Model Selection

Charlie Qu

Oct 24 2017

We have seen that as the number of features in a model increases, the training error will necessarily decrease, but the test error may not.
This writing will explore this using simulation of data to compare methods for estimation and model selection.

Some "guideposts" for when to finish parts are provided within the problem set.

1. Generate a dataset with $p = 20$ features and $n = 1000$ as follows: First let's set our random seed in case we need to rerun parts later.

```
# set the random seed so that we can replicate results.
set.seed(8675309)
```

In order to simulate data, we need to specify the values of the "true" parameters. For this study we will use

```
# true parameters
sigma = 2.5
betatrue = c(4,2,0,0,0,-1,0,1.5, 0,0,0,1,0,.5,0,0,0,0,-1,1,4)
#          int|    X1                            | X2     |X3

truemodel = betatrue != 0
```

Generate Data with correlated columns.

```
#sample size
n = 1000

# generate some standard normals
  Z = matrix(rnorm(n*10, 0, 1), ncol=10, nrow=n)

#  Create X1 by taking linear cominations of Z to induce correlation among X1 components

  X1 = cbind(Z,
             (Z[,1:5] %*% c(.3, .5, .7, .9, 1.1) %*% t(rep(1,5)) +
             matrix(rnorm(n*5, 0, 1), ncol=5, nrow=n))
             )
# generate X2 as a standard normal
  X2 <- matrix(rnorm(n*4,0,1), ncol=4, nrow=n)

# Generate X3 as a linear combination of X2 and noise
  X3 <- X2[,4]+rnorm(n,0,sd=0.1)

# combine them
  X <- cbind(X1,X2,X3)

# Generate mu
# X does not have a column of ones for the intercept so need to add the intercept
# for true mu
mu = betatrue[1] + X %*% betatrue[-1]

# now generate Y
```

```
Y = mu + rnorm(n,0,sigma)

# make a dataframe and save it
df = data.frame(Y, X, mu)
```

2. Split your data set into a training set containing 100 observations and a test set containing 900 observations. Before splitting reset the random seed based on your team number

```
set.seed(1)   # replace 0 with team number before runing
n = nrow(df)
n.train = floor(.10*n)
train = sample(1:n, size=n.train, replace=FALSE)
df.train = df[train,]
df.test = df[-train,]
```

3. Using Ordinary Least squares based on fitting the full model for the training data, compute the average RMSE for a) estimating β_{true}, b) estimating $\mu_{true} = X_{\text{test}}\beta_{true}$ and c) out of sample prediction of Y_{test} for the test data. Note for a vector of length d, RMSE is defined as

$$RMSE(\hat{\theta}) = \sqrt{\sum_{i=1}^{d}(\hat{\theta}_j - \theta_j)^2/d}$$

   Provide Confidence/prediction intervals for β, and μ, and Y in the test data and report what percent of the intervals contain the true values. Do any estimates seem surprising?

(0) Preparation: fit the full model for the train data and construct functions for computing rmse and pentage. In the following chunks, (1),(2) and (3) are for RMSE; (4),(5) and (6) are for intervals and percentages.

```
#rmse function:
rmse=function(y,y.pred){
  rmse.val=sqrt(mean((y-y.pred)^2))
  return(rmse.val)
}

#percentage function
perc=function(d.f,beta_true){
  perc.val=sum(beta_true>d.f[,1] & beta_true<d.f[,2])/nrow(d.f)
  return(perc.val)
}

#fit the full model
fit.full=lm(formula = Y~.-mu,data = df.train)
results=summary(fit.full)

#store betatrue as a data frame for later use
df.beta.true=as.data.frame(t(betatrue))
colnames(df.beta.true)=c("(Intercept)",colnames(df)[2:21])
```

(1) Compute average RMSE for a) estimating β_{true}:

```
beta.pred.full=as.data.frame(results$coefficients)$Estimate
rmse.beta.full=rmse(beta.pred.full,betatrue)
print(rmse.beta.full)
```

```
## [1] 1.41319
```

(2) Compute average RMSE for b) estimating $\mu_{true} = X_{\text{test}}\beta_{true}$:

```
X.test=as.matrix(df.test%>%
  dplyr::select(-c(Y,mu)))
mu.true.full=X.test%*%betatrue[2:21]+betatrue[1]
mu.pred.full=X.test%*%beta.pred.full[2:21]+beta.pred.full[1]
rmse.mu.full=rmse(mu.pred.full,mu.true.full)
print(rmse.mu.full)
```

```
## [1] 1.055904
```

(3) Compute average RMSE for c) out of sample prediction of Y_{test} for the test data:

```
y.pred.full=predict(fit.full,newdata = df.test)
rmse.y.full=rmse(y.pred.full,df.test$Y)
print(rmse.y.full)
```

```
## [1] 2.720919
```

Summary:

The average RMSE for estimating β_{true} is 1.4132

The average RMSE for estimating $\mu_{true} = X_{\text{test}}\beta_{true}$ is 1.0559

The average RMSE for estimating out of sample prediction of Y_{test} for the test data is 2.7209

(4) Compute the confidence intervals for β:

```
CI.beta.full=as.data.frame(confint(fit.full)) %>%
  dplyr::mutate(beta.true=betatrue)%>%
  dplyr::mutate(beta.fit=fit.full[["coefficients"]])
row.names(CI.beta.full)=colnames(df.beta.true)
kable(CI.beta.full)
```

	2.5 %	97.5 %	beta.true	beta.fit
(Intercept)	3.4291887	4.7080065	4.0	4.0685976
V1	1.0611723	2.7390871	2.0	1.9001297
V2	-1.0674421	0.8425919	0.0	-0.1124251
V3	-1.0672896	1.1880179	0.0	0.0603642
V4	-1.4335532	1.3833834	0.0	-0.0250849
V5	-2.5281478	0.7265552	-1.0	-0.9007963
V6	-1.1192621	0.2928517	0.0	-0.4132052
V7	0.9763946	2.1363654	1.5	1.5563800
V8	-0.4424226	0.7307077	0.0	0.1441425
V9	-0.7139387	0.4280127	0.0	-0.1429630
V10	-0.2186895	1.0325409	0.0	0.4069257
V11	0.4190834	1.5595759	1.0	0.9893297
V12	-0.7874625	0.5776361	0.0	-0.1049132
V13	-0.2565923	1.0286902	0.5	0.3860490
V14	-0.2996105	1.1707256	0.0	0.4355575
V15	-0.6911102	0.6115857	0.0	-0.0397623
V16	-0.5732270	0.9227430	0.0	0.1747580
V17	-0.4063022	0.7301734	0.0	0.1619356
V18	-1.9166841	-0.7390837	-1.0	-1.3278839
V19	0.0741771	11.2134426	1.0	5.6438099
X3	-6.0331556	5.1850015	4.0	-0.4240770

(4) Compute the percentage of the confidence intervals of β containing β_{true}:

```
perc(CI.beta.full,betatrue)
```

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

(5) Compute the confidence interval for μ

```
CI.mu.full=as.data.frame(predict(fit.full,newdata = df.test,interval = "confidence"))%>%
  select(-fit)
```

(5) Compute the percentage of the intervals containing μ in test data:

```
perc(CI.mu.full,df.test$mu)
```

```
## [1] 0.9988889
```

(6) Compute the prediction intervals of Y:

```
PI.y.full=as.data.frame(predict(fit.full,newdata = df.test,interval = "prediction",level=0.95))%>%
  select(-fit)
```

(6) Compute the percentage of the prediction intervals for Y containing Y in the test data:

```
perc(PI.y.full,df.test$Y)
```

```
## [1] 0.9822222
```

Summary:

The percentage of the confidence intervals of β containing β_{true} is 100%

The percentage of the intervals of μ containing μ_{true} in the test data is 99.89%

The percentage of the prediction intervals for Y containing Y in the test data is 98.22%

Discussion:

It is suprising that all the β_{true} is contained in the confidence intervals of β although some intervals are somewhat not symmetric about β_{true} (V10,V14,V19 and X3). And we can see that for these predictors, there are relatively large differences between their fitted and true values of β. In spite of these differences, the other fitted β values are suprisingly close to the true value. Therefore, the full model is relatively close to the true model. We also notice that the percentage of prediction intervals for Y containing Y in the test data is also suprisingly high. If we compute the training rmse(calculated below), we can see that the training and testing rmse are relatively close to each other. Since the data are generated by the true model, we can also say that the full model by OLS is relatively close to the true model.

```
rmse(predict(fit.full,newdata = df.train),df.train$Y)
```

```
## [1] 2.508306
```

4. Perform best subset selection on the training data, and plot the training set RMSE for fitting associated with the best model of each size.

Perform best subset selection on the training data:

```
df.best.subset=df.train%>%
  dplyr::select(-c(Y,mu))
require(leaps)
```

```
## Loading required package: leaps
```

```
p=20
res=regsubsets(df.best.subset,df.train$Y,nvmax=1000)
```

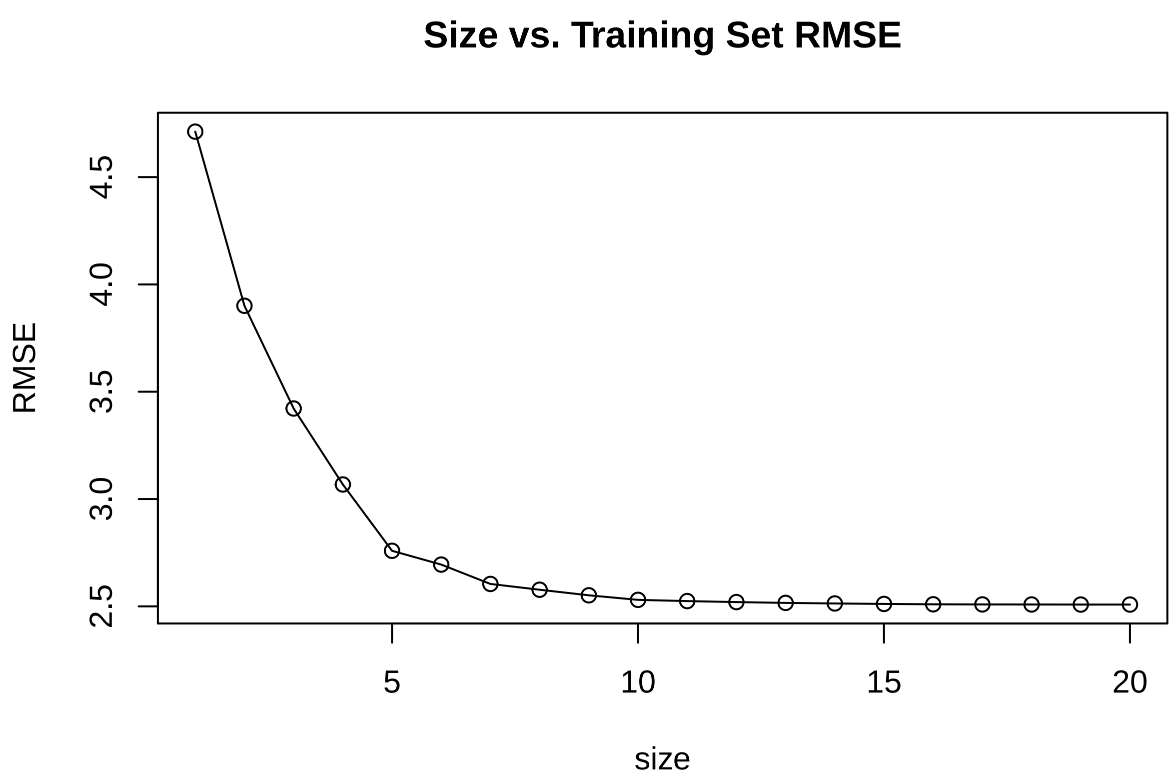
```r
res_summary = summary(res)
print(res_summary)
```

```
## Subset selection object
## 20 Variables  (and intercept)
##     Forced in Forced out
## V1      FALSE      FALSE
## V2      FALSE      FALSE
## V3      FALSE      FALSE
## V4      FALSE      FALSE
## V5      FALSE      FALSE
## V6      FALSE      FALSE
## V7      FALSE      FALSE
## V8      FALSE      FALSE
## V9      FALSE      FALSE
## V10     FALSE      FALSE
## V11     FALSE      FALSE
## V12     FALSE      FALSE
## V13     FALSE      FALSE
## V14     FALSE      FALSE
## V15     FALSE      FALSE
## V16     FALSE      FALSE
## V17     FALSE      FALSE
## V18     FALSE      FALSE
## V19     FALSE      FALSE
## X3      FALSE      FALSE
## 1 subsets of each size up to 20
## Selection Algorithm: exhaustive
##           V1  V2  V3  V4  V5  V6  V7  V8  V9  V10 V11 V12 V13 V14 V15 V16
## 1  ( 1 )  " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 2  ( 1 )  " " " " " " " " " " " " " " " " " " " " "*" " " " " " " " " " "
## 3  ( 1 )  " " " " " " " " " " " " "*" " " " " " " "*" " " " " " " " " " "
## 4  ( 1 )  "*" " " " " " " " " " " "*" " " " " " " "*" " " " " " " " " " "
## 5  ( 1 )  "*" " " " " " " " " " " "*" " " " " " " "*" " " " " " " " " " "
## 6  ( 1 )  "*" " " " " " " "*" " " "*" " " " " " " "*" " " " " " " " " " "
## 7  ( 1 )  "*" " " " " " " "*" " " "*" " " " " " " "*" " " " " "*" " " " "
## 8  ( 1 )  "*" " " " " " " "*" " " "*" " " " " "*" "*" " " " " "*" " " " "
## 9  ( 1 )  "*" " " " " " " "*" " " "*" " " " " "*" "*" " " "*" "*" " " " "
## 10  ( 1 ) "*" " " " " " " "*" "*" "*" " " " " "*" "*" " " "*" "*" " " " "
## 11  ( 1 ) "*" " " " " " " "*" "*" "*" " " " " "*" "*" " " "*" "*" " " " "
## 12  ( 1 ) "*" " " " " " " "*" "*" "*" "*" " " "*" "*" " " "*" "*" " " " "
## 13  ( 1 ) "*" " " " " " " "*" "*" "*" "*" "*" "*" "*" " " "*" "*" " " " "
## 14  ( 1 ) "*" " " " " " " "*" "*" "*" "*" "*" "*" "*" " " "*" "*" " " "*"
## 15  ( 1 ) "*" " " " " " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " "*"
## 16  ( 1 ) "*" "*" " " " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " "*"
## 17  ( 1 ) "*" "*" "*" " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " "*"
## 18  ( 1 ) "*" "*" "*" " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " "*"
## 19  ( 1 ) "*" "*" "*" " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
## 20  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
##           V17 V18 V19 X3
## 1  ( 1 )  " " " " " " "*"
## 2  ( 1 )  " " " " " " "*"
## 3  ( 1 )  " " " " "*" " "
## 4  ( 1 )  " " " " "*" " "
```

```
## 5  ( 1 )  " " "*" "*" " "
## 6  ( 1 )  " " "*" "*" " "
## 7  ( 1 )  " " "*" "*" " "
## 8  ( 1 )  " " "*" "*" " "
## 9  ( 1 )  " " "*" "*" " "
## 10  ( 1 ) " " "*" "*" " "
## 11  ( 1 ) "*" "*" "*" " "
## 12  ( 1 ) "*" "*" "*" " "
## 13  ( 1 ) "*" "*" "*" " "
## 14  ( 1 ) "*" "*" "*" " "
## 15  ( 1 ) "*" "*" "*" " "
## 16  ( 1 ) "*" "*" "*" " "
## 17  ( 1 ) "*" "*" "*" " "
## 18  ( 1 ) "*" "*" "*" "*"
## 19  ( 1 ) "*" "*" "*" "*"
## 20  ( 1 ) "*" "*" "*" "*"
```

Plot the training set RMSE for fitting associated with the best model at each size:

```r
plot(1:p, sqrt(res_summary$rss/100),"l",main = "Size vs. Training Set RMSE",xlab = "size",ylab = "RMSE")
points(1:p,sqrt(res_summary$rss/100))
```



5. Plot the test set RMSE for prediction associated with the best model of each size. For which model size does the test set RMSE take on its minimum value? Comment on your results. If it takes on its minimum value for a model with only an intercept or a model containing all of the predictors, adjust σ used in generating the data until the test set RMSE is minimized for an intermediate point.

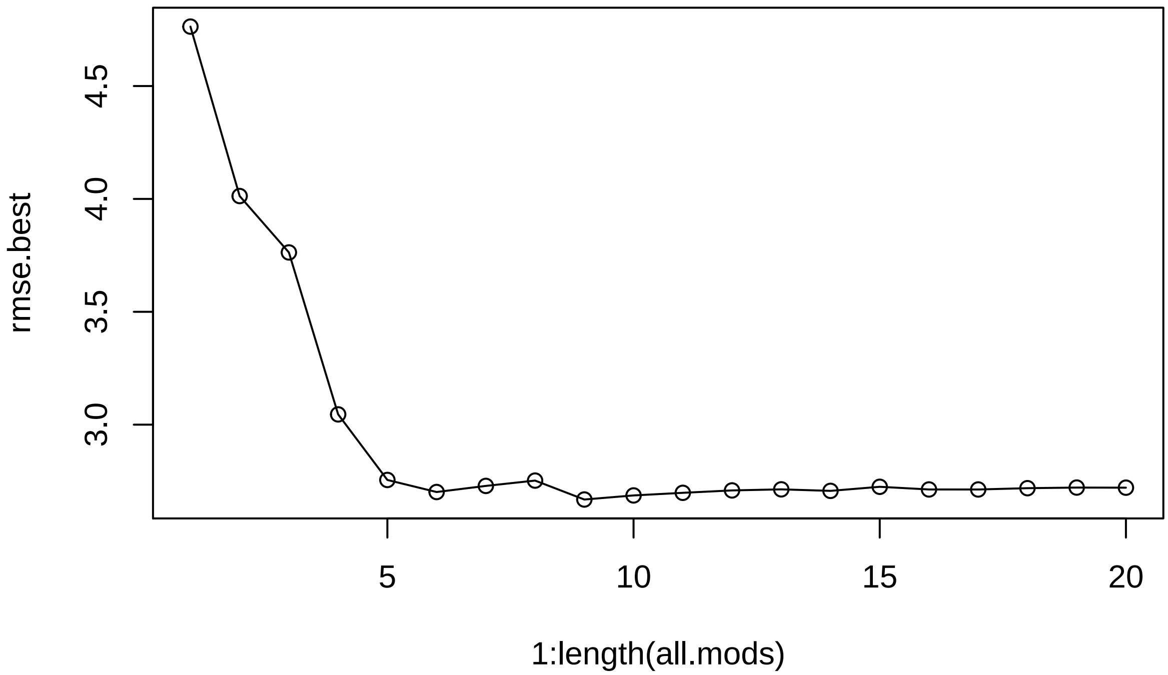
Compute test set RMSE for prediction associated with best model of each size:

```
#extract all models as what we did in lab
all.mods = lapply(1:nrow(res_summary$which), function(x) paste("Y~", paste(names(which(res_summary$whic

#for loop to calculate RMSE for best model of each size
rmse.best=c()
for (i in 1:length(all.mods)) {
  fo.temp=as.formula(all.mods[i] %>% unlist())
  fit.temp=lm(formula=fo.temp,data = df.train)
  y.pred=predict(fit.temp,df.test)
  rmse.temp=rmse(y.pred,df.test$Y)
  rmse.best=c(rmse.best,rmse.temp)
}
```

Plot the RMSEs with respect to size:

```
plot(1:length(all.mods),rmse.best,"l")
points(1:length(all.mods),rmse.best)
```



Find the model size corresponding to the lowest test set RMSE:

```
which(rmse.best==min(rmse.best))
```

```
## [1] 9
```

The corresponding model:

```
all.mods[which(rmse.best==min(rmse.best))]
```

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

```
## [1] "Y~ V1+V5+V7+V10+V11+V13+V14+V18+V19"
```

Comment:

from the plot, we can see that best model for size 9 corresponds to the lowest testing RMSE. And the model is $Y \sim V1 + V5 + V7 + V10 + V11 + V13 + V14 + V18 + V19$. However, we can also see from the plot that, for the best models for size larger than 5(excluding 8), the rmses are relatively close to each other. The model with the lowest test set rmse is indeed a better choice than others but we cannot simply pick it with out any consideration. It also noticeable that the testing rmse does not necessarily decreases as the training rmse decreases with the increase of size. We can see that the testing RMSE variates within a certain interval from size 5 to 20, indicating training rmse is not necessarily related to testing rmse.

6. How does the model at which the test set RMSE is minimized compare to the true model used to generate the data? Comment on the coefficient values and confidence intervals obtained from using all of the data. Do the intervals include the true values?

Fit the model with minimum test set RMSE with full data:

```
fo.best=as.formula(all.mods[which(rmse.best==min(rmse.best))] %>% unlist())
fit.best=lm(formula=fo.best,data = df)
summary(fit.best)
```

```
##
## Call:
## lm(formula = fo.best, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.3495 -1.8276 -0.0008  1.5929  7.6541
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.06107    0.08157  49.789   <2e-16 ***
## V1           1.94969    0.08251  23.631   <2e-16 ***
## V5          -0.90849    0.10293  -8.826   <2e-16 ***
## V7           1.61598    0.08064  20.039   <2e-16 ***
## V10          0.04771    0.08385   0.569    0.570
## V11          0.90868    0.07046  12.896   <2e-16 ***
## V13          0.57106    0.06867   8.315    3e-16 ***
## V14          0.03883    0.07089   0.548    0.584
## V18         -1.14948    0.08128 -14.141   <2e-16 ***
## V19          4.93197    0.08131  60.659   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.571 on 990 degrees of freedom
## Multiple R-squared:  0.8575, Adjusted R-squared:  0.8562
## F-statistic: 661.9 on 9 and 990 DF,  p-value: < 2.2e-16
```

Compare with the true model, NAs indicate that the predictor is not included in the fitted model:

```
df.compare.best=cbind(fit.best$coefficients,confint(fit.best))
df.compare.best=merge(t(df.beta.true),df.compare.best,by="row.names",all=T,sort=F)%>%
  mutate(contain.true=V1.x>`2.5 %` &V1.x<`97.5 %`)
colnames(df.compare.best)=c("predictor","beta.true","beta.fit","2.5%","97.5%","Contain.True.Beta")
kable(df.compare.best)
```

predictor	beta.true	beta.fit	2.5%	97.5%	Contain.True.Beta
(Intercept)	4.0	4.0610727	3.9010121	4.2211333	TRUE
V1	2.0	1.9496854	1.7877767	2.1115941	TRUE
V5	-1.0	-0.9084862	-1.1104777	-0.7064948	TRUE
V7	1.5	1.6159800	1.4577282	1.7742319	TRUE
V10	0.0	0.0477079	-0.1168345	0.2122503	TRUE
V11	1.0	0.9086848	0.7704136	1.0469560	TRUE
V13	0.5	0.5710610	0.4362968	0.7058253	TRUE
V14	0.0	0.0388257	-0.1002882	0.1779397	TRUE
V18	-1.0	-1.1494762	-1.3089865	-0.9899658	TRUE
V19	1.0	4.9319662	4.7724137	5.0915186	FALSE
V4	0.0	NA	NA	NA	NA
V9	0.0	NA	NA	NA	NA
V2	0.0	NA	NA	NA	NA
V3	0.0	NA	NA	NA	NA
V8	0.0	NA	NA	NA	NA
V17	0.0	NA	NA	NA	NA
V6	0.0	NA	NA	NA	NA
V15	0.0	NA	NA	NA	NA
V12	0.0	NA	NA	NA	NA
V16	0.0	NA	NA	NA	NA
X3	4.0	NA	NA	NA	NA

Intervals does not contain the true value:

```
df.compare.best$predictor[which(df.compare.best$Contain.True.Beta==0)]
```

```
## [1] "V19"
```

Discussion:

from the table, we can see that, 9 out of 10 (i.e. 90%) intervals of estimated βs contain β_{true} except for V19. X3, whose β_{true} is equal to 4 is missing in the model while V10 and V14, whose β_{true}s are equal to 0, are included. For V10 and V14, their estimated coefficients are very close to 0, indicating these predictors can possibly be dropped from the model. For V19 and X3, we notice that the estimated β for V19 is significantly larger than β_{true} and this model fails to include X3 as a predictor. From the generation of the data, we can see that X3 is generated from V19 with the addition of a noise(with $N(mean = 0, sd = 0.1)$). Therefore, X3 is excluded from the model because V19 and X3 are highly correlated. Thus, the large difference between the fitted and ture β for V19 is probably because V19 needs to account for the contribution of the missing of X3. Aside from V10, V14, V19 and X3, all the other estimated values for β are very close to β_{true}. Overall, the model from best subset selection is very close to the true model.

Moreover, we need to pay attention to the data we are using. Since the data is generated by the true model, among same models, the ones fitted with more data are generally more close to the true model than those with less data. Therefore, As we can see in AIC model in nest questions, it is the same model as we have found by best subset selection. However, β for V10 and V14 in the AIC model is not close to 0 as we will see, indicating that the model is less close to the true model. It is because we only use the training data for AIC model and, with less data, the model would be less close to the true model. If we use the training data for the model in best subset selection, the outputted βs would be same as those in AIC model.

7. Use AIC with stepwise or all possible subsets to select a model based on the training data and then use OLS to estimate the parameters under that model. Using the estimates to compute the RMSE for a) estimating β^{true}, b) estimating μ_{true} in the test data, and c) predicting Y_{test}. For prediction, does this find the best model in terms of RMSE? Does AIC find the true model? Comment on your findings.

Use AIC with direction=backwards:

```r
res.step.aic = step(lm(Y~.-mu,data=df.train),k=2,direction="backward",trace=0)
results.aic=summary(res.step.aic)
print(summary(res.step.aic))
```

```
##
## Call:
## lm(formula = Y ~ V1 + V5 + V7 + V10 + V11 + V13 + V14 + V18 +
##     V19, data = df.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.6008 -1.6527  0.1735  1.2128  7.0758
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.0255     0.2855  14.100  < 2e-16 ***
## V1            1.8405     0.2898   6.350 8.58e-09 ***
## V5           -1.0313     0.3328  -3.099  0.00259 **
## V7            1.5796     0.2619   6.031 3.55e-08 ***
## V10           0.4245     0.2814   1.508  0.13496
## V11           1.0522     0.2302   4.571 1.54e-05 ***
## V13           0.3380     0.2492   1.357  0.17829
## V14           0.3650     0.2574   1.418  0.15971
## V18          -1.3392     0.2725  -4.914 3.97e-06 ***
## V19           5.1939     0.3208  16.192  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.689 on 90 degrees of freedom
## Multiple R-squared:  0.8234, Adjusted R-squared:  0.8057
## F-statistic: 46.61 on 9 and 90 DF,  p-value: < 2.2e-16
```

(1) Compute average RMSE for a) estimating β_{true}:

```r
beta.pred.aic=as.data.frame(results.aic$coefficients)$Estimate
beta.true.aic=as.vector(t(df.beta.true[,c(names(res.step.aic$coefficients))]))
rmse.beta.aic=rmse(beta.pred.aic,beta.true.aic)
print(rmse.beta.aic)
```

```
## [1] 1.344595
```

(2) Compute average RMSE for b) estimating μ_{true} in the test data:

```r
n.aic=length(beta.true.aic)
mycolumn1=rownames(results.aic[["coefficients"]])[2:n.aic]
col.number.1=match(mycolumn1,names(df.test))
X.test.aic=as.matrix(df.test%>%
  dplyr::select(col.number.1))
mu.pred.aic=X.test.aic%*%beta.pred.aic[2:n.aic]+beta.pred.aic[1]
rmse.mu.aic=rmse(mu.pred.aic,df.test$mu)
print(rmse.mu.aic)
```

```
## [1] 0.9040591
```

(3) Compute average RMSE for c) predicting Y_{test}:

```r
y.pred.aic=predict(res.step.aic,newdata = df.test)
rmse.y.aic=rmse(y.pred.aic,df.test$Y)
print(rmse.y.aic)
```

```
## [1] 2.668482
```

Summary:

The average RMSE for estimating β_{true} is 1.3446

The average RMSE for estimating $\mu_{true} = X_{\text{test}}\beta_{true}$ is 0.9041

The average RMSE for estimating out of sample prediction of Y_{test} for the test data is 2.6685

Discussion:

AIC with backwards steps succeeded in finding the best model in terms of testing RMSE: $Y \sim V1 + V5 + V7 + V10 + V11 + V13 + V14 + V18 + V19$. The same as the best model in terms of rmse, the AIC model still failed in finding the true model. It failed to include the predictor X3 which has $\beta_{true} = 4$. However, we used the full data for the best model in terms of rmse. As we discussed in the previous question, the use of full data would make the model more close to the model model. If we use training data for both model, they will be the same.

In terms of rmse, the AIC model is the same as the model we found by best subset selection. Also,all the rmse's of AIC model is smaller than those of full model. From the next couple of questions, we get the rmse's of BIC model, which are all larger than those of AIC. Therefore, from the perspective of rmse or prediction, AIC model is the best model among the 3 models: full model, AIC model and BIC model.

8. Take a look at the summaries from the estimates under the best AIC model fit to the training data. Create confidence intervals for the β's and comment on whether they include zero or not or the true value.

Compute confidence intervals for the β's for AIC model:

```r
rownames.aic=names(res.step.aic$coefficients)

CI.beta.aic=as.data.frame(confint(res.step.aic))

CI.beta.aic=CI.beta.aic%>%
  mutate(beta.fit=res.step.aic[["coefficients"]])%>%
  mutate(beta.true=beta.true.aic)%>%
  mutate(Include.True.Beta=beta.true>`2.5 %` & beta.true<`97.5 %`)%>%
  mutate(Include.Zero=0>`2.5 %` & 0<`97.5 %`)
rownames(CI.beta.aic)=rownames.aic
kable(CI.beta.aic)
```

	2.5 %	97.5 %	beta.fit	beta.true	Include.True.Beta	Include.Zero
(Intercept)	3.4583228	4.5926979	4.0255103	4.0	TRUE	FALSE
V1	1.2646426	2.4162978	1.8404702	2.0	TRUE	FALSE
V5	-1.6923898	-0.3701990	-1.0312944	-1.0	TRUE	FALSE
V7	1.0592343	2.0999980	1.5796162	1.5	TRUE	FALSE
V10	-0.1346060	0.9835648	0.4244794	0.0	TRUE	TRUE
V11	0.5949199	1.5094432	1.0521816	1.0	TRUE	FALSE
V13	-0.1569852	0.8330561	0.3380354	0.5	TRUE	TRUE
V14	-0.1464528	0.8764043	0.3649757	0.0	TRUE	TRUE
V18	-1.8805995	-0.7978355	-1.3392175	-1.0	TRUE	FALSE
V19	4.5566190	5.8311154	5.1938672	1.0	FALSE	FALSE

Find confidence intervals not including β_{true}:

```r
rownames(subset(CI.beta.aic,Include.True.Beta==0))
```

```
## [1] "V19"
```

Find confidence intervals including 0:

```r
rownames(subset(CI.beta.aic,Include.Zero==1))
```

```
## [1] "V10" "V13" "V14"
```

Comment:

The confidence intervals of V10,V13 and V14 contains 0. For V10 and V14, the intervals containing 0 is resonable(or a good sign for predicting true model) since β_{true} for these 2 predictors are 0. For V13, the confidence interval containing 0 indicating we can not reject that β for V13 is 0. Since β_{true} for V13 is equal to 0.5 not 0, the confidence interval signals the discrepency betwwen the AIC model and the true model. The confidence interval for β for V19 does not contain β_{true}. The reason is probably that V19 and X3 are highly correlated and V19 needs to account for the contribution of the missing of X3.

9. Use BIC with either stepwise or all possible subsets to select a model and then use OLS to estimate the parameters under that model. Use the estimates to compute the RMSE for a) estimating β^{true}, b) μ_{true} for the test data, and c) predicting Y_{test}. For prediction, does this find the best model in terms of RMSE? Does BIC find the true model? Comment on your findings.

Use BIC with direction=backwards and n=100:

```r
n.log=100
res.step.bic <- step(lm(Y~.-mu,data=df.train),k=log(n.log),direction="backward",trace = 0)
results.bic=summary(res.step.bic)
print(results.bic)
```

```
##
## Call:
## lm(formula = Y ~ V1 + V5 + V7 + V11 + V14 + V18 + V19, data = df.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.4681 -1.6455  0.1049  1.4254  7.0705
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.9220     0.2807  13.971  < 2e-16 ***
## V1            1.7856     0.2906   6.144 2.03e-08 ***
## V5           -0.9544     0.3336  -2.861  0.00522 **
## V7            1.6315     0.2598   6.281 1.10e-08 ***
## V11           1.1583     0.2114   5.480 3.69e-07 ***
## V14           0.5578     0.2190   2.547  0.01253 *
## V18          -1.3009     0.2727  -4.771 6.86e-06 ***
## V19           5.1637     0.3234  15.968  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.715 on 92 degrees of freedom
## Multiple R-squared:  0.816,  Adjusted R-squared:  0.802
## F-statistic: 58.27 on 7 and 92 DF,  p-value: < 2.2e-16
```

(1) Compute average RMSE for a) estimating β_{true}:

```
beta.pred.bic=as.data.frame(results.bic$coefficients)$Estimate
beta.true.bic=as.vector(t(df.beta.true[,c(names(res.step.bic$coefficients))]))
rmse.beta.bic=rmse(beta.pred.bic,beta.true.bic)
print(rmse.beta.bic)
```

```
## [1] 1.493102
```

(2) Compute average RMSE for b) estimating μ_{true} in the test data:

```
n.bic=length(beta.true.bic)
mycolumn2=rownames(results.bic[["coefficients"]])[2:n.bic]
col.number.2=match(mycolumn2,names(df.test))

X.test.bic=as.matrix(df.test%>%
  dplyr::select(col.number.2))
mu.pred.bic=X.test.bic%*%beta.pred.bic[2:n.bic]+beta.pred.bic[1]
rmse.mu.bic=rmse(mu.pred.bic,df.test$mu)
print(rmse.mu.bic)
```

```
## [1] 1.015786
```

(3) Compute average RMSE for c) predicting Y_{test}:

```
y.pred.bic=predict(res.step.bic,newdata = df.test)
rmse.y.bic=rmse(y.pred.bic,df.test$Y)
print(rmse.y.bic)
```

```
## [1] 2.728731
```

Summary:

The average RMSE for estimating β_{true} is 1.4931

The average RMSE for estimating $\mu_{true} = X_{\text{test}}\beta_{true}$ is 1.0158

The average RMSE for estimating out of sample prediction of Y_{test} for the test data is 2.7287

Discussion:

BIC with backwards steps failed in finding both the best model in terms of testing RMSE and the true model. The predictor X3 is excluded from the model for the reason as discussed before. We can see that BIC also failed to inlcude V13 in β_{true}. The model includes V14, which has 0 value in β_{true}, instead. Compared to AIC model, the BIC model succeeded in excluding V10, whose β_{true} equals to 0. Therefore, from the perspective of finding true model, BIC model is relatively better than the AIC model .

In terms of rmse, the rmse for β and Y of BIC model is larger than those of full model while the rmse for μ is smaller than that of full model but is still larger than the AIC model. Therefore, from the perspective of rmse or prediction, AIC model is better than BIC model.

10. Take a look at the summaries from the estimates under the best BIC model fit to the training data. Create confidence intervals for the β's and comment on whether they include zero or not or the true value.

Compute confidence intervals for the β's for BIC model:

```
rownames.bic=names(res.step.bic$coefficients)

CI.beta.bic=as.data.frame(confint(res.step.bic))

CI.beta.bic=CI.beta.bic%>%
  mutate(fitted.val=res.step.bic[["coefficients"]])%>%
```

```r
  mutate(true.val=beta.true.bic)%>%
  mutate(Include.True=true.val>`2.5 %` & true.val<`97.5 %`)%>%
  mutate(Include.Zero=0>`2.5 %` & 0<`97.5 %`)
rownames(CI.beta.bic)=rownames.bic
kable(CI.beta.bic)
```

	2.5 %	97.5 %	fitted.val	true.val	Include.True	Include.Zero
(Intercept)	3.3644311	4.4795443	3.9219877	4.0	TRUE	FALSE
V1	1.2084340	2.3627968	1.7856154	2.0	TRUE	FALSE
V5	-1.6168590	-0.2919157	-0.9543874	-1.0	TRUE	FALSE
V7	1.1155863	2.1474284	1.6315074	1.5	TRUE	FALSE
V11	0.7384925	1.5780602	1.1582763	1.0	TRUE	FALSE
V14	0.1228207	0.9928120	0.5578164	0.0	FALSE	FALSE
V18	-1.8424371	-0.7593497	-1.3008934	-1.0	TRUE	FALSE
V19	4.5214828	5.8059773	5.1637301	1.0	FALSE	FALSE

```r
rownames(subset(CI.beta.bic,Include.True==0))
```

```
## [1] "V14" "V19"
```

```r
rownames(subset(CI.beta.bic,Include.Zero==1))
```

```
## character(0)
```

Comment:

none of the confidence intervals contain 0. However, for V14, the confidence interval should contain 0 if BIC model is close to the true model since β_{true} is equal to 0. Therefore, the confidence interval of V14 indicates the discrepency between the BIC model and the true model. The confidence interval for β for V14 and V19 does not contain β_{true}. For V19, the reason is probably that V19 and X3 are highly correlated and V19 needs to account for the contribution of the missing of X3. For V14,

11. Provide a paragraph summarizing your findings and any recommendations for model selection and inference for the tasks of prediction of future data, estimation of parameters or selecting the true model.

Generate a table for comparing testing rmse for each model(with train data):

```r
df.compare.model=as.data.frame(t(c(min(rmse.best),rmse.y.aic,rmse.y.bic,rmse.y.full)))
colnames(df.compare.model)=c("best subset selection","AIC","BIC","Full Model")
row.names(df.compare.model)="test set rmse"
kable(df.compare.model)
```

	best subset selection	AIC	BIC	Full Model
test set rmse	2.668482	2.668482	2.728731	2.720919

Summary:
We have come up with 4 models, namely AIC model, BIC model, minimized RMSE model, and the original Full model, which are shown below: AIC model

$$Y \sim V_1 + V_5 + V_7 + V_{11} + V_{13} + V_{14} + V_{18} + V_{19}$$

BIC model

$$Y \sim V_1 + V_5 + V_7 + V_{11} + V_{14} + V_{18} + V_{19}$$

Best Subset Selection model

$$Y \sim V_1 + V_5 + V_7 + V_{10} + V_{11} + V_{13} + V_{14} + V_{18} + V_{19}$$

Full model

$$Y \sim V_1 + V_2 + V_3 + V_4 + V_5 + V_6 + V_7 + V_8 + V_9 + V_{10} + V_{11} + V_{12} + V_{13} + V_{14} + V_{15} + V_{16} + V_{17} + V_{18} + V_{19} + X_3$$

Upon scrutiny, we observe that AIC model is slightly more complicated than BIC model. This is understandable, because BIC penalizes harder than AIC – recall that BIC has a penalty of $log(n)$, where $n = 100$ in this case, while AIC has a penalty of 2. Notably, AIC and BIC are both approximately correct according to a different goal and a different set of asymptotic assumptions. We have to adjust based on our prediction needs. It is interesting to know that the RMSE of our best subset selection model is the same as that of our AIC model. This shows that AIC finds the best model in terms of RMSE.

Upon scrutiny, we observe that AIC model is slightly more complicated than BIC model. This is understandable, because BIC penalizes harder than AIC – recall that BIC has a penalty of $log(n)$, where $n = 100$ in this case, while AIC has a penalty of 2. Notably, AIC and BIC are both approximately correct according to a different goal and a different set of asymptotic assumptions. We have to adjust based on our prediction needs. It is interesting to know that the RMSE of our best subset selection model is the same as that of our AIC model. This shows that AIC finds the best model in terms of RMSE.