Assignment 5

Statistical Modelling I

Clinton Morris
Mauricio Garcia Tec

October 2017

Introduction

This is our report for the assignment 5 of Statistical Modelling I. The online version of this documents can be found at https://sds383team.github.io/HW5/HW5.html.

We run five different regression models for the Concrete Compressive Strength Data of the UCI machine learning repository. After running the models we present a comparison table. All of them perform roughly the same, Ridge being the best, but with a very small margin.

These are the R libraries we will use:

```
library(glmnet) # Library for penalized regression models and cross-validation
library(tidyverse) # Efficient data manipulation and I/O
library(ggplot2) # A grammar of graphics for plotting
library(ggthemes) # Improves layout adn colors for ggplot2 graphics
```

The Task

We obtain the data

```
concrete <- read_csv("https://raw.githubusercontent.com/SDS383team/HW5/master/data/Con
creteData.csv")
```

Here is how the first rows look like:

Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Compressive strength
540.0	0.0	0	162	2.5	1040.0	676.0	28	79.99
540.0	0.0	0	162	2.5	1055.0	676.0	28	61.89
332.5	142.5	0	228	0.0	932.0	594.0	270	40.27
332.5	142.5	0	228	0.0	932.0	594.0	365	41.05
198.6	132.4	0	192	0.0	978.4	825.5	360	44.30

Our **task** is to compare regression models where the response variable is **Compressive strength** and the rest are used as predictors.

To begin, we first create train and test sets splitting the data by half; then we define the design matrix X and response vector y of the testing and training sets. The train and test data will be stored in matrix and numeric vector form since the package `glmnet` which we use later needs it.

```
set.seed(999) # for reproducibility
train_idx <-  sample(1:nrow(concrete), size = nrow(concrete) / 2)
test_data <- concrete %>%
  slice(train_idx)
train_data <- concrete %>%
  slice(-train_idx)
X_test <- test_data %>%
  select(-`Compressive strength`) %>%
  data.matrix()
X_train <- train_data %>%
  select(-`Compressive strength`) %>%
  data.matrix()
y_test <- test_data %>%
  pull(`Compressive strength`)
y_train <- train_data %>%
  pull(`Compressive strength`)
```

Running the Models

Below are the commands to compute five different regression models and estimate their generalization errors.

(1) Multiple Linear Regression

We run and save the RMSE for linear regression on the full training with all the variables using R's function `lm`.

```
mlr <- lm(`Compressive strength` ~ ., data = train_data)
rmse_train_mlr <- sqrt(mean((y_train - predict(mlr, train_data))^2))
rmse_test_mlr <- sqrt(mean((y_test - predict(mlr, test_data))^2))
```

We now repeat the same training but with **10-fold cross validation** on the training set. Since we are not tuning any parameter the average RMSE should be very similar to the rmse on the test set.

```
# Number of folds for cross-validation
nfolds <- 10
size <- nrow(train_data)
# Create 10 equal size folds as a list of validation indexes
folds <- split(sample(size), cut(1:size, breaks = nfolds, labels = FALSE))
# Perform 10 fold cross validation
rmse_cv_mlr_k <- numeric(nfolds)
for (k in 1:nfolds) {
  idx <- folds[[k]]
  mod <- lm(`Compressive strength` ~ ., data = train_data[-idx, ])
  rmse_cv_mlr_k[k] <- sqrt(mean((y_train[idx] - predict(mod, train_data[idx, ]))^2))
}
rmse_cv_mlr <- mean(rmse_cv_mlr_k)
```

The RMSE for each cross-fold is shown at the appendix.

(2) Multiple Linear Regression & Model Selection with BIC

We will now choose the best model using the Bayesian Information Criterion. The BIC is a function that penalizes each free variable, selecting the one that best minimizes $p\log(n) - 2\log(\hat{L})$ where n is the number of observations on the dataset and p the number of variables. The functionality of model selection using BIC in R is given by the function `step` with the option `k = log(n)`. In addition we set `trace FALSE` to avoid verbose printing. The output input of `step` is our linear model and the output is another linear model that contains the selected variables only.

```
bicmlr <- lm(`Compressive strength` ~ ., data = train_data) %>%
  step(k = log(nrow(train_data)), trace = FALSE)
rmse_train_bicmlr <- sqrt(mean((y_train - predict(bicmlr, train_data))^2))
rmse_test_bicmlr <- sqrt(mean((y_test - predict(bicmlr, test_data))^2))
```

We now repeat the cross validation on the same sets as before

```
rmse_cv_bicmlr_k <- numeric(nfolds)
nvariables_bic <- numeric(nfolds) # used in appendix
for (k in 1:10) {
  idx <- folds[[k]]
  mod <- lm(`Compressive strength` ~ ., data = train_data[-idx, ]) %>%
    step(k = log(nrow(train_data) - length(idx)), trace = FALSE)
  rmse_cv_bicmlr_k[k] <- sqrt(mean((y_train[idx] - predict(mod, train_data[idx,
]))^2))
  nvariables_bic[k] <- length(mod$coefficients)
}
rmse_cv_bicmlr <-  mean(rmse_cv_bicmlr_k)
```

Again, plots and values of the quality of the model are shown in the appendix and the comparison table after running every model.

(3) Lasso

For the penalized models we now use the `glm` package, which takes numeric matrices as input. The function `glmnet` has an elastic mixing input `alpha`, which does the lasso regression when set to `alpha = 1`. The Lasso adds a penalty of the form $\lambda\|\beta\|_1$ to the multiple linear regression model. While we could just "guess" a value for λ, it is usually obtained from cross validation or from comparing on a test set, this is implemented on the `cv.glmnet` function.

```
lasso = cv.glmnet(X_train, y_train, alpha = 1, type.measure = "mse", nfolds = 10)
rmse_cv_lasso <- sqrt(min(lasso$cvm))
rmse_train_lasso <- sqrt(mean((predict(lasso, X_train, s = "lambda.min") -
y_train)^2))
rmse_test_lasso <- sqrt(mean((predict(lasso, X_test, s = "lambda.min") - y_test)^2))
```

The MSE for different values of λ and the optimal values are in the appendix.

(4) Ridge

Setting `alpha = 0` the `glmnet` regression yields Ridge.

```
ridge = cv.glmnet(X_train, y_train, alpha = 0, type.measure = "mse", nfolds = 10)
rmse_cv_ridge <- sqrt(min(ridge$cvm))
rmse_train_ridge <- sqrt(mean((predict(ridge, X_train, s = "lambda.min") -
y_train)^2))
rmse_test_ridge <- sqrt(mean((predict(ridge, X_test, s = "lambda.min") - y_test)^2))
```

(5) Elastic Net

An elastic net interpolates the lasso and ridge regularization. By setting `alpha = 0.5` in the `cv.glmnet` function we compute an elastic net.

```
enet = cv.glmnet(X_train, y_train, alpha = 0.5, type.measure = "mse", nfolds = 10)
rmse_cv_enet <- sqrt(min(enet$cvm))
rmse_train_enet <- sqrt(mean((predict(enet, X_train, s = "lambda.min") - y_train)^2))
rmse_test_enet <- sqrt(mean((predict(enet, X_test, s = "lambda.min") - y_test)^2))
```

Comparison

Here is a table with the performance of each model using the Root MSE.

Model	Training	10-Fold CV	Test
MLR	10.3459	10.5037	10.5281
MLR with BIC	10.4303	10.6091	10.5453
Lasso	10.3466	10.5932	10.5206
Ridge	10.4959	10.6980	10.4797
Elastic Net (alpha=0.5)	10.3467	10.5641	10.5205

In the table above it is shown that Ridge is the best performer, but after trying different random seed is the way we splitted the data, we saw that this is not always the case; sometimes even Multiple Linear Regression can be the best one.

Appendix

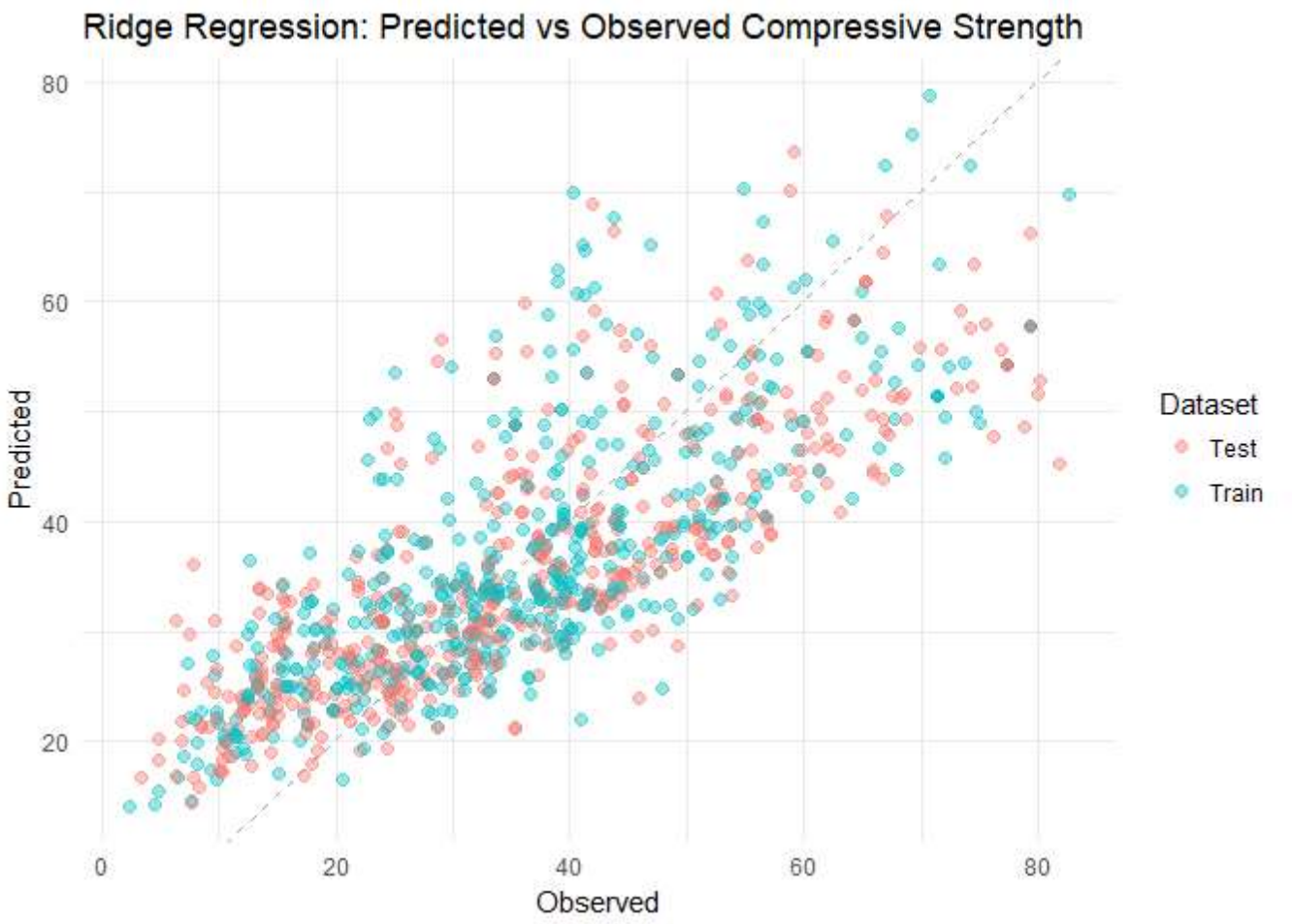
We add some plots and information to our analysis to better understand what happened.

(A) Quality of Fit: Observed vs Fitted

It is always good to compare the predicted vs the fitted values. Since Ridge regression was the best in our test, and most models performed the same, we show the plot for the Ridge model only.

```
plotdata <- concrete %>%
  rename(Observed = `Compressive strength`) %>%
  mutate(Dataset = ifelse(1:nrow(concrete) %in% train_idx, "Train", "Test")) %>%
  mutate(Predicted = as.numeric(predict(ridge, as.matrix(concrete[ ,-
ncol(concrete)]))))
ggplot(plotdata, aes(x = Observed, y = Predicted, colour = Dataset)) +
  geom_point(size = 2, alpha = 0.4) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", colour = "darkgray") +
```

```
  theme_minimal() +
  ggtitle("Ridge Regression: Predicted vs Observed Compressive Strength")
```



(B) RMSE & Number of Variables in MLR with BIC Model Section

We first print the number of selected variables that the BIC selected for each of the k-folds in cross validation,

```
print(nvariables_bic)
```

```
##  [1] 6 6 6 6 6 6 6 6 8 7
```

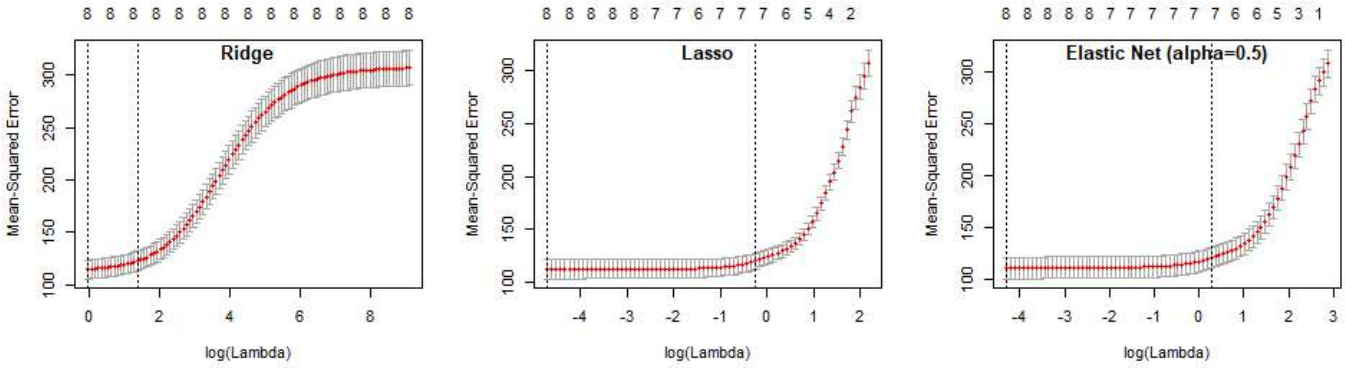
The next plot compares the RMSE in each cross-validation fold with and without the Model Selection. We do not see any improvement from the BIC step.

```
plotdata <- data.frame(
  RMSE = c(rmse_cv_mlr_k, rmse_cv_bicmlr_k),
  Model = c(rep("MLR", nfolds), rep("MLR with BIC", nfolds)),
  Fold = factor(rep(1:nfolds, 2))
)
ggplot(plotdata, aes(x = Fold, y = RMSE, fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  ggtitle("MLR vs MLR wit BIC Model Selection")
```

(C) Comparing the MSE of Penalized Models and optimal lambdas

For the penalized models we are actually using the cross-validation step to fine the best penalization parameter λ. The following plots show the MSE (no the RMSE but it's a monotone transform) for different values of λ. The plot shows also error bars for the MSE.

```
par(mfrow = c(1,3))
plot(ridge)
title("Ridge", line = -1)
plot(lasso)
title("Lasso", line = -1)
plot(enet)
title("Elastic Net (alpha=0.5)", line = -1)
```



The following table compares the optimal values of the penalization for each model.

Model	Lambda
Lasso	0.0090668
Ridge	0.9722062
Elastic Net (alpha=0.5)	0.0137174