Linear regression for data science

Contents

Introduction

In this do-it-yourself lab at home due to Ascension Day, you will learn how to handle many variables with regression by using variable selection techniques, shrinkage techniques, and how to tune hyper-parameters for these techniques. This practical has been derived from chapter 6 of ISLR. You can download the student zip including all needed files for practical 4 here.

Note: the completed lab has to be **handed in** on Black Board and will be **graded** (pass/fail, counting towards your grade for assignment 2). The deadline is Friday May 19th, 5PM. Hand-in should be a **PDF** file. If you know how to knit pdf files (see below), you can hand in the knitted pdf file. However, if you have not done this before, you are advised to knit to a html file as sepecified below, and within the html browser, 'print' your file as a pdf file.

In addition, you will need for loops (see also lab 1), data manipulation techniques from Dplyr, and the `caret` package (see lab week 3) to create a training, validation and test split for the used dataset. Another package we are going to use is `glmnet`. For this, you will probably need to `install.packages("glmnet")` before running the `library()` functions.

```
library(ISLR)
library(glmnet)
library(tidyverse)
library(caret)
library(ggthemes)
```

Best subset selection

Our goal is to to predict `Salary` from the `Hitters` dataset from the `ISLR` package. In this at home section, we will do the pre-work for best-subset selection. During the lab, we will continue with the actual best subset selection. First, we will prepare a dataframe `baseball` from the `Hitters` dataset where you remove the baseball players for which the `Salary` is missing. Use the following code:

```
baseball <- Hitters %>% filter(!is.na(Salary))
```

We can check how many baseball players are left using:

```
nrow(baseball)
```

```
## [1] 263
```

---

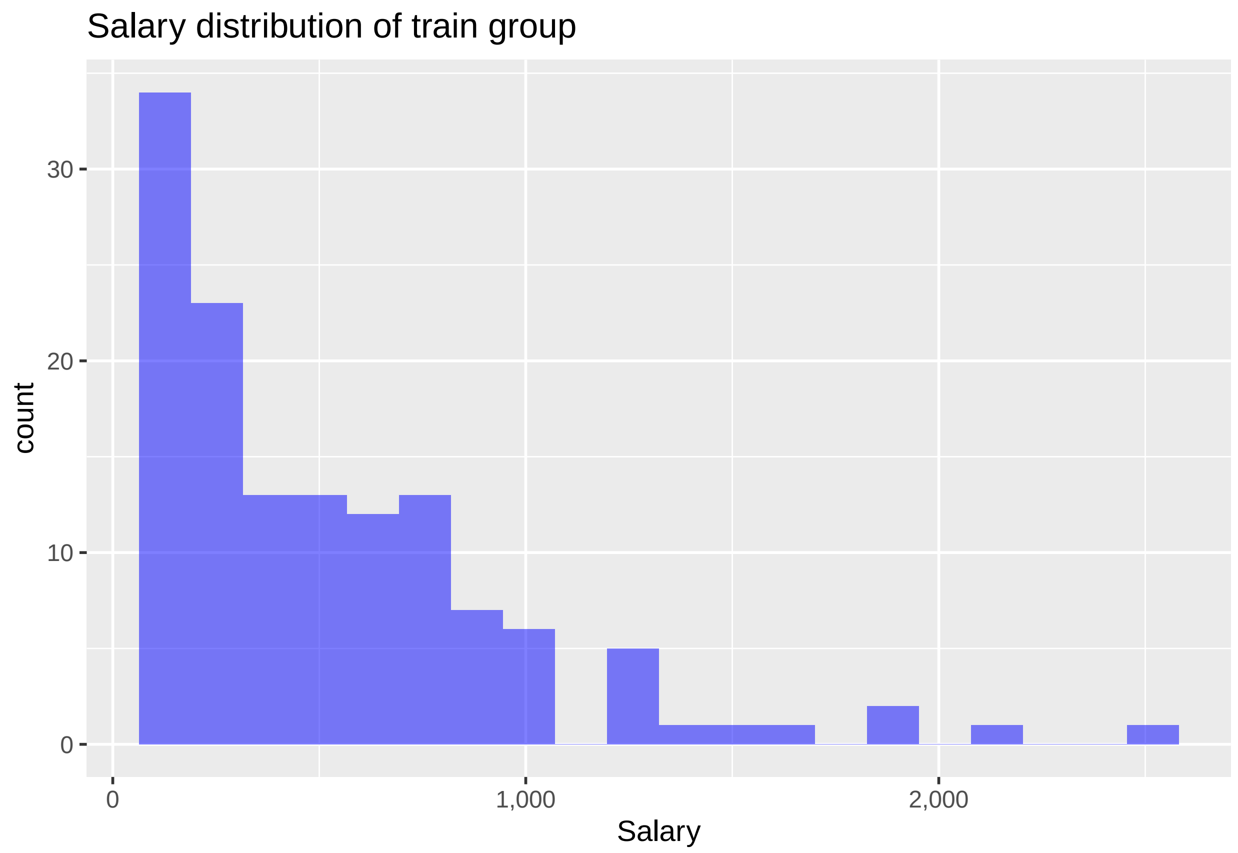
1. a) **Create `baseball_train` (50%), `baseball_valid` (30%), and `baseball_test` (20%) datasets using the `createDataPartition()` function of the `caret` package.**

```
set.seed(45)

# training
train_data <- createDataPartition(baseball$Salary, p = 0.5, list = FALSE)

baseball_train <- baseball[train_data, ]

# validation
valid_data <- createDataPartition(baseball[-train_data, ]$Salary, p = 0.6,
                                  list = FALSE)

baseball_valid <- baseball[-train_data, ][valid_data, ]


# testing
baseball_test <- baseball[-train_data, ][-valid_data, ]
```

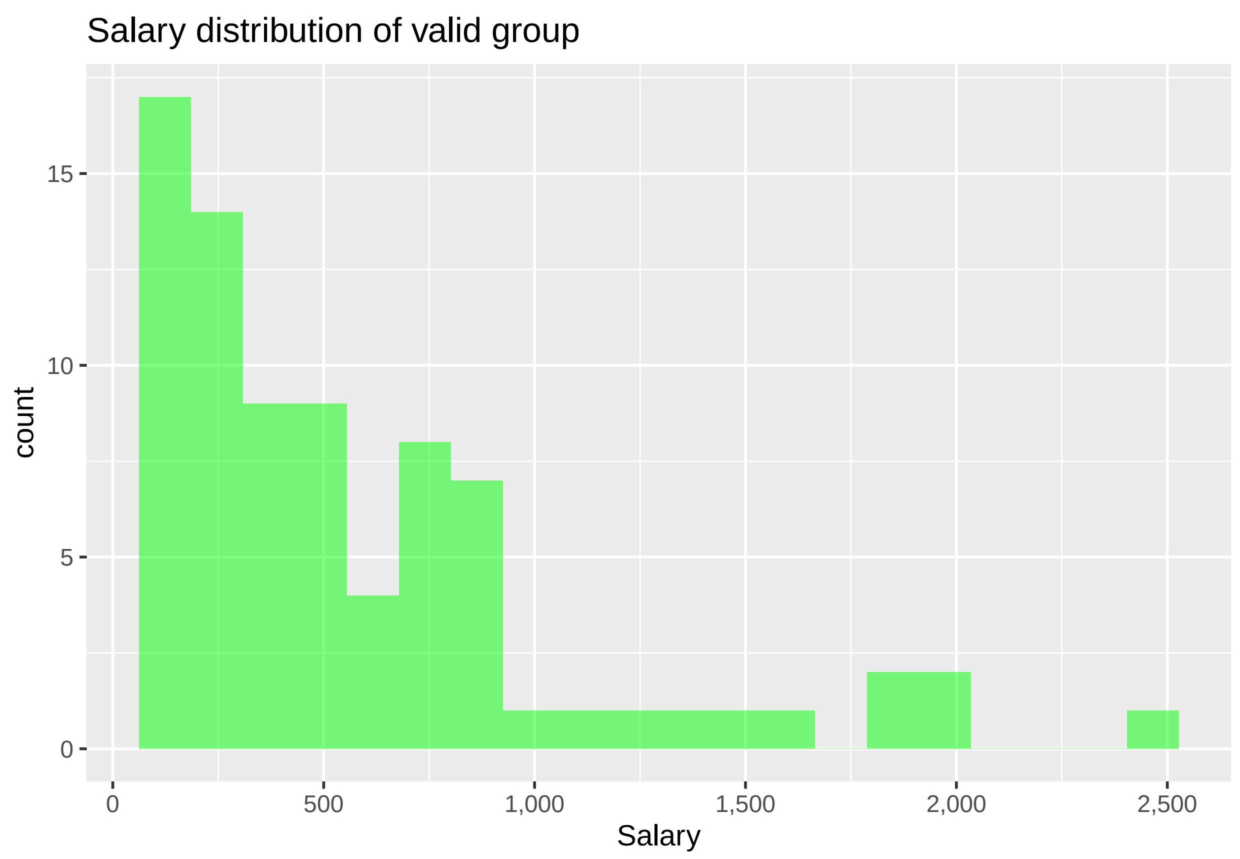
---

1. b) **Using your knowledge of `ggplot` from lab 2, plot the salary information of the train, validate and test groups using `geom_histogram()` or `geom_density()`**

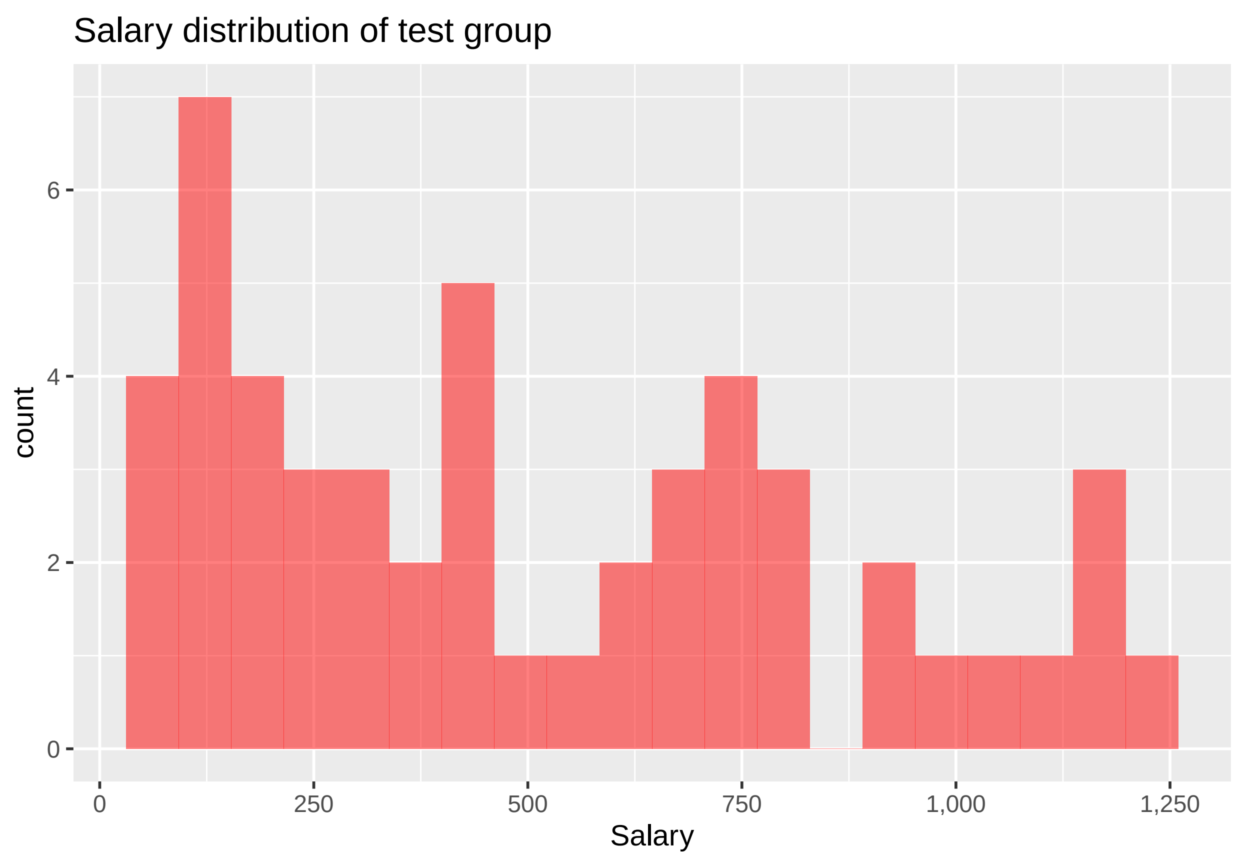
```
library(ggplot2)
ggplot(baseball_train, aes(x = Salary)) +
  geom_histogram(bins = 20, fill = "blue", alpha = 0.5) +
  labs(title = "Salary distribution of train group") +
  scale_x_continuous(labels = scales::comma)
```



```
ggplot(baseball_valid, aes(x = Salary)) +
  geom_histogram(bins = 20, fill = "green", alpha = 0.5) +
  labs(title = "Salary distribution of valid group") +
  scale_x_continuous(labels = scales::comma)
```



```r
ggplot(baseball_test, aes(x = Salary)) +
  geom_histogram(bins = 20, fill = "red", alpha = 0.5) +
  labs(title = "Salary distribution of test group") +
  scale_x_continuous(labels = scales::comma)
```



We will use the following function which we called `lm_mse()` to obtain the mse on the validation dataset for predictions from a linear model:

```
lm_mse <- function(formula, train_data, valid_data) {
  y_name <- as.character(formula)[2]
  y_true <- valid_data[[y_name]]

  lm_fit <- lm(formula, train_data)
  y_pred <- predict(lm_fit, newdata = valid_data)

  mean((y_true - y_pred)^2)
}
```

Note that the input consists of (1) a formula, (2) a training dataset, and (3) a test dataset.

---

2. **Try out the function with the formula `Salary ~ Hits + Runs`, using `baseball_train` and `baseball_valid`.**

```r
lm_mse(Salary ~ Hits + Runs, baseball_train, baseball_valid)
```

```
## [1] 192681.2
```

We have pre-programmed a function for you to generate a character vector for *all* formulas with a set number of `p` variables. You can load the function into your environment by *sourcing* the `.R` file it is written in:

```r
source("generate_formulas.R")
```

You can use it like so:

```r
generate_formulas(p = 2, x_vars = c("x1", "x2", "x3", "x4"), y_var = "y")
```

```
## [1] "y ~ x1 + x2" "y ~ x1 + x3" "y ~ x1 + x4" "y ~ x2 + x3" "y ~ x2 + x4"
## [6] "y ~ x3 + x4"
```

---

3. **Create a character vector of all predictor variables from the `Hitters` dataset. `colnames()` may be of help. Note that `Salary` is not a predictor variable.**

```r
all_columns <- colnames(Hitters)

(x_vars <- all_columns[all_columns != "Salary"])
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
##  [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
## [19] "NewLeague"
```

---

4. **Using the function `generate_formulas()` (which is inlcuded in your project folder for lab week 4), generate all formulas with as outcome `Salary` and 3 predictors from the `Hitters` data. Assign this to a variable called `formulas`. There should be 969 elements in this vector.**

```r
formulas <- generate_formulas(p = 3,

                              x_vars = colnames(
                                Hitters)[-which(colnames(Hitters) ==
                                                  "Salary")],

                              y_var = "Salary")

length(formulas)
```

```
## [1] 969
```

---

5. **Use a `for loop` to find the best set of 3 predictors in the `Hitters` dataset based on MSE. Use the `baseball_train` and `baseball_valid` datasets.**

When creating the `for loop`, use the function `as.formula()` from the stats package to loop over all the equations contained in `formulas`. `as.formula()` transforms the characters of the input to a formula, so we can actually use it as a formula in our code.

To select the best formula with the best MSE, use the function `which.min()`, which presents the lowest value from the list provided.

```r
library(stats)

best_formula <- NULL
best_mse <- Inf

# Iterate over all formulas
for (formula in formulas) {

  formula_obj <- as.formula(formula) #formula string to a formula object

  # Fit the linear model and make predictions
  lm_fit <- lm(formula_obj, data = baseball_train)
  y_pred <- predict(lm_fit, newdata = baseball_valid)
  mse <- mean((baseball_valid$Salary - y_pred)^2) ## Calculate MSE

  # Check if current formula has the lowest MSE
  if (mse < best_mse) {
    best_formula <- formula
    best_mse <- mse
  }
}

(best_formula)
```

```
## [1] "Salary ~ Hits + CRBI + Division"
```

```
(best_mse)
```

```
## [1] 121754.9
```

---

6. **Do the same for 1, 2 and 4 predictors. Now select the best model from the models with the best set of 1, 2, 3, or 4 predictors in terms of its out-of-sample MSE**

```
library(stats)

best_formula <- NULL
best_mse <- Inf

# Iterate over different numbers of predictors
for (p in 1:4) {
  # Generate formulas with the current number of predictors
  formulas <- generate_formulas(p = p,

                               x_vars = colnames(
                                 Hitters)[-which(colnames(Hitters)
                                                 == "Salary")],

                               y_var = "Salary")

  # Iterate over all formulas
  for (formula in formulas) {

    formula_obj <- as.formula(formula) # string to a formula object

    # Fit the linear model and make predictions
    lm_fit <- lm(formula_obj, data = baseball_train)
    y_pred <- predict(lm_fit, newdata = baseball_valid)


    mse <- mean((baseball_valid$Salary - y_pred)^2) #    # Calculate MSE

    # lowest MSE check
    if (mse < best_mse) {
      best_formula <- formula
      best_mse <- mse
    }
  }
}

# Print the best formula and its MSE
(best_formula)
(best_mse)
```

```
## [1] "Salary ~ AtBat + Hits + CRBI + Division"
## [1] 115821.2
```

---

7. a) **Calculate the test MSE for the model with the best number of predictors.**

```
best_formula_obj <- as.formula(best_formula)

lm_fit_best <- lm(best_formula_obj, data = baseball_train)

# baseball_test prediction
y_pred_best <- predict(lm_fit_best, newdata = baseball_test)

# test MSE calculation
test_mse_best <- mean((baseball_test$Salary - y_pred_best)^2)
```

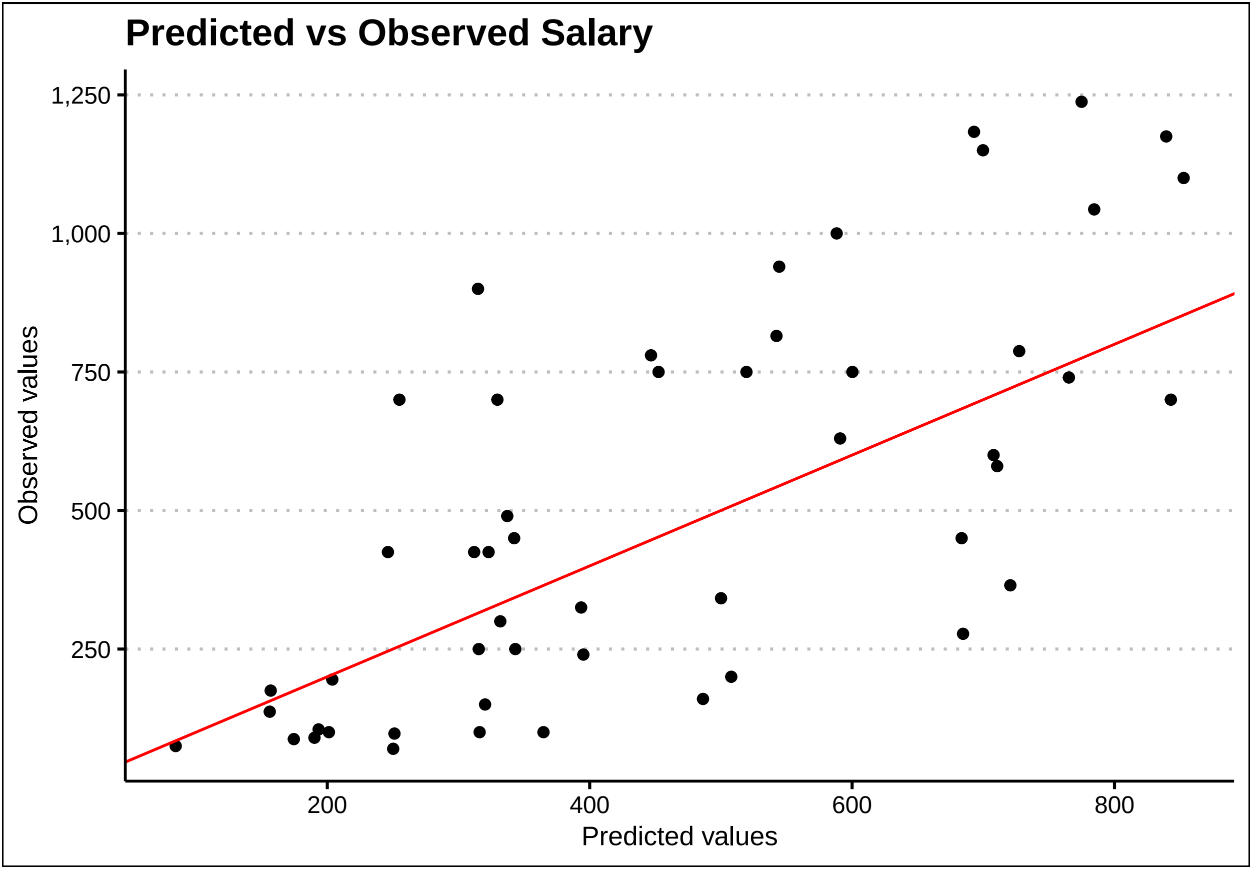
```
(test_mse_best)
```

```
## [1] 63730.97
```

---

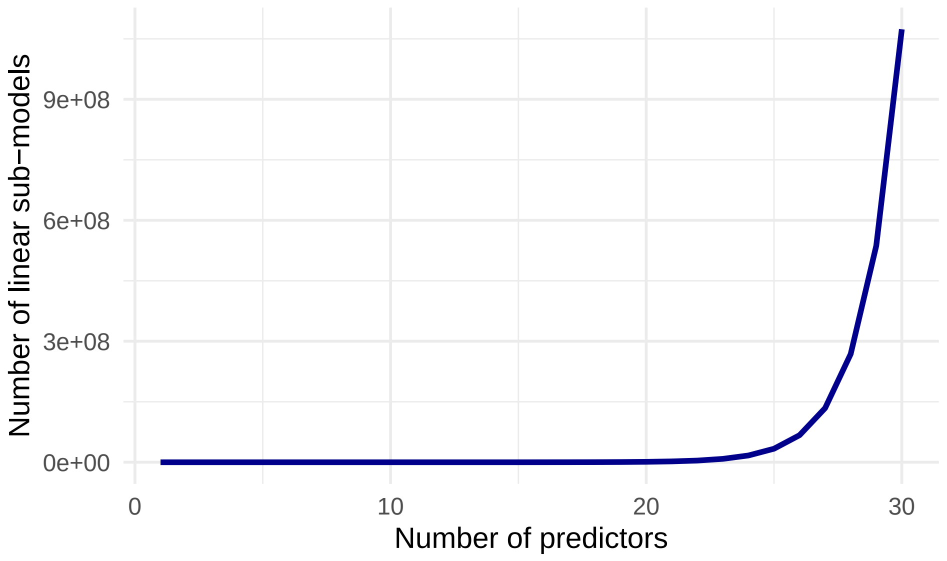
7. b) **Using the model with the best number of predictors, create a plot comparing predicted values (mapped to x position) versus observed values (mapped to y position) of `baseball_test`.**

```
plotting_data <- data.frame(
  Observed = baseball_test$Salary,
  Predicted = y_pred_best
)

plotting_data %>%
  ggplot(aes(x = Predicted, y = Observed)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  labs(x = "Predicted values",
       y = "Observed values",
       title = "Predicted vs Observed Salary") +
  scale_y_continuous(labels = scales::comma) +
  theme_clean()
```



Through enumerating all possibilities, we have selected the best subset of at most 4 non-interacting predictors for the prediction of baseball salaries. This method works well for few predictors, but the computational cost of enumeration increases quickly to the point where it is not feasible to enumerate all combinations of variables:

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



Regularization with glmnet

`glmnet` is a package that implements efficient (quick!) algorithms for LASSO and ridge regression, among other things.

---

8. **Skim through the help file of `glmnet`. We are going to perform a linear regression with normal (gaussian) error terms. What format should our data be in?**

```
?glmnet

# Outcome variable should be a numeric vector of
# length n (number of observations)
# Predictor variables should be a numeric matrix of size n by p
# or a sparse matrix of the same size. Predictor variables should be
# standardized (mean zero and unit variance) before fitting the model.
```

Again, we will try to predict baseball salary, this time using all the available variables and using the LASSO penalty to perform subset selection. For this, we first need to generate an input matrix.

---

9. **First generate the input matrix using (a variation on) the following code. Remember that the "." in a formula means "all available variables". Make sure to check that this `x_train` looks like what you would expect.**

```
x_train <- model.matrix(Salary ~ ., data = baseball_train)
```

```
x_train <- model.matrix(Salary ~ ., data = baseball_train)
dim(x_train)
```

```
## [1] 133  20
```

```
# Yes, it looks as expected.
```

The `model.matrix()` function takes a dataset and a formula and outputs the predictor matrix where the categorical variables have been correctly transformed into dummy variables, and it adds an intercept. It is used internally by the `lm()` function as well!

---

10. **Using `glmnet()`, perform a LASSO regression with the generated `x_train` as the predictor matrix and `Salary` as the response variable. Set the `lambda` parameter of the penalty to 15. NB: Remove the intercept column from the `x_matrix` – `glmnet` adds an intercept internally.**

```
x_train <- x_train[, -1] # removal intercept column

# LASSO reg fit
lasso_model <- glmnet(x_train, baseball_train$Salary, alpha = 1, lambda = 15)

(lasso_model)
```

```
##
## Call:  glmnet(x = x_train, y = baseball_train$Salary, alpha = 1, lambda = 15)
##
##   Df  %Dev Lambda
## 1  9 44.68     15
```

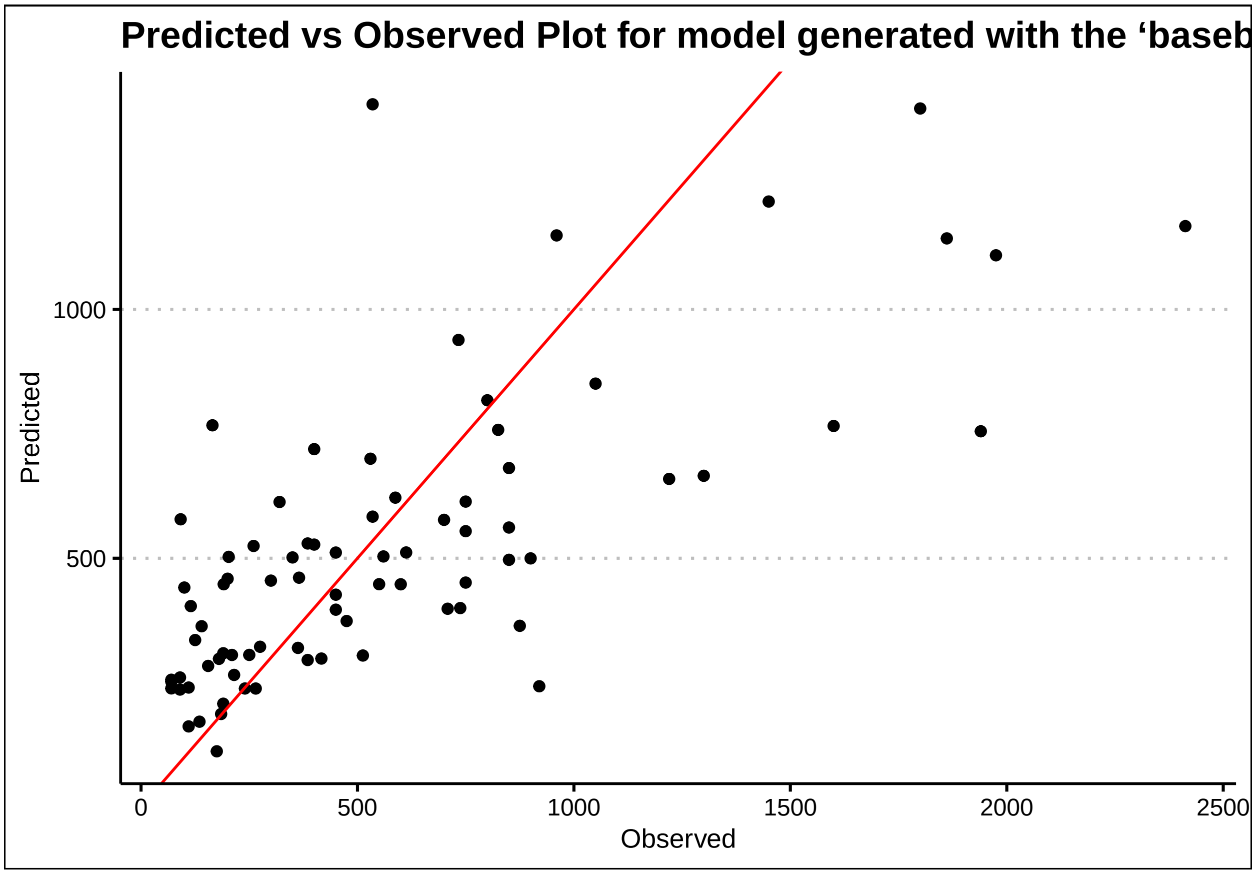
11. **The coefficients for the variables are in the `beta` element of the list generated by the `glmnet()` function. Which variables have been selected? You may use the `coef()` function.**

```
coef_lasso <- coef(lasso_model)

# selected variables identification
(selected_vars <- which(coef_lasso != 0))
```

```
##  [1]  1  3  7 11 12 13 16 17 18 20
```

12. **Create a predicted versus observed plot for the model you generated with the `baseball_valid` data. Use the `predict()` function for this! What is the MSE on the validation set?**

```
x_valid <-   model.matrix(Salary ~ ., data = baseball_valid)[,-1]
y_valid <-  baseball_valid$Salary

plot_model <- data.frame(
  x_valid =  model.matrix(Salary ~ ., data = baseball_valid)[,-1],
  y_valid = baseball_valid$Salary)

y_pred <- predict(lasso_model, x_valid)

ggplot(plot_model, aes(x = y_valid, y = y_pred)) +
         geom_point() +
         geom_abline(intercept = 0, slope = 1, color = "red") +
         labs(x = "Observed",
              y = "Predicted",
              title = "Predicted vs Observed Plot for model generated with the `baseball
         theme_clean()
```



```
(mse <- mean((y_valid - y_pred)^2))
```

```
## [1] 127722.9
```

Tuning lambda

Like many methods of analysis, regularized regression has a *tuning parameter*. In the previous section, we've set this parameter to 15. The `lambda` parameter changes the strength of the shrinkage in `glmnet()`. Changing the tuning parameter will change the predictions, and thus the MSE. In this section, we will select the tuning parameter based on out-of-sample MSE.

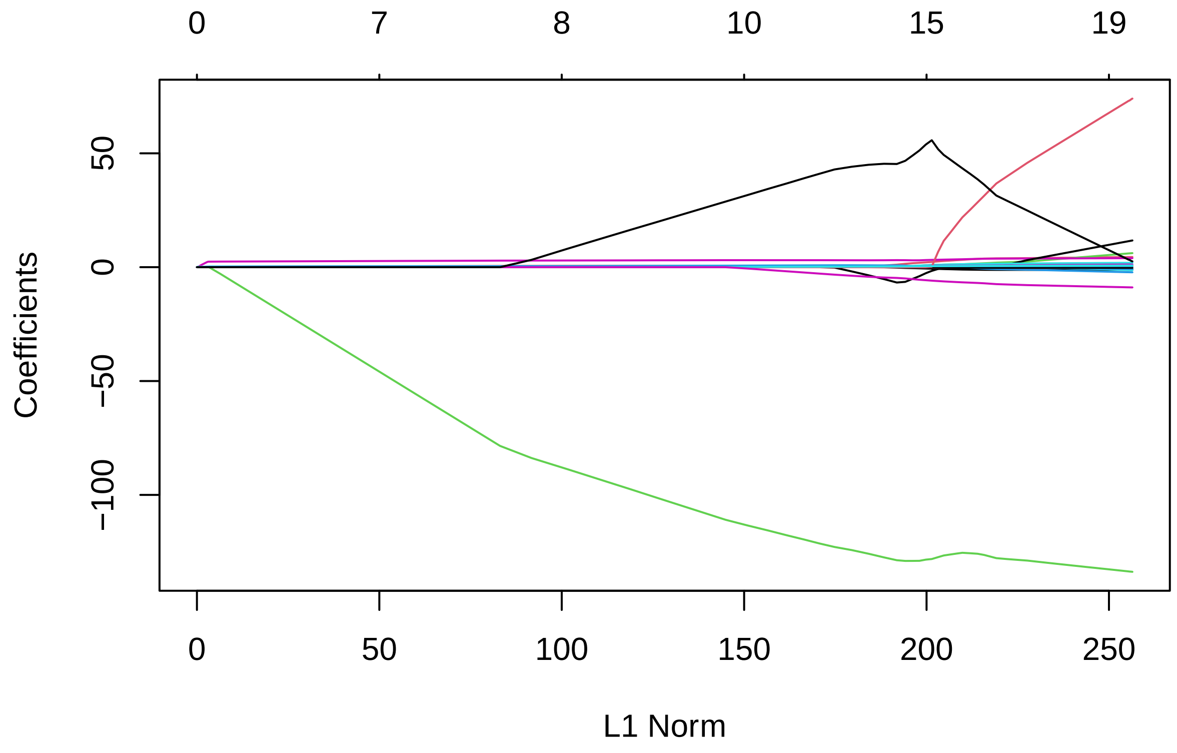
---

13. a) **Fit a LASSO regression model on the same data as before, but now do not enter a specific `lambda` value. What is different about the object that is generated? Hint: use the `coef()` and `plot()` methods on the resulting object.**

```r
(lasso_model2 <- glmnet(x_train, baseball_train$Salary, alpha = 1))
```

```
##
## Call:  glmnet(x = x_train, y = baseball_train$Salary, alpha = 1)
##
##    Df  %Dev  Lambda
## 1   0  0.00 238.300
## 2   1  4.79 217.100
## 3   1  8.76 197.800
## 4   1 12.06 180.200
## 5   3 15.69 164.200
## 6   3 19.11 149.600
## 7   4 22.57 136.300
## 8   4 25.82 124.200
## 9   4 28.51 113.200
## 10  4 30.74 103.100
## 11  4 32.60  93.980
## 12  5 34.13  85.630
## 13  6 35.50  78.020
## 14  6 36.68  71.090
## 15  7 37.75  64.780
## 16  7 38.90  59.020
## 17  7 39.87  53.780
## 18  7 40.67  49.000
## 19  7 41.33  44.650
## 20  7 41.89  40.680
## 21  7 42.34  37.070
## 22  7 42.72  33.770
## 23  7 43.04  30.770
## 24  7 43.30  28.040
## 25  8 43.56  25.550
## 26  8 43.81  23.280
## 27  9 44.05  21.210
## 28  9 44.27  19.330
## 29  9 44.44  17.610
## 30  9 44.59  16.050
## 31  9 44.71  14.620
## 32  9 44.81  13.320
## 33 10 44.99  12.140
## 34 10 45.14  11.060
## 35 10 45.27  10.080
## 36 10 45.38   9.182
## 37 10 45.47   8.366
## 38 11 45.56   7.623
## 39 11 45.74   6.946
## 40 11 45.89   6.329
## 41 12 46.10   5.767
## 42 12 46.44   5.254
## 43 14 47.12   4.787
## 44 14 47.96   4.362
## 45 15 48.67   3.975
## 46 15 49.33   3.622
## 47 15 49.84   3.300
## 48 15 50.20   3.007
## 49 14 50.51   2.740
## 50 14 50.74   2.496
## 51 14 50.94   2.274
## 52 15 51.11   2.072
## 53 15 51.29   1.888
## 54 16 51.44   1.721
## 55 16 51.58   1.568
## 56 17 51.74   1.428
## 57 17 51.87   1.302
## 58 18 52.02   1.186
## 59 18 52.14   1.081
## 60 19 52.25   0.985
## 61 19 52.34   0.897
## 62 19 52.42   0.817
## 63 19 52.49   0.745
## 64 19 52.54   0.679
## 65 19 52.59   0.618
## 66 19 52.63   0.563
## 67 19 52.66   0.513
## 68 19 52.68   0.468
## 69 19 52.71   0.426
## 70 19 52.72   0.388
## 71 19 52.74   0.354
## 72 19 52.75   0.322
## 73 19 52.76   0.294
## 74 19 52.77   0.268
## 75 19 52.78   0.244
## 76 19 52.79   0.222
## 77 19 52.79   0.202
## 78 19 52.79   0.184
## 79 19 52.80   0.168
## 80 19 52.80   0.153
## 81 19 52.80   0.140
## 82 19 52.81   0.127
## 83 19 52.81   0.116
```

```
lasso_coefs2 <- coef(lasso_model2)

plot(lasso_model2)
```



```
#  lasso_model2 without the lambda computes the regularization path for a grid
# of lambda values instead of a single lambda value like in lasso_model
# containing a lambda
```

---

13. b) **To help you interpret the obtained plot, Google and explain the qualitative relationship between L1 norm (the maximum allowed sum of `coefs`) and `lambda`.**

```
# Plot illustrates the effect of the L1 norm constraint on the coefficients.
# L1 norm is the sum of the absolute values of the coefficients.
# L1 is inversely related to lambda, that is, as lambda increases it decreases,
# meaning that the coefficients are shrunk towards zero.
```

For deciding which value of lambda to choose, we could work similarly to what we have don in the best subset selection section before. However, the `glmnet` package includes another method for this task: cross validation.

---

14. **Use the `cv.glmnet` function to determine the `lambda` value for which the out-of-sample MSE is lowest using 15-fold cross validation. As your dataset, you may use the training and validation sets bound together with bind_rows(). What is the best lambda value?**

Note You can remove the first column of the `model.matrix` object, which contains the intercept, for use in `cv.glmnet`. In addition, To obtain the best lambda value, you can call the output value `lambda.min` from the object in which you stored the results of calling `cv.glmnet`.

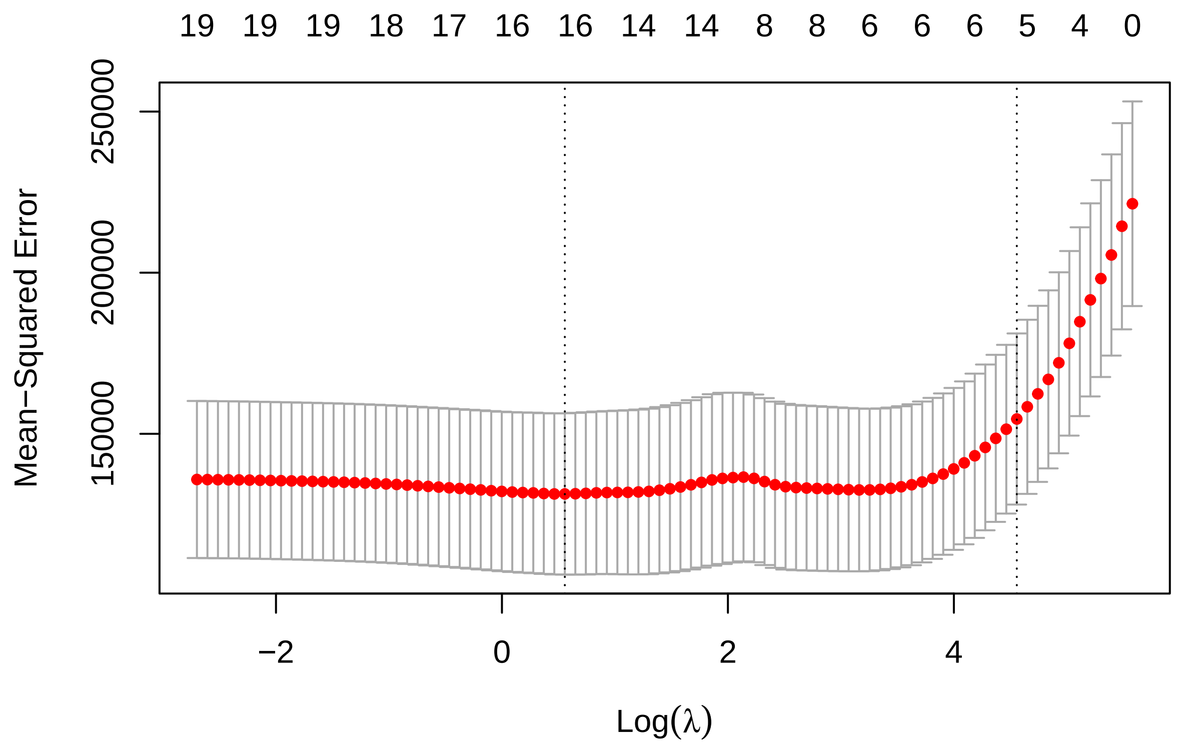
```r
combined_data <- bind_rows(baseball_train, baseball_valid)

x_combined <- model.matrix(Salary ~ ., data = combined_data)[, -1]

cv_result <- cv.glmnet(x_combined, combined_data$Salary, alpha = 1,
                       nfolds = 15)

(best_lambda <- cv_result$lambda.min)
```

```
## [1] 1.744613
```

---

15. **Try out the plot() method on this object. What do you see? What does this tell you about the bias-variance tradeoff?**

```r
plot(cv_result)
```



```
# As lambda decreases, the MSE initially decreases as well, meaning that the
# model is decreasing its bias and fitting the data better but after a certain
# point, the MSE starts to increase again, showing that the model is increasing
# its variance and over fitting the data.
```

It should be noted, that for all these previous exercises they can also be completed using the **Ridge Method** which is not covered in much depth during this practical session. To learn more about this method please refer back Section 6.2 in the An Introduction to Statistical Learning Textbook.
