STAT 2270 Homework 3 Fall 2020

Manuel Alejandro Garcia Acosta

10/9/2020

Exercise 1

Here we'll look at the relationship between undergraduate GPA and LSAT scores amongst individuals applying to law school.

NOTE: For this exercise I'll be using $\alpha = 0.05$.

Part (a)

Load the 'law' data is stored in the bootstrap package in R. Does there appear to be a strong relationship? Calculate the correlation.

```
# Load the 'law' dataset
fix(law)
# Computing the correlation between GPA and LSAT
corr <- cor(law$LSAT,law$GPA)
```

The correlation between GPA and LSAT is 0.7763745. Correlation seems to be moderate.

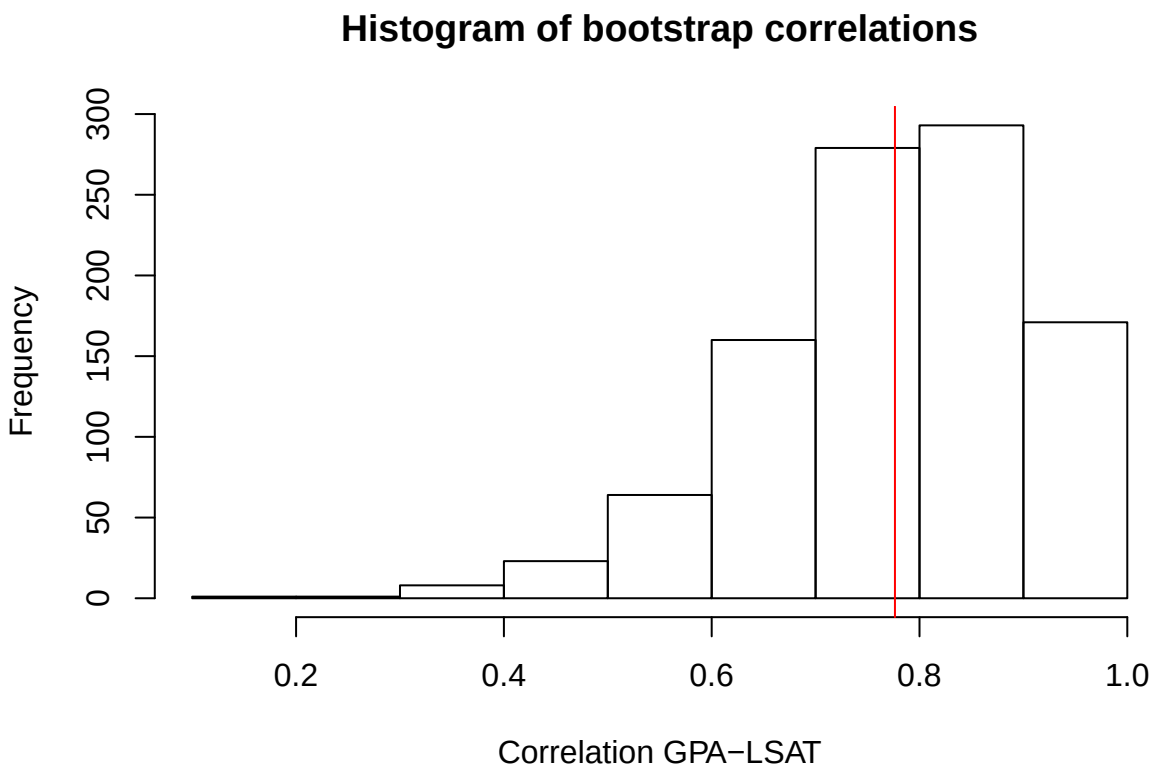
Part (b)

Take $B = 1000$ bootstrap replicates of the data to get 1000 bootstrap estimates of correlation and make a histogram of the bootstrap correlations. Insert a red vertical line in the histogram showing the correlation calculated on the original data.

Here we take the 1000 bootstrap samples and compute the estimates for correlation.

```
set.seed(1910)
# bootstrapping with 1000 replications
output <- boot(data = law, statistic = correlation, i = 1, j =2, R = 1000)
# Observed value of correlation for original data
theta0 <- output$t0
# Bootstrap replicates of the statistic (correlation)
thetas <- output$t
```

Here is the histogram of the bootstrap correlations. The red vertical line marks the correlation on the original data.

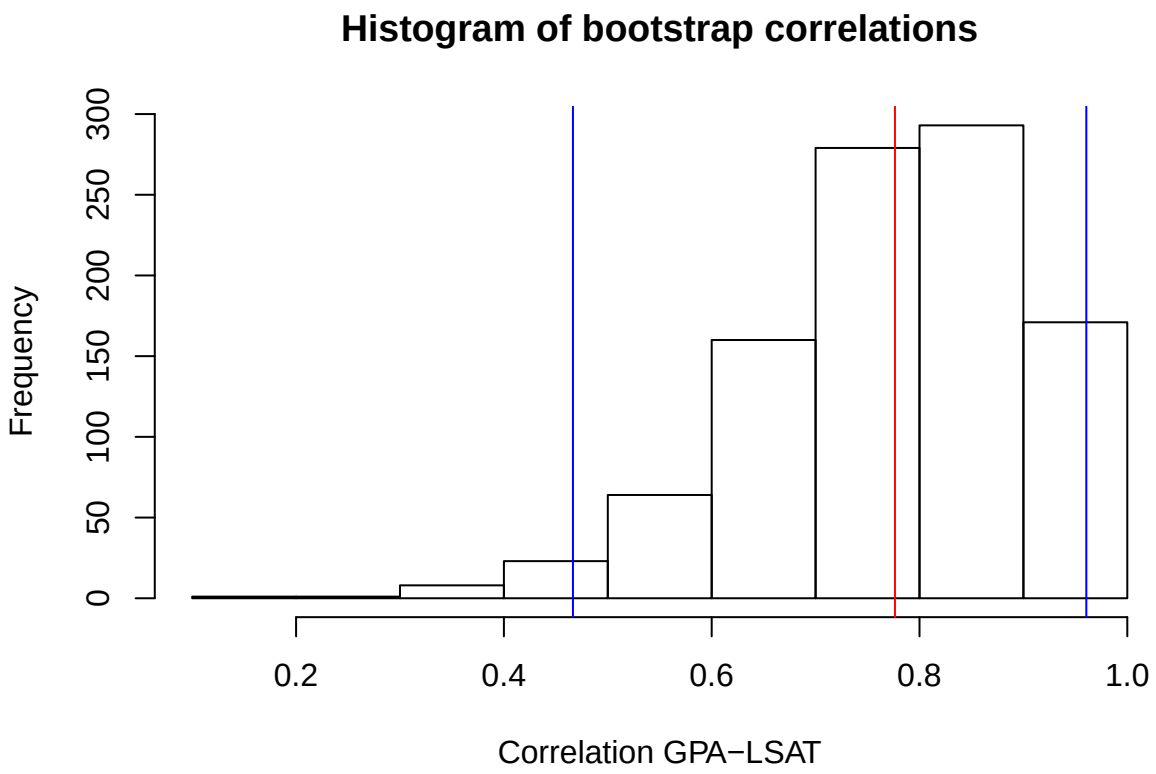


Part (c)

Calculate the bootstrap percentile confidence interval for correlation and insert blue vertical lines in the histogram from part (b) at the upper and lower limits. Based on this, could we reject the null hypothesis that the true correlation is equal to 0.5?

We know that the 95% boostrap percentile CI has limits $(q_{0.025}, q_{0.975}) = (0.4664544, 0.9605491)$. We add these limits as blue vertical lines to the histogram from part (b).

```
# Calculate the bootstrap percentile confidence interval for correlation
# Using boot package
boot.limits <- boot.ci(output, conf = 0.95, type = 'perc')
# Using quantile function
limits <- quantile(output$t, probs = c(0.025,0.975))
```



Since 0.5 is in the confidence interval, at confidence level $\alpha = 0.05$ we fail to reject the null hypothesis that the true correlation is equal to 0.5.

Part (d)

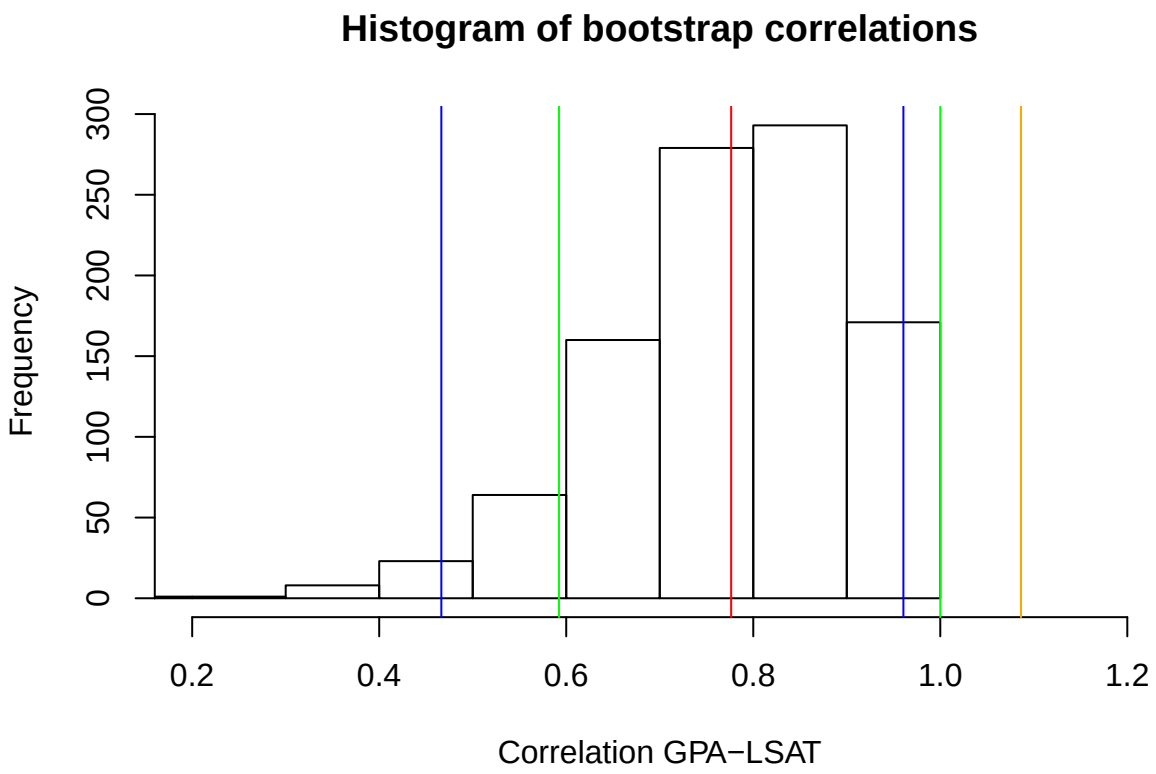
Calculate the bootstrap estimate of bias as well as the (standard) bias corrected bootstrap percentile confidence interval. Insert additional green vertical lines in the histogram showing the bounds for this new interval. According to this, could we reject the null hypothesis that the true correlation is equal to 0.5?

Here we compute the bias corrected point estimate and confidence interval.

```
# Bias corrected point estimate
bc <- 2*theta0-mean(as.numeric(thetas))
# Bias corrected CIs
lower <- 2*theta0-limits[2]
upper <- 2*theta0-limits[1]
```

The bias corrected estimate for the correlation was $\hat{\theta}^*_{BC} = 0.7788417$ and the bias corrected CI was $(0.5921999, 1.086295)$. Notice the bias corrected estimate is close to the correlation in the original dataset. Also, notice that the upper bound of the bias corrected CI is larger than 1 (so we'll truncate ir at 1).

I add the limits of the truncated CI as green vertical lines to the histogram. The original upper bound of the bias corrected CI interval is plotted in orange.



Notice 0.5 is not in the new interval, thus we would reject the null hypothesis that the true correlation is equal to 0.5 (with $\alpha = 0.05$).

Part (e)

Based on these confidence intervals, you should see strong evidence that the true correlation is not equal to 0. Design and carry out a permutation test to explicitly test this.

Here we would like to test

$$H_0 : Corr(X_1, X_2) = 0$$
$$H_1 : Corr(X_1, X_2) \neq 0$$

We already computed the original correlation in part (a). We need also to do the following

1. Fix the order of X_1, shuffle the order of X_2, and compute $r_i^* = \widehat{Corr}(X_1, X_2^*)$
2. Repeat the step above $B = 1000$ times to get $r_1^*, \ldots, r_{1000}^*$
3. Look at the distribution of the permuted test statistics (r_i^*'s). If the original statistic $r = \widehat{Corr}(X_1, X_2)$ lies below the $1 - \alpha/2$ quantile, we fail to reject H_0. Otherwise, we reject.

```
# Computing the permuted correlations
set.seed(1911)
permuted <- sapply(1:1000, function(z){
  shuffled.cor(data = law, fix = 1, change = 2)
  })
# Computing the 0.95 quantile of the permuted correlations
quant <- quantile(permuted, probs = 0.975)
```

The $1 - \alpha/2$ quantile is $q_{1-\alpha/2} = 0.5143565$. Since the original statistic is $r = 0.7763745$ and lies above $q_{1-\alpha/2}$, we reject the null hypothesis and conclude that GAP and $LSAT$ variables are correlated.

Exercise 2

As we discussed at the beginning of this course, we don't need strong assumptions about linear models in order to derive the optimal parameter estimates with respect to squared error. We do, however, need relatively strong assumptions about the underlying relationships and distributions to provide standard inferential results (confidence intervals and hypothesis tests). Now we'll look at how we could carry out corresponding versions of these with permutation-style-tests. *Note: These tests are not necessarily valid or exact, but they should give you a feel for how you might go about trying to measure familiar characteristics without needing distributional results.*

NOTE: For this exercise I'll be using $\alpha = 0.05$.

Part (a)

Generate 50 observations from the model $Y = X_1 + X_2 + \epsilon$ where X_1 and X_2 are independent and sampled uniformly at random from $[0, 1]$ and ϵ $N(0, 0.25^2)$. Generate another test dataset with 30 observations using the same setup. Use the training data to construct a linear model with X_1 and X_2 – no interaction terms or higher-order polynomial terms. Calculate the MSE on the test set and call this MSE_0.

Here we generate the train dataset and train the model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

```
# Generating the dataset
set.seed(1920)
x1 <- runif(50, min = 0, max = 1)
x2 <- runif(50, min = 0, max = 1)
e <- rnorm(50, mean = 0, sd=0.25)
y <- x1+x2+e
# Training the model with training data
train <- data.frame(x1,x2,y)
fit <- lm(y~., data = train)
```

Next, we compute the error on the test set and get $MSE_0 = 0.06330103$.

```
# Creating the test dataset
x1.t <- runif(30, min = 0, max = 1)
x2.t <- runif(30, min = 0, max = 1)
e.t <- rnorm(30, mean = 0, sd=0.25)
y.t <- x1.t+x2.t+e.t
test <- data.frame('x1' = x1.t, 'x2' = x2.t, 'y' = y.t)
# Computing the test error
fitted.test <- predict.lm(object = fit, newdata = test[,1:2])
mse0 <- mean( (test$y - fitted.test)^2 )
```

Part (b)

Using the test MSE as our statistic, devise a permutation-test-equivalent to the (overall) F-test. Carry out the test with 1000 permutations. Can you reject the null hypothesis that none of the predictors are significant

using your test?

The steps we'll follow for the permutation test are as follow

1. Build model and compute the error MSE_0. We did this in part (a).
2. Permute the values of X_1 and X_2 at the same time, i.e., not independently. Fit the model and compute the new test error.
3. Repeat step (2) 1000 times to get $MSE_1^*, ..., MSE_{1000}^*$.
4. Look at the distribution of model errors when X_1, X_2 are permuted. If MSE_0 lies above the α quantile of the permuted errors, then the original model isn't doing significantly better with the true values of X_1, X_2. If this happens, we could conclude that such features aren't important.

We can summarize the last part of the procedure with the hypotheses

$$\begin{aligned} H_0 &: \beta_1 = \beta_2 = 0 \\ H_1 &: \text{Not both equal 0} \end{aligned} \tag{1}$$

Notice these are equivalent to an F-test for testing the effect of both features. Next, I compute $MSE_1^*, ..., MSE_{1000}^*$.

```
# Computing the permuted test errors
set.seed(1921)
test.errors <- sapply(1:1000, function(z){
  shuffled.mse(data = train, features = c(1,2), test = test)
})
# Computing the alpha quantile of the permuted errors
quant <- quantile(test.errors, probs = 0.05)
```

The original error $MSE_0 = 0.06330103$ is below $q_\alpha = 0.1305771$, the $\alpha = 0.05$ quantile of the permuted errors. Thus the original model is doing significantly better with the true values of X_1, X_2. We reject the null H_0 and conclude that at least one of the two features is important.

Part (c)

As we know, we can examine whether an individual feature is significant by looking at its corresponding t-test. Again using test MSE as the statistic of interest, devise a permutation-test-equivalent to the individual t-test. Carry out the test with 1000 permutations to test whether X_2 is significant. Can you reject the null hypothesis that $\beta_2 = 0$ using your test?

The usual t-test has the following alternatives

$$\begin{aligned} H_0 &: \beta_2 = 0 \\ H_1 &: \beta_2 \neq 0 \end{aligned} \tag{2}$$

The permutation-test-equivalent to the t-test will be rather similar to the one in part (b). The steps we'll follow are:

1. Build model and compute the error MSE_0. We did this in part (a).
2. Fix X_1 and permute the values of X_2. Fit the model and compute the new test error.
3. Repeat step (2) 1000 times to get $MSE_1^*, ..., MSE_{1000}^*$.
4. Look at the distribution of model errors when X_2 is permuted. If MSE_0 lies above the α quantile of the permuted errors, then the original model isn't doing significantly better with the true values of X_2.

Here I compute the errors $MSE_1^*, ..., MSE_{1000}^*$.

```
# Computing the permuted errors
set.seed(1922)
errors.2c <- sapply(1:1000, function(z){
  shuffled.mse(data = train, features = 2, test = test)
})
# Computing the alpha quantile of the permuted errors
quant.2c <- quantile(errors.2c, probs = 0.05)
```

Since the original error $MSE_0 = 0.06330103$ is below $q_\alpha = 0.09723249$, the α quantile of the permuted errors, the original model is doing significantly better with the true values of X_2. We reject the null H_0 and conclude that $\beta_2 \neq 0$.

Part (d)

In this case, we have only two features so an individual t-test is equivalent to a partial F-test. Let's scale things up a bit. Using the same general procedure and model as above (with all coefficients equal to 1), create a training set with 100 observations on 10 features. Also create a test set with 50 observations.

Here I just create the training and test datasets. Recall that the true model is the same as before

$$Y = X_1 + X_2 + \epsilon$$

where $\epsilon \sim N(0, 0.25^2)$. Here we create the train dataset

```
# Creating the train dataset
set.seed(1923)
observations <- 100
variables <- 10

x.2d <- matrix(ncol=variables, nrow=observations, byrow = FALSE)
nom <- c()
for(i in 1:variables){
  temp <- runif(observations, min = 0, max = 1)
  x.2d[,i] <- temp
  nom[i] <- paste0('x',i)
}
colnames(x.2d) <- nom
y <- x.2d[,1]+ x.2d[,2] +rnorm(100, mean = 0, sd=0.25)
train.2d <- data.frame(cbind(x.2d,y) )
```

Next, we create the test dataset

```
# Creating the test dataset
set.seed(1924)
observations <- 50
variables <- 10

x.2dt <- matrix(ncol=variables, nrow=observations, byrow = FALSE)
nom <- c()
for(i in 1:variables){
  temp <- runif(observations, min = 0, max = 1)
  x.2dt[,i] <- temp
  nom[i] <- paste0('x',i)
```

```
}
colnames(x.2dt) <- nom
y <- x.2dt[,1]+x.2dt[,2]+rnorm(50, mean = 0, sd=0.25)
test.2d <- data.frame(cbind(x.2dt,y) )
```

Part (e)

Using the data from part (d), devise a permutation-test-equivalent to the partial F-test that will evaluate whether any subset of the features is significant. Carry out the test with 1000 permutations on X_8, X_9 and X_{10}. Can you reject the null hypothesis?

This permutation test is going to follow a similar procedure to the one in part (c). First we fit the model using the training data and get the test error MSE_0. Next, we'll fix the variables X_i, $i \in \{1, ..., 7\}$ and permute X_8, X_9 and X_{10} together to get the 1000 permuted errors. Finally, if MSE_0 lies above the α quantile of the permuted errors, then the original model isn't doing significantly better with the true values of X_8, X_9 and X_{10}.

```
# Computing the original test error
fit <- lm(y~., data = train.2d)
fitted.test <- predict.lm(object = fit, newdata = subset(test.2d, select = -y) )
mse0 <- mean( (test.2d$y - fitted.test)^2 )
```

Here $MSE_0 = 0.05031948$. Next, we need to compute the permuted errors.

```
# Computing the permuted errors
set.seed(1925)
errors.2c <- sapply(1:1000, function(z){
  shuffled.mse(data = train.2d, features = c(8,9,10), test = test.2d)
})
# Computing the alpha quantile of the permuted errors
quant.2e <- quantile(errors.2c, probs = 0.05)
```

Since the original error $MSE_0 = 0.05031948$ is above $q_\alpha = 0.04379429$, the α quantile of the permuted errors. Thus the model is not performing significantly better with the true values of X_8, X_9, X_{10}. We fail to reject the null hypothesis H_0 and conclude that $\beta_8 = \beta_9 = \beta_{10} = 0$.

Exercise 3

This problem will deal with *StabilitySelection*. You can find the original paper by Meinshausen and Buhlmann at https://arxiv.org/pdf/0809.2932.pdf.

Consider the (true) model below consisting of 4 sets of predictors:

$$Y = \sum_{i=1}^{3} X_i + \sum_{j=4}^{6} X_j + \sum_{k=7}^{9} X_k + 0 \cdot \sum_{l=10}^{20} X_l + \epsilon$$

NOTE: I don't evaluate some chunks of code while compiling the *Rmd* file since it takes a while, the results of such code are in the file 'ex3.RData'.

Part (a)

Suppose that each set of features is independent from all others but that within the sets, the correlation is quite strong – let's say 0.9. Also suppose that each feature is normally distributed with mean and variance 1 and that $\epsilon \sim N(0,1)$ is independent of all of the features. Generate 50 observations from this model.

Here we generate the data. The groups

- Group 1: X_1, X_2, X_3
- Group 2: X_4, X_5, X_6
- Group 3: X_7, X_8, X_9
- Group 4: $X_{10}, \ldots, X_{20}$

were set to have within-group sample correlation equal to 0.9.

```
# Creating the dataset
set.seed(1930)
group1 <- simulate.normal(n = 50, features = 3, sample.corr = 0.9,
                          empirical = TRUE)
group2 <- simulate.normal(n = 50, features = 3, sample.corr = 0.9,
                          empirical = TRUE)
group3 <- simulate.normal(n = 50, features = 3, sample.corr = 0.9,
                          empirical = TRUE)
group4 <- simulate.normal(n = 50, features = 11, sample.corr = 0.9,
                          empirical = TRUE)
e.3 <- rnorm(n = 50, mean = 0, sd = 1)
y <- rowSums(group1)+rowSums(group2)+rowSums(group3)+e.3
data <- data.frame(group1, group2, group3, group4, y)

nom <- c()
for(i in 1:20){
  nom[i] <- paste0('x',i)
}
names(data) <- c(nom,'y')
```

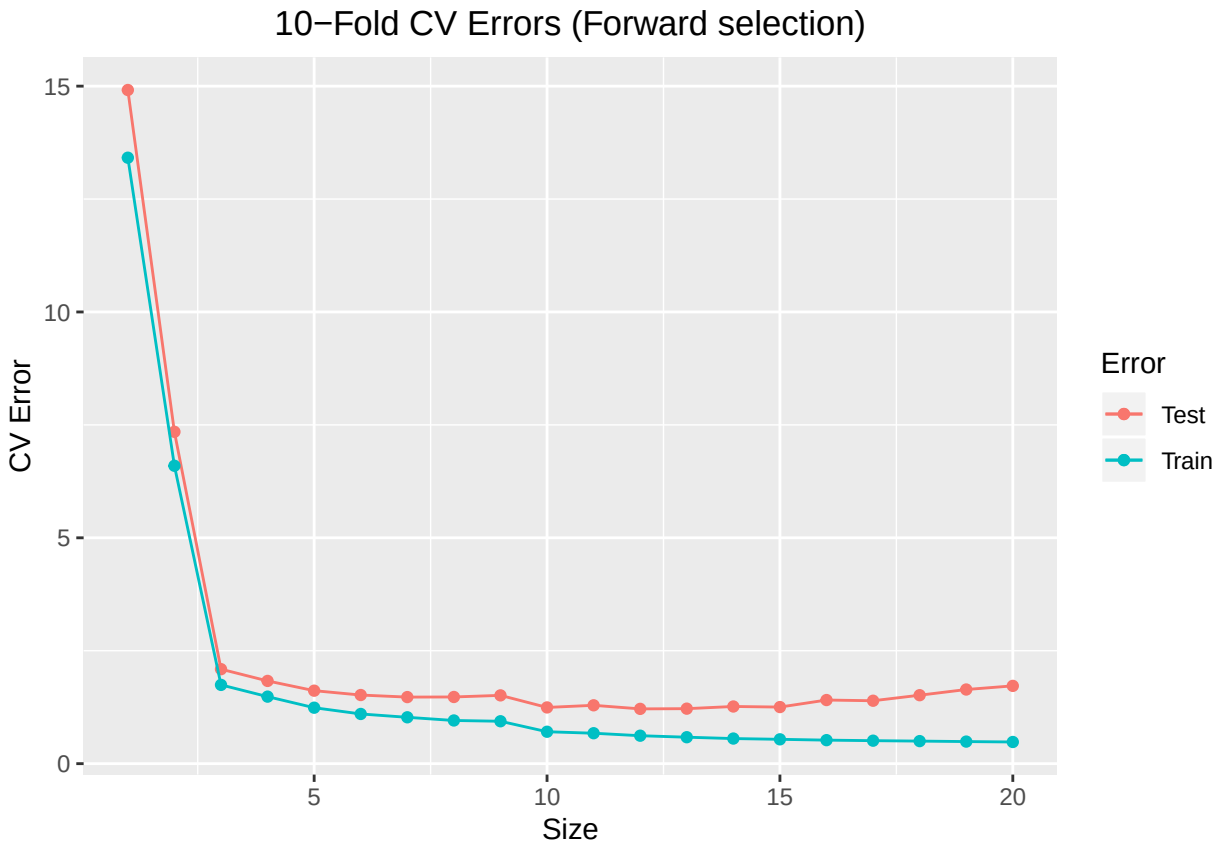
Part (b)

Using forward stepwise selection, find the best model of each size and make a 10- fold cross-validation plot of the errors. What size model should you think you ought to pick (knowing the truth)? What size model would you pick based on the cross-validation results?

With the dataset created in part (a), I obtained the best models of sizes 1 through 20 using forward stepwise selection with regsubsets() from the 'leaps' package. I ran 10-fold CV ten times and averaged the errors to get the train and test errors I'm reporting.

```
# Getting the best models by size
fit.step <- regsubsets(y~., data = data, nvmax = 20, method = 'forward')
all.models <- summary(fit.step)
# Running 10-fold CV
set.seed(1940)
models <- lapply(1:20, best.model, summ.models = all.models)
errors <- lapply(models, cv.lm, data = data, k = 10, repeats =10)
cv.errors <- as.data.frame(do.call(rbind,errors))
```

Here is the plot of the 10-fold CV Errors.



Knowing the truth, I think we should pick a model with size 6, or at least a model with around 6 features. Based on the cross-validation results, I think a model of size 3-6 would be a good pick since models with more features don't seem to reduce the error considerably. The 3-sized model improves greatly upon the 1-sized and 2-sized models, size 4, 5 and 6 appear to improve error-wise with respect to the 3-sized model but the improvement is not as big.

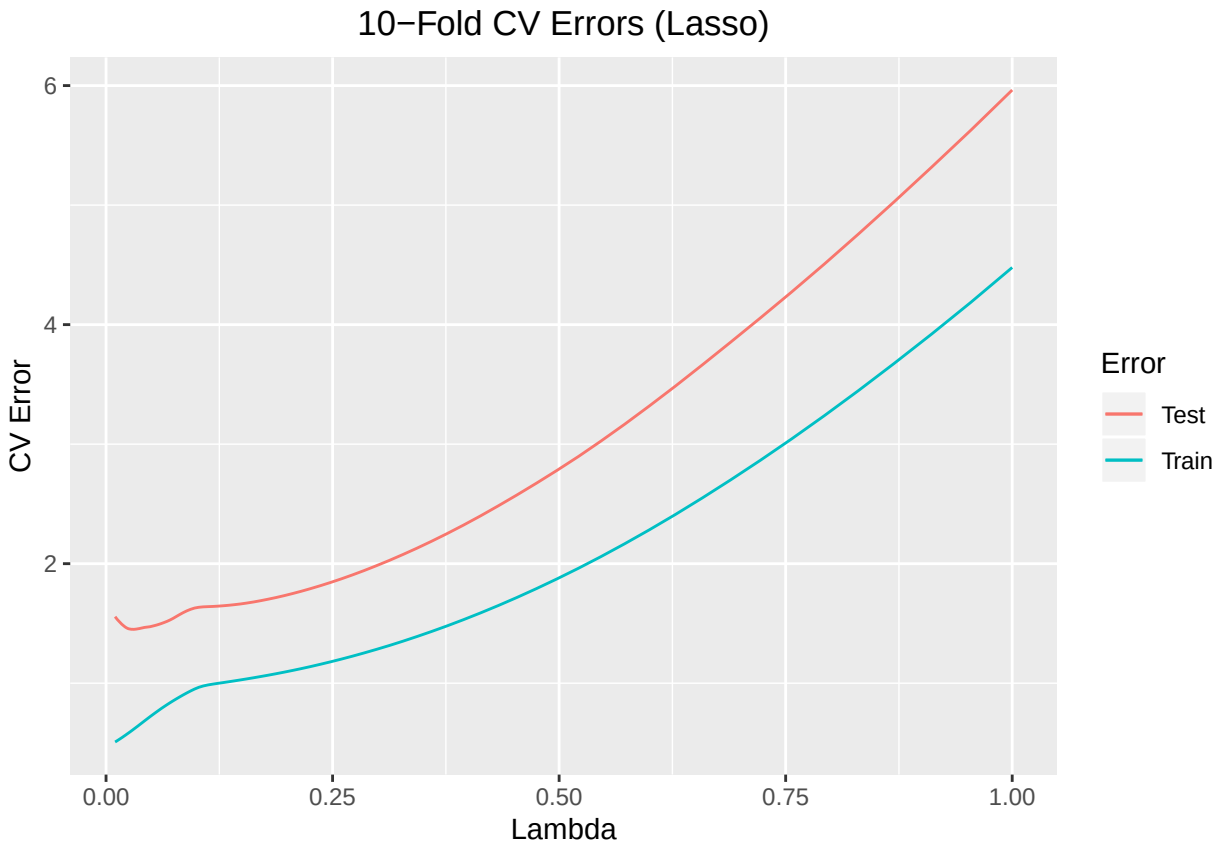
Part (c)

Fit a model via lasso using cross-validation to select the value of the tuning parameter.

I'll use 10-fold CV to select the value of the tunning parameter. The set of lambdas we'll consider range from 1 to 1/100.

```
# Creating the set of lambdas
lambdas <- 10^seq(0,-2,length.out=100)
```

```
# Running 10-fold CV with Lasso
set.seed(1941)
x <- as.matrix(subset(data, select = -y))
fit4 <- cv.lasso(x = x, y = y, alpha = 1, lambda = lambdas, k = 10, repeats = 10)
```



By the results above, we get that the best value for lambda is $\lambda = 0.03053856$ with corresponding CV error 1.450561.

```
lambda <- best.lambda.test(fit4)
fit4[ which.min(fit4$test), ]
```

```
##        lambda    train     test
## 25 0.03053856 0.615555 1.450561
```

Part (d)

Perform stability selection based on 1000 bootstrap samples using lasso to build each model and using the same value of the tuning parameter you chose in part (c). Make a histogram of the selection frequencies for each variable ordered from highest to lowest. What do you observe? Give some intuition for seeing the results you do based on what you know about how lasso behaves.

This is what we need to do for stability selection

1. Take $B = 1000$ resamples of the original data $D_1^*, \ldots, D_B^*$.
2. Perform model selection on each D_i^* for λ. We got lambda in part (c).
3. Calculate the proportion $\hat{\theta}_j(\lambda)$ of times each X_j was selected.

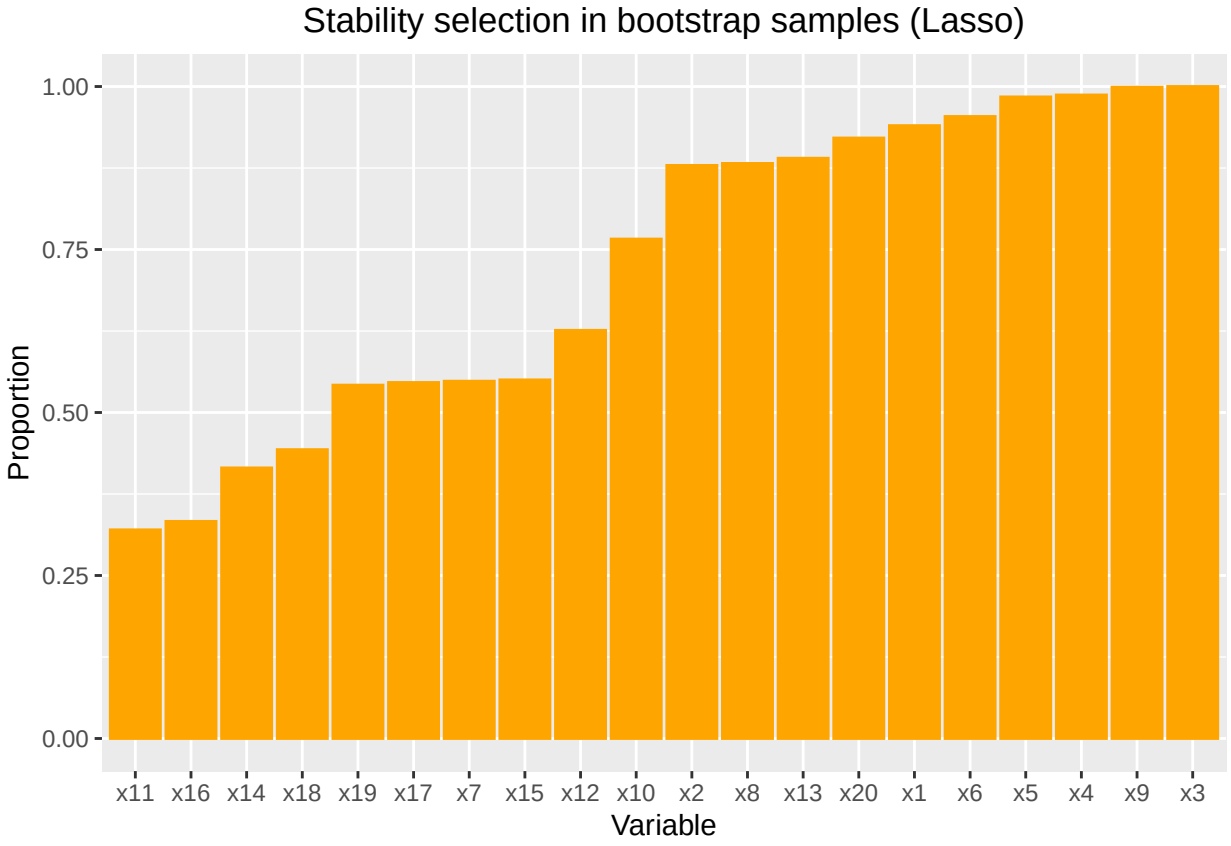
NOTE: We'll be using the tuned lambda obtained in part (c), $\lambda = 0.05857021$.

```
# Creating the bootstrap samples
set.seed(1942)
boot <- lapply(1:1000, function(z){
  boots.sample(data = data)
```

```
} )

# Getting the proportion of times each variable is selected using Lasso
prop <- lapply(boot, lasso.non.zero, lambda = lambda)
proportions <- do.call(rbind,prop)
proportions <- colSums(proportions)/1000
data1 <- as.data.frame(proportions)
data1 <- cbind( names(subset(data, select = -y)) , data1)
names(data1) <- c('variable','prop')
```

As we know, the true model was constructed using features $X_1, \ldots, X_9$ with coefficients equal to 1. With exception of feature X_7, the rest of the features in the true model were selected about 80% of the time or more. Features $X_{11}, \ldots, X_{20}$ weren't selected as often with the exception of X_{10}, X_{13}, X_{20} which were selected at least 75% of the time.



Part (e)

Repeat parts (c) and (d) using the group lasso. How does your histogram compare to the one using the standard lasso? Give some intuition for the similarities/differences.

Here I run 10-fold CV for grouped lasso once and obtain the CV error for each value of lambda. Considered groups were

- Group 1: X_1, X_2, X_3
- Group 2: X_4, X_5, X_6
- Group 3: X_7, X_8, X_9

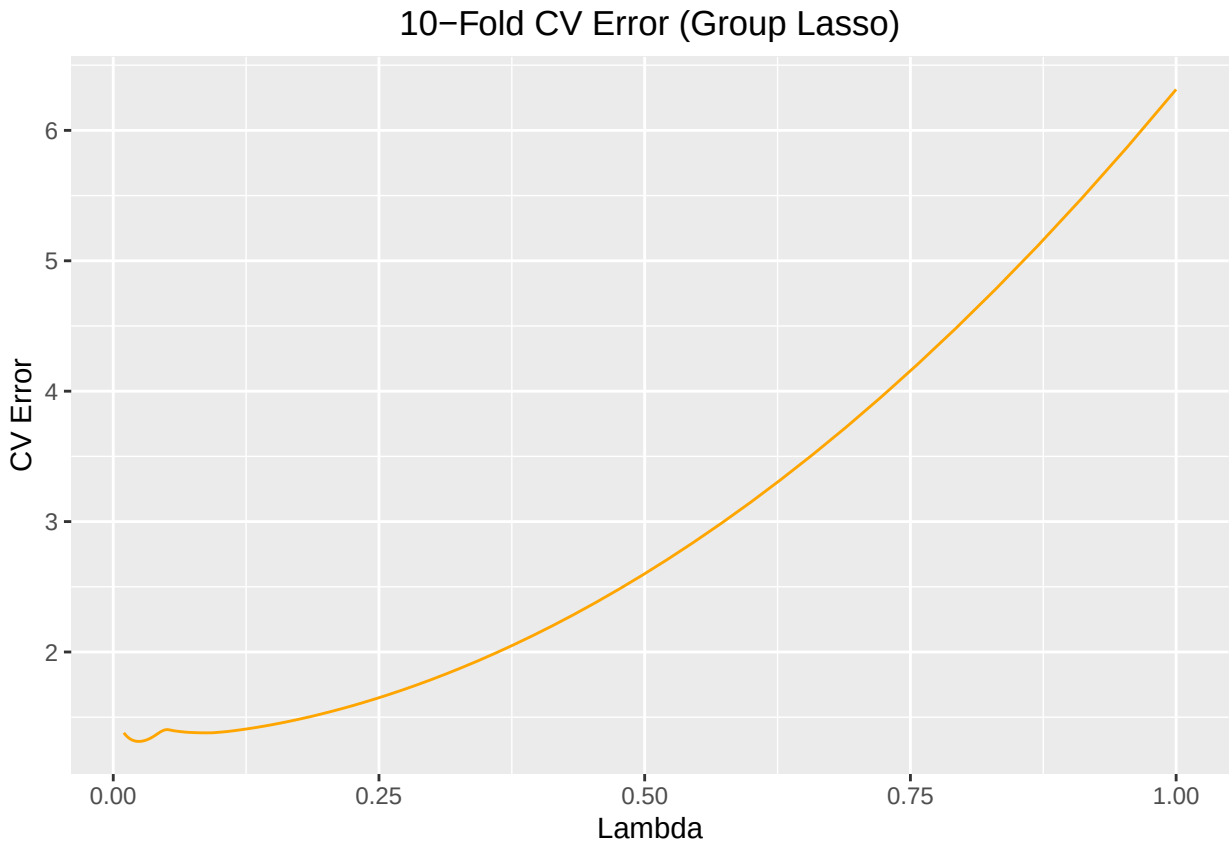
- Group 4: $X_{10}, \ldots, X_{20}$

NOTE: The vector of lambdas used to tune the model was the same as in part (c).

```
# Running 10-fold CV with group Lasso
set.seed(1943)
x <- as.matrix(subset(data, select = -y))
groups <- rep(1:4, times = c(3,3,3,11))
fit4e <- cv.gglasso(x = x, y = y, group = groups, lambda = lambdas,
                    pred.loss = 'L2', nfolds = 10)
data2 <- data.frame('lambda' = fit4e$lambda, 'cv.error' = fit4e$cvm)
```

```
# Getting the best lambda
lambda.group <- fit4e$lambda.min
# Getting the CV error for the best lambda
fit4e$cvm[which(fit4e$lambda == fit4e$lambda.min)  ]
```

```
## [1] 1.314286
```

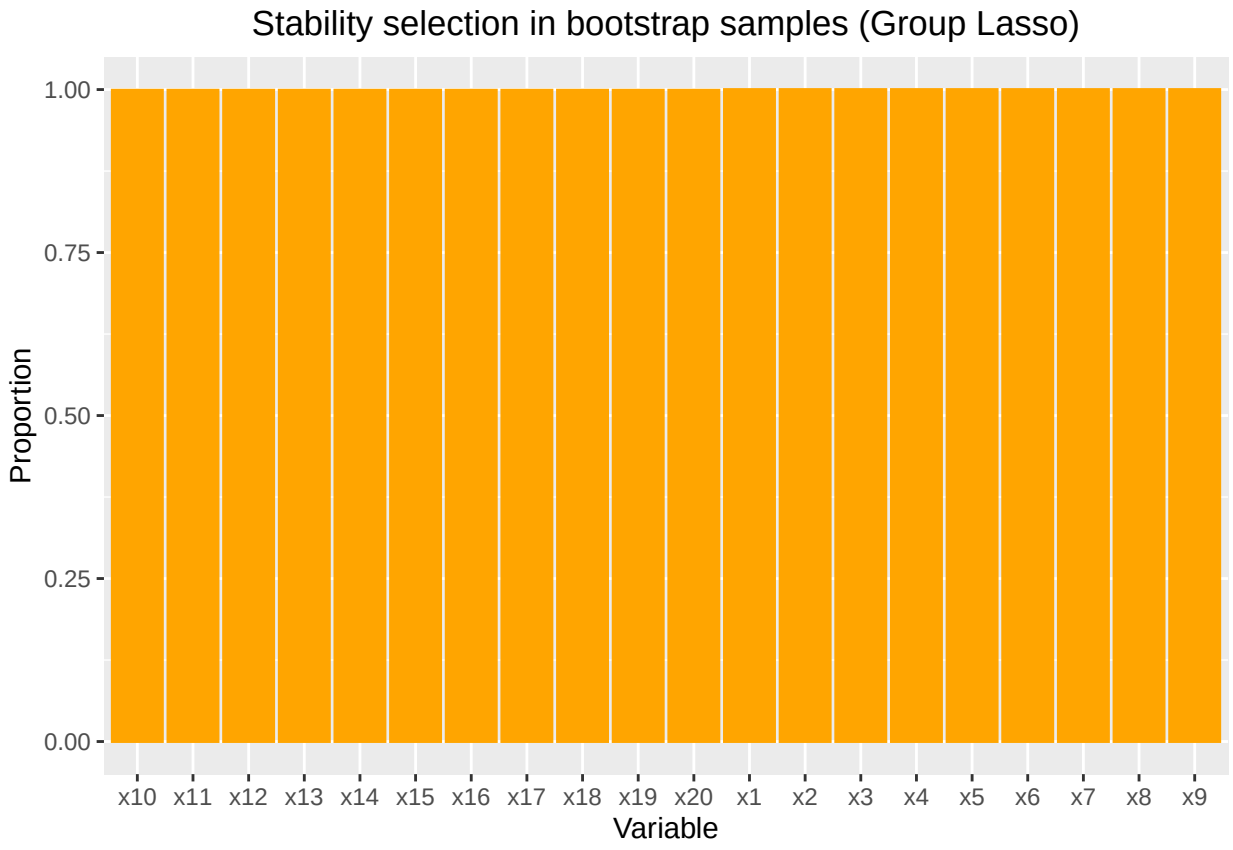


According to our results, the best value for lambda was $\lambda_G = 0.02420128$ with corresponding CV error 1.314286. Next, we perform stability selection based on 1000 bootstrap samples using group lasso to build a histogram of the selection frequencies.

```
# Creating the bootstrap samples
set.seed(1944)
boot.group <- lapply(1:1000, function(z){
  boots.sample(data = data)
} )
# Getting the proportion of times each variable is selected using Lasso
prop.group <- lapply(boot.group, group.lasso.non.zero, group= groups,
```

```
                    lambda = lambda.group)
proportions.group <- do.call(rbind,prop.group)
proportions.group <- colSums(proportions.group)/1000
data3 <- as.data.frame(proportions.group)
data3 <- cbind( names(subset(data, select = -y)) , data3)
names(data3) <- c('variable','prop')
```

As discussed in part (d), variables $X_1, \ldots, X_6, X_8, X_9$ and X_{10}, X_{13}, X_{20} were selected with high frequency. Since the 4 groups defined for group lasso have at least one of these variables, it makes sense (although it is a little surprising) that all features did not have zero coefficient estimate assigned to them ever, i.e., group lasso chose all groups in every bootstrap sample.



Part (f)

Depending on the particular sample you drew in part (a), you may see different things happen in parts (d) and (e). Given a (true) model of the form shown above, what model(s) do you think would be reasonable to use in practice if you only wanted a model with 3 terms (covariates). That is, knowing the truth, what do you think would be the best 3-variable models? Is this what was selected in part (d)? In this context, why might stability selection pick a model different from what you think is best? Play around with the within-group correlation strength and variance of the error term until you find two different settings: one in which stability selection does the "right" thing and one where it does the "wrong" thing.

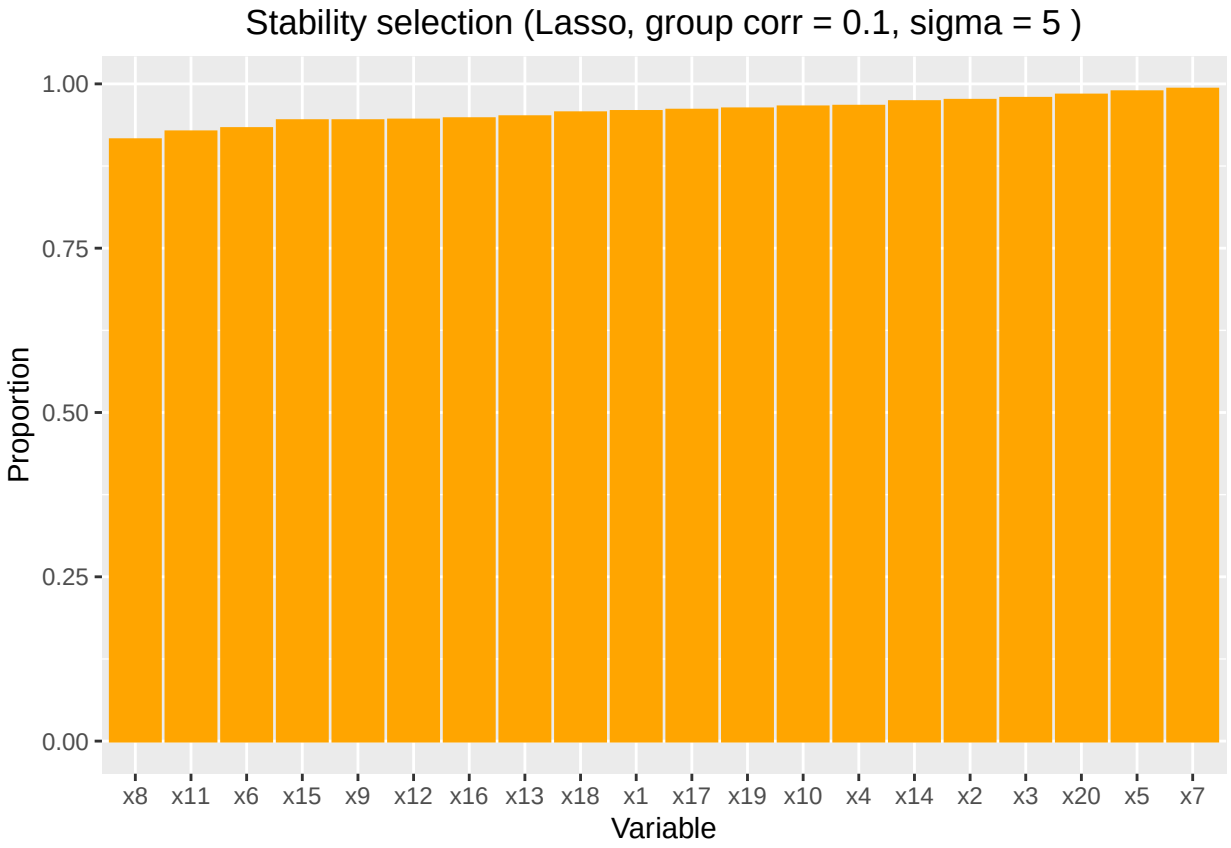
I think that the best 3-variable models would be those which include one feature from each of the first three groups from the true model, i.e, the model will have one feature from $\{X_1, X_2, X_3\}$, one from $\{X_4, X_5, X_5\}$ and one from $\{X_7, X_8, X_9\}$. In part (d) the most selected features were X_3, X_4, X_9, this model agrees with what I just said.

I think that stability selection will pick a model different from what we think is best depending on how noisy the response variable is, i.e., is will depend on the variance of ϵ. We can see this in the next examples.

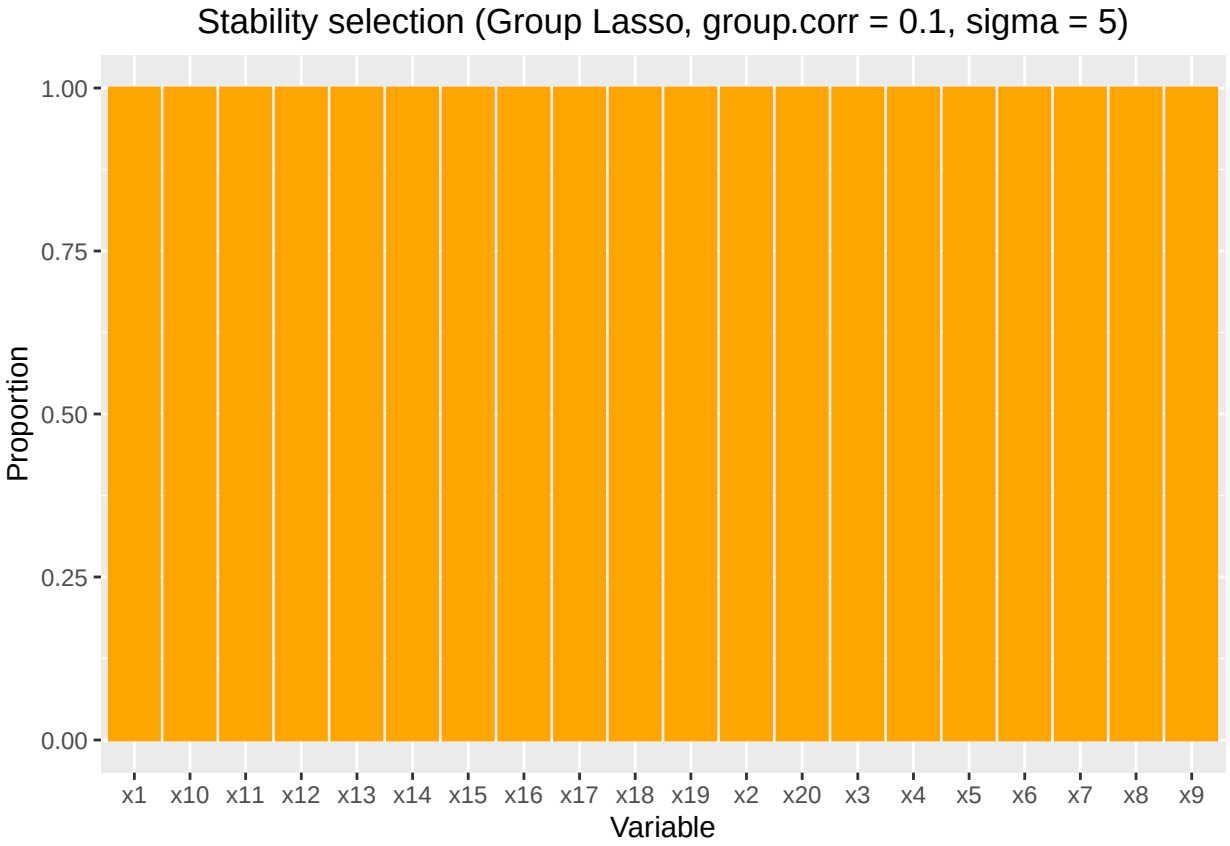
'Wrong' thing

I got stability selection to do the 'wrong thing' when the data was noisy (used $Var(\epsilon) = 25$) and within-group correlation was 0.1.

As we can see in the histogram from lasso, in the noisy dataset lasso selected each feature almost uniformly. All features had no-zero coefficient estimates at least 75% of the time. This differs greatly from the histogram from part (d).

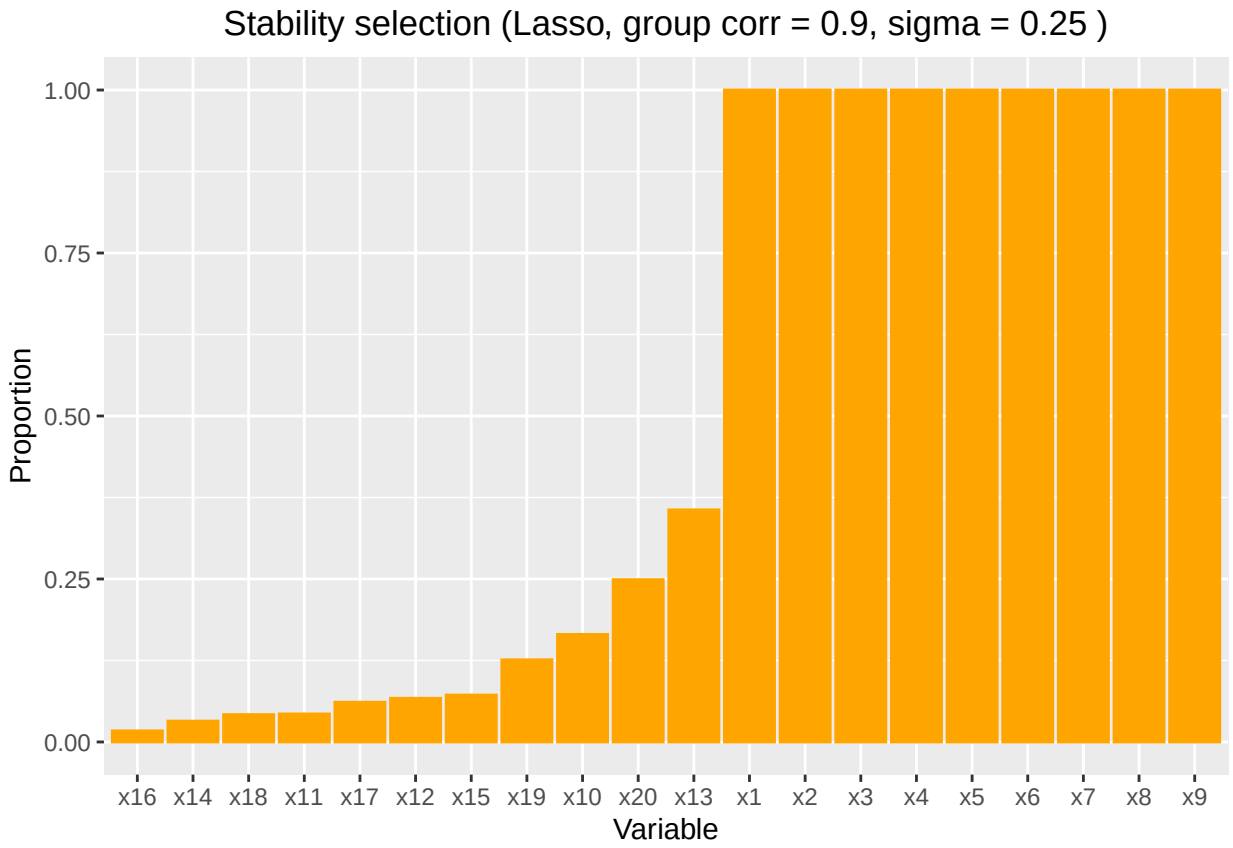


Not surprisingly, group lasso selected all groups in every bootstrap sample. This comes as a result of all the features being assigned non-zero coefficients most of the time.

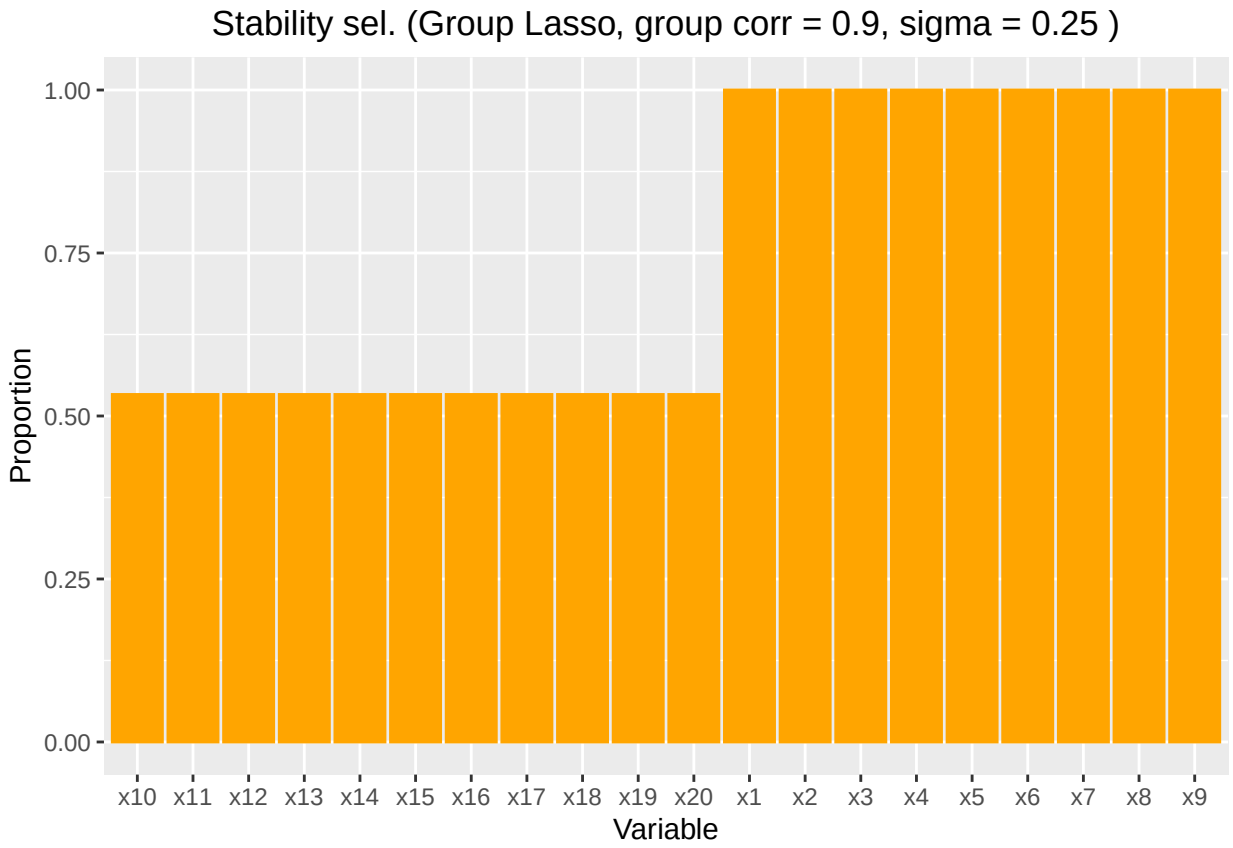


'Right' thing

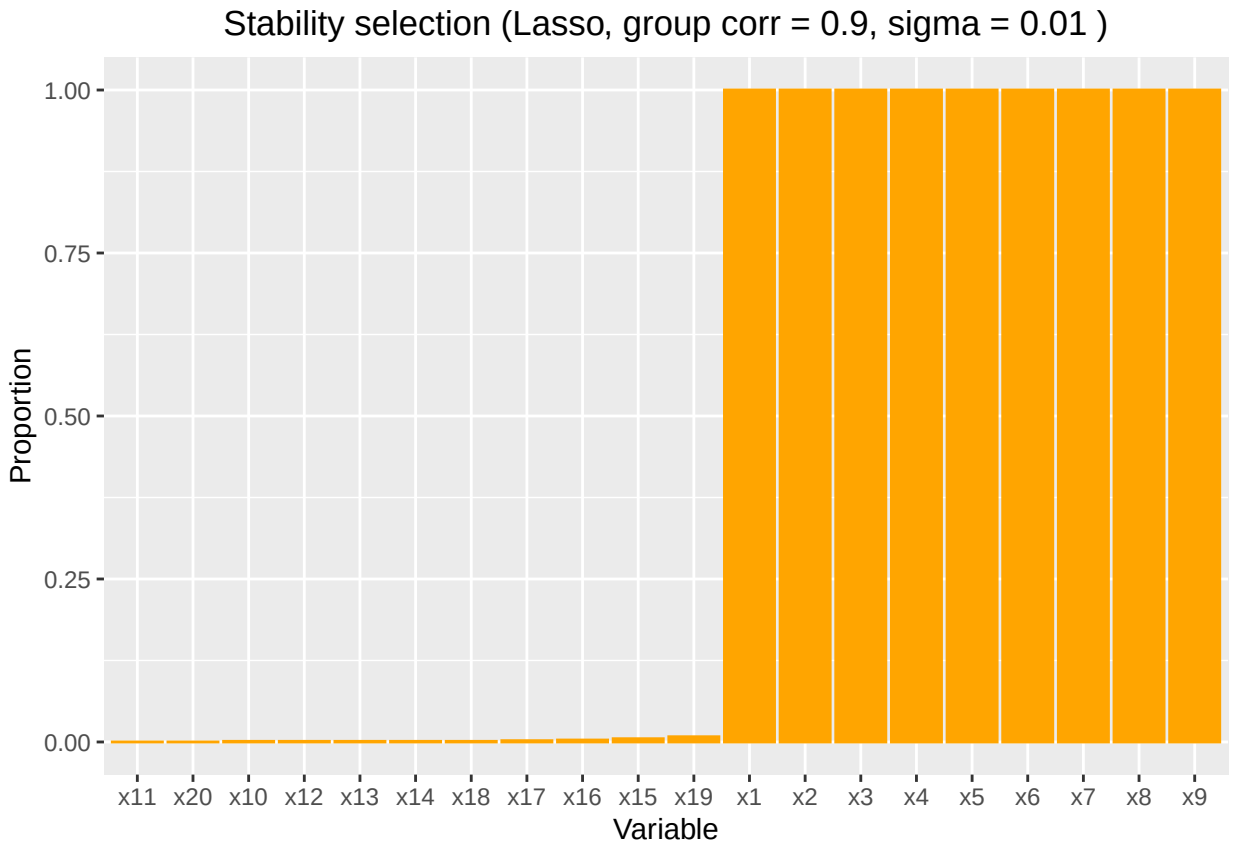
I got stability selection to do the 'right thing' when the data had almost no noise (used $Var(\epsilon) = 0.25^2$) and within-group correlation was set to 0.9. As we can see in the histogram, lasso does a good job choosing the variables from the true model.

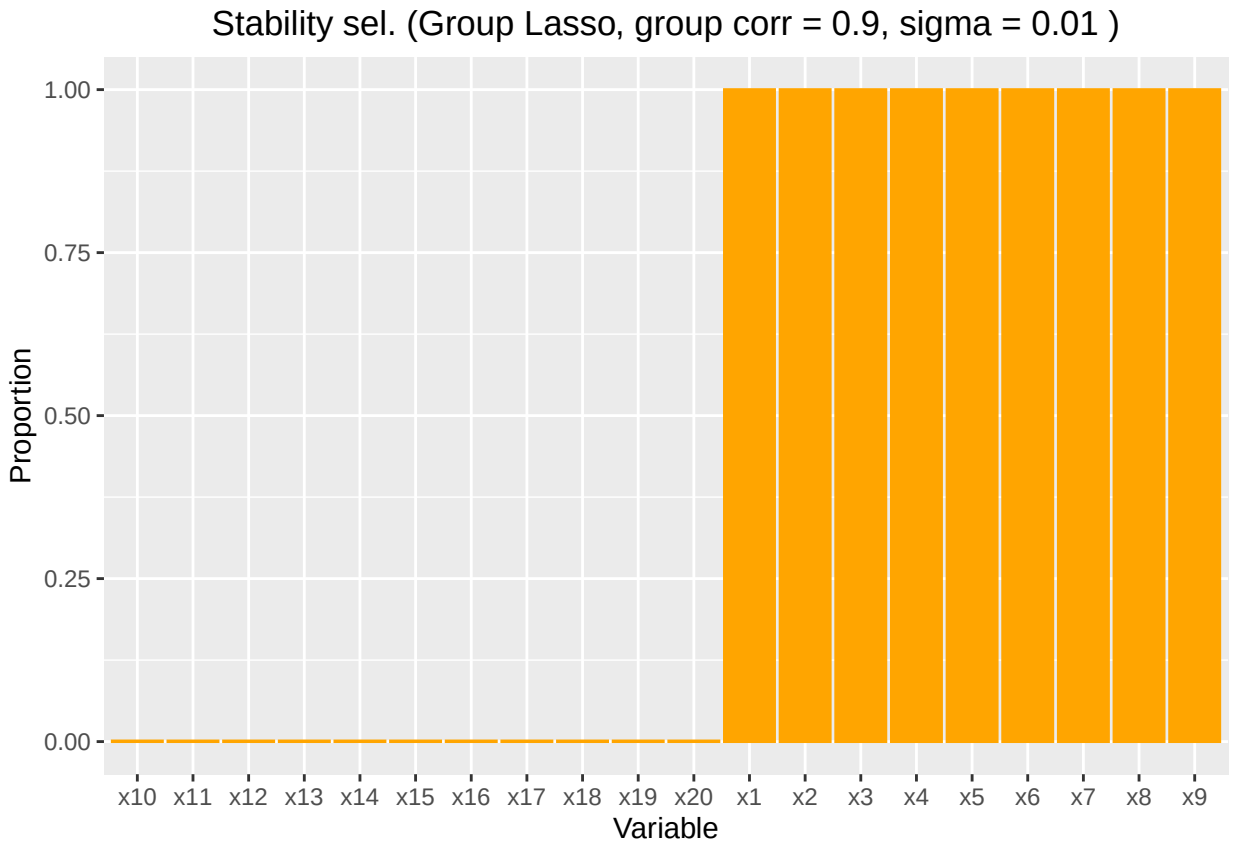


As opposed to the 'wrong' thing, here group lasso did a good job of leaving out the set of variables $X_{10} \ldots, X_{20}$. This set of variables was assigned non-zero coefficients in around 50% of the boostrap samples. The variables from the true model were assigned non-zero coefficients all the time.



We can see a more extreme result when we set an even lower variance. These are the histograms when $\sigma = 0.01$ and within-group correlation is 0.9.


Stability selection (Lasso, group corr = 0.9, sigma = 0.01)
Proportion
1.00
0.75
0.50
0.25
0.00
x11 x20 x10 x12 x13 x14 x18 x17 x16 x15 x19 x1 x2 x3 x4 x5 x6 x7 x8 x9
Variable


Stability sel. (Group Lasso, group corr = 0.9, sigma = 0.01)
Proportion
1.00
0.75
0.50
0.25
0.00
x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20 x1 x2 x3 x4 x5 x6 x7 x8 x9
Variable

Exercise 4

In addition to the kind of permutation tests we discussed in class, there are also rank-based tests that rely on permutations and work in a very similar fashion. Suppose that instead of working with the original (raw) observations, we replaced each observation with it's rank. That is, given say two groups of observations, I find my smallest overall observation and replace that value with "1". Then I find the next smallest and replace it with "2" etc. The idea here is that under the null hypothesis, no group should have "ranks" (values) systematically higher than the other. Once I've done the replacing, I could do something like permuting which ranks belong to which group and comparing the differences in sample means (i.e. means of the ranks) to the original difference in means of the ranks.

Part (a)

What are the advantages to doing something like this as opposed to the standard permutation tests? *(Hint: Suppose you have a small number of samples in each group).*

I think some of the advantages are:

1. The impact of outliers is mitigated. Regardless of how extreme an outlier is, it receives the same rank as if it were just slightly larger than the second-largest observation.
2. Eliminate skewness of data since all ranks are equally apart from each other.

Part (b)

What are the disadvantages to doing something like this as opposed to the standard permutation tests? *(Hint: Suppose you have a small number of samples in each group).* Does this actually "solve a problem" or does it just add an extra assumption? Think carefully about this – the answer isn't as obvious as you might first think.)

I think that:

1. When we switch to using the ranks of the observations instead of the observations themselves we are loosing information. This could have as a consequence that the power of the rank-based test to be smallet than the original permutation test.
2. Often we would not know which observations are in fact outliers.

Exercise 5

We haven't talked much yet about "variable importance" but we will when we talk about tree-based methods in the coming weeks. Think of this as a primer for the discussion to come.

Part (a)

Based on the name "variable importance", what do you think we're actually trying to measure? Write down a formal mathematical definition of this. (Note: This doesn't need to be "right". Just based on your intuition, write down a definition that you think would be appropriate and justify it.)

I think we're trying to measure the impact of the variable in the performance of the model. For example, in a linear model, I think "variable importance" could be the amount by which the performance of the model is improved when we include a given feature X_i. I think this definition makes sense since when we ask: Is this feature important? We usually want to check the effect of the feature in our model.

Part (b)

Suppose I want to define the notion of "importance rank." That is, I just want to order my predictor variables from most important to least important. Given a particular dataset, I propose to do this by something like forward stepwise selection. The variable that goes in first I'll call most important, the next variable to go in would be the second most important etc. Does this seem like a good way to define this? Why or why not – justify your answer.

I think there might be a problem here. Suppose we have a noisy dataset with 20 features where the best 1-sized model uses X_1 and the next variables by rank are X_2, X_3, X_4. Now, suppose the best model is a 4-sized model with features X_5, X_6, X_7, X_8. Using the proposed definition, we would assign higher ranks to X_1, X_2, X_3, X_4 than the features X_5, X_6, X_7, X_8.

Another possible problem with this definition is that no two features can be seen as having the same importance. Consider a model were X_1 and X_2 are part of the true model and they are highly correlated. Since forward selection chooses only one feature at a time, we would be assigning more importance to either X_1 or X_2.

Part (c)

Suppose that based on the measure we defined in part (b), you're not actually sure whether the variables are in the right order. To compensate for this, you take B bootstrap samples and do the forward stepwise procedure on each, each time recording the variable order. After that's done, you order the variables by their average rank. Does that solve any issues you may have had with the definition in part (b)?

We discussed in class that using boostrap will provide good estimates as long as the boostrap distribution approximates the true distribution. Now, consider the last scenario from part (b). Even with the boostrapped samples, the correlation between the features will still be present and we'll still have the problem of selecting one feature over the other.

Part (d)

Let's try it. Generate $n = 25$ observations from the model $Y = X_1 + X_2 + \epsilon$ where X_1 and X_2 are independent and sampled uniformly at random from $[0, 1]$ and ϵ $N(0, 0.25^2)$. Knowing the true model here, is either variable more "important" than the other? Note: This answer probably seems obvious – it's not.

Here we generate the dataset.

```
set.seed(1950)
x1 <- runif(25,min = 0,max = 1)
x2 <- runif(25,min = 0,max = 1)
e5 <- rnorm(25,mean = 0,sd = 0.25)
y <- x1+x2+e5
data5 <- data.frame(x1,x2,y)
#cor(data5$x1,data5$y)
#cor(data5$x2,data5$y)
```

In the original dataset $Corr(X_1, Y) = 0.6483806$ and $Corr(X_2, Y) = 0.6980927$. Moreover, while fitting a linear model using the original data we can notice that the p-value for the t-test

$$H_0 : \beta_2 = 0$$
$$H_0 : \beta_2 \neq 0$$

is smaller than the p-value for the t-test for β_1. This will lead us to think that X_2 is going to be selected first by forward selection more often than X_1. Nevertheless, from the true model we could argue that both features have the same importance.

```
mod <- lm(y~., data = data5)
summary(mod)$coefficients
```

```
##              Estimate Std. Error  t value    Pr(>|t|)
## (Intercept) 0.1493642  0.1423652 1.049163 0.305492725
## x1          0.7558535  0.2524308 2.994300 0.006682284
## x2          0.8433490  0.2338232 3.606781 0.001565681
```

Part (e)

Using the dataset from part (d), carry out the procedure from part (c) with $B = 500$. (Note: with only two variables here and knowing what you do about the model, you don't *actually* need to code a forward selection procedure to accomplish this.) Let p_1 denote the percentage of times X_1 was selected first and similarly for $p_2 = 1 - p_1$. Define $p = \max\{p_1, p_2\}$.

Here we create 500 bootstrap samples, run forward selection and then compute p_1, p_2 and p.

```
# Creating the bootstrap samples
set.seed(1951)
bot <- lapply(1:500, function(z){
  boots.sample(data5)
})
# Computing p1, p2 and p
#result <- lapply(bot, selected)
result <- lapply(bot, bot.proportion)
results <- do.call(rbind,result)[,1:2]
results <- colSums(results)/500
```

For this first run we have $p_1 = 0.344$, $p_2 = 0.656$ and thus $p = 0.656$.

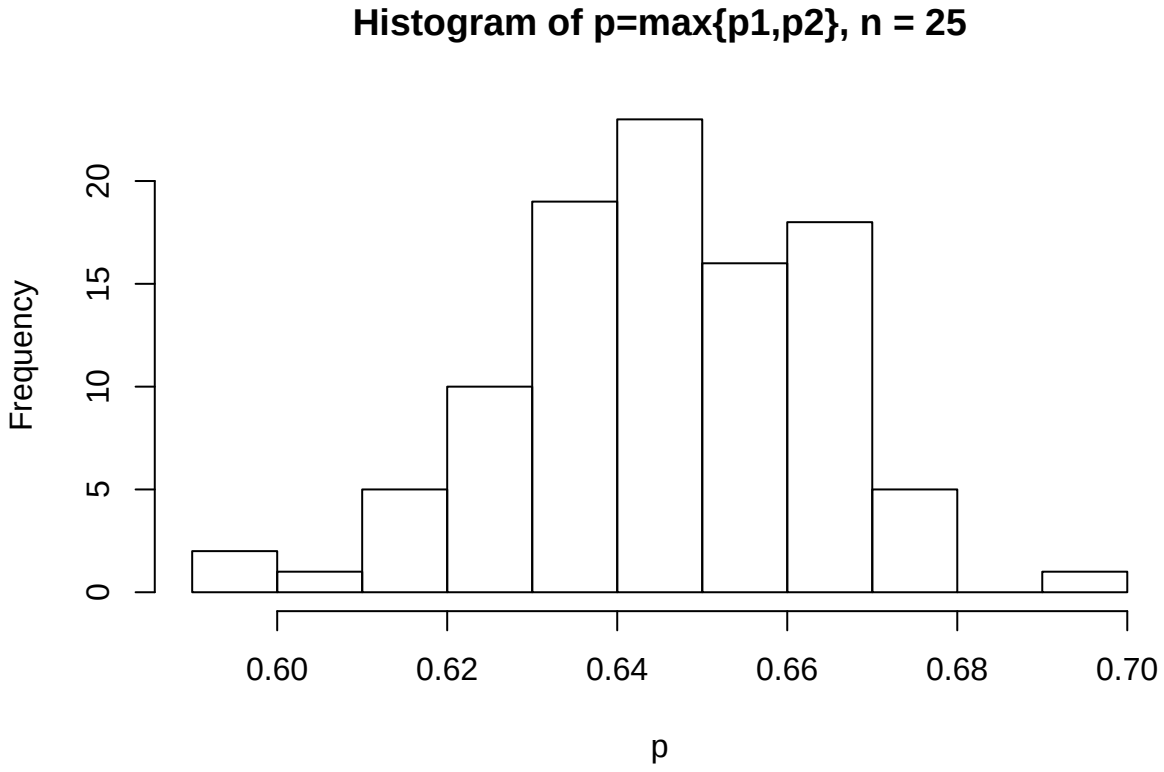
Part (f)

Repeat part (e) 100 times and make a histogram of p. Is this centered where you'd expect?

Here we just repeat the procedure from (e) 100 times.

```
proportions5 <- lapply(1:100, function(z){
  bot.proportions(data = data5)
})
prop5 <- do.call(rbind,proportions5)
```

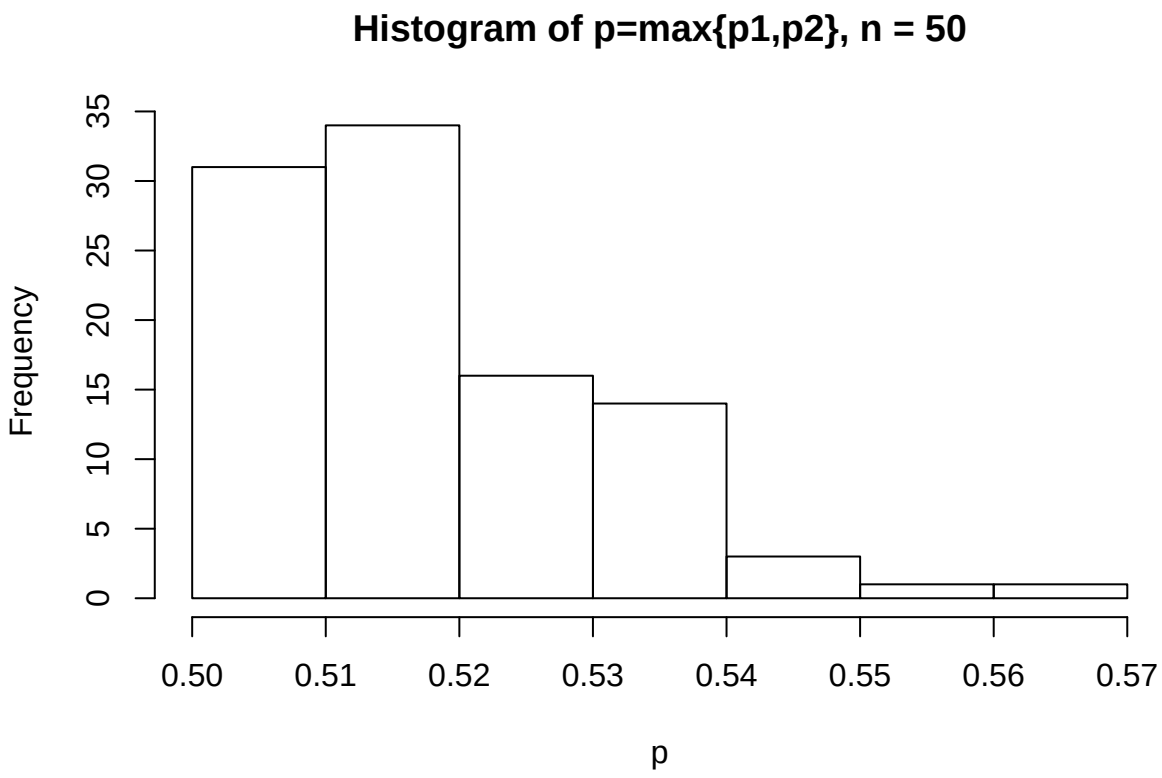
In theory, we would expect the histogram to be centered around 0.5. This didn't happen. Histogram seems to be centered around $p = 0.64$.

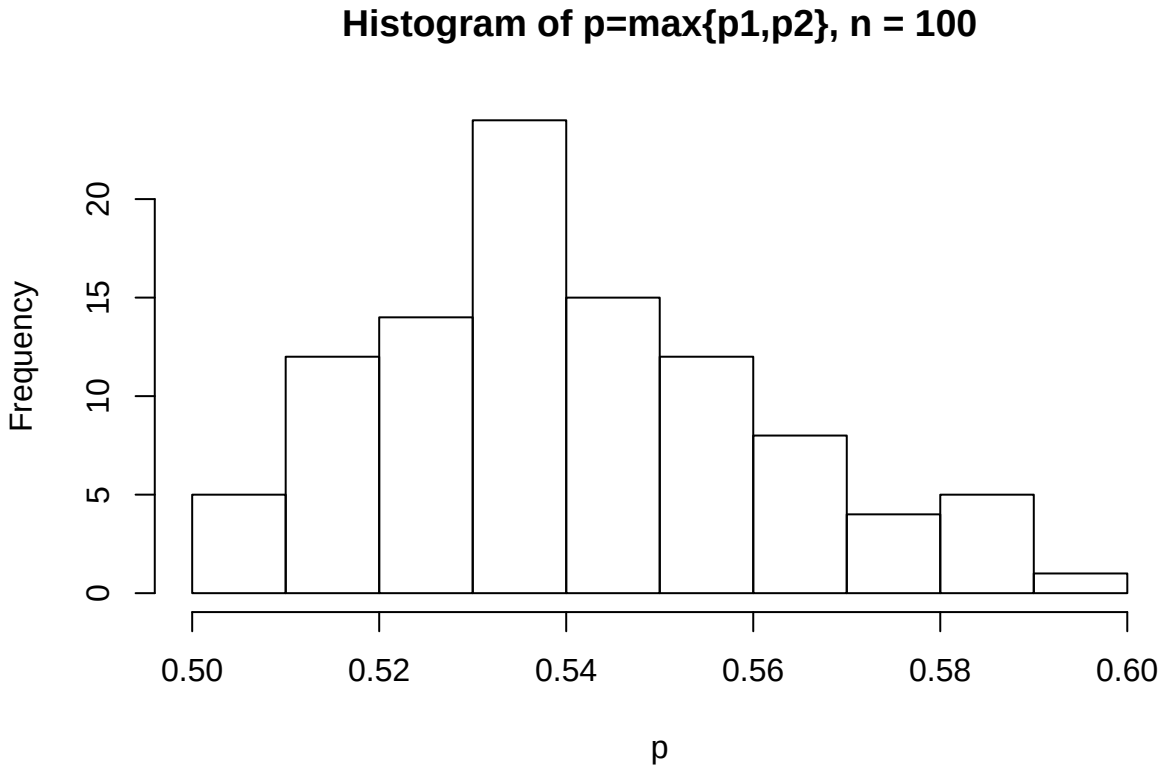


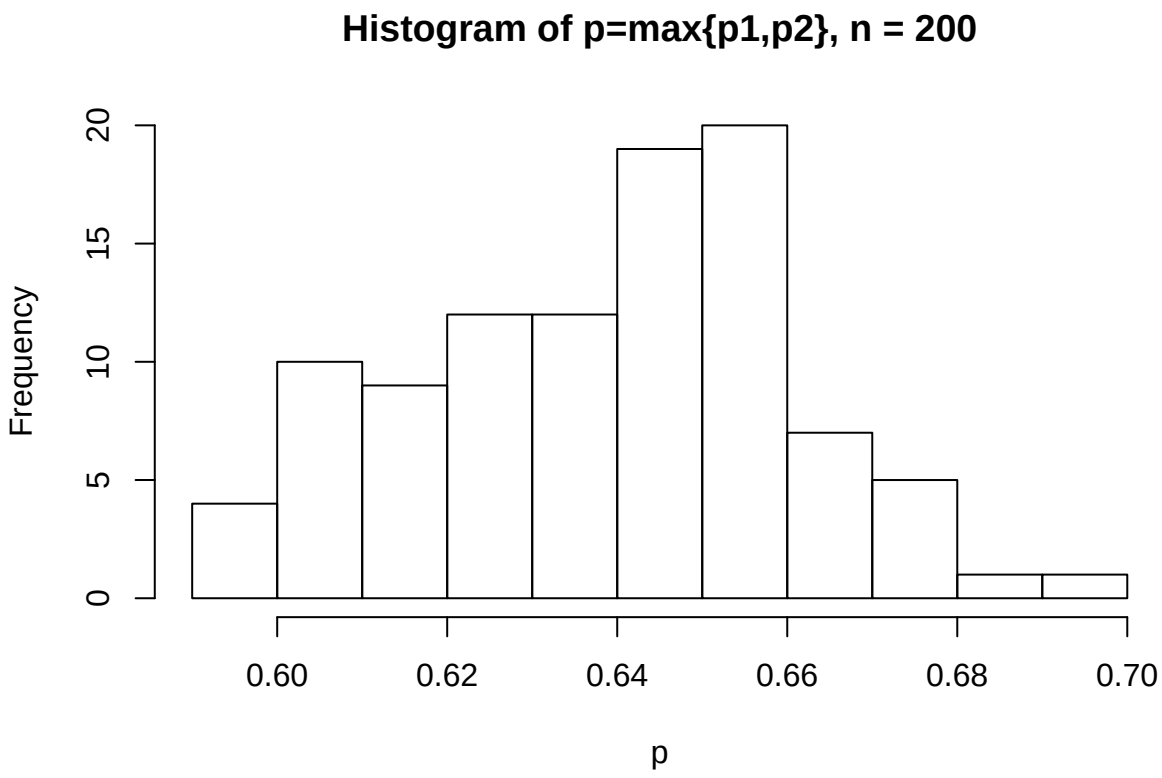
Part (g)

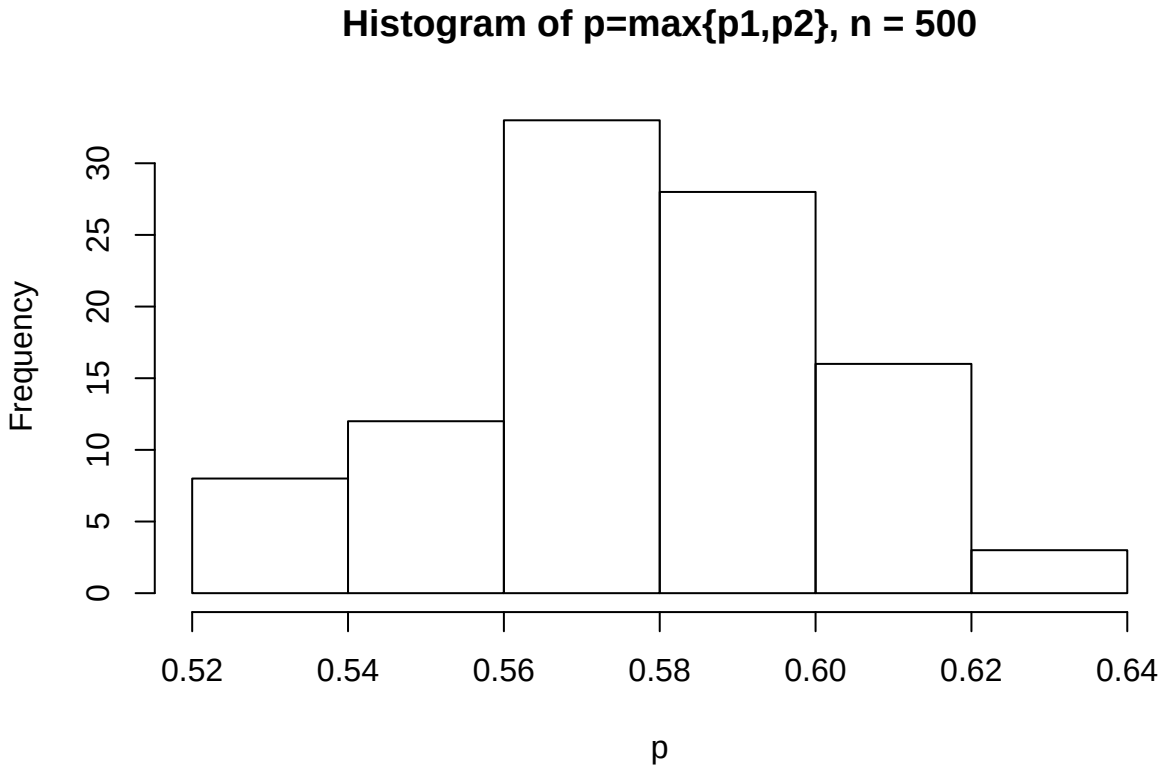
Repeat (e) and (f) for a few larger values of n. Are these histograms centered where you'd expect?

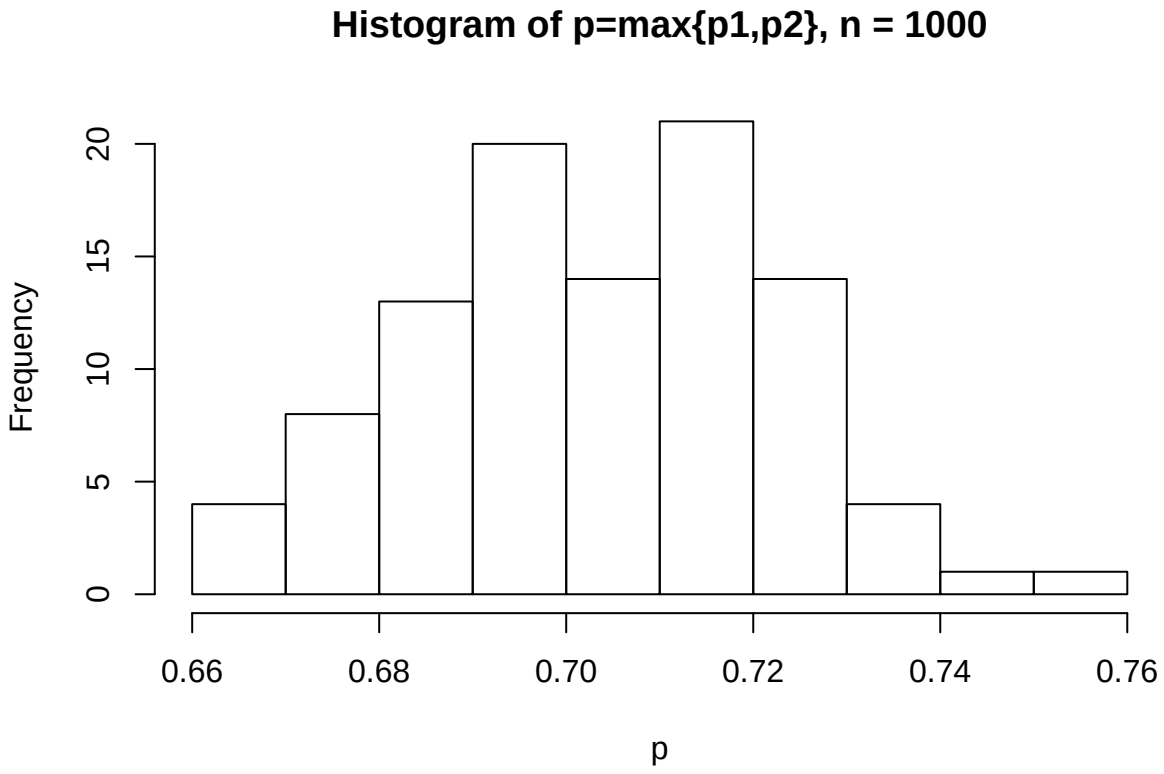
We'll do this for $n = 50, 100, 200, 500$. Once again, the histograms aren't centered around 0.5, though for bigger sample sizes the histogram seems to be centered around 0.7 (sample size = 1000). Here are the histograms.


Histogram of p=max{p1,p2}, n = 50
Frequency
0
5
10
15
20
25
30
35
0.50
0.51
0.52
0.53
0.54
0.55
0.56
0.57
p


Histogram of p=max{p1,p2}, n = 100
Frequency
0
5
10
15
20
0.50
0.52
0.54
0.56
0.58
0.60
p


Histogram of p=max{p1,p2}, n = 200
Frequency
0
5
10
15
20
0.60
0.62
0.64
0.66
0.68
0.70
p


Histogram of p=max{p1,p2}, n = 500
Frequency
0
5
10
15
20
25
30
0.52
0.54
0.56
0.58
0.60
0.62
0.64
p



Part (g)

You should be seeing a somewhat counterintuitive result here. Something that a larger n – our usual knight in shining armor as statisticians – doesn't necessarily help fix. Explain what's going on and why you're seeing the results that you are.

What is happening is that one of the features is being selected first by forward selection way more often than the other even tough the two of them appear to have the same importance in the true model $Y = X_1 + X_2 + \epsilon$. I think this is happening because in the original dataset X_2 is more highly correlated with the response variable than X_1 and taking the bootstrap samples could not account for this.