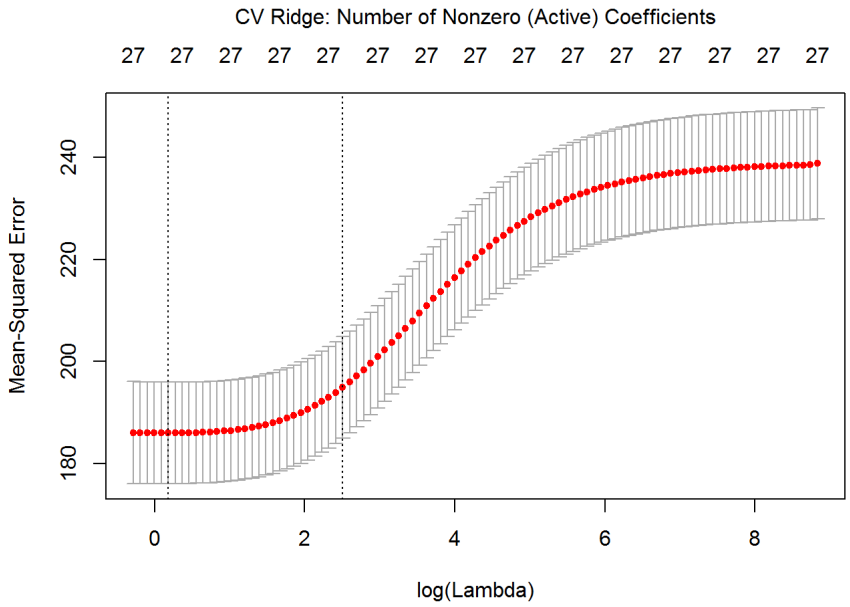
```
#### 10-fold cross validation ####
# LASSO
library("glmnet")
set.seed(seed)  # set seed
# (10-fold) cross validation for the LASSO
cvLASSO = cv.glmnet(XTrain, yTrain, alpha = 1)
plot(cvLASSO)
mtext("CV LASSO: Number of Nonzero (Active) Coefficients", side=3, line=2.5)
```




```
# Report MSE LASSO
predLASSO <-  predict(cvLASSO, s = cvLASSO$lambda.1se, newx = XTest)
testMSE_LASSO <- mean((predLASSO - yTest)^2); testMSE_LASSO
```

```
## [1] 200.5609
```

```
#### 10-fold cross validation ####
# Ridge Regression
set.seed(seed)  # set seed
# (10-fold) cross validation for Ridge Regression
cvRidge = cv.glmnet(XTrain, yTrain, alpha = 0)
plot(cvRidge)
mtext("CV Ridge: Number of Nonzero (Active) Coefficients", side=3, line=2.5)
```




```
# Report MSE Ridge
predRidge <-  predict(cvRidge, s = cvRidge$lambda.1se, newx = XTest)
testMSE_Ridge <- mean((predRidge - yTest)^2); testMSE_Ridge
```

```
## [1] 195.7406
```

Note that the `predict()` method applied to `cv.gmlnet` or `glmnet` forecasting models is effectively a function wrapper to `predict.gmlnet()`. According to what you would like to get as a **prediction output**, you can use `type="..."` to specify one of the following types of prediction outputs:

- `type="link"`, reports the linear predictors for "binomial", "multinomial", "poisson" or "cox" models; for "gaussian" models it gives the fitted values.
- `type="response"`, reports the fitted probabilities for "binomial" or "multinomial", fitted mean for "poisson" and the fitted relative-risk for "cox"; for "gaussian" type "response" is equivalent to type "link".
- `type="coefficients"`, reports the coefficients at the requested values for `s`. Note that for "binomial" models, results are returned only for the class corresponding to the second level of the factor response.
- `type="class"`, applies only to "binomial" or "multinomial" models, and produces the class label corresponding to the maximum probability.
- `type="nonzero"`, returns a list of the indices of the nonzero coefficients for each value of `s`.

8.9 Stepwise OLS (ordinary least squares)

For a fair comparison, let's also obtain an OLS stepwise model selection, see Chapter 16 (http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650/notes/16_FeatureSelection.html).

```
dt = as.data.frame(cbind(yTrain,XTrain))
ols_step <- lm(yTrain ~., data = dt)
ols_step <- step(ols_step, direction = 'both', k=2, trace = F)
summary(ols_step)
```

```
##
## Call:
## lm(formula = yTrain ~ L_cingulate_gyrus_ComputeArea + R_cingulate_gyrus_Volume +
##     L_caudate_Volume + L_putamen_ComputeArea + L_putamen_Volume +
##     R_putamen_ComputeArea + Weight + Age + chr17_rs11012_GT +
##     chr17_rs393152_GT + chr17_rs12185268_GT + UPDRS_part_I, data = dt)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.990  -9.098  -0.310   8.373  49.027
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  -2.8179771  4.5458868  -0.620  0.53548
## L_cingulate_gyrus_ComputeArea  0.0045203  0.0013422   3.368  0.00079 ***
## R_cingulate_gyrus_Volume      -0.0010036  0.0003461  -2.900  0.00382 **
## L_caudate_Volume              -0.0021999  0.0011054  -1.990  0.04686 *
## L_putamen_ComputeArea         -0.0087295  0.0045925  -1.901  0.05764 .
## L_putamen_Volume               0.0035419  0.0017969   1.971  0.04902 *
## R_putamen_ComputeArea          0.0029862  0.0019036   1.569  0.11706
## Weight                         0.0424646  0.0268088   1.584  0.11355
## Age                            0.2198283  0.0522490   4.207 2.84e-05 ***
## chr17_rs11012_GT              -4.2408237  1.8122682  -2.340  0.01950 *
## chr17_rs393152_GT             -3.5818432  2.2619779  -1.584  0.11365
## chr17_rs12185268_GT            8.2990131  2.7356037   3.034  0.00248 **
## UPDRS_part_I                   3.8780897  0.2541024  15.262  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.41 on 911 degrees of freedom
## Multiple R-squared:  0.2556, Adjusted R-squared:  0.2457
## F-statistic: 26.06 on 12 and 911 DF,  p-value: < 2.2e-16
```

We use `direction=both` for both *forward* and *backward* selection and choose the optimal one. `k=2` specifies AIC and BIC criteria, and you can choose $k \sim \log(n)$.

Then, we use the `ols_step` model to predict the outcome Y for some new test data.

```
betaHatOLS_step = ols_step$coefficients
var_step <- colnames(ols_step$model)[-1]
XTestOLS_step = cbind(rep(1, nrow(XTest)), XTest[,var_step])
predOLS_step = XTestOLS_step%*%betaHatOLS_step
testMSEOLS_step = mean((predOLS_step - yTest)^2)
# Report MSE OLS Stepwise feature selection
testMSEOLS_step
```

```
## [1] 186.3043
```

Alternatively, we can predict the outcomes directly using the `predict()` function, and the results should be identical:

```
pred2 <- predict(ols_step,as.data.frame(XTest))
any(pred2 == predOLS_step)
```

```
## [1] TRUE
```

8.10 Final Models

Let's identify the most important (predictive) features, which can then be interpreted in the context of the specific data.

```
# Determine final models

# Extract Coefficients
# OLS coefficient estimates
betaHatOLS = fitOLS$coefficients
# LASSO coefficient estimates
betaHatLASSO = as.double(coef(fitLASSO, s = cvLASSO$lambda.1se))  # s is lambda
# Ridge  coefficient estimates
betaHatRidge = as.double(coef(fitRidge, s = cvRidge$lambda.1se))

# Test Set MSE
# calculate predicted values

XTestOLS = cbind(rep(1, nrow(XTest)), XTest) # add intercept to test data
predOLS = XTestOLS%*%betaHatOLS
predLASSO = predict(fitLASSO, s = cvLASSO$lambda.1se, newx = XTest)
predRidge = predict(fitRidge, s = cvRidge$lambda.1se, newx = XTest)

# calculate test set MSE
testMSEOLS = mean((predOLS - yTest)^2)
testMSELASSO = mean((predLASSO - yTest)^2)
testMSERidge = mean((predRidge - yTest)^2)
```

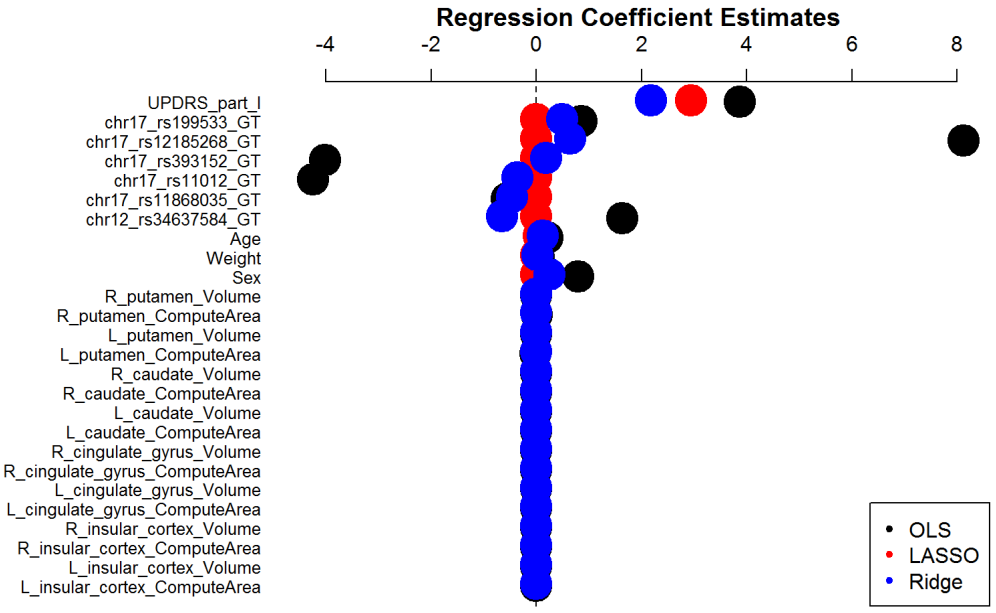
This plot shows a rank-ordered list of hte key predictors of the clinical outcome variable (total UPDRS, `y <- data1$UPDRS_part_I + data1$UPDRS_part_II + data1$UPDRS_part_III`).

```
# Plot Regression Coefficients
# create variable names for plotting
library("arm")
par(mar=c(2, 13, 1, 1))   # extra large left margin
varNames <- colnames(data1[ , !(names(data1) %in% drop_features)]); varNames; length(varNames)
```

```
##  [1] "L_insular_cortex_ComputeArea"  "L_insular_cortex_Volume"
##  [3] "R_insular_cortex_ComputeArea"  "R_insular_cortex_Volume"
##  [5] "L_cingulate_gyrus_ComputeArea" "L_cingulate_gyrus_Volume"
##  [7] "R_cingulate_gyrus_ComputeArea" "R_cingulate_gyrus_Volume"
##  [9] "L_caudate_ComputeArea"         "L_caudate_Volume"
## [11] "R_caudate_ComputeArea"         "R_caudate_Volume"
## [13] "L_putamen_ComputeArea"         "L_putamen_Volume"
## [15] "R_putamen_ComputeArea"         "R_putamen_Volume"
## [17] "Sex"                           "Weight"
## [19] "Age"                           "chr12_rs34637584_GT"
## [21] "chr17_rs11868035_GT"           "chr17_rs11012_GT"
## [23] "chr17_rs393152_GT"             "chr17_rs12185268_GT"
## [25] "chr17_rs199533_GT"             "UPDRS_part_I"
## [27] "time_visit"
```

```
## [1] 27
```

```
# Graph 27 regression coefficients (exclude intercept [1], betaHat indices 2:27)
coefplot(betaHatOLS[2:27], sd = rep(0, 26), cex.pts = 5, main = "Regression Coefficient Estimates", varnames = varNames)
coefplot(betaHatLASSO[2:27], sd = rep(0, 26), add = TRUE, col.pts = "red", cex.pts = 5)
coefplot(betaHatRidge[2:27], sd = rep(0, 26), add = TRUE, col.pts = "blue", cex.pts = 5)
legend("bottomright", c("OLS", "LASSO", "Ridge"), col = c("black", "red", "blue"), pch = c(20, 20 , 20), bty = "o")
```




```
# par()
```

8.11 Model Performance

We next quantify the performance of the model.

```
# Test Set MSE Table
# create table as data frame
MSETable = data.frame(OLS=testMSEOLS, OLS_step=testMSEOLS_step, LASSO=testMSELASSO, Ridge=testMSERidge)

# convert to markdown
kable(MSETable, format="pandoc", caption="Test Set MSE", align=c("c", "c", "c", "c"))
```

Test Set MSE

OLS	OLS_step	LASSO	Ridge
183.3239	186.3043	200.5609	195.7406

8.12 Compare the selected variables

```
var_step = names(ols_step$coefficients)[-1]
var_lasso = colnames(XTrain)[which(coef(fitLASSO, s = cvLASSO$lambda.min)!=0)-1]
intersect(var_step,var_lasso)
```

```
## [1] "L_cingulate_gyrus_ComputeArea" "R_putamen_ComputeArea"
## [3] "Weight"                        "Age"
## [5] "chr17_rs12185268_GT"           "UPDRS_part_I"
```

```
coef(fitLASSO, s = cvLASSO$lambda.min)
```

```
## 28 x 1 sparse Matrix of class "dgCMatrix"
##                                      1
## (Intercept)                 1.7142107049
## L_insular_cortex_ComputeArea   .
## L_insular_cortex_Volume        .
## R_insular_cortex_ComputeArea   .
## R_insular_cortex_Volume        .
## L_cingulate_gyrus_ComputeArea  0.0003399436
## L_cingulate_gyrus_Volume       0.0002099980
## R_cingulate_gyrus_ComputeArea  .
## R_cingulate_gyrus_Volume       .
## L_caudate_ComputeArea          .
## L_caudate_Volume               .
## R_caudate_ComputeArea          .
## R_caudate_Volume               .
## L_putamen_ComputeArea          .
## L_putamen_Volume               .
## R_putamen_ComputeArea          0.0010417502
## R_putamen_Volume               .
## Sex                            .
## Weight                         0.0336216322
## Age                            0.2097678904
## chr12_rs34637584_GT            .
## chr17_rs11868035_GT           -0.0094055047
## chr17_rs11012_GT               .
## chr17_rs393152_GT              .
## chr17_rs12185268_GT            0.2688574886
## chr17_rs199533_GT              0.3730813890
## UPDRS_part_I                   3.7697168303
## time_visit                     .
```

Stepwise variable selection for OLS selects 12 variables, whereas LASSO selects 9 variables with the best λ. There are 6 common variables common for both OLS and LASSO.

8.13 Summary

Traditional linear models are useful but also have their shortcomings:

- Prediction accuracy may be sub-optimal.
- Model interpretability may be challenging (especially when a large number of features are used as regressors).
- Stepwise model selection may improve the model performance and add some interpretations, but still may not be optimal.

Regularization adds a penalty term to the estimation:

- Enables exploitation of the *bias-variance* tradeoff.
- Provides flexibility on specifying penalties to allow for continuous variable selection.
- Allows incorporation of prior knowledge.

9 Knock-off Filtering: Simulated Example

Variable selection that controls the false discovery rate (FDR) of *salient features* can be accomplished in different ways. The knockoff filtering (https://web.stanford.edu/~candes/Knockoffs/) represents one strategy for controlled variable selection. To show the usage of `knockoff.filter` we start with a synthetic dataset constructed so that the true coefficient vector β has only a few nonzero entries.

The essence of the knockoff filtering is based on the following three-step process:

- Construct the decoy features (knockoff variables), one for each real observed feature. These act as controls for assessing the importance of the real variables.
- For each feature, X_i, compute the knockoff statistic, W_j, which measures the importance of the variable, relative to its decoy counterpart, $\tilde{X}_i$.
- Determine the overall knockoff threshold. This is computed by rank-ordering the W_j statistics (from large to small), walking down the list of W_j's, selecting variables X_j corresponding to positive W_j's, and terminating this search the last time the ratio of negative to positive W_j's is below the default FDR q value, e.g., $q = 0.10$.

Mathematically, we consider X_j to be *unimportant* (i.e., peripheral or extraneous) if the conditional distribution of Y given $X_1, \ldots, X_p$ does not depend on X_j. Formally, X_j is unimportant if it is conditionally independent of Y given all other features, X_{-j}:

$$Y \perp X_j | X_{-j}.$$

We want to generate a Markov Blanket of Y, such that the smallest set of features J satisfies this condition. Further, to make sure we do not make too many mistakes, we search for a set $\hat{S}$ controlling the false discovery rate (FDR):

$$FDR(\hat{S}) = \mathrm{E}\left(\frac{\#j \in \hat{S}:\ x_j\ unimportant}{\#j \in \hat{S}}\right) \leq q\ (e.g.\ 10\%).$$

Let's look at one simulation example.

```
# Problem parameters
n = 1000          # number of observations
p = 300           # number of variables
k = 30            # number of variables with nonzero coefficients
amplitude = 3.5   # signal amplitude (for noise level = 1)

# Problem data
X = matrix(rnorm(n*p), nrow=n, ncol=p)
nonzero = sample(p, k)
beta = amplitude * (1:p %in% nonzero)
y.sample <- function() X %*% beta + rnorm(n)
```

To begin with, we will invoke the `knockoff.filter` using the default settings.

```
# install.packages("knockoff")
library(knockoff)
y = y.sample()
result = knockoff.filter(X, y)
print(result)
```

```
## Call:
## knockoff.filter(X = X, y = y)
##
## Selected variables:
##  [1]   6  29  30  42  52  54  63  68  70  83  88  96 102 113 115 135 138
## [18] 139 167 176 179 194 212 220 225 228 241 248 265 273 287 288 295
```

The false discovery proportion (fdp) is:

```
fdp <- function(selected) sum(beta[selected] == 0) / max(1, length(selected))
fdp(result$selected)
```

```
## [1] 0.09090909
```

This yields an approximate FDR of 0.10.

The default settings of the knockoff filter uses a test statistic based on LASSO – `knockoff.stat.lasso_signed_max` , which computes the W_j statistics that quantify the discrepancy between a real (X_j) and a decoy, knockoff ($\tilde{X}_j$), feature:

$$W_j = \max(X_j, \tilde{X}_j) \times sgn(X_j - \tilde{X}_j).$$

Effectively, the W_j statistics measures how much more important the variable X_j is relative to its decoy counterpart $\tilde{X}_j$. The strength of the importance of X_j relative to $\tilde{X}_j$ is measured by the magnitude of W_j.

The `knockoff` package includes several other test statistics, with appropriate names prefixed by *knockoff.stat*. For instance, we can use a statistic based on forward selection (fs) and a lower target FDR of 0.10.

```
result = knockoff.filter(X, y, fdr = 0.10, statistic = knockoff.stat.fs)
fdp(result$selected)
```

```
## [1] 0.1428571
```

One can also define additional test statistics, complementing the ones included in the package already. For instance, if we want to implement the following test-statistics:

$$W_j = ||X_j^t . y|| - ||\tilde{X^t} . y||.$$

We can code it as:

```
new_knockoff_stat <- function(X, X_ko, y) {
  abs(t(X) %*% y) - abs(t(X_ko) %*% y)
}
result = knockoff.filter(X, y, statistic = new_knockoff_stat)
fdp(result$selected)
```

```
## [1] 0.3333333
```

9.1 Notes

The `knockoff.filter` function is a wrapper around several simpler functions that (1) construct knockoff variables (*knockoff.create*); (2) compute the test statistic W (various functions with prefix *knockoff.stat*); and (3) compute the threshold for variable selection (*knockoff.threshold*).

The high-level function *knockoff.filter* will automatically `normalize the columns` of the input matrix (unless this behavior is explicitly disabled). However, all other functions in this *package assume that the columns of the input matrix have unitary Euclidean norm*.

9.2 PD Neuroimaging-genetics Case-Study

Let's illustrate controlled variable selection via knockoff filtering using the real PD dataset.

The goal is to determine which imaging, genetics and phenotypic covariates are associated with the clinical diagnosis of PD. The dataset is publicly available online (https://umich.instructure.com/files/330397/download?download_frd=1)

9.2.1 Preparing the data

The data set consists of clinical, genetics, and demographic measurements. To evaluate our results, we will compare diagnostic predictions created by the model for the *UPDRS scores* and the *ResearchGroup* factor variable.

9.2.2 Fetching and cleaning the data

First, we download the data and read it into data frames.

```
data1 <- read.table('https://umich.instructure.com/files/330397/download?download_frd=1', sep=",", header=T)
# we will deal with missing values using multiple imputation later. For now, let's just ignore incomplete cases
data1.completeRowIndexes <- complete.cases(data1) # table(data1.completeRowIndexes)
prop.table(table(data1.completeRowIndexes))
```

```
## data1.completeRowIndexes
##     FALSE      TRUE
## 0.3452381 0.6547619
```

```
# attach(data1)
# View(data1[data1.completeRowIndexes, ])
data2 <- data1[data1.completeRowIndexes, ]
Dx_label  <- data2$ResearchGroup; table(Dx_label)
```

```
## Dx_label
## Control      PD   SWEDD
##     121     897     137
```

9.2.3 Preparing the design matrix

We now construct the design matrix X and the response vector Y. The features (columns of X) represent covariates that will be used to explain the response Y.

```
# Construct preliminary design matrix.
# define response and predictors
Y <- data1$UPDRS_part_I + data1$UPDRS_part_II + data1$UPDRS_part_III
table(Y)   # Show Clinically relevant classification
```

```
## Y
##  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
## 54 20 25 12  8  7 11 16 16  9 21 16 13 13 22 25 21 31 25 29 29 28 20 25 28
## 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
## 26 35 41 23 34 32 31 37 34 28 36 29 27 22 19 17 18 18 19 16  9 10 12  9 11
## 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 66 68 69 71 75 80 81 82
##  7 10 11  5  7  4  1  5  9  4  3  2  1  6  1  2  1  2  1  1  2  3  1
```

```
Y <- Y[data1.completeRowIndexes]

# X = scale(ncaaData[, -20])  # Explicit Scaling is not needed, as glmnet auto standardizes predictors
# X = as.matrix(data1[, c("R_caudate_Volume", "R_putamen_Volume", "Weight", "Age", "chr17_rs12185268_GT")])  # X needs to be a matri
x, not a data frame
drop_features <- c("FID_IID", "ResearchGroup", "UPDRS_part_I", "UPDRS_part_II", "UPDRS_part_III")
X <- data1[ , !(names(data1) %in% drop_features)]
X = as.matrix(X)   # remove columns: index, ResearchGroup, and y=(PDRS_part_I + UPDRS_part_II + UPDRS_part_III)
X <- X[data1.completeRowIndexes, ]; dim(X)
```

```
## [1] 1155   26
```

```
summary(X)
```

```
##  L_insular_cortex_ComputeArea L_insular_cortex_Volume
##  Min.   :  50.03              Min.   :   22.63
##  1st Qu.:2174.57              1st Qu.: 5867.23
##  Median :2522.52              Median : 7362.90
##  Mean   :2306.89              Mean   : 6710.18
##  3rd Qu.:2752.17              3rd Qu.: 8483.80
##  Max.   :3650.81              Max.   :13499.92
##  R_insular_cortex_ComputeArea R_insular_cortex_Volume
##  Min.   :  40.92              Min.   :  11.84
##  1st Qu.:1647.69              1st Qu.:3559.74
##  Median :1931.21              Median :4465.12
##  Mean   :1758.64              Mean   :4127.87
##  3rd Qu.:2135.57              3rd Qu.:5319.13
##  Max.   :2791.92              Max.   :8179.40
##  L_cingulate_gyrus_ComputeArea L_cingulate_gyrus_Volume
##  Min.   : 127.8                Min.   :   57.33
##  1st Qu.:2847.4                1st Qu.: 6587.07
##  Median :3737.7                Median : 8965.03
##  Mean   :3411.3                Mean   : 8265.03
##  3rd Qu.:4253.7                3rd Qu.:10815.06
##  Max.   :5944.2                Max.   :17153.19
##  R_cingulate_gyrus_ComputeArea R_cingulate_gyrus_Volume
##  Min.   : 104.1                Min.   :   47.67
##  1st Qu.:2829.4                1st Qu.: 6346.31
##  Median :3719.4                Median : 9094.15
##  Mean   :3368.4                Mean   : 8194.07
##  3rd Qu.:4261.8                3rd Qu.:10832.53
##  Max.   :6593.7                Max.   :19761.77
##  L_caudate_ComputeArea L_caudate_Volume    R_caudate_ComputeArea
##  Min.   :   1.782      Min.   :   0.1928   Min.   :   1.782
##  1st Qu.: 318.806      1st Qu.: 264.0013   1st Qu.: 660.696
##  Median : 710.779      Median : 998.2269   Median :1063.046
##  Mean   : 657.442      Mean   : 992.2892   Mean   : 894.806
##  3rd Qu.: 951.868      3rd Qu.:1568.3643   3rd Qu.:1183.659
##  Max.   :1453.506      Max.   :2746.6208   Max.   :1684.563
##  R_caudate_Volume   L_putamen_ComputeArea L_putamen_Volume
##  Min.   :   0.193   Min.   :   6.76       Min.   :   1.228
##  1st Qu.: 893.637   1st Qu.: 775.73       1st Qu.:1234.601
##  Median :1803.281   Median :1029.17       Median :1911.089
##  Mean   :1548.739   Mean   : 959.15       Mean   :1864.390
##  3rd Qu.:2152.509   3rd Qu.:1260.56       3rd Qu.:2623.722
##  Max.   :3579.373   Max.   :2129.67       Max.   :4712.661
##  R_putamen_ComputeArea R_putamen_Volume        Sex            Weight
##  Min.   :  13.93       Min.   :   3.207   Min.   :1.000   Min.   : 43.20
##  1st Qu.:1255.62       1st Qu.:2474.041   1st Qu.:1.000   1st Qu.: 69.90
##  Median :1490.05       Median :3510.249   Median :1.000   Median : 80.90
##  Mean   :1332.01       Mean   :3083.007   Mean   :1.347   Mean   : 82.06
##  3rd Qu.:1642.41       3rd Qu.:3994.733   3rd Qu.:2.000   3rd Qu.: 90.70
##  Max.   :2251.41       Max.   :7096.580   Max.   :2.000   Max.   :135.00
##       Age        chr12_rs34637584_GT chr17_rs11868035_GT chr17_rs11012_GT
##  Min.   :31.18   Min.   :0.00000     Min.   :0.0000      Min.   :0.0000
##  1st Qu.:53.87   1st Qu.:0.00000     1st Qu.:0.0000      1st Qu.:0.0000
##  Median :62.16   Median :0.00000     Median :1.0000      Median :0.0000
##  Mean   :61.25   Mean   :0.01212     Mean   :0.6364      Mean   :0.3654
##  3rd Qu.:68.83   3rd Qu.:0.00000     3rd Qu.:1.0000      3rd Qu.:1.0000
##  Max.   :83.03   Max.   :1.00000     Max.   :2.0000      Max.   :2.0000
##  chr17_rs393152_GT chr17_rs12185268_GT chr17_rs199533_GT   time_visit
##  Min.   :0.0000    Min.   :0.0000      Min.   :0.0000    Min.   : 0.00
##  1st Qu.:0.0000    1st Qu.:0.0000      1st Qu.:0.0000    1st Qu.: 9.00
##  Median :0.0000    Median :0.0000      Median :0.0000    Median :24.00
##  Mean   :0.4468    Mean   :0.4268      Mean   :0.4052    Mean   :23.83
##  3rd Qu.:1.0000    3rd Qu.:1.0000      3rd Qu.:1.0000    3rd Qu.:36.00
##  Max.   :2.0000    Max.   :2.0000      Max.   :2.0000    Max.   :54.00
```

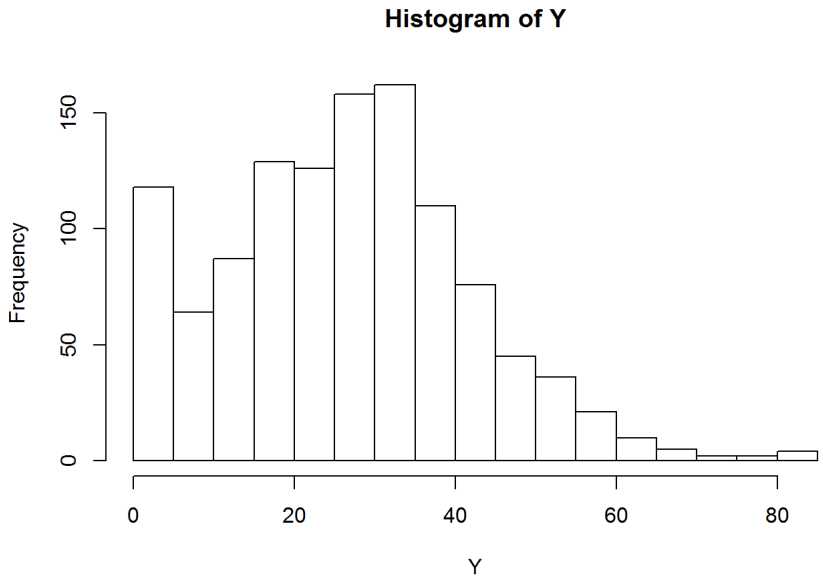
```
mode(X) <- 'numeric'

Dx_label <- Dx_label[data1.completeRowIndexes]; length(Dx_label)
```

```
## [1] 1155
```

9.2.4 Preparing the response vector

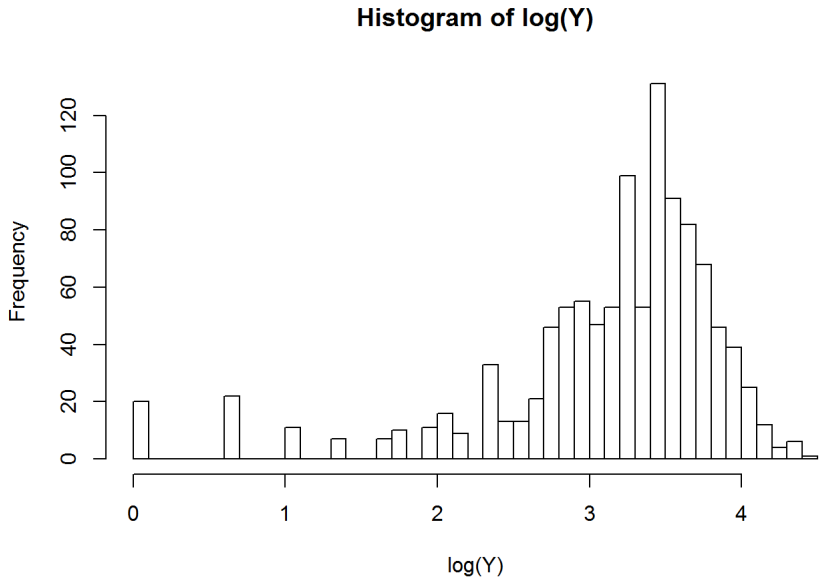
The knockoff filter is designed to control the FDR under Gaussian noise. A quick inspection of the response vector shows that it is highly non-Gaussian.

```
hist(Y, breaks='FD')
```



A `log-transform` may help to stabilize the clinical response measurements:

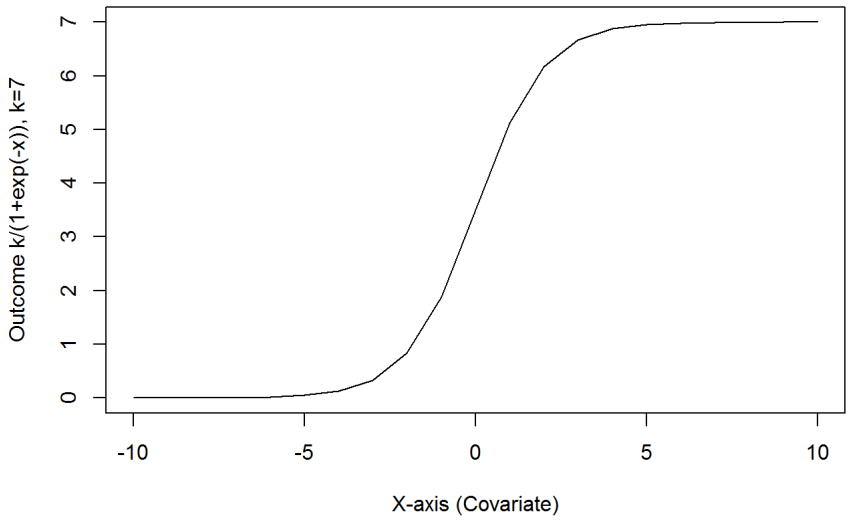
```
hist(log(Y), breaks='FD')
```



For **binary outcome variables**, or **ordinal categorical variables**, we can employ the `logistic curve` to transform the polytomous outcomes into real values.

The Logistic curve is $y = f(x) = \frac{1}{1+e^{-x}}$, where y and x represent probability and quantitative-predictor values, respectively. A slightly more general form is: $y = f(x) = \frac{K}{1+e^{-x}}$, where the covariate $x \in (-\infty, \infty)$ and the response $y \in [0, K]$. For example,

```
library("ggplot2")
k=7
x <- seq(-10, 10, 1)
plot(x, k/(1+exp(-x)), xlab="X-axis (Covariate)", ylab="Outcome k/(1+exp(-x)), k=7", type="l")
```



The point of this logistic transformation is that:

$$y = \frac{1}{1+e^{-x}} \Longleftrightarrow x = \ln \frac{y}{1-y},$$

which represents the `log-odds` (when y is the probability of an event of interest)!!!

We use the logistic regression equation model to estimate the probability of specific outcomes:

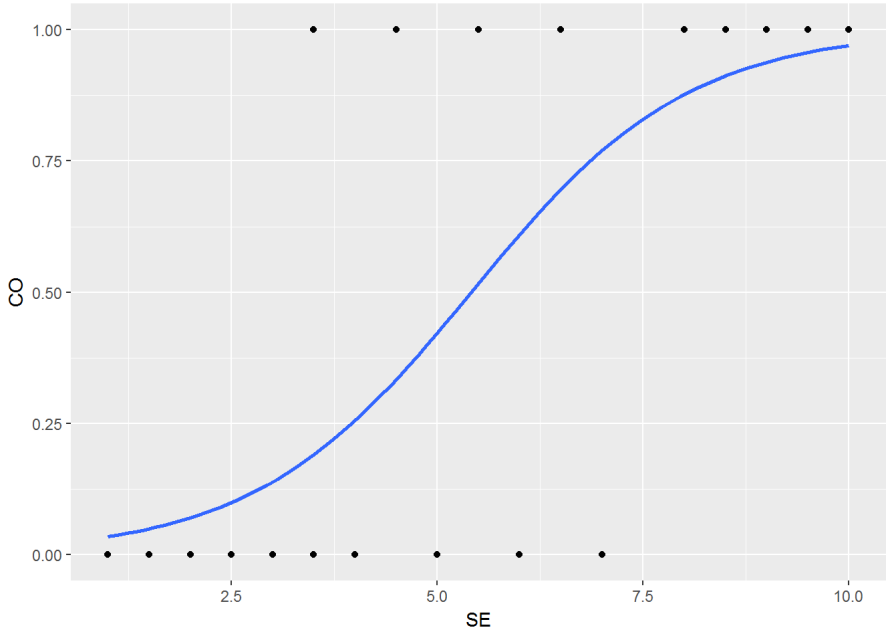
(Estimate of)$P(Y=1|x_1,x_2,.,x_l) = \frac{1}{1+e^{-(a_o+\sum_{k=1}^{l} a_k x_k)}}$, where the coefficients a_o (intercept) and effects $a_k, k=1,2,\ldots,l$, are estimated using GLM according to a maximum likelihood approach. Using this model allows us to estimate the probability of the dependent (clinical outcome) variable $Y=1$ (CO), i.e., surviving surgery, given the observed values of the predictors $X_k, k=1,2,\ldots,l$.

Probability of surviving a heart transplant based on surgeon's experience. A group of 20 patients undergo heart transplantation with different surgeons having experience in the range {0(least), 2, …, 10(most)}, representing 100's of operating/surgery hours. How does the surgeon's experience affect the probability of the patient survival?

The data below shows the outcome of the surgery (1=survival) or (0=death) according to the surgeons' experience in 100's of hours of practice.

Surgeon's Experience (SE)	**1**	**1.5**	**2**	**2.5**	**3**	**3.5**	**3.5**	**4**	**4.5**	**5**	**5.5**	**6**	**6.5**	**7**	**8**	**8.5**	**9**	**9.5**	**10**	**10**
Clinical Outcome (CO)	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

```
mydata <- read.csv("https://umich.instructure.com/files/405273/download?download_frd=1")  # 01_HeartSurgerySurvivalData.csv
# estimates a logistic regression model for the clinical outcome (CO), survival, using the glm
# (generalized linear model) function.
# convert Surgeon's Experience (SE) to a factor to indicate it should be treated as a categorical variable.
# mydata$rank <- factor(mydata$SE)
mylogit <- glm(CO ~ SE, data = mydata, family = "binomial")

# library(ggplot2)
ggplot(mydata, aes(x=SE, y=CO)) + geom_point() +
    stat_smooth(method="glm", method.args=list(family = "binomial"), se=FALSE)
```



Graph of a logistic regression curve showing probability of surviving the surgery versus surgeon's experience.

The graph shows the probability of the clinical outcome, survival, (Y-axis) versus the surgeon's experience (X-axis), with the logistic regression curve fitted to the data.

```
mylogit <- glm(CO ~ SE, data = mydata, family = "binomial")
summary(mylogit)
```

```
##
## Call:
## glm(formula = CO ~ SE, family = "binomial", data = mydata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7131  -0.5719  -0.0085   0.4493   1.8220
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -4.1030     1.7629  -2.327   0.0199 *
## SE            0.7583     0.3139   2.416   0.0157 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 27.726  on 19  degrees of freedom
## Residual deviance: 16.092  on 18  degrees of freedom
## AIC: 20.092
##
## Number of Fisher Scoring iterations: 5
```

The output indicates that surgeon's experience (SE) is significantly associated with the probability of surviving the surgery (0.0157, Wald test). The output also provides the coefficients for:

- Intercept = -4.1030 and SE = 0.7583.

These coefficients can then be used in the logistic regression equation model to estimate the probability of surviving the heart surgery:

Probability of surviving heart surgery $CO = 1/(1 + exp(-(-4.1030 + 0.7583 \times SE)))$

For example, for a patient who is operated by a surgeon with 200 hours of operating experience (SE=2), we plug in the value 2 in the equation to get an estimated probability of survival, $p = 0.07$:

```
SE=2
CO =1/(1+exp(-(-4.1030+0.7583*SE)))
CO
```

```
## [1] 0.07001884
```

[1] 0.07001884

Similarly, a patient undergoing heart surgery with a doctor that has 400 operating hours experience (SE=4), the estimated probability of survival is p=0.26:

```
SE=4; CO =1/(1+exp(-(-4.1030+0.7583*SE))); CO
```

```
## [1] 0.2554411
```

```
CO
```

```
## [1] 0.2554411
```

```
for (SE in c(1:5)) {
  CO <- 1/(1+exp(-(-4.1030+0.7583*SE)));
  print(c(SE, CO))
}
```

```
## [1] 1.00000000 0.03406915
## [1] 2.00000000 0.07001884
## [1] 3.0000000 0.1384648
## [1] 4.0000000 0.2554411
## [1] 5.0000000 0.4227486
```

[1] 0.2554411

The table below shows the probability of surviving surgery for several values of surgeons' experience.

Surgeon's Experience (SE)	Probability of patient survival (Clinical Outcome)
1	0.034
2	0.07
3	0.14
4	0.26
5	0.423

The output from the logistic regression analysis gives a p-value of $p = 0.0157$, which is based on the Wald z-score. In addition to the Wald method, we can calculate the p-value for logistic regression using the Likelihood Ratio Test (LRT), which for these data yields 0.0006476922.

```
mylogit <- glm(CO ~ SE, data = mydata, family = "binomial")
summary(mylogit)
```

```
##
## Call:
## glm(formula = CO ~ SE, family = "binomial", data = mydata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7131  -0.5719  -0.0085   0.4493   1.8220
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -4.1030     1.7629  -2.327   0.0199 *
## SE            0.7583     0.3139   2.416   0.0157 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 27.726  on 19  degrees of freedom
## Residual deviance: 16.092  on 18  degrees of freedom
## AIC: 20.092
##
## Number of Fisher Scoring iterations: 5
```

.	Estimate	Std. Error	z value	$Pr(
SE	0.7583	0.3139	2.416	0.0157 *

The *logit* of a number $0 \leq p \leq 1$ is given by the formula: $logit(p) = log\frac{p}{1-p}$, and represents the log-odds ratio (of survival in this case).

```
       confint(mylogit)
```

```
## Waiting for profiling to be done...
```

```
##                 2.5 %    97.5 %
## (Intercept) -8.6083535 -1.282692
## SE           0.2687893  1.576912
```

So, why `exponentiating the coefficients` ? Because,

$$logit(p) = log\frac{p}{1-p} \longrightarrow e^{logit(p)} = e^{log\frac{p}{1-p}} \longrightarrow RHS = \frac{p}{1-p}, \text{ (odds-ratio, OR)}.$$

```
exp(coef(mylogit))    # exponentiated logit model coefficients
```

```
## (Intercept)          SE
##  0.01652254  2.13474149
```

- (Intercept) SE
- 0.01652254 2.13474149 == exp(0.7583456)
- coef(mylogit) # raw logit model coefficients
- (Intercept) SE
- -4.1030298 0.7583456

```
exp(cbind(OR = coef(mylogit), confint(mylogit)))
```

```
## Waiting for profiling to be done...
```

```
##                    OR        2.5 %   97.5 %
## (Intercept) 0.01652254 0.0001825743 0.277290
## SE          2.13474149 1.3083794719 4.839986
```

.	OR	2.5%	97.5%
(Intercept)	0.01652254	0.0001825743	0.277290
SE	2.13474149	1.3083794719	4.839986

We can compute the LRT and report its p-values by using the *with()* function:

`with(mylogit, df.null - df.residual)`

```
with(mylogit, pchisq(null.deviance - deviance, df.null - df.residual, lower.tail = FALSE))
```

```
## [1] 0.0006476922
```

[1] 0.0006476922

LRT p-value < 0.001 tells us that our model as a whole fits significantly better than an empty model. The deviance residual is `-2*log likelihood`, and we can report the model's log likelihood by:

```
logLik(mylogit)
```

```
## 'log Lik.' -8.046117 (df=2)
```

$$\log \text{Lik.} = -8.046117 \ (df = 2).$$

9.3 Note

The LRT compares the data fit of two models. For instance, removing predictor variables from a model may reduce the model quality (i.e., a model will have a lower log likelihood). To statistically assess whether the observed difference in model fit is significant, the LRT compares the difference of the log likelihoods of the two models. When this difference is statistically significant, the full model (the one with more variables) represents a better fit to the data, compared to the reduced model. LRT is computed using the log likelihoods (ll) of the two models:

$$LRT = -2 \ln\left(\frac{L(m_1)}{L(m_2)}\right) = 2(ll(m_2) - ll(m_1)),$$

where:

- m_1 and m_2 are the reduced and the full models, respectively,
- $L(m_1)$ and $L(m_2)$ denote the likelihoods of the 2 models, and
- $ll(m_1)$ and $ll(m_2)$ represent the *log likelihood* (natural log of the model likelihood function).

As $n \longrightarrow \infty$, the distribution of the LRT is asymptotically chi-squared with degrees of freedom equal to the number of parameters that are reduced (i.e., the number of variables removed from the model). In our case, $LRT \sim \chi^2_{df=2}$, as we have an intercept and one predictor (SE), and the null model is empty (no parameters).

9.3.1 False Discovery Rate (FDR)

A measure of performance of a test is the FDR rate:

$$\underbrace{FDR}_{\text{False Discovery Rate}} = \underbrace{E}_{\text{expectation}}\underbrace{\left(\frac{\#FalsePositives}{\text{total number of selected features}}\right)}_{\text{False Discovery Proportion}}.$$

The Benjamini-Hochberg (BH) FDR procedure involves ordering the p-values, specifying a target FDR, calculating and applying the threshold. Below we show how this is accomplished in R.

```
#p-values entered from smallest to largest
pvals <- c(0.9, 0.35, 0.01, 0.013, 0.014, 0.19, 0.35, 0.5, 0.63, 0.67, 0.75, 0.81, 0.01, 0.051)
length(pvals)
```

```
## [1] 14
```

```
#enter the target FDR
alpha.star <- 0.05

# order the p-values small to large
pvals <- sort(pvals); pvals
```

```
##  [1] 0.010 0.010 0.013 0.014 0.051 0.190 0.350 0.350 0.500 0.630 0.670
## [12] 0.750 0.810 0.900
```

```
#calculate the threshold for each p-value
threshold<-alpha.star*(1:length(pvals))/length(pvals)

#compare the p-value against its threshold and display results
cbind(pvals, threshold, pvals<=threshold)
```

```
##       pvals   threshold
##  [1,] 0.010 0.003571429 0
##  [2,] 0.010 0.007142857 0
##  [3,] 0.013 0.010714286 0
##  [4,] 0.014 0.014285714 1
##  [5,] 0.051 0.017857143 0
##  [6,] 0.190 0.021428571 0
##  [7,] 0.350 0.025000000 0
##  [8,] 0.350 0.028571429 0
##  [9,] 0.500 0.032142857 0
## [10,] 0.630 0.035714286 0
## [11,] 0.670 0.039285714 0
## [12,] 0.750 0.042857143 0
## [13,] 0.810 0.046428571 0
## [14,] 0.900 0.050000000 0
```

Start with the smallest p-value and move up we find that the largest k for which the p-value is less than its threshold, α^*, which is $\hat{k} = 4$.

Next, the algorithm rejects the null hypotheses for the tests that correspond to the p-values $p_{(1)}, p_{(2)}, p_{(3)}, p_{(4)}$.

Note: that since we controlled FDR at $\alpha^* = 0.05$, we are guaranteed that on average only 5% of the tests that we rejected are spurious. Since $\alpha^* = 0.05$ of 4 is quite small and less than 1, we are confident that none of our rejections are spurious ones.

The Bonferroni corrected α for these data is $\frac{0.05}{14} = 0.0036$. If we had used this family-wise error rate in our individual hypothesis tests, then we would have concluded that none of our 14 results were significant!

9.3.2 Graphical Interpretation of the Benjamini-Hochberg (BH) Method

There's a graphical interpretation of the BH calculations.

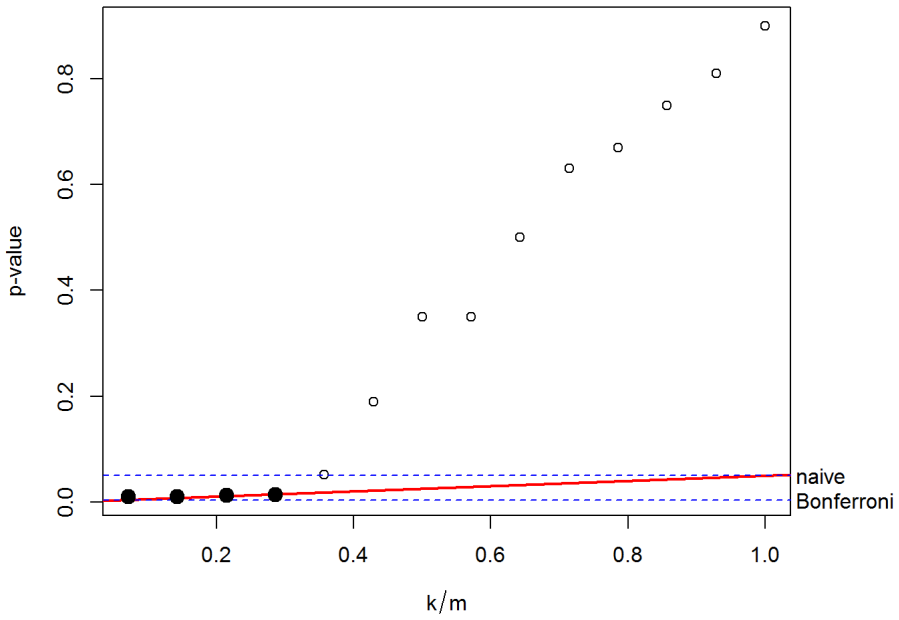
- Sort the p-values from largest to smallest.
- Plot the ordered p-values $p_{(k)}$ on the y-axis versus their indexes on the x-axis.
- Superimpose on this plot a line that passes through the origin and has slope α^*.

Any p-value that falls on or below this line corresponds to a significant result.

```
#generate the values to be plotted on x-axis
x.values<-(1:length(pvals))/length(pvals)

#widen right margin to make room for labels
par(mar=c(4.1, 4.1, 1.1, 4.1))

#plot points
plot(x.values, pvals, xlab=expression(k/m), ylab="p-value") #, ylim=c(0.0, 0.4))

#add FDR line
abline(0, .05, col=2, lwd=2)

#add naive threshold line
abline(h=.05, col=4, lty=2)

#add Bonferroni-corrected threshold line
abline(h=.05/length(pvals), col=4, lty=2)

#label lines
mtext(c('naive', 'Bonferroni'), side=4, at=c(.05, .05/length(pvals)), las=1, line=0.2)

#select observations that are less than threshold
for.test <- cbind(1:length(pvals), pvals)
pass.test <- for.test[pvals <= 0.05*x.values, ]
pass.test
```

```
##       pvals
## 4.000 0.014
```

```
#use largest k to color points that meet Benjamini-Hochberg FDR test
last<-ifelse(is.vector(pass.test), pass.test[1], pass.test[nrow(pass.test), 1])
points(x.values[1:last], pvals[1:last], pch=19, cex=1.5)
```



9.3.3 FDR Adjusting the p-values

R can automatically performs the Benjamini-Hochberg procedure. The adjusted p-values are obtained by

```
pvals.adjusted <- p.adjust(pvals, "BH")
```

The adjusted p-values indicate the corresponding null hypothesis we need to reject to preserve the initial α^* false-positive rate. We can also compute the adjusted p-values as follows:

```
#calculate the term that appears in the innermost minimum function
test.p <- length(pvals)/(1:length(pvals))*pvals
test.p
```

```
##  [1] 0.14000000 0.07000000 0.06066667 0.04900000 0.14280000 0.44333333
##  [7] 0.70000000 0.61250000 0.77777778 0.88200000 0.85272727 0.87500000
## [13] 0.87230769 0.90000000
```

```
#use a loop to run through each p-value and carry out the adjustment
adj.p <- numeric(14)
for(i in 1:14) {
    adj.p[i]<-min(test.p[i:length(test.p)])
    ifelse(adj.p[i]>1, 1, adj.p[i])
}
adj.p
```

```
##  [1] 0.0490000 0.0490000 0.0490000 0.0490000 0.1428000 0.4433333 0.6125000
##  [8] 0.6125000 0.7777778 0.8527273 0.8527273 0.8723077 0.8723077 0.9000000
```

Note that the manually computed (`adj.p`) and the automatically computed (`pvals.adjusted`) adjusted-p-values are the same.

9.4 Running the knockoff filter

We now run the knockoff filter on each drug separately. We also run the Benjamini-Hochberg (BH) procedure for later comparison. More details about the Knock-off filtering methods are available here (https://www.stat.uchicago.edu/~rina/knockoff/knockoff_slides.pdf).

Before running either selection procedure, remove rows with missing values, reduce the design matrix by removing predictor columns that do not appear frequently (e.g., at least three times in the sample), and remove any columns that are duplicates.

```
library(knockoff)

# Direct call to knockoff filtering looks like this:
fdr <- 0.1
result = knockoff.filter(X, Y, fdr = fdr, knockoffs = 'equicorrelated'); names(result$selected)
```

```
## [1] "L_cingulate_gyrus_ComputeArea" "R_putamen_ComputeArea"
## [3] "Sex"                           "Weight"
## [5] "Age"                           "chr12_rs34637584_GT"
## [7] "chr17_rs11012_GT"              "chr17_rs199533_GT"
```

```
knockoff_selected <- names(result$selected)

# Run BH (Benjamini-Hochberg)
k = ncol(X)
lm.fit = lm(Y ~ X - 1) # no intercept
# Alternatively: dat = as.data.frame(cbind(Y,X))
# lm.fit = lm(Y ~ . -1,data=dat ) # no intercept

p.values = coef(summary(lm.fit))[, 4]
cutoff = max(c(0, which(sort(p.values) <= fdr * (1:k) / k)))
BH_selected = names(which(p.values <= fdr * cutoff / k))

knockoff_selected; BH_selected
```

```
## [1] "L_cingulate_gyrus_ComputeArea" "R_putamen_ComputeArea"
## [3] "Sex"                           "Weight"
## [5] "Age"                           "chr12_rs34637584_GT"
## [7] "chr17_rs11012_GT"              "chr17_rs199533_GT"
```

```
## [1] "XL_putamen_ComputeArea" "XL_putamen_Volume"
## [3] "XSex"                   "XWeight"
## [5] "XAge"                   "Xchr17_rs11868035_GT"
## [7] "Xchr17_rs11012_GT"      "Xchr17_rs12185268_GT"
```

```
# Housekeeping: remove the "X" prefixes in the BH_selected list of features
for(i in 1:length(BH_selected)){
  BH_selected[i] <- substring(BH_selected[i], 2)
}

intersect(BH_selected,knockoff_selected)
```

```
## [1] "Sex"              "Weight"           "Age"
## [4] "chr17_rs11012_GT"
```

We see that there are some features that are selected by both methods suggesting they may be indeed salient.

Try to apply some of these techniques to other data from the list of our Case-Studies (https://umich.instructure.com/courses/38100/files/).