STA9890: Regression

Jonathan Ng

```r
library(tidyverse)
library(gridExtra)
library(reshape2)
library(stringr)
library(FSA)
library(ISLR)
library(glmnet)
library(randomForest)
library(kableExtra)
library(plotly)
library(rbenchmark)

set.seed(1)
data <- read_csv("OnlineNewsPopularity.csv") %>%
  select(-url, -timedelta) %>%
  sample_n(5000)
```

1. Mashable Online News Popularity

https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity

(a) Describe the response variable and the predictors. How was the data collected?

This dataset from *Mashable* summarizes sets of features about articles published on their website www.mashable.com. The goal is to predict response variable, the number of shares in social networks for a given article, which is an indicator of popularity.

The data was collected solely from articles published on www.mashable.com for a two year period. The data does not contain the actual content of articles, but rather various summary statistics and metadata extracted from the published articles.

The Response Variable: `shares`

The Predictor Variables:

1. n_tokens_title: Number of words in the title
2. n_tokens_content: Number of words in the content
3. n_unique_tokens: Rate of unique words in the content
4. n_non_stop_words: Rate of non-stop words in the content
5. n_non_stop_unique_tokens: Rate of unique non-stop words in the content
6. num_hrefs: Number of links
7. num_self_hrefs: Number of links to other articles published by Mashable
8. num_imgs: Number of images

9. num_videos: Number of videos
10. average_token_length: Average length of the words in the content
11. num_keywords: Number of keywords in the metadata
12. data_channel_is_lifestyle: Is data channel ‘Lifestyle’?
13. data_channel_is_elnettertainment: Is data channel ‘Entertainment’?
14. data_channel_is_bus: Is data channel ‘Business’?
15. data_channel_is_socmed: Is data channel ‘Social Media’?
16. data_channel_is_tech: Is data channel ‘Tech’?
17. data_channel_is_world: Is data channel ‘World’?
18. kw_min_min: Worst keyword (min. shares)
19. kw_max_min: Worst keyword (max. shares)
20. kw_avg_min: Worst keyword (avg. shares)
21. kw_min_max: Best keyword (min. shares)
22. kw_max_max: Best keyword (max. shares)
23. kw_avg_max: Best keyword (avg. shares)
24. kw_min_avg: Avg. keyword (min. shares)
25. kw_max_avg: Avg. keyword (max. shares)
26. kw_avg_avg: Avg. keyword (avg. shares)
27. self_reference_min_shares: Min. shares of referenced articles in Mashable
28. self_reference_max_shares: Max. shares of referenced articles in Mashable
29. self_reference_avg_sharess: Avg. shares of referenced articles in Mashable
30. weekday_is_monday: Was the article published on a Monday?
31. weekday_is_tuesday: Was the article published on a Tuesday?
32. weekday_is_wednesday: Was the article published on a Wednesday?
33. weekday_is_thursday: Was the article published on a Thursday?
34. weekday_is_friday: Was the article published on a Friday?
35. weekday_is_saturday: Was the article published on a Saturday?
36. weekday_is_sunday: Was the article published on a Sunday?
37. is_weekend: Was the article published on the weekend?
38. LDA_00: Closeness to LDA topic 0
39. LDA_01: Closeness to LDA topic 1
40. LDA_02: Closeness to LDA topic 2
41. LDA_03: Closeness to LDA topic 3
42. LDA_04: Closeness to LDA topic 4
43. global_subjectivity: Text subjectivity
44. global_sentiment_polarity: Text sentiment polarity
45. global_rate_positive_words: Rate of positive words in the content
46. global_rate_negative_words: Rate of negative words in the content
47. rate_positive_words: Rate of positive words among non-neutral tokens
48. rate_negative_words: Rate of negative words among non-neutral tokens
49. avg_positive_polarity: Avg. polarity of positive words
50. min_positive_polarity: Min. polarity of positive words
51. max_positive_polarity: Max. polarity of positive words
52. avg_negative_polarity: Avg. polarity of negative words
53. min_negative_polarity: Min. polarity of negative words
54. max_negative_polarity: Max. polarity of negative words
55. title_subjectivity: Title subjectivity
56. title_sentiment_polarity: Title polarity
57. abs_title_subjectivity: Absolute subjectivity level
58. abs_title_sentiment_polarity: Absolute polarity level

Impute missing data-points with their mean. What is n and p?

The data does not contain missing values, so imputation is not required.

$$n = 39644$$
$$p = 58$$

Standardize the numerical predictors using equation (6.6) in the ISLR book.

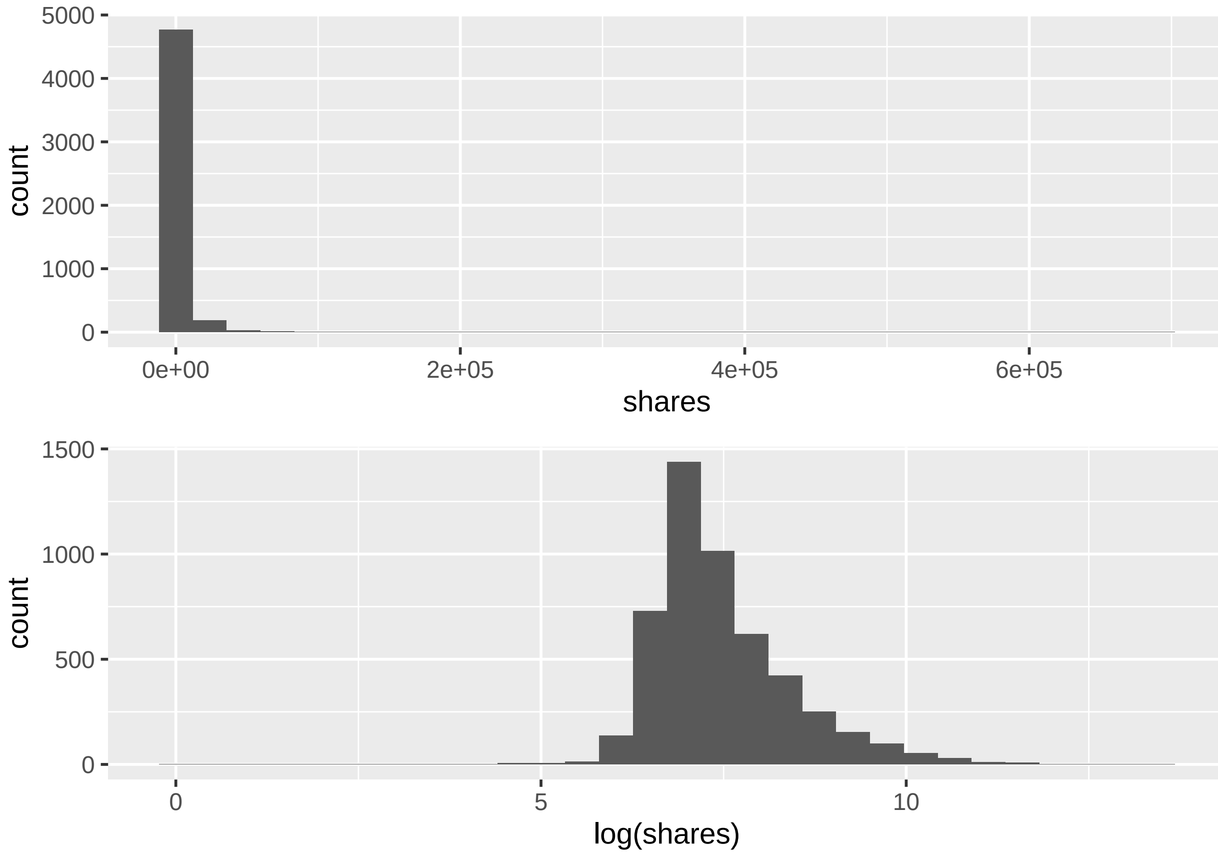
Equation 6.6:

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

```r
data <- data %>%
  select(-shares) %>%
  mutate_all(function(x) {x / sd(x)}) %>%
  mutate(shares = data$shares)
```

Distribution of y

```r
hist_y <- data %>%
  ggplot(aes(x=shares))+
  geom_histogram(bins = 30)

hist_log_y <- data %>%
  ggplot(aes(x=log(shares)))+
  geom_histogram(bins = 30)

grid.arrange(hist_y, hist_log_y, nrow=2)
```




```r
data <- data %>%
  mutate(shares=log(shares))

data %>%
  glimpse()
```

```
## Observations: 5,000
## Variables: 59
## $ n_tokens_title                <dbl> 3.757529, 5.166603, 4.696912, 3.75752...
## $ n_tokens_content              <dbl> 0.6308615, 0.3724882, 1.3198571, 0.78...
## $ n_unique_tokens               <dbl> 4.505415, 4.780077, 3.230670, 4.25455...
## $ n_non_stop_words              <dbl> 6.213983, 6.213983, 6.213983, 6.21398...
## $ n_non_stop_unique_tokens      <dbl> 4.985482, 5.522474, 4.253439, 4.51426...
## $ num_hrefs                     <dbl> 0.4520988, 0.5425185, 0.5425185, 0.27...
## $ num_self_hrefs                <dbl> 0.2619090, 0.7857269, 0.7857269, 0.78...
## $ num_imgs                      <dbl> 0.1250814, 0.1250814, 0.1250814, 1.25...
## $ num_videos                    <dbl> 0.239961, 0.000000, 0.000000, 0.00000...
## $ average_token_length          <dbl> 6.468896, 5.413295, 5.417059, 6.30705...
## $ num_keywords                  <dbl> 3.184113, 4.776170, 4.776170, 3.71479...
## $ data_channel_is_lifestyle     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ data_channel_is_entertainment <dbl> 0.000000, 0.000000, 0.000000, 2.59590...
## $ data_channel_is_bus           <dbl> 2.771672, 2.771672, 0.000000, 0.00000...
## $ data_channel_is_socmed        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ data_channel_is_tech          <dbl> 0.000000, 0.000000, 2.611734, 0.00000...
## $ data_channel_is_world         <dbl> 0.00000, 0.00000, 0.00000, 0.00000, 2...
## $ kw_min_min                    <dbl> -0.01450154, 3.14683521, 0.05800618, ...
## $ kw_max_min                    <dbl> 0.10223094, 0.22377504, 0.07184492, 0...
```

```
## $ kw_avg_min                   <dbl> 0.24407628, 0.62642821, 0.16799824, 0...
## $ kw_min_max                   <dbl> 0.1424320, 0.0000000, 0.0000000, 0.29...
## $ kw_max_max                   <dbl> 3.963373, 2.904029, 3.963373, 3.96337...
## $ kw_avg_max                   <dbl> 2.8477590, 0.7746449, 1.3885462, 1.98...
## $ kw_min_avg                   <dbl> 1.4141508, 0.0000000, 0.0000000, 1.17...
## $ kw_max_avg                   <dbl> 0.5822269, 0.8806021, 0.6095905, 0.94...
## $ kw_avg_avg                   <dbl> 1.928369, 2.099917, 1.938905, 2.80282...
## $ self_reference_min_shares    <dbl> 0.55256923, 0.17047349, 0.09993273, 0...
## $ self_reference_max_shares    <dbl> 0.23231072, 0.14086927, 0.12356953, 0...
## $ self_reference_avg_sharess   <dbl> 0.43392180, 0.18157011, 0.12925330, 0...
## $ weekday_is_monday            <dbl> 0.000000, 0.000000, 0.000000, 0.00000...
## $ weekday_is_tuesday           <dbl> 0.00000, 2.59926, 0.00000, 0.00000, 0...
## $ weekday_is_wednesday         <dbl> 2.544717, 0.000000, 0.000000, 2.54471...
## $ weekday_is_thursday          <dbl> 0.000000, 0.000000, 2.576164, 0.00000...
## $ weekday_is_friday            <dbl> 0.000000, 0.000000, 0.000000, 0.00000...
## $ weekday_is_saturday          <dbl> 0.00000, 0.00000, 0.00000, 0.00000, 0...
## $ weekday_is_sunday            <dbl> 0.000000, 0.000000, 0.000000, 0.00000...
## $ is_weekend                   <dbl> 0.000000, 0.000000, 0.000000, 0.00000...
## $ LDA_00                       <dbl> 2.97415241, 0.96870559, 0.50493613, 0...
## $ LDA_01                       <dbl> 0.14810054, 0.60831183, 0.09873790, 2...
## $ LDA_02                       <dbl> 0.11703301, 0.07783261, 0.07782465, 0...
## $ LDA_03                       <dbl> 0.11417523, 0.07611679, 0.07720218, 0...
## $ LDA_04                       <dbl> 0.40764143, 1.97650377, 2.80658497, 0...
## $ global_subjectivity          <dbl> 2.798987, 4.148770, 3.839035, 3.89829...
## $ global_sentiment_polarity    <dbl> 2.3225839, 2.2398209, 2.5514693, 1.67...
## $ global_rate_positive_words   <dbl> 2.1917583, 3.0371344, 2.7618820, 2.55...
## $ global_rate_negative_words   <dbl> 0.3211055, 0.0000000, 0.1534811, 1.80...
## $ rate_positive_words          <dbl> 4.887703, 5.332040, 5.154305, 3.70924...
## $ rate_negative_words          <dbl> 0.5307563, 0.0000000, 0.2123025, 1.93...
## $ avg_positive_polarity        <dbl> 3.382688, 2.892794, 3.987211, 3.75457...
## $ min_positive_polarity        <dbl> 0.4613762, 0.6920643, 0.4613762, 0.46...
## $ max_positive_polarity        <dbl> 2.057821, 3.292513, 3.498295, 3.29251...
## $ avg_negative_polarity        <dbl> -0.9819048, 0.0000000, -1.5710477, -2...
## $ min_negative_polarity        <dbl> -0.4286202, 0.0000000, -0.6857923, -1...
## $ max_negative_polarity        <dbl> -1.3309854, 0.0000000, -2.1295767, -1...
## $ title_subjectivity           <dbl> 0.0000000, 1.9620591, 1.3873145, 0.00...
## $ title_sentiment_polarity     <dbl> 0.0000000, 0.8109844, 0.5160810, 0.00...
## $ abs_title_subjectivity       <dbl> 2.6523409, 0.7578117, 0.2411219, 2.65...
## $ abs_title_sentiment_polarity <dbl> 0.0000000, 0.9503838, 0.6047897, 0.00...
## $ shares                       <dbl> 7.313220, 8.556414, 8.342840, 6.53087...
```

For each $n_{train} = 0.8n$, repeat the following 100 times, do the following for the different models mentioned below.

(a) Randomly split the dataset into two mutually exclusive datasets D_{test} and D_{train} with size n_{test} and n_{train} such that $n_{train} + n_{test} = n$.

(b) Use D_{train} to fit lasso, elastic-net $\alpha = 0.5$, ridge, and random forrests.

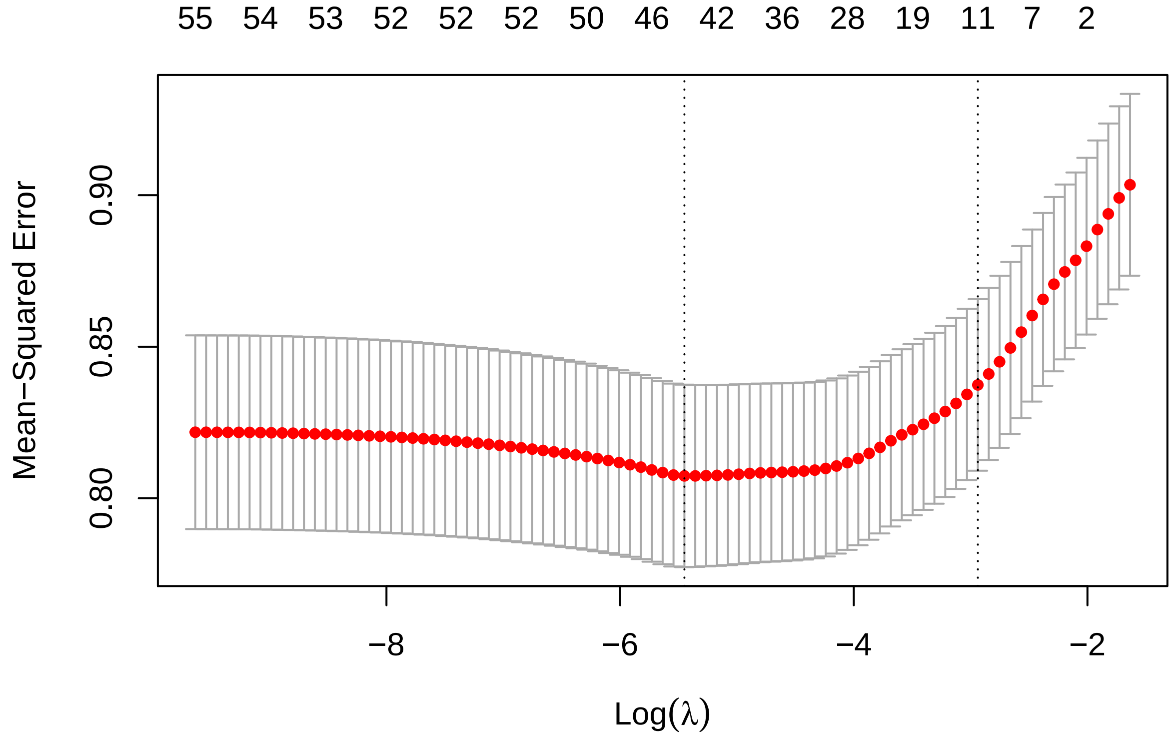
(c) Tune the λs using 10-fold CV.

(d) For each estimated model calculate

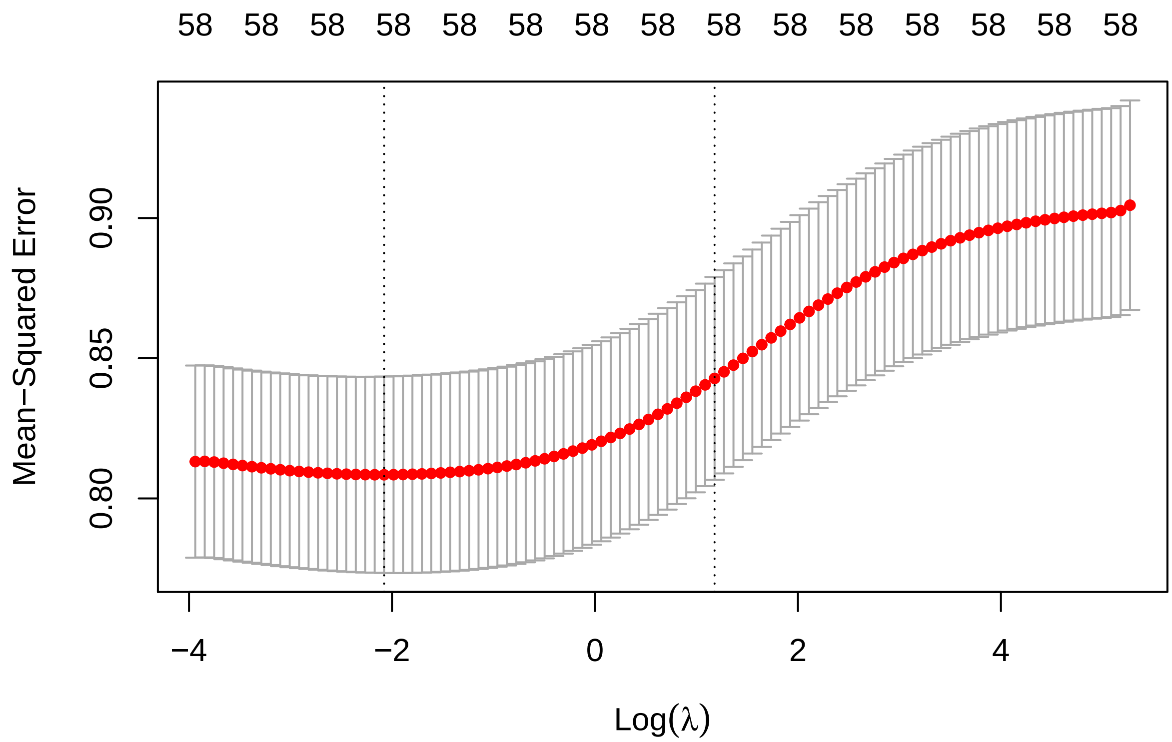
$$R^2_{test} = 1 - \frac{\frac{1}{n_{test}} \sum_{i \in D_{test}} (y_i - \hat{y}_i)^2}{\frac{1}{n} \sum_{i=1}^{n} (y_i - \bar{y})^2}$$

```r
n <- 5000
p <- 58

sample_and_train <- function(plots=F) {
  # (a)
  n_train <- as.integer(0.8 * n)
  n_test = n - n_train
  train_inds <- sample.int(n, n_train)
  D_train <- data[train_inds, ]
  D_test <- data[-train_inds, ]
  # (b) & (c)
  X_train <- select(D_train, -shares) %>% data.matrix()
  X_test <- select(D_test, -shares) %>% data.matrix()
  y_train <- D_train$shares
  y_test <- D_test$shares
  y <- data$shares
  # (d)
  ## lasso
  lasso_time_start <- Sys.time()
  lasso_cv <- cv.glmnet(X_train, y_train, alpha = 1)
  lasso_fit <- glmnet(X_train, y_train, alpha = 1, lambda = lasso_cv$lambda.min)
  lasso_y_train_hat <- predict(lasso_fit, X_train)
  lasso_y_test_hat <- predict(lasso_fit, X_test)
  lasso_resid_train <- as.vector(y_train - lasso_y_train_hat)
  lasso_resid_test <- as.vector(y_test - lasso_y_test_hat)
  lasso_Rsq_train <- 1 - mean((lasso_resid_train)^2) / mean((y - mean(y))^2)
  lasso_Rsq_test <- 1 - mean((lasso_resid_test)^2) / mean((y - mean(y))^2)
  lasso_time <- Sys.time() - lasso_time_start

  ## ridge
  ridge_time_start <- Sys.time()
  ridge_cv <- cv.glmnet(X_train, y_train, alpha = 0)
  ridge_fit <- glmnet(X_train, y_train, alpha = 0, lambda = ridge_cv$lambda.min)
  ridge_y_train_hat <- predict(ridge_fit, X_train)
  ridge_y_test_hat <- predict(ridge_fit, X_test)
  ridge_resid_train <- as.vector(y_train - ridge_y_train_hat)
  ridge_resid_test <- as.vector(y_test - ridge_y_test_hat)
  ridge_Rsq_train <- 1 - mean((ridge_resid_train)^2) / mean((y - mean(y))^2)
  ridge_Rsq_test <- 1 - mean((ridge_resid_test)^2) / mean((y - mean(y))^2)
  ridge_time <- Sys.time() - ridge_time_start

  ## elastic-net
  elnet_time_start <- Sys.time()
  elnet_cv <- cv.glmnet(X_train, y_train, alpha = 0.5)
  elnet_fit <- glmnet(X_train, y_train, alpha = 0.5, lambda = elnet_cv$lambda.min)
  elnet_y_train_hat <- predict(elnet_fit, X_train)
  elnet_y_test_hat <- predict(elnet_fit, X_test)
  elnet_resid_train <- as.vector(y_train - elnet_y_train_hat)
  elnet_resid_test <- as.vector(y_test - elnet_y_test_hat)
  elnet_Rsq_train <- 1 - mean((elnet_resid_train)^2) / mean((y - mean(y))^2)
  elnet_Rsq_test <- 1 - mean((elnet_resid_test)^2) / mean((y - mean(y))^2)
  elnet_time <- Sys.time() - elnet_time_start
```

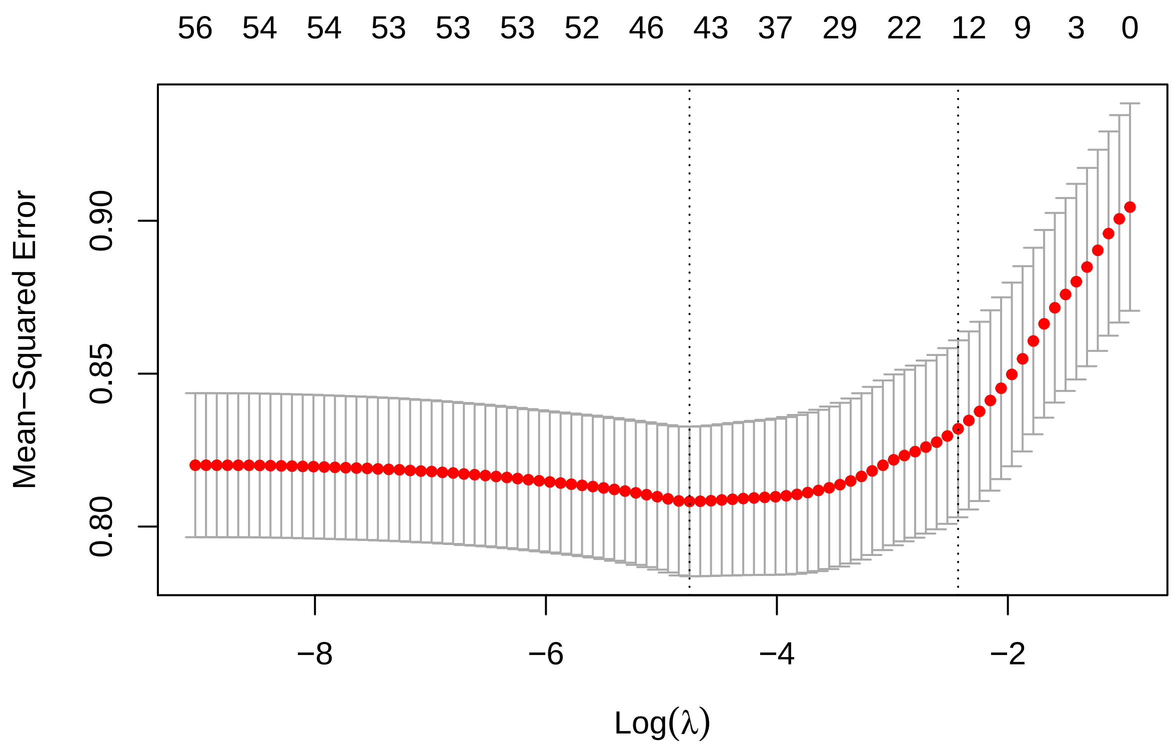
```r
  ## random forest
  rf_time_start <- Sys.time()
  rf_fit <-randomForest(X_train, y_train, mtry = sqrt(p), importance = T)
  rf_y_train_hat <- predict(rf_fit, X_train)
  rf_y_test_hat <- predict(rf_fit, X_test)
  rf_resid_train <- y_train - rf_y_train_hat
  rf_resid_test <- y_test - rf_y_test_hat
  rf_Rsq_train <- 1 - mean((rf_resid_train)^2) / mean((y - mean(y))^2)
  rf_Rsq_test <- 1 - mean((rf_resid_test)^2) / mean((y - mean(y))^2)
  rf_time <- Sys.time() - rf_time_start

  if(plots) {
    ## 10 fold CV Plots
    plot(lasso_cv, sub = paste("Lasso:", lasso_cv$lambda.min))
    plot(ridge_cv, sub = paste("Ridge", ridge_cv$lambda.min))
    plot(elnet_cv, sub = paste("Elastic Net:", elnet_cv$lambda.min))

    ## Residuals Boxplots
    resid_train <- data.frame(lasso = lasso_resid_train, ridge = ridge_resid_train,
                              elnet = elnet_resid_train, rf = rf_resid_train, dataset="train")
    resid_test <- data.frame(lasso = lasso_resid_test, ridge = ridge_resid_test,
                             elnet = elnet_resid_test, rf = rf_resid_test, dataset="test")
    resid_models <- rbind(resid_train, resid_test)
    resid_plot <- resid_models %>%
      gather(model, residuals, lasso:rf) %>%
      ggplot(aes(x=model, y=residuals, fill=model)) +
      geom_boxplot() +
      facet_wrap(~dataset)

    # resid_plot < ggplotly(resid_plot) # Un-comment for ggplotly plot
    print(resid_plot)
  }

  Rsq_train <- list(lasso = lasso_Rsq_train, ridge = ridge_Rsq_train,
                    elnet = elnet_Rsq_train, rf = rf_Rsq_train)
  Rsq_test <- list(lasso = lasso_Rsq_test, ridge = ridge_Rsq_test,
                   elnet = elnet_Rsq_test, rf = rf_Rsq_test)
  times <- list(lasso = lasso_time, ridge = ridge_time,
                elnet = elnet_time, rf = rf_time)

  return(list(Rsq_train, Rsq_test, times))
}

set.seed(2)

one_sample <- sample_and_train(plots = T)
```

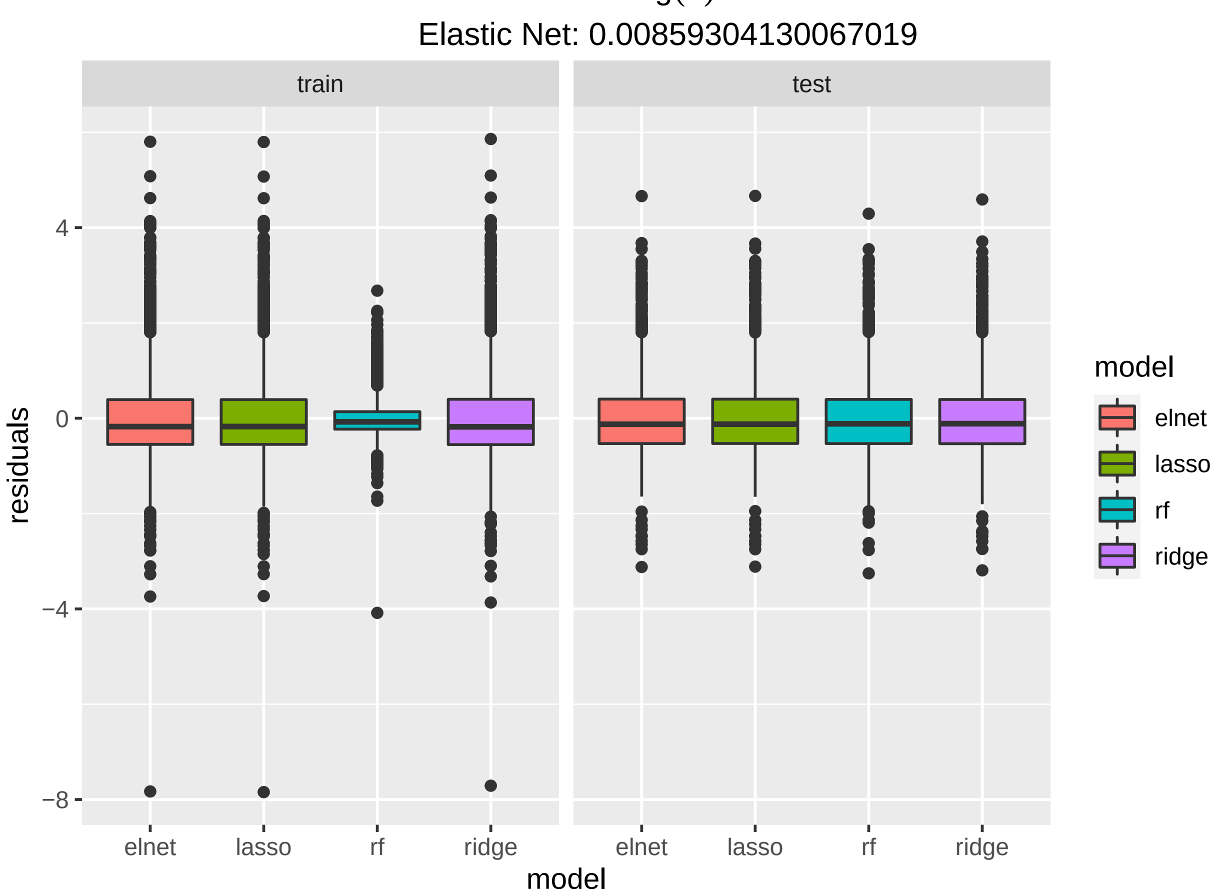


Lasso: 0.0042965206503351



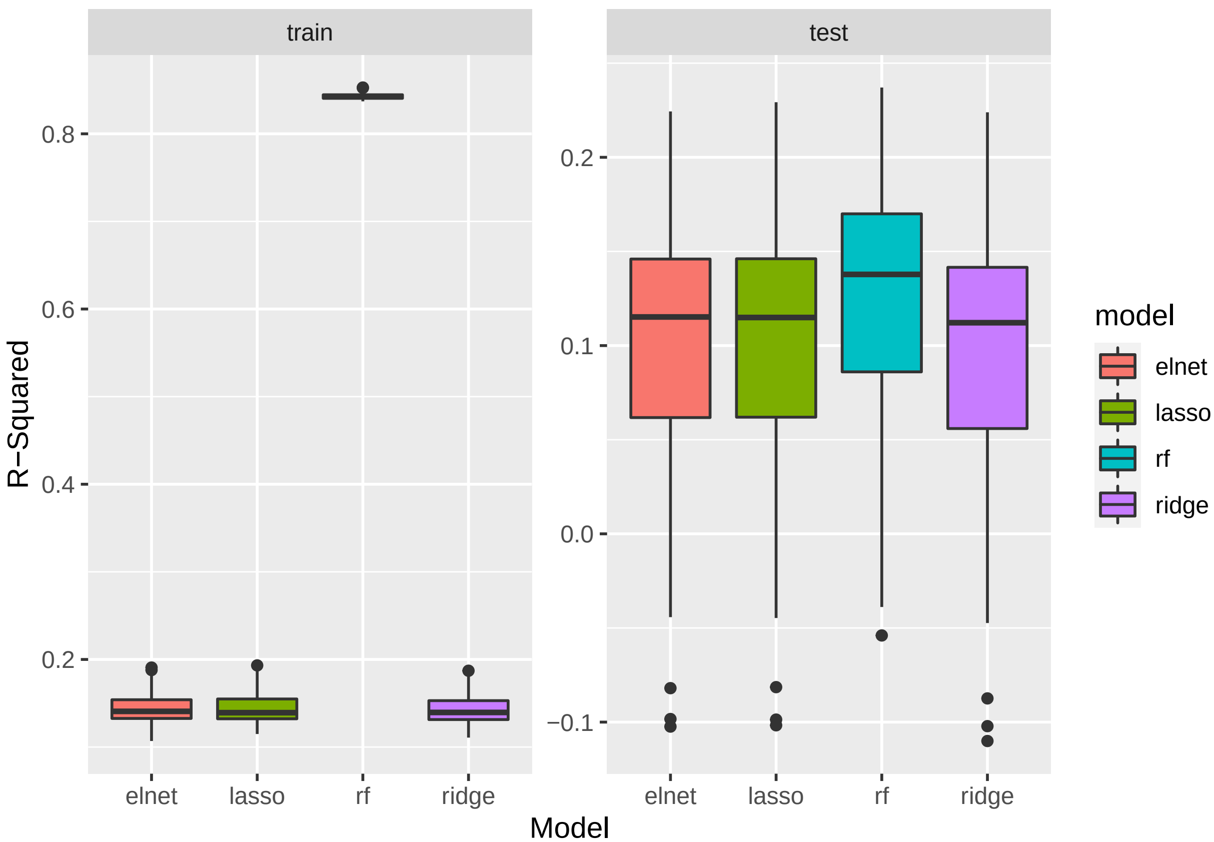
Ridge 0.12524585722488






```r
set.seed(2)
M <- 100
Rsq_time_models <- replicate(M, unlist(sample_and_train(plots = F))) %>% t()
```

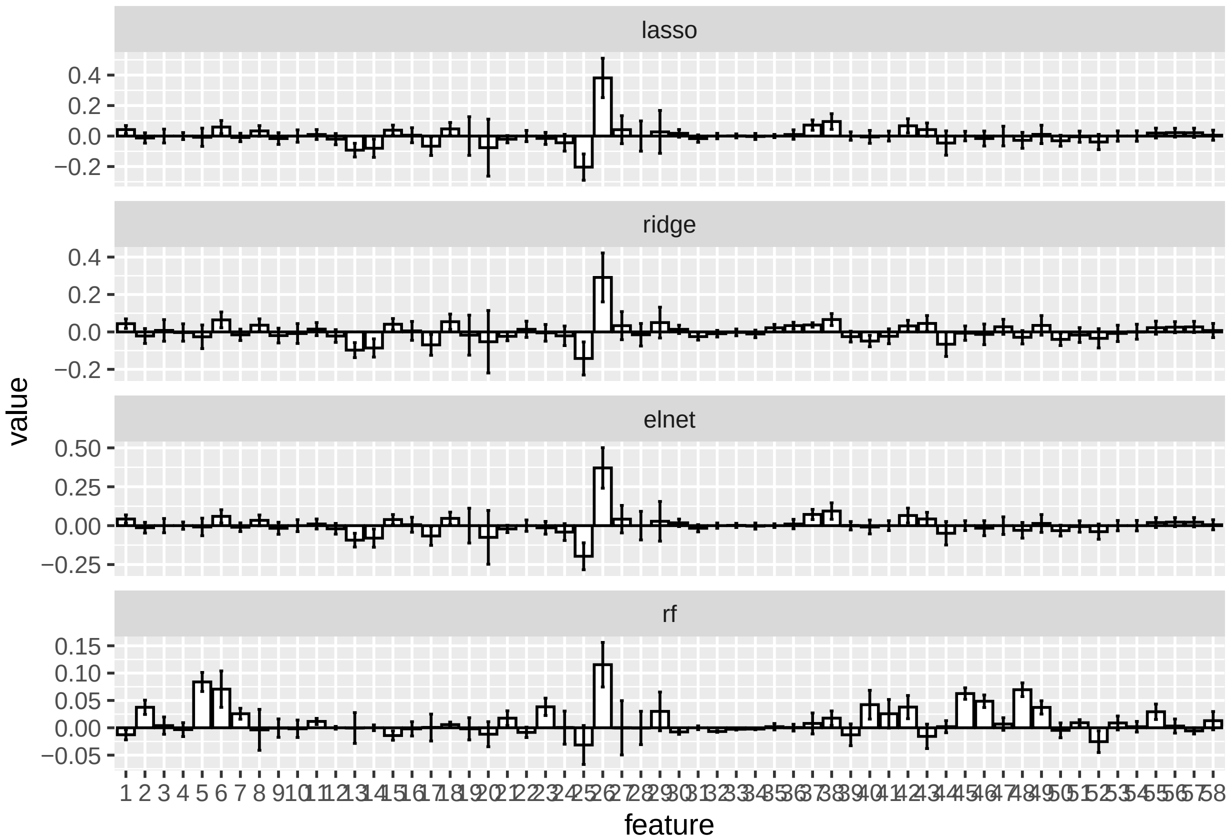
```r
Rsq_train <- Rsq_time_models[, 1:4] %>% data.frame() %>% mutate(dataset="train")
Rsq_test <- Rsq_time_models[, 5:8] %>% data.frame() %>% mutate(dataset="test")
time_models <- Rsq_time_models[, 9:12] %>% data.frame()

Rsq_plot_data <- rbind(Rsq_train, Rsq_test) %>%
  gather(model, Rsq, lasso:rf)

Rsq_plot <- Rsq_plot_data %>%
  mutate(dataset = factor(dataset, levels=c("train", "test"))) %>%
  ggplot(aes(x = model, y = Rsq)) +
  geom_boxplot(aes(fill = model)) +
  facet_wrap(~ dataset, scales = "free_y") +
  labs(x = "Model", y = "R-Squared")

#Rsq_plot <- ggplotly(Rsq_plot) # Un-comment for ggplotly plot
Rsq_plot
```



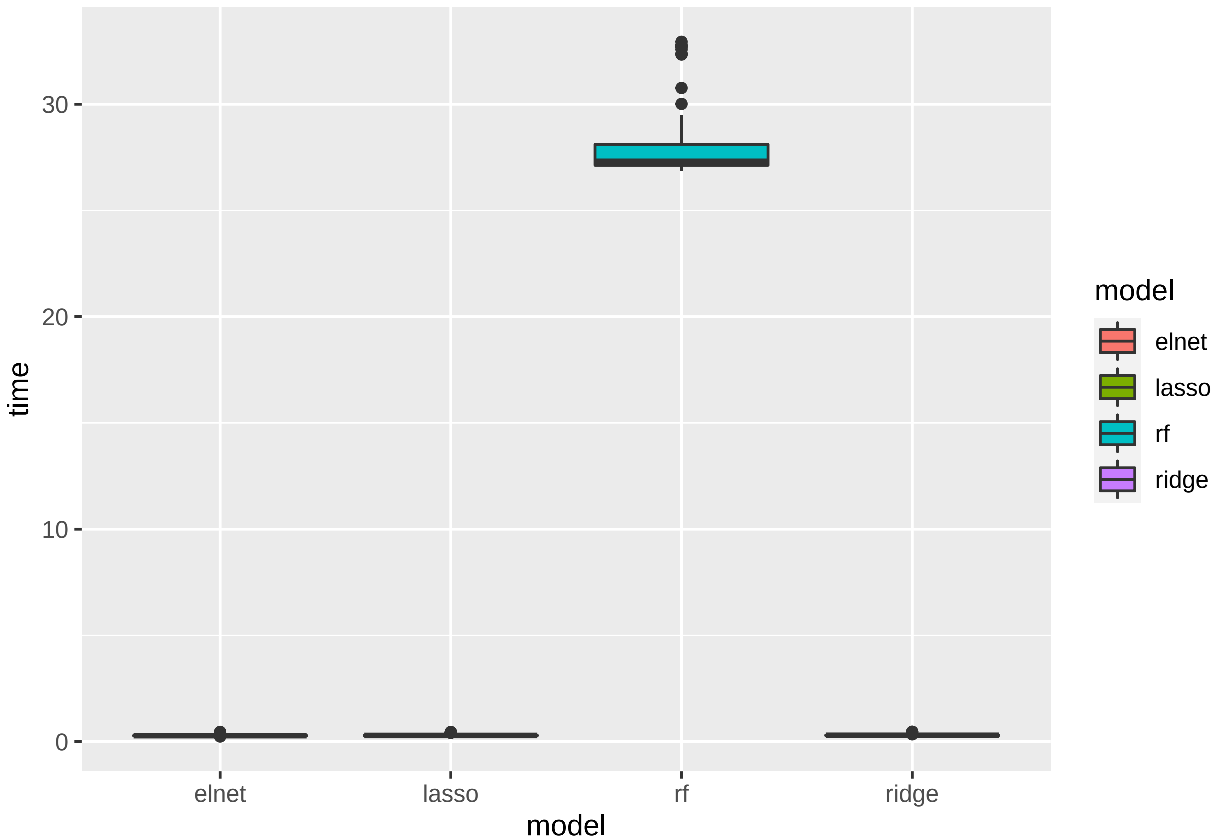

```r
sample_and_train_bs <- function() {
  bs_inds <- sample(n, replace=T)

  X_bs <- data[bs_inds, ] %>%
    select(-shares) %>%
    data.matrix()
  y_bs <- data$shares[bs_inds]

  # bootstrap lasso
```

```r
  lasso_cv <- cv.glmnet(X_bs, y_bs, alpha = 1)
  lasso_fit <- glmnet(X_bs, y_bs, alpha = 1, lambda = lasso_cv$lambda.min)
  beta_lasso_bs <- as.vector(lasso_fit$beta)
  # bootstrap ridge
  ridge_cv <- cv.glmnet(X_bs, y_bs, alpha = 0)
  ridge_fit <- glmnet(X_bs, y_bs, alpha = 0, lambda = ridge_cv$lambda.min)
  beta_ridge_bs <- as.vector(ridge_fit$beta)
  # bootstrap elastic-net
  elnet_cv <- cv.glmnet(X_bs, y_bs, alpha = 0.5)
  elnet_fit <- glmnet(X_bs, y_bs, alpha = 0.5, lambda = elnet_cv$lambda.min)
  beta_elnet_bs <- as.vector(elnet_fit$beta)
  # bootstrap random forest
  rf_fit <- randomForest(X_bs, y_bs, mtry = sqrt(p), importance = TRUE)
  beta_rf_bs <- as.vector(rf_fit$importance[,1])

  return(list(beta_lasso_bs=beta_lasso_bs, beta_ridge_bs=beta_ridge_bs,
              beta_elnet_bs=beta_elnet_bs, beta_rf_bs=beta_rf_bs))
}
set.seed(3)
bootstrapSamples = 100
beta_models <- replicate(bootstrapSamples, sample_and_train_bs())

beta_lasso_bs <- do.call(rbind, beta_models[1,])
beta_ridge_bs <- do.call(rbind, beta_models[2,])
beta_elnet_bs <- do.call(rbind, beta_models[3,])
beta_rf_bs <- do.call(rbind, beta_models[4,])

# calculate bootstrapped standard errors
lasso_bs_sd  <- apply(beta_lasso_bs, 2, "sd")
ridge_bs_sd  <- apply(beta_ridge_bs, 2, "sd")
rf_bs_sd  <- apply(beta_rf_bs, 2, "sd")
elnet_bs_sd  <- apply(beta_elnet_bs, 2, "sd")

X <- data %>% select(-shares) %>% data.matrix()
y <- data$shares

# fit lasso to the whole data
lasso_cv <- cv.glmnet(X, y, alpha = 1)
lasso_fit <- glmnet(X, y, alpha = 1, lambda = lasso_cv$lambda.min)
# fit ridge to the whole data
ridge_cv <- cv.glmnet(X, y, alpha = 0)
ridge_fit <- glmnet(X, y, alpha = 0, lambda = ridge_cv$lambda.min)
# fit elnet to the whole data
elnet_cv <- cv.glmnet(X, y, alpha = 0.5)
elnet_fit <- glmnet(X, y, alpha = 0.5, lambda = elnet_cv$lambda.min)
# fit rf to the whole data
rf_fit <- randomForest(X, y, ntree = 1, mtry = sqrt(p), importance = TRUE)

features <- 1:length(names(X[1,])) %>% as.factor()

betaS_lasso <- data.frame(feature = features, value = as.vector(lasso_fit$beta),
                          error = 2*lasso_bs_sd, model = "lasso")
betaS_ridge <- data.frame(feature = features, value = as.vector(ridge_fit$beta),
```

```
                                error = 2*ridge_bs_sd, model = "ridge")
betaS_elnet <- data.frame(feature = features, value = as.vector(elnet_fit$beta),
                                error = 2*elnet_bs_sd, model = "elnet")
betaS_rf <- data.frame(feature = features, value = as.vector(rf_fit$importance[,1]),
                             error = 2*rf_bs_sd, model = "rf")

betaS_models <- rbind(betaS_lasso, betaS_ridge, betaS_elnet, betaS_rf)
betaS_plot <- betaS_models %>%
  ggplot(aes(x=feature, y=value)) +
  geom_bar(stat = "identity", fill="white", colour="black") +
  geom_errorbar(aes(ymin=value-error, ymax=value+error), width=.2) +
  facet_wrap(~ model, scales = "free_y", ncol = 1)
#betaS_plot <- ggplotly(betaS_plot) # Un-comment for ggplotly plot
betaS_plot
```




```
time_plot <- time_models %>%
  gather(model, time, lasso:rf) %>%
  ggplot(aes(x=model, y=time, fill=model)) +
  geom_boxplot()

#time_plot <- ggplotly(time_plot) # Un-comment for ggplotly plot
time_plot
```




```
avg_model_times <- time_models %>%
  gather(model, time, lasso:rf) %>%
  group_by(model) %>%
  summarize(mean=mean(time), median=median(time), min=min(time), max=max(time))
kable(avg_model_times, digits=2)
```

model	mean	median	min	max
elnet	0.29	0.28	0.24	0.46
lasso	0.30	0.29	0.24	0.46
rf	27.96	27.31	26.85	32.94
ridge	0.30	0.30	0.26	0.47