Homework 3: Linear Model Selection and Regularization

Wanitchaya Poonpatanapricha

Conceptual exercises

1)

```
n <- 1000
err <- rnorm(n, mean = 0, sd = 1)
df <- vector("list", 20)
name <- vector(typeof("x") , 20)
for (i in seq(20)){
  name[i] <- paste0("p",i)
  if (i%%2 == 0) {
    df[[i]] <- runif(n, min = i, max = i^2)
  }
  else {
    df[[i]] <- runif(n, min = -1*i, max = i)
  }
}
df <- as.data.frame(df) %>%
  unnest()
names(df) <- name
beta <- rnorm(20, mean = 0, sd = 15)
beta[c(2,5,10,15)] <- 0
beta
```

```
##  [1] -19.312   0.000  12.530  -4.798   0.000   4.095   4.388  20.242  12.108
## [10]   0.000   9.942 -27.396  27.725   3.983   0.000  -3.255  26.999 -21.581
## [19]   4.476   2.416
```

```
y <- as.matrix(df) %*% as.matrix(beta) + as.matrix(err)
df$y <- y
str(df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1000 obs. of  21 variables:
##  $ p1 : num  -0.644 0.37 -0.531 -0.651 0.847 ...
##  $ p2 : num  3.11 2.81 3.87 2.26 3.14 ...
##  $ p3 : num  -2.798 0.604 -0.999 0.71 -2.663 ...
##  $ p4 : num  4.11 12.79 8.2 15.66 8.26 ...
##  $ p5 : num  -2.71 -2.31 3.72 5 3.28 ...
##  $ p6 : num  28.5 29.7 34.6 33.8 16.7 ...
##  $ p7 : num  5.23 -5.36 -4.42 2.54 -2.76 ...
##  $ p8 : num  37.7 31.5 24.6 45.7 27.8 ...
##  $ p9 : num  -2.294 2.923 0.588 -3.528 8.561 ...
##  $ p10: num  82.6 32.9 64.7 99.8 18.9 ...
##  $ p11: num  10.282 -6.727 -0.393 -6.689 -5.815 ...
```

```
##  $ p12: num  39.8 101.5 16.1 43 135.5 ...
##  $ p13: num  -0.0715 12.7568 10.0798 -6.2146 -11.7315 ...
##  $ p14: num  185.4 43.1 22 180.7 113.9 ...
##  $ p15: num  2.82 -8.6 7.89 -4.08 13.55 ...
##  $ p16: num  83.1 162.1 136.7 214.7 210.4 ...
##  $ p17: num  -2.62 -7.07 3.35 -6.67 -16.19 ...
##  $ p18: num  134 137 209 182 178 ...
##  $ p19: num  -13.965 -1.673 -1.249 -14.25 -0.125 ...
##  $ p20: num  180.3 132.5 254.6 149.2 37.8 ...
##  $ y  : num [1:1000, 1] -2277 -4973 -3742 -4231 -7873 ...
```

2)

```
split <- initial_split(data = df, prop = 0.1)
train <- training(split)
test <- testing(split)
dim(train)
```

```
## [1] 100  21
```

```
dim(test)
```

```
## [1] 900  21
```

3)

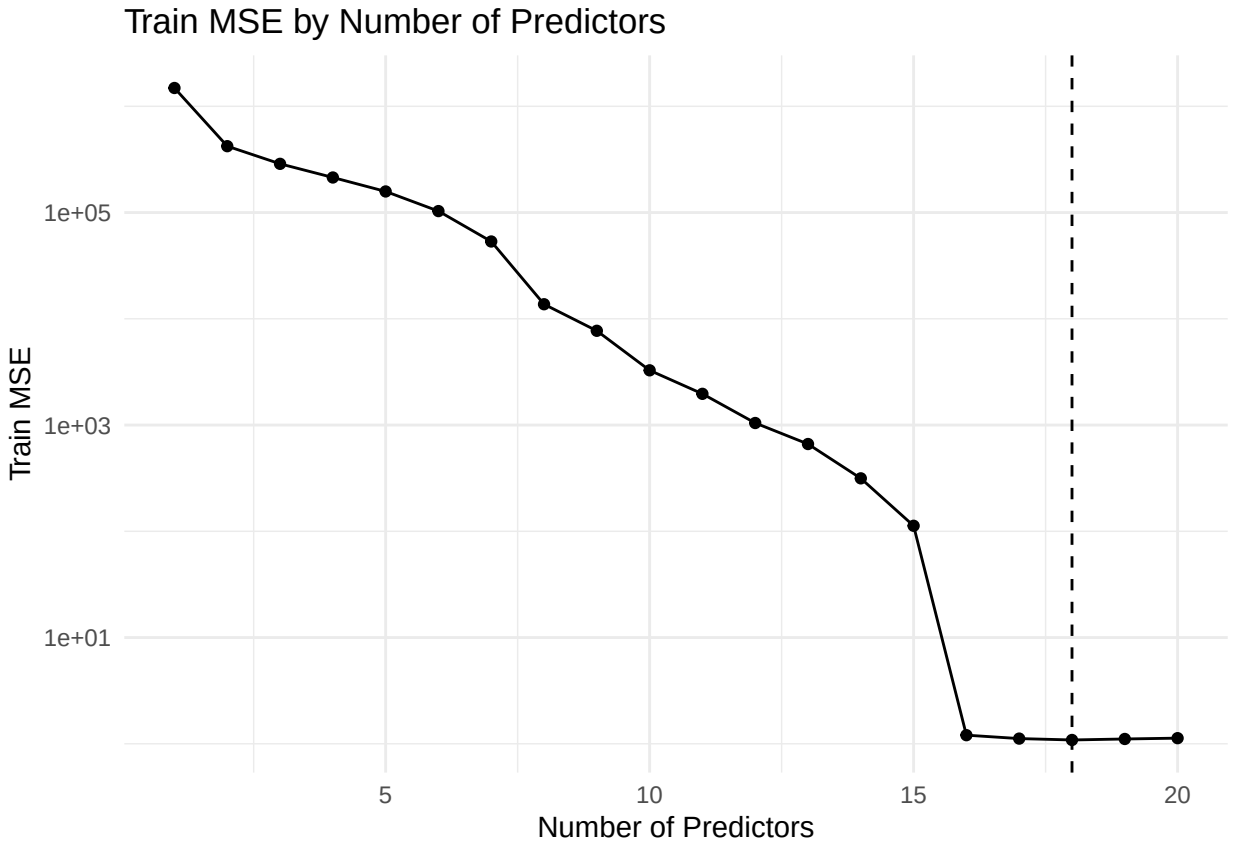
```
best_subset <- regsubsets(y ~ ., data = train, nvmax = 20)

kable(summary(best_subset)$outmat, caption = "Subset Selections")
```

Table 1: Subset Selections

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20
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)	*	*	*	*		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20
20 (1)	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

```
results <- summary(best_subset)

predict.regsubsets <- function(object, newdata, id ,...) {
  form <- as.formula(y ~ .)
  mat <- model.matrix(form, newdata)
  coefi <- coef(object, id = id)
  xvars <- names(coefi)
  as.vector(mat[, xvars] %*% coefi, mode = "integer")
}

predict_all <- function(object, newdata, k){
  map(k, ~ predict.regsubsets(object, newdata, id = .x)) %>%
    set_names(k)
}

train_mse <- as.data.frame(predict_all(best_subset, train, k = 1:20)) %>%
  mutate(truth = train$y) %>%
  gather(k, pred,-truth) %>%
  group_by(k) %>%
  mse(truth = truth, estimate = pred) %>%
  mutate(mse = .estimate) %>%
  select(k, mse)
train_mse$k <- c(1,10:19,2,20,3:9)

train_mse <- train_mse %>%
  arrange(k)

ggplot(train_mse, aes(k, mse)) +
  geom_point() +
  geom_line() +
  geom_vline(xintercept = which.min(train_mse$mse), linetype = 2) +
  scale_y_log10() +
  labs(title = "Train MSE by Number of Predictors",
       y = "Train MSE",
       x = "Number of Predictors")
```

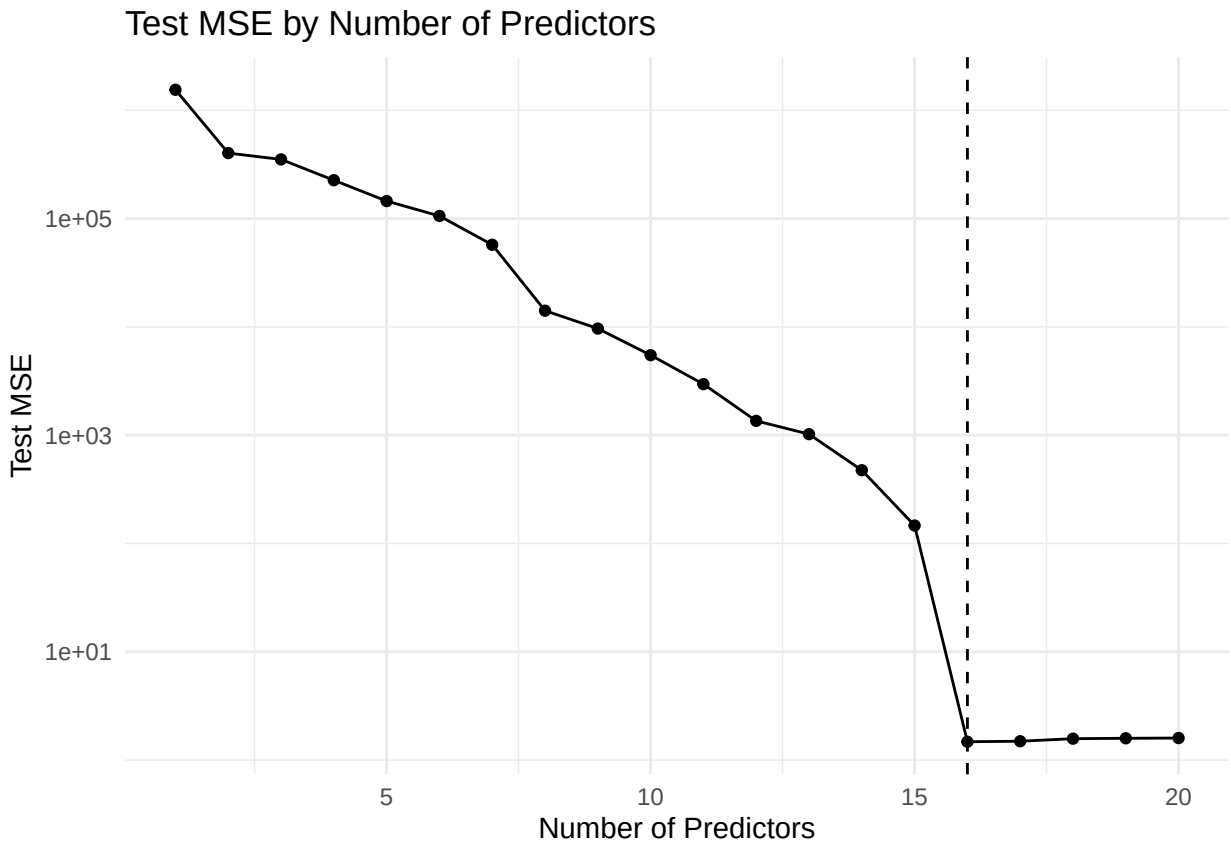


From the plot, we can see that the model with 18 predictors has the lowest training MSE. However, its MSE is not drastically different from models with 16, 17, 19, and 20. Moreover, we would expect only 16 predictors to matter since we set 4 betas to 0 in the data simulation. Hence, it would be better to see the test MSE. We expect the test MSE to be able to discriminate the best model better than the train MSE.

4)

```
test_mse <- as.data.frame(predict_all(best_subset, test, k = 1:20)) %>%
  mutate(truth = test$y) %>%
  gather(k, pred,-truth) %>%
  group_by(k) %>%
  mse(truth = truth, estimate = pred) %>%
  mutate(mse = .estimate) %>%
  select(k, mse)
test_mse$k <- c(1,10:19,2,20,3:9)

test_mse <- test_mse %>%
  arrange(k)

ggplot(test_mse,aes(k, mse)) +
  geom_point() +
  geom_line() +
  geom_vline(xintercept = which.min(test_mse$mse), linetype = 2) +
  scale_y_log10() +
  labs(title = "Test MSE by Number of Predictors",
       y = "Test MSE",
```

```
       x = "Number of Predictors")
```



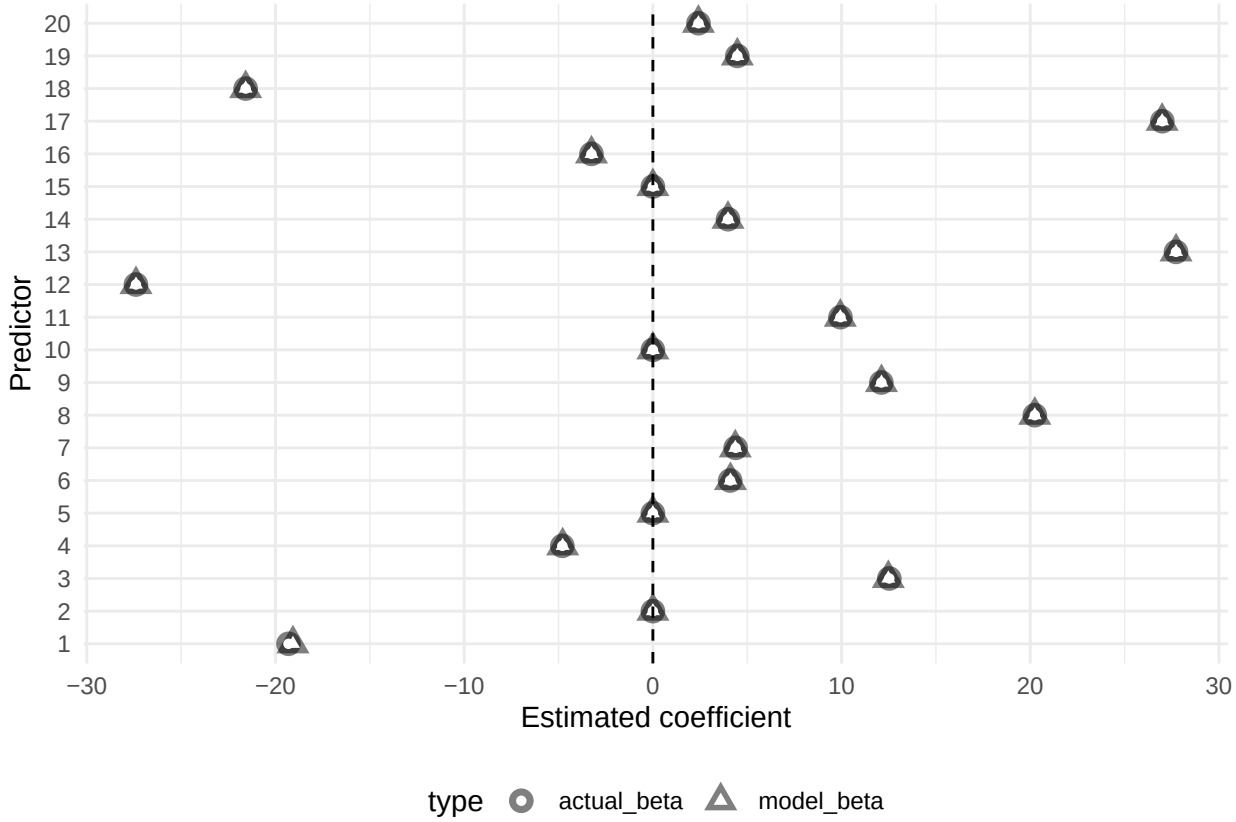
5)

The test MSE is lowest when the number of predictors is 16. This matches the actual data simulation where 4 of the betas are set to 0. And indeed, if we look at table 2, we can see that the predictors not included in this best model are p2, p5, p10, and p15, those that have the simulated betas as 0.

6)

```
df_1 <- coef(best_subset, id = 16) %>%
  as.data.frame()
names(df_1) <- c("model_beta")
df_1$predictor <- rownames(df_1)
rownames(df_1) <- NULL
df_0 <- data.frame("model_beta" = c(0, 0, 0, 0),
           "predictor" = c("p2", "p5", "p10", "p15"))
df_1 <- rbind(df_1, df_0) %>%
  filter(predictor != "(Intercept)") %>%
  mutate(predictor_num = as.integer(substring(predictor, 2, last = 1000000L))) %>%
  arrange(predictor_num)
df_1$actual_beta <- beta

df_1 %>%
```

```
  gather(type, beta, actual_beta, model_beta) %>%
  ggplot(aes(factor(predictor_num), beta, shape = type)) +
  geom_point(size = 4, alpha = 0.5) + #aes(colour = type),
  geom_point(colour = "white", size = 1.5) +
  #geom_point(aes(shape = type), alpha = 0.3, size = 3, fill = "white", color = "black") + #
  #scale_color_brewer(type = "qual") +
  geom_hline(yintercept = 0, linetype = 2) +
  coord_flip() +
  labs(x = "Predictor",
       y = "Estimated coefficient",
       color = "Type") +
  theme(legend.position = "bottom")
```




```
df_1 %>%
  select(predictor, model_beta, actual_beta) %>%
  kable(caption = "Betas from best model vs from actual data")
```

Table 2: Betas from best model vs from actual data

predictor	model_beta	actual_beta
p1	-19.076	-19.312
p2	0.000	0.000
p3	12.478	12.530
p4	-4.780	-4.798
p5	0.000	0.000
p6	4.111	4.095
p7	4.367	4.388

predictor	model_beta	actual_beta
p8	20.240	20.242
p9	12.114	12.108
p10	0.000	0.000
p11	9.932	9.942
p12	-27.393	-27.396
p13	27.726	27.725
p14	3.982	3.983
p15	0.000	0.000
p16	-3.254	-3.255
p17	26.989	26.999
p18	-21.582	-21.581
p19	4.476	4.476
p20	2.414	2.416

From the table and the plot, we can see that for the best model (where p = 16 amd with lowest test MSE), there are very little differences between the actual and the estimated betas. This further confirms that this best model can capture the true structure of the simulated data pretty well.

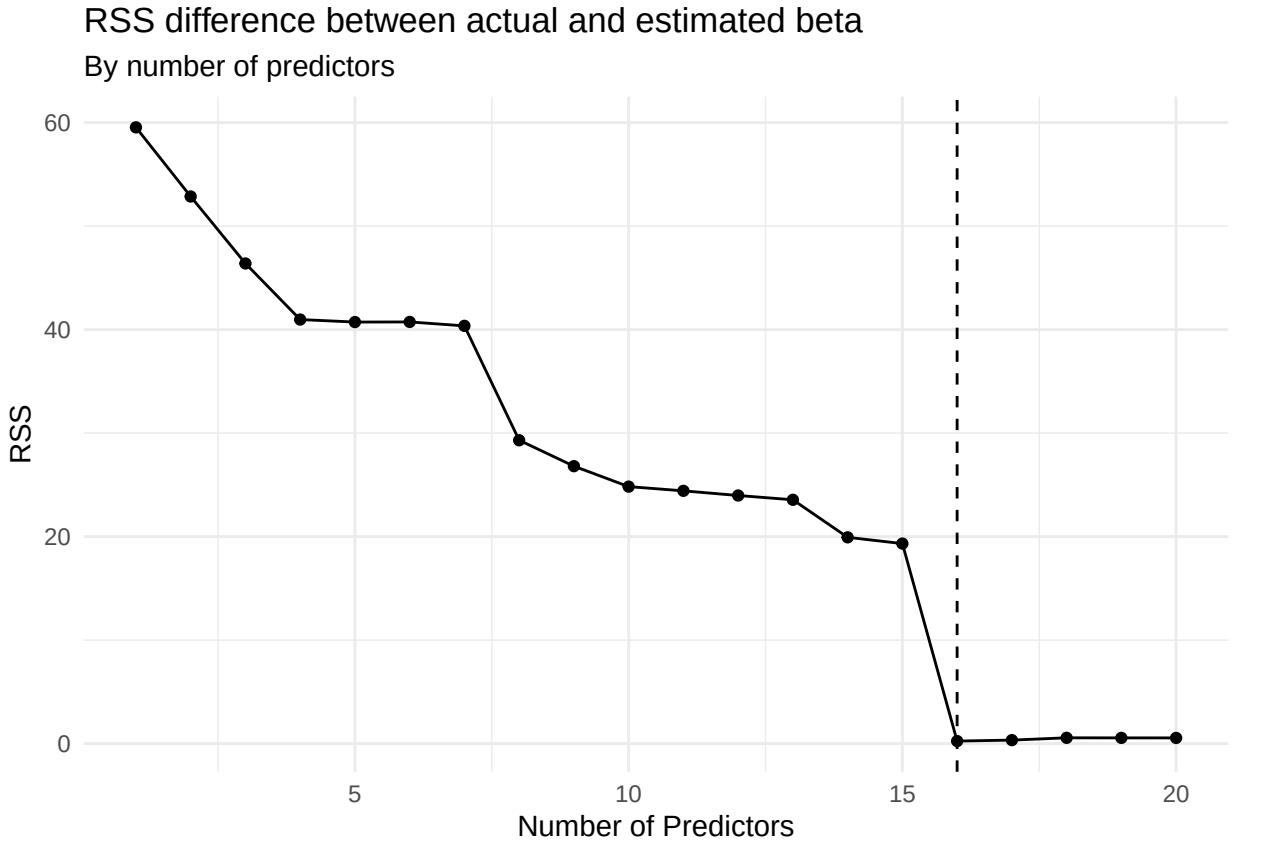
7)

Note that I simulate 2 different plots based on different interpretation of p as a constant (p = 20) or p as the size of current model (p = r)

7.1) p as a constant (p = 20)

```
# get all beta vectors from all model with 1 <= p <= 20
df_beta <- map(1:20, ~ coef(best_subset, id = .x)) %>%
  set_names(1:20)
predictors <- map(1:20, ~ paste0("p", .x)) %>% unlist()
# dataframe of all predictor betas set to 0
zero_df <- data.frame(predictor = predictors, beta = rep.int(0, 20)) %>%
    mutate(predictor = as.character(predictor))
# dataframe of actual betas
actual_beta_df <- data.frame(actual_beta = beta, predictor = predictors)
# function to calculate the RSS between actual and estimated beta
# for one model with p = i
cal_bse <- function(i){
  # get beta vector for model with  p = i
  df_beta <- df_beta[[i]] %>%
    as.data.frame()
  # tidy the beta dataframe
  df_beta$predictor <- rownames(df_beta)
  names(df_beta) <- c("beta", "predictor")
  rownames(df_beta) <- NULL
  # remove intercept
  df_beta <- df_beta %>%
    filter(predictor != "(Intercept)")
  # fill non-present predictors with beta = 0
  beta_df <- rbind(zero_df, df_beta) %>%
    group_by(predictor) %>%
    summarise(beta = sum(beta))
```

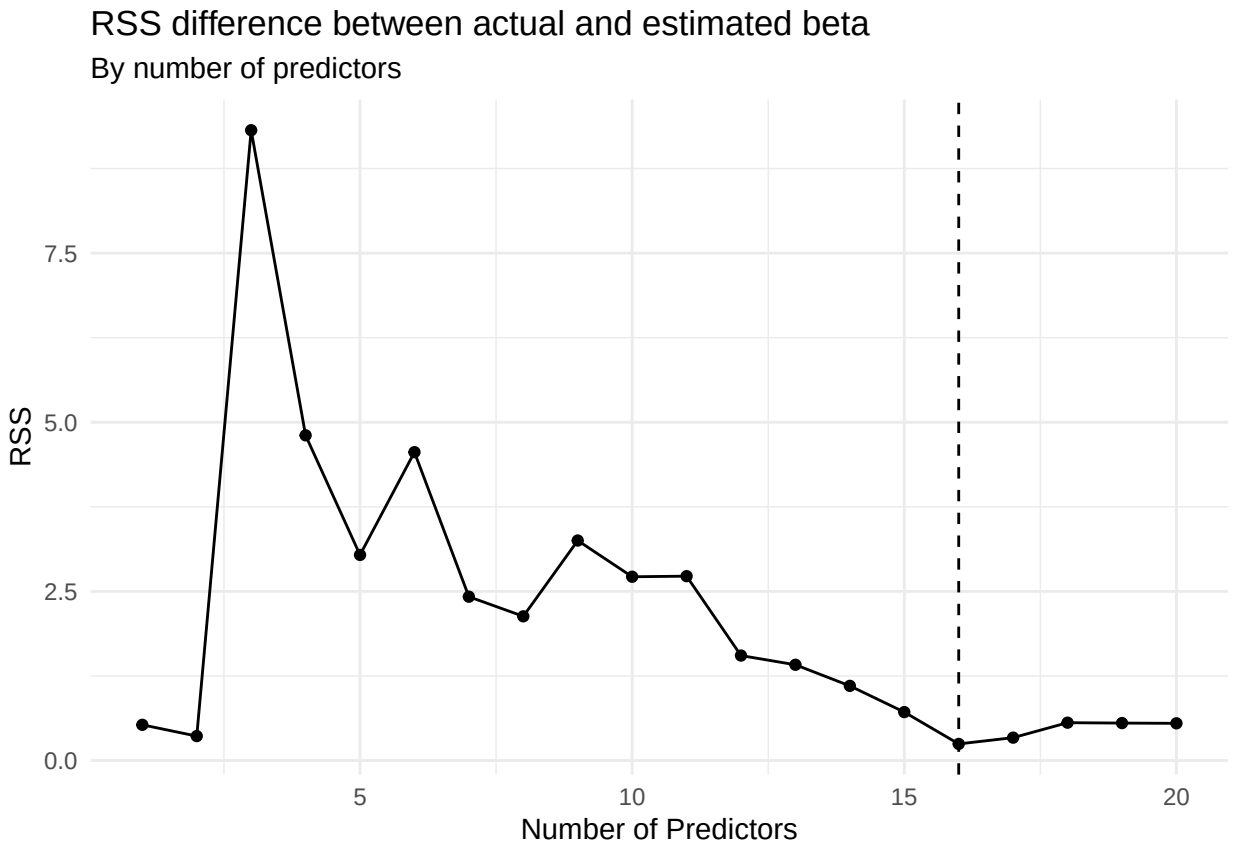
```r
  # calculate (true beta - estimate beta) ^ 2 for each predictor
  diff_beta_df <- full_join(actual_beta_df, beta_df) %>%
    mutate(diff_beta = (actual_beta - beta)^2)
  # calculate the RSS and RMS
  return (sum(diff_beta_df$diff_beta)^0.5)
}
# calculate all RSSs for all models
bse <- map(1:20, ~cal_bse(.x)) %>%
  set_names(1:20) %>%
  as.data.frame() %>%
  gather(num_pred, bse)
bse$num_pred <- c(1:20)
# plot
bse %>%
  ggplot(aes(num_pred, bse)) +
  geom_point() +
  geom_line() +
  geom_vline(xintercept = which.min(bse$bse), linetype = 2) +
  labs(x = "Number of Predictors",
       y = "RSS",
       title = "RSS difference between actual and estimated beta",
       subtitle = "By number of predictors")
```



In this case, the RSS of difference between actual and estimated beta reflects the test MSE trend! This make very much sense as the lower the test MSE, the better the model captures the true beta.

7.2) p as the size of current model (p = r, 1 <= r <= 20)

```r
# get all beta vectors from all model with 1 <= p <= 20
df_beta <- map(1:20, ~ coef(best_subset, id = .x)) %>%
  set_names(1:20)
predictors <- map(1:20, ~ paste0("p", .x)) %>% unlist()
# dataframe of all predictor betas set to 0
zero_df <- data.frame(predictor = predictors, beta = rep.int(0, 20)) %>%
    mutate(predictor = as.character(predictor))
# dataframe of actual betas
actual_beta_df <- data.frame(actual_beta = beta, predictor = predictors)
# function to calculate the RSS between actual and estimated beta
# for one model with p = i
cal_bse <- function(i){
  # get beta vector for model with  p = i
  df_beta <- df_beta[[i]] %>%
    as.data.frame()
  # tidy the beta dataframe
  df_beta$predictor <- rownames(df_beta)
  names(df_beta) <- c("beta", "predictor")
  rownames(df_beta) <- NULL
  # remove intercept
  df_beta <- df_beta %>%
    filter(predictor != "(Intercept)")
  # fill non-present predictors with beta = 0
  beta_df <- rbind(zero_df, df_beta) %>%
    group_by(predictor) %>%
    summarise(beta = sum(beta))
  # calculate (true beta - estimate beta) ^ 2 for each predictor
  diff_beta_df <- full_join(actual_beta_df, beta_df) %>%
    filter(beta != 0) %>%
    mutate(diff_beta = (actual_beta - beta)^2)
  # calculate the RSS and RMS
  return (sum(diff_beta_df$diff_beta)^0.5)
}
# calculate all RSSs for all models
bse <- map(1:20, ~cal_bse(.x)) %>%
  set_names(1:20) %>%
  as.data.frame() %>%
  gather(num_pred, bse)
bse$num_pred <- c(1:20)
# plot
bse %>%
  ggplot(aes(num_pred, bse)) +
  geom_point() +
  geom_line() +
  geom_vline(xintercept = which.min(bse$bse), linetype = 2) +
  labs(x = "Number of Predictors",
       y = "RSS",
       title = "RSS difference between actual and estimated beta",
       subtitle = "By number of predictors")
```



In this case, the pattern of the RSS between actual and estimated beta has a different trend from the test MSE. However, we still see the dip at p = 16 where test MSE is at minimum. The increase in RSS during the immediate p is likely due to OLS trying to use the available p predictors to account for all the variance in the data but cannot do so well because the number of p is less than the true number p (in this case true p = 16).

Application exercises

```
set.seed(232)
gss_train <- read_csv("data/gss_train.csv")
gss_test <- read_csv("data/gss_test.csv")
# use the same cv splits throughout all regularizations
fold_id <- sample(1:10, size = nrow(gss_train), replace = TRUE)
traintest <- rbind(gss_train, gss_test)
X <- model.matrix(egalit_scale ~., data = traintest)[, -1]
```

1)

```
# linear model
lm_model <- lm(egalit_scale ~ ., data = gss_train)
lm_pred <- predict(lm_model, gss_test)
# linear test mse
(mse_lm <- mse_vec(truth = gss_test$egalit_scale, estimate = lm_pred))
```

```
## [1] 63.21
```

2)

```
# train ridge model (alpha = 0)
ridge_model <- cv.glmnet(X[1:nrow(gss_train),],
                         gss_train$egalit_scale,
                         alpha = 0,
                         foldid = fold_id)
ridge_pred <- predict(ridge_model, s='lambda.min', newx=X[-(1:nrow(gss_train)),])
# ridge test mse
(mse_ridge <- mse_vec(truth = gss_test$egalit_scale, estimate = as.vector(ridge_pred)))
```

```
## [1] 61.04
```

3)

```
# train lasso model (alpha = 1)
lasso_model <- cv.glmnet(X[1:nrow(gss_train),],
                         gss_train$egalit_scale,
                         alpha = 1,
                         foldid = fold_id)
lasso_pred <- predict(lasso_model, s='lambda.min', newx=X[-(1:nrow(gss_train)),])
# lasso test mse
(mse_lasso <- mse_vec(truth = gss_test$egalit_scale, estimate = as.vector(lasso_pred)))
```

```
## [1] 61.22
```

```
# number of non-zero predictors
lasso <- coef(lasso_model, s=lasso_model$lambda.min) %>%
  as.matrix() %>%
  as.data.frame()
names(lasso) <- c("value")
lasso %>%
   filter(value != 0) %>%
  nrow() - 1 # minus intercept
```

```
## [1] 29
```

4)

```
# search across a range of alphas
tuning_grid <- tibble::tibble(
  alpha      = seq(0, 1, by = .1),
  mse_min    = NA,
  mse_1se    = NA,
  lambda_min = NA,
  lambda_1se = NA
)

for(i in seq_along(tuning_grid$alpha)) {
  # fit CV model for each alpha value
```

```
  fit <- cv.glmnet(X[1:nrow(gss_train),],
                   gss_train$egalit_scale,
                   alpha = tuning_grid$alpha[i],
                   foldid = fold_id)

  # extract train MSE and lambda values
  tuning_grid$mse_min[i]    <- fit$cvm[fit$lambda == fit$lambda.min]
  tuning_grid$mse_1se[i]    <- fit$cvm[fit$lambda == fit$lambda.1se]
  tuning_grid$lambda_min[i] <- fit$lambda.min
  tuning_grid$lambda_1se[i] <- fit$lambda.1se
}
# find the top combinations of alpha and lambda by train mse
elas_all <- tuning_grid %>%
  arrange(mse_min)
elas_all %>% head() %>% kable(caption = "Top alpha and lambda combinations")
```

Table 3: Top alpha and lambda combinations

alpha	mse_min	mse_1se	lambda_min	lambda_1se
0.7	60.2	61.06	0.3040	0.5830
0.8	60.2	61.05	0.2660	0.5101
0.9	60.2	61.04	0.2364	0.4534
1.0	60.2	61.03	0.2128	0.4081
0.6	60.2	61.08	0.3546	0.6802
0.5	60.2	61.11	0.4256	0.8162

```
# train best elastic model with best alpha
best_elas <- cv.glmnet(X[1:nrow(gss_train),],
                   gss_train$egalit_scale,
                   alpha = elas_all[1,]$alpha,
                   foldid = fold_id)
elas_pred <- predict(best_elas, s='lambda.min', newx=X[-(1:nrow(gss_train)),])
# elastic test mse
(mse_elas <- mse_vec(truth = gss_test$egalit_scale, estimate = as.vector(elas_pred)))
```

```
## [1] 61.19
```

```
# number of non-zero predictors
elas <- coef(lasso_model, s=lasso_model$lambda.min) %>%
  as.matrix() %>%
  as.data.frame()
names(elas) <- c("value")
elas %>%
   filter(value != 0) %>%
  nrow() - 1 # minus intercept
```

```
## [1] 29
```

5)

From the results, we can see that all regularizations improve the test MSE slightly from the linear regression without regularization. This is expected because the regularization will make the resulted model less overfit

to the training data. The regularization with the lowest test MSE is the ridge regression, followed by elastic and lasso regressions repetively. Theoritically, elastic regression should give us the best regulation because we search from alpha = 0 (ridge) to 1 (lasso). However, since the alpha is chosen by train MSE, elastic regression does not guarantee the best test MSE. This suggests that we might need a validation set on top of just training and testing set too. So that for the alpha tuning for the elastic regression, we can choose the best alpha-lambda combination based on the validation MSE instead of the train MSE.