BUS41204: Week 5 Review Session

JungHo Lee

Variable Selection

Agenda:

1. Stepwise Selection Methods
2. Shrinkage Methods and Cross-Validation
3. Boruta Method

Why Select Variables?

- Performance: Predictive performance is often degraded as the number of uninformative predictors (noise) increases
- Interpretability: Simpler models are easier to interpret
- Computational ease: Simpler models require less computational resources
- Trade-off between interpretability/low variance for a small model vs. reducing bias/model error in the larger model.

Stepwise Selection Methods

We will use simulated data to see how backward elimination and forward selection can be implemented in practice. We can also combine the two (add a covariate, check if any can be removed, add another, etc).

Simulate our data:

```
set.seed(41204)

n = 30
p = 15 # total number of predictors (without the intercept)
s = 10 # true model size
X = scale(matrix(rnorm(n*p),n,p), center=FALSE) # covariate matrix

beta = numeric(p)
beta[1:s] = runif(s,-5,5)
y = X%*%beta + rnorm(n) # response
```

Backward Elimination

Backward elimination: Start with a model with all covariates, remove one at a time.

If we use p-value as our stopping criterion, at each step, we

- Remove the variable with the highest (least significant) p-value,

- Stop if all p-values are $\leq$ threshold.

```
#' @param y: a vector of responses
#' @param X: covariate matrix
#' @param alpha: threshold p-value
#' @return XS: matrix of selected covariates
run_backward_elimination = function(y, X, alpha){
  p = dim(X)[2]; S = 1:p

    while(TRUE){
    pvals = summary(lm(y~X[,S]))$coefficients[-1,4]

    # stopping criterion
    if(max(pvals) <= alpha){
      break
    }

    # remove var with highest p-val
        remove_ind = S[which.max(pvals)]
        S = setdiff(S,remove_ind)
    }
    XS = X[,S,drop=FALSE]; colnames(XS) = S

    return(XS)
}
```

Run backward elimination with the threshold p-value of 0.1:

```
XS = run_backward_elimination(y, X, 0.1)
summary(lm(y~XS))$coefficients
```

```
##                Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) -0.2833323  0.2335809  -1.212995 2.408112e-01
## XS1          4.2855572  0.2149754  19.935103 1.018806e-13
## XS2          4.1100551  0.2468707  16.648612 2.224626e-12
## XS3          4.2201468  0.2797961  15.082937 1.175373e-11
## XS4         -0.7169119  0.1976271  -3.627599 1.925272e-03
## XS5          3.6609668  0.2032968  18.007988 5.844390e-13
## XS6          2.1171880  0.1720068  12.308746 3.348967e-10
## XS7         -2.7291292  0.1938002 -14.082181 3.689710e-11
## XS8         -4.6479553  0.2149009 -21.628361 2.483644e-14
## XS9          1.3244227  0.2756905   4.804021 1.421450e-04
## XS10         2.4304440  0.1874281  12.967343 1.434512e-10
## XS13         0.4392526  0.2121373   2.070606 5.305145e-02
```

Run backwards elimination with the threshold p-value of 0.01:

```
XS = run_backward_elimination(y, X, 0.01)
summary(lm(y~XS))$coefficients
```

```
##                Estimate Std. Error     t value     Pr(>|t|)
## (Intercept) -0.2351602  0.2517244  -0.9341972 3.619238e-01
```

```
## XS1          4.2890745  0.2328243  18.4219390 1.411117e-13
## XS2          3.8687117  0.2356987  16.4138020 1.117434e-12
## XS3          4.5618390  0.2447194  18.6411027 1.139851e-13
## XS4         -0.8330615  0.2052391  -4.0589812 6.696228e-04
## XS5          3.6702672  0.2201292  16.6732449 8.449237e-13
## XS6          2.1993166  0.1812732  12.1326077 2.151493e-10
## XS7         -2.6935055  0.2090687 -12.8833501 7.735793e-11
## XS8         -4.4946894  0.2185087 -20.5698413 1.909890e-14
## XS9          1.0139362  0.2505604   4.0466734 6.887129e-04
## XS10         2.3829594  0.2014708  11.8278128 3.305918e-10
```

Just by random chance, some of the covariates might come with a low p-value, more so when the number of predictors is large.

```r
y_noise = rnorm(n)
XS = run_backward_elimination(y_noise, X, 0.1)
summary(lm(y_noise~XS))$coefficients
```

```
##                Estimate Std. Error    t value    Pr(>|t|)
## (Intercept)  0.06345579  0.1522714  0.4167281 0.680054824
## XS          -0.43975648  0.1548745 -2.8394367 0.008323167
```

How do we address this issue?

Bayesian Information Criterion (BIC) $=$ deviance $+\log(n) * (\text{length}(\text{variable}) + 1)$ takes into account the size of the model.

Up to a constant, this is equal to $\text{BIC(S)} = n\log(RSS(\text{model S})/n) + |S|log(n)$.

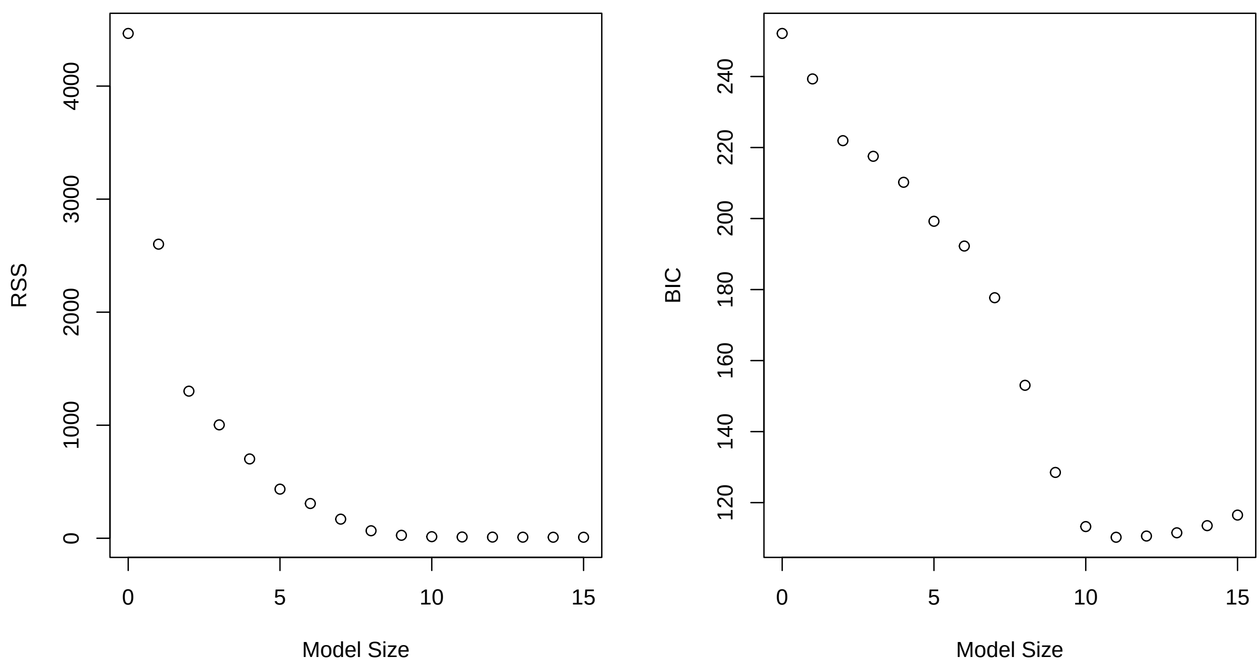
Apply the BIC to backward elimination:

```r
#' @param y: a vector of responses
#' @param X: covariate matrix
#' @return plots of RSS and BIC as a function of model size
run_backward_elimination_BIC = function(y, X){
    S = 1:p
    S_in_order = rep(0,p)
    store_RSS = rep(0,p+1)

    store_RSS[p+1] = sum((y-lm(y~X[,S])$fitted.values)^2)

    for(i in 1:p){
        pvals = summary(lm(y~X[,S]))$coefficients[-1,4]
      remove_ind = S[which.max(pvals)]
        S_in_order[p+1-i] = remove_ind
        S = setdiff(S,remove_ind)

        if(length(S)>0){
          store_RSS[p+1-i] = sum((y-lm(y~X[,S])$fitted.values)^2)
        }
        else{store_RSS[p+1-i] = sum((y-lm(y~1)$fitted.values)^2)}}

    BIC = n*log(store_RSS) + (0:p)*log(n)
    modelsize = which.min(BIC)-1
```

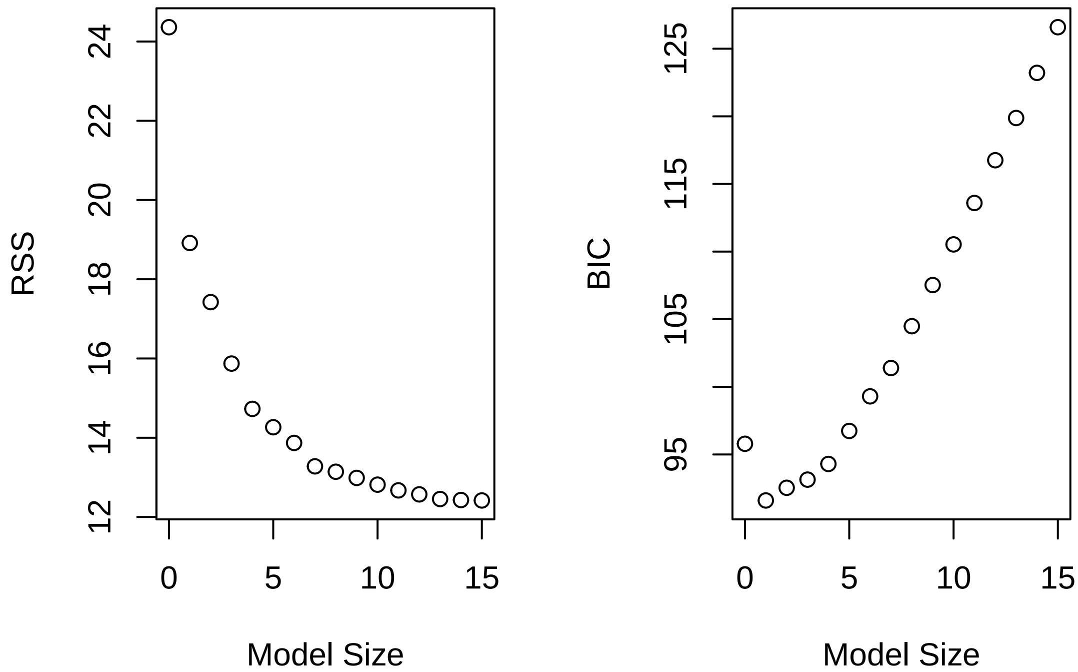
```
    par(mfrow=c(1,2))
    plot(0:p, store_RSS, xlab='Model Size', ylab='RSS')
    plot(0:p, BIC, xlab='Model Size', ylab='BIC')
}
```

Run with signal:

```
run_backward_elimination_BIC(y, X)
```




```
run_backward_elimination_BIC(y_noise, X)
```



Forward Selection

Start with a model with no covariates, add in one at a time.

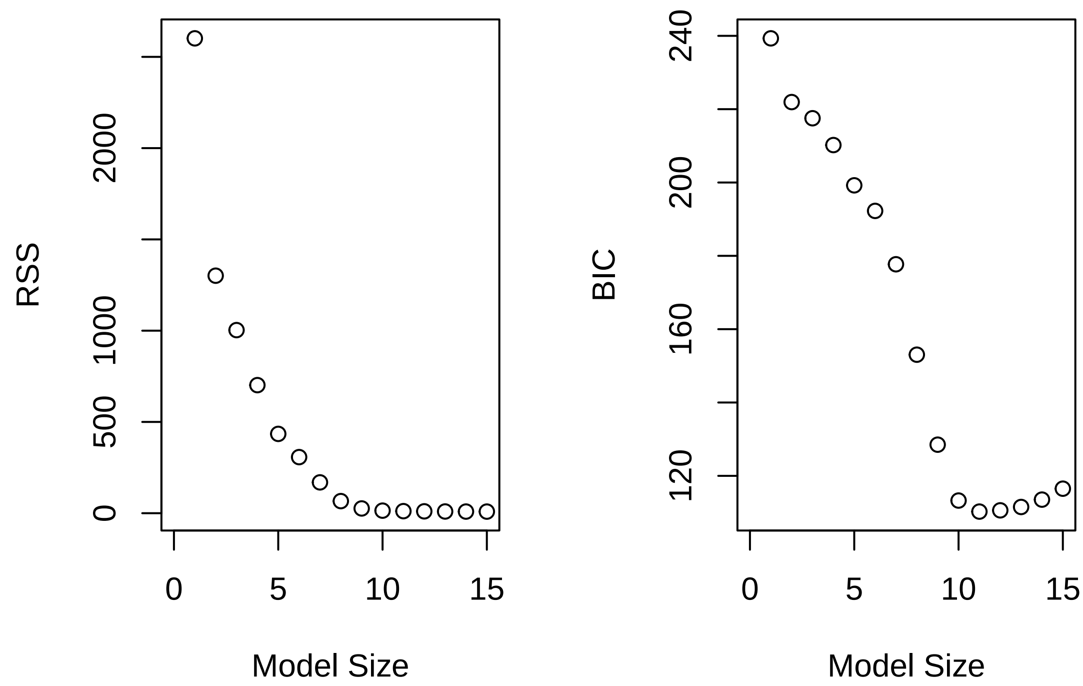
At each step,

- Select the variable that decreases the RSS the most
- Stop if all p-values are $\geq$ threshold

```
#' @param y: a vector of responses
#' @param X: covariate matrix
#' @return plots of RSS and BIC as a function of model size
run_forward_selection_BIC = function(y, X){
  S = store_RSS = c()
  p = dim(X)[2]

  for(i in 1:p){
        RSS = rep(0,p)
        for(j in 1:p){
          S0 = append(S,j)
          RSS[j] = sum((y-lm(y~X[,S0])$fitted.values)^2)
        }
        store_RSS[i+1] = min(RSS)
        ind = which.min(RSS)
        S = append(S,ind)
  }

    BIC = n*log(store_RSS) + (0:p)*log(n)
    modelsize = which.min(BIC)-1

        par(mfrow=c(1,2))
```

```
        plot(0:p, store_RSS, xlab='Model Size', ylab='RSS')
    plot(0:p, BIC, xlab='Model Size', ylab='BIC')
}
```

```
run_forward_selection_BIC(y, X)
```



Best Subset

Compares all models for each possible combination of the p predictors. Since there are 2^p models to compare, it cannot be applied when p is large.

```
library(bestglm)

data = cbind(X,y)
bestglm(as.data.frame(data), family = gaussian, IC = "BIC", method = "exhaustive")
```

```
## BIC
## BICq equivalent for q in (0.181806937963117, 0.542773421034033)
## Best Model:
##                Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) -0.2833323  0.2335809  -1.212995 2.408112e-01
## V1           4.2855572  0.2149754  19.935103 1.018806e-13
## V2           4.1100551  0.2468707  16.648612 2.224626e-12
## V3           4.2201468  0.2797961  15.082937 1.175373e-11
## V4          -0.7169119  0.1976271  -3.627599 1.925272e-03
## V5           3.6609668  0.2032968  18.007988 5.844390e-13
## V6           2.1171880  0.1720068  12.308746 3.348967e-10
## V7          -2.7291292  0.1938002 -14.082181 3.689710e-11
## V8          -4.6479553  0.2149009 -21.628361 2.483644e-14
```

```
## V9           1.3244227  0.2756905   4.804021 1.421450e-04
## V10          2.4304440  0.1874281  12.967343 1.434512e-10
## V13          0.4392526  0.2121373   2.070606 5.305145e-02
```

Shrinkage Methods

Main idea: By shrinking the coefficient, we compromise some bias for less variance. As a result, we get better prediction performance.

Lasso and Ridge Methods

Lasso: $\hat{\beta} = \text{argmin}_\beta \Big\{ \text{deviance} + \lambda \|\beta\|_1 \Big\}$ **Ridge**: $\hat{\beta} = \text{argmin}_\beta \Big\{ \text{deviance} + \lambda \|\beta\|_2^2 \Big\}$

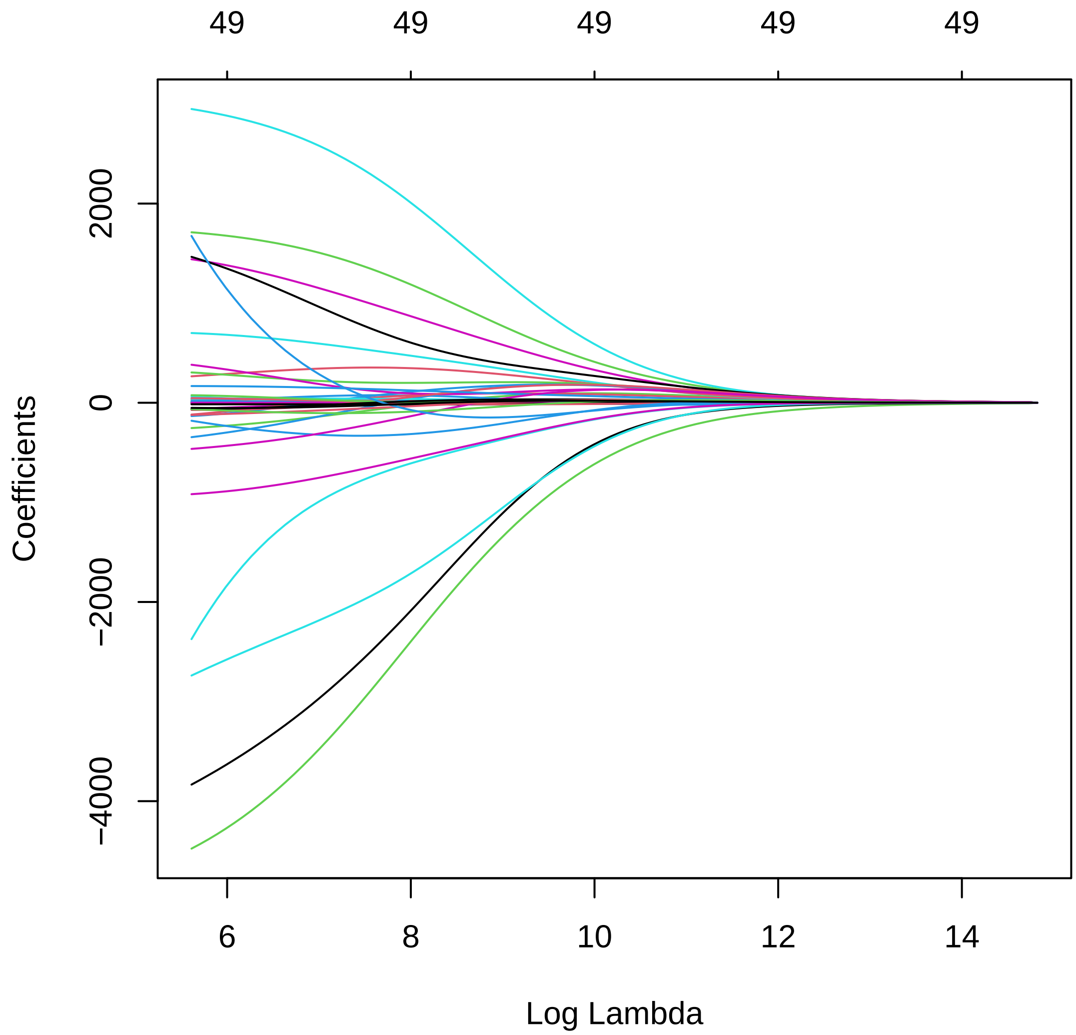
```
ames = as.matrix(read.table('ames_data.txt', header=TRUE))

y_name = 'LotArea'

ind = which(ames[,colnames(ames)=='YearBuilt']>=2000) # only those built after 2000

X = ames[ind, colnames(ames)!=y_name]
y = ames[ind, colnames(ames)==y_name]

n = length(y); p = dim(X)[2]
```

```
colnames(X)
```

```
##  [1] "SalePrice"     "LotFrontage"   "OverallQual"   "OverallCond"
##  [5] "YearBuilt"     "YearRemod.Add" "MasVnrArea"    "BsmtFinSF1"
##  [9] "BsmtFinSF2"    "BsmtUnfSF"     "X1stFlrSF"     "X2ndFlrSF"
## [13] "LowQualFinSF"  "BsmtFullBath"  "BsmtHalfBath"  "FullBath"
## [17] "HalfBath"      "BedroomAbvGr"  "KitchenAbvGr"  "TotRmsAbvGrd"
## [21] "Fireplaces"    "GarageCars"    "GarageArea"    "WoodDeckSF"
## [25] "OpenPorchSF"   "EnclosedPorch" "X3SsnPorch"    "ScreenPorch"
## [29] "PoolArea"      "MiscVal"       "MoSold"        "YrSold"
## [33] "LotShape"      "LandSlope"     "Railroad"      "ExterQual"
## [37] "ExterCond"     "BsmtQual"      "BsmtCond"      "BsmtExposure"
## [41] "CentralAir"    "KitchenQual"   "Functional"    "FireplaceQu"
## [45] "GarageFinish"  "GarageQual"    "GarageCond"    "PavedDrive"
## [49] "Fence"
```

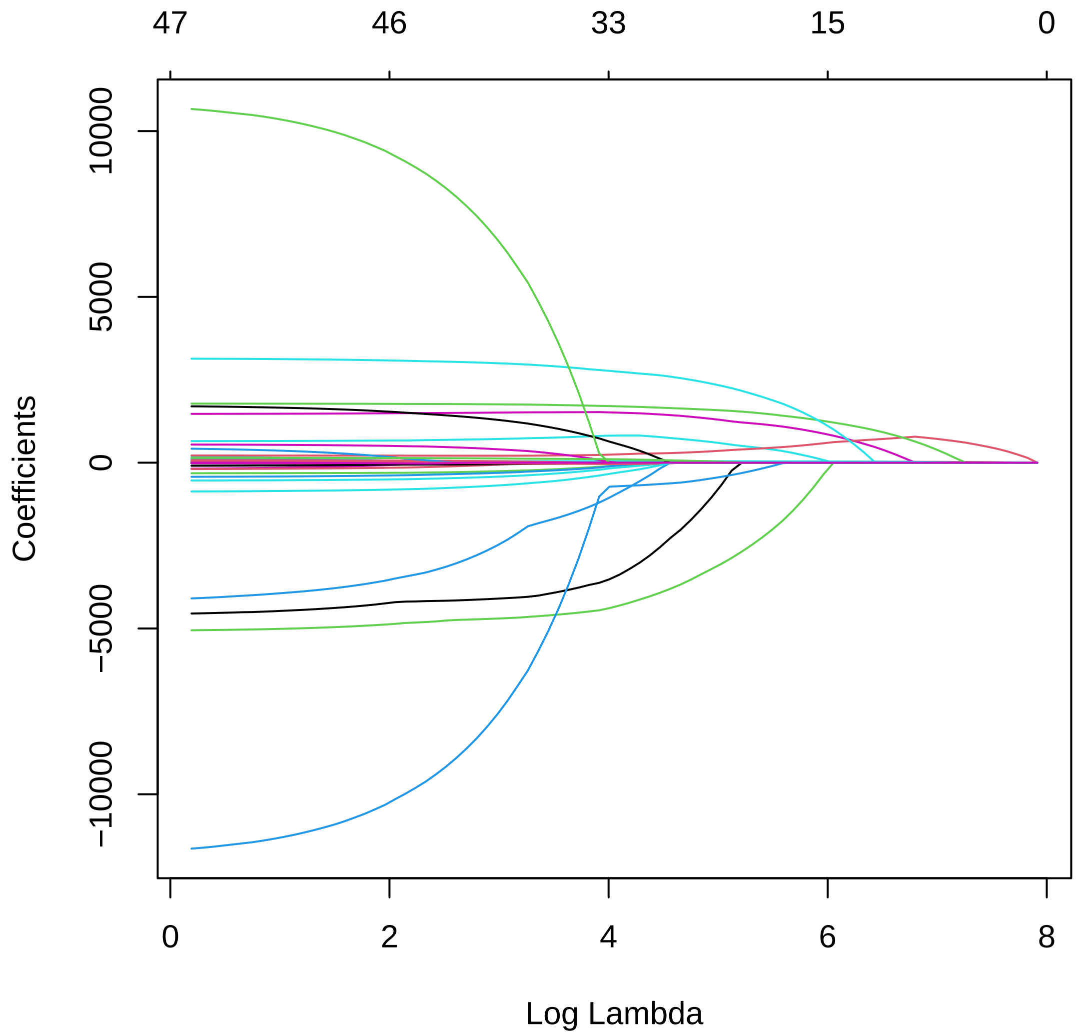
Ridge regression:

```
library(glmnet)

plot(glmnet(X, y, alpha=0), xvar = "lambda") # alpha = 0 is ridge penalty
```



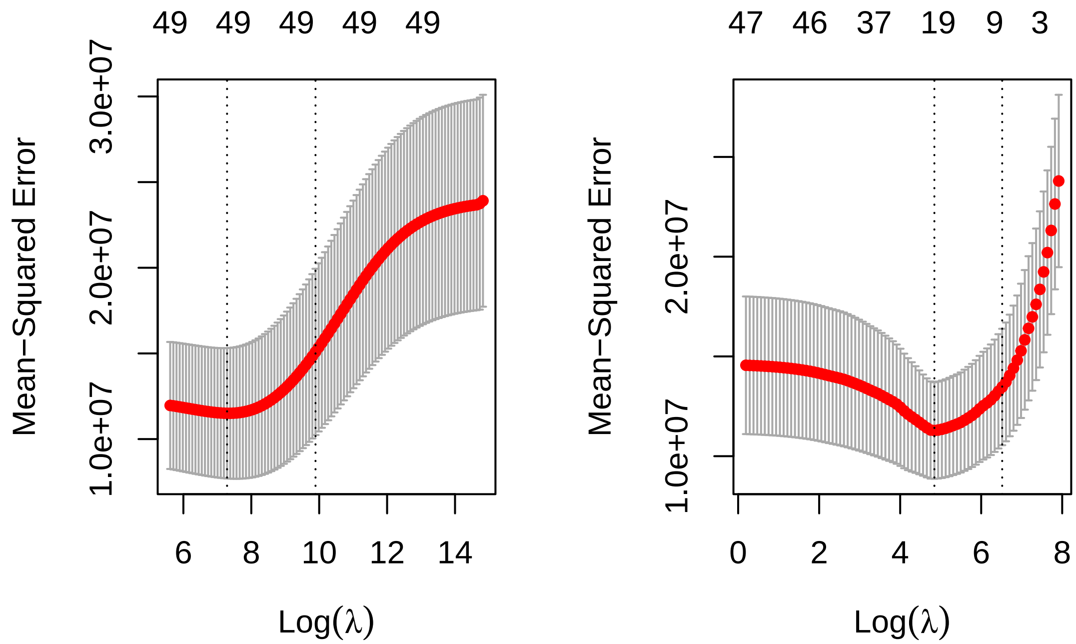
Lasso regression:

```
plot(glmnet(X, y, alpha=1), xvar = "lambda") # alpha = 1 is lasso penalty
```



Cross-Validation

```
# default is 10-fold cv
cv_ridge = cv.glmnet(X, y, alpha=0) # ridge
cv_lasso = cv.glmnet(X, y, alpha=1) # lasso

par(mfrow=c(1,2))
plot(cv_ridge)
plot(cv_lasso)
```




```r
coef(cv_ridge)
```

```
## 50 x 1 sparse Matrix of class "dgCMatrix"
##                          s1
## (Intercept)   -4.121003e+04
## SalePrice      1.994301e-03
## LotFrontage    1.051496e+01
## OverallQual    9.547285e+01
## OverallCond   -1.308557e+00
## YearBuilt      1.725133e+01
## YearRemod.Add  1.659878e+01
## MasVnrArea     7.022319e-01
## BsmtFinSF1     2.916358e-01
## BsmtFinSF2     8.545437e-02
## BsmtUnfSF      1.070839e-01
## X1stFlrSF      5.325704e-01
## X2ndFlrSF      2.180455e-01
## LowQualFinSF  -1.044580e+01
## BsmtFullBath   8.741627e+01
## BsmtHalfBath  -6.759623e+02
## FullBath       1.772554e+02
## HalfBath       2.183957e+02
## BedroomAbvGr   3.539865e+02
## KitchenAbvGr  -4.695120e+02
## TotRmsAbvGrd   1.987698e+02
## Fireplaces     1.918232e+02
## GarageCars     1.800561e+02
## GarageArea     1.020518e+00
## WoodDeckSF     9.336234e-01
## OpenPorchSF    2.143925e+00
## EnclosedPorch -1.552856e-01
## X3SsnPorch     3.291962e-01
## ScreenPorch    2.424142e+00
```

```
## PoolArea       1.613713e+01
## MiscVal        2.386155e-01
## MoSold        -4.291208e+00
## YrSold        -1.267719e+01
## LotShape       4.431284e+02
## LandSlope     -8.645766e+01
## Railroad       6.460126e+02
## ExterQual      1.330267e+02
## ExterCond      2.818435e+02
## BsmtQual       1.810896e+02
## BsmtCond       2.560575e+01
## BsmtExposure   6.913018e+01
## CentralAir    -4.904661e+02
## KitchenQual    1.246437e+02
## Functional     3.322517e+01
## FireplaceQu    2.687281e+01
## GarageFinish   9.423348e+00
## GarageQual    -8.755350e+01
## GarageCond    -1.860231e+02
## PavedDrive    -1.798139e+02
## Fence          2.051726e+01
```

```
coef(cv_lasso)
```

```
## 50 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)   -520.3328559
## SalePrice        .
## LotFrontage     27.4219543
## OverallQual      .
## OverallCond      .
## YearBuilt        .
## YearRemod.Add    .
## MasVnrArea       .
## BsmtFinSF1       0.2040883
## BsmtFinSF2       .
## BsmtUnfSF        .
## X1stFlrSF        1.3843547
## X2ndFlrSF        .
## LowQualFinSF     .
## BsmtFullBath     .
## BsmtHalfBath     .
## FullBath         .
## HalfBath         .
## BedroomAbvGr   372.5190961
## KitchenAbvGr     .
## TotRmsAbvGrd   718.4876000
## Fireplaces       .
## GarageCars       .
## GarageArea       0.3962021
## WoodDeckSF       .
## OpenPorchSF      .
## EnclosedPorch    .
## X3SsnPorch       .
```

```
## ScreenPorch      .
## PoolArea        29.8705286
## MiscVal          .
## MoSold           .
## YrSold           .
## LotShape       907.6041627
## LandSlope        .
## Railroad         .
## ExterQual        .
## ExterCond        .
## BsmtQual         .
## BsmtCond         .
## BsmtExposure     .
## CentralAir       .
## KitchenQual      .
## Functional       .
## FireplaceQu      .
## GarageFinish     .
## GarageQual       .
## GarageCond       .
## PavedDrive       .
## Fence            .
```

```
# predict(cv_lasso, newdata)
```

Boruta Method

Stochastic wrapper procedure that uses random forest to compute variable importance measures

Overview of the Boruta algorithm:

- Adds randomness to data by creating shuffled copies of all features (shadow features)
- Train a random forest on the extended data set to compute feature importance
- Iteratively remove features that are less important than the best shadow features
- Stops when all features are confirmed or rejected or a specified limit of random forest runs is reached.

We import `BankChurners.csv` data from Kaggle. We then delete the last 2 variables suggested from Kaggle data description, and `CLIENTNUM` since it's Client's ID number which is unique and does not affect our prediction.

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
churners = read.csv('BankChurners.csv')
str(churners)
```

```
## 'data.frame':    10127 obs. of  23 variables:
##  $ CLIENTNUM
```

```
##  $ Attrition_Flag
##  $ Customer_Age
##  $ Gender
##  $ Dependent_count
##  $ Education_Level
##  $ Marital_Status
##  $ Income_Category
##  $ Card_Category
##  $ Months_on_book
##  $ Total_Relationship_Count
##  $ Months_Inactive_12_mon
##  $ Contacts_Count_12_mon
##  $ Credit_Limit
##  $ Total_Revolving_Bal
##  $ Avg_Open_To_Buy
##  $ Total_Amt_Chng_Q4_Q1
##  $ Total_Trans_Amt
##  $ Total_Trans_Ct
##  $ Total_Ct_Chng_Q4_Q1
##  $ Avg_Utilization_Ratio
##  $ Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Educati
##  $ Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Educati
```

```
churners= churners[,-c(1,22,23)]
```

```
churners$Attrition_Flag = as.factor(churners$Attrition_Flag)

set.seed(41204)

n = dim(churners)[1]
ind = sample(n, floor(0.75*n))
train = churners[ind,]
test = churners[-ind,]

table(train$Attrition_Flag)
```

```
##
## Attrited Customer Existing Customer
##              1206              6389
```

```
table(test$Attrition_Flag)
```

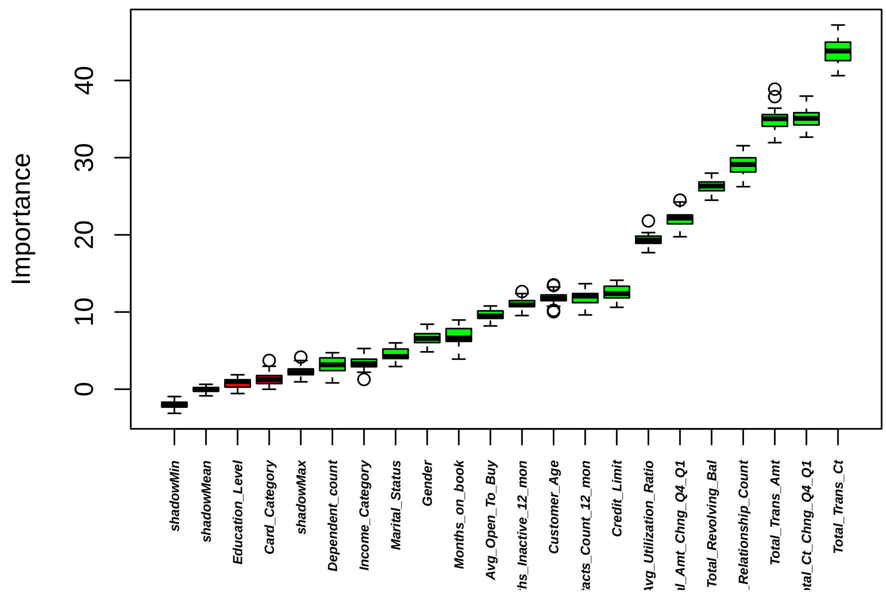
```
##
## Attrited Customer Existing Customer
##               421              2111
```

We use 30% of the training data to select the input variables by Boruta method.

```
library(Boruta)

inVars = sample(nrow(train),nrow(train)*.3)

(boruta = Boruta(Attrition_Flag~., data=train[inVars,], maxRuns=500, doTrace=0))
```

```
## Boruta performed 37 iterations in 9.617218 secs.
##  17 attributes confirmed important: Avg_Open_To_Buy,
## Avg_Utilization_Ratio, Contacts_Count_12_mon, Credit_Limit,
## Customer_Age and 12 more;
##  2 attributes confirmed unimportant: Card_Category, Education_Level;
```

```r
plot(boruta, xlab="", xaxt="n")

lz = lapply(1:ncol(boruta$ImpHistory), function(i)
  boruta$ImpHistory[is.finite(boruta$ImpHistory[,i]),i])

names(lz) = colnames(boruta$ImpHistory)
lb = sort(sapply(lz, median))
axis(side=1, las=2, labels=names(lb), at=1:ncol(boruta$ImpHistory), cex.axis=0.5, font=4)
```



Variables with green boxes are more important than the ones represented with red boxes, and we can see the range of importance scores within a single variable in the graph.

```r
selected_vars = names(boruta$finalDecision)[boruta$finalDecision %in% c("Confirmed","Tentative")]
print(selected_vars)
```

```
##  [1] "Customer_Age"             "Gender"
##  [3] "Dependent_count"          "Marital_Status"
##  [5] "Income_Category"          "Months_on_book"
##  [7] "Total_Relationship_Count" "Months_Inactive_12_mon"
##  [9] "Contacts_Count_12_mon"    "Credit_Limit"
## [11] "Total_Revolving_Bal"      "Avg_Open_To_Buy"
## [13] "Total_Amt_Chng_Q4_Q1"     "Total_Trans_Amt"
## [15] "Total_Trans_Ct"           "Total_Ct_Chng_Q4_Q1"
## [17] "Avg_Utilization_Ratio"
```

```r
length(selected_vars)
```

```
## [1] 17
```

References

- https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/26/lecture-26.pdf
- https://bookdown.org/max/FES/selection.html
- Lectures Notes from STAT34300: Applied Linear Statistical Methods by Prof. Rina Foygel Barber