Computational Data Analysis, Case 1

Anders Olsen (s154043) & Peyman Kor (s191828)

March 2020

1 Introduction

This report summarizes the work done in the first case study of the course 02582 Computational Data Analysis regarding model building with a data set of 95 continuous and 5 categorical predictors and one continuous response variable. Only 100 observations are provided, i.e. $p = N$, and when partitioning the data into training and test sets, $p > N$. Thus, dimensionality issues will be prominent and methods for shrinkage will be necessary to explore.

Initial data exploration showed that all continuous predictors are correlated with each other to some degree. Pooling the values of the correlation matrix reveal that they possibly are Gaussian distributed with mean $\mu = 0.46$ and standard deviation $\sigma^2 = 0.10$. Figure 1(left) shows this, and also indicates that the distribution is possibly a tad right-skewed. None of the coefficients are high enough to remove variables directly, but all predictors share some information, indicating that regularization is crucial. Likewise, the correlation coefficients between the continuous predictors and the response variable could stem from a Gaussian distribution with $\mu = 0.01$ and $\sigma^2 = 0.10$ (Figure 1right). The continuous predictors were normalized and the response was centered prior to analysis.

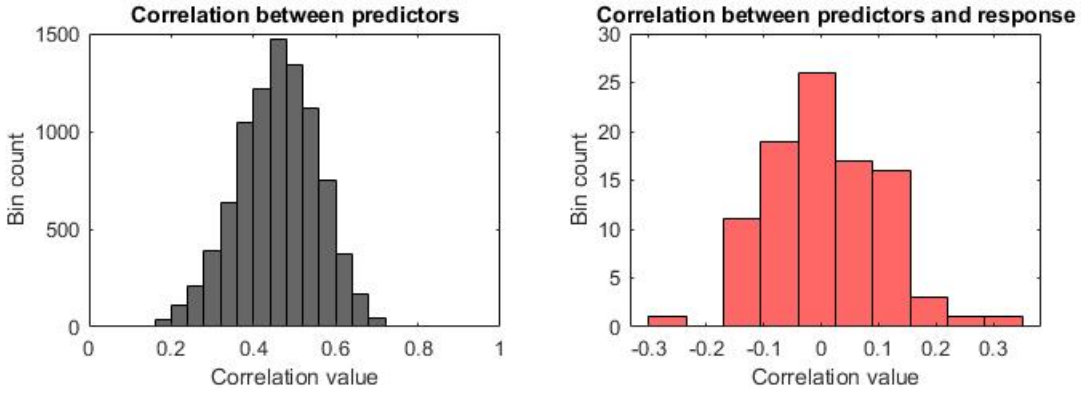

Figure 1: Correlation coefficients between predictors (left) and between predictors and the response variable (right).

2 Model and Method

Having a very high number of predictors compared to observations renders many machine learning methods useless. For example, the ordinary least-squares has no solution. Hence, the focus of this task is to implement models that can solve high-dimensional problems. In particular, we look at regularization

methods such as ridge regression, least absolute shrinkage and selection operator (LASSO) and the mix of the two, elastic net. Furthermore, we also take a look at tree-based methods.

2.1 Missing Values

The five categorical variables, which are all multinomial with class names G, H, I, J and K, contain missing values that need to be dealt with. A way to solve this issue is to build a model to predict the values of the missing data, such as any multinomial classifier, e.g. logistic regression. We opted to use k-Nearest Neighbor (kNN) that identifies the k closest (using Euclidean distance) observations in 95-dimensional space and predicts the new observation as the majority vote of these points. We treated each categorical variable one at a time as the response variable, and the 95 continuous variables as the *data*. To select the value k, a leave-one-out cross-validation (CV) loop was employed N times since data is scarce. Within each fold, a model with k ranging from 1 to 20 was fitted and the 0-1 loss was noted. The k corresponding to the lowest average error was $k = 3$. The imputed values are then used for subsequent model building. Optimally, values would be imputed within model selection loops but this was not done here.

2.2 Factor Handling

After imputing missing values, the categorical variables were hardcoded to a dummy variable of zeros and ones. For each variable, there is one column per category, yielding a total of 25 columns (5 variables with 5 categories).

2.3 Elastic net

The elastic net involves the minimization problem in Eq. 1, which is a compromise between ridge regression, which uses L_2 regularization, and Lasso which penalizes the L_1-norm. Lasso is known for inducing sparsity in the solution, forcing more beta estimates to zero the higher the regularization parameter λ is. This is a type of subset selection and thus produces simpler models. On the contrary, ridge regression shrinks parameter estimates, though rarely to zero. In general, regularization attempts to lower the variance of the solution while adding bias in the hope of decreasing the overall prediction error. Regularization generalizes the model by reducing overfitting.

$$\underset{\beta}{\operatorname{argmin}} \frac{1}{2n}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\left(\frac{1}{2}(1-\alpha)\boldsymbol{\beta}^T\boldsymbol{\beta} + \alpha\sum_{i=1}^{p}|\beta_i|\right) \tag{1}$$

Two parameters need to be estimated, the regularization parameter λ, and α which controls the influence that each norm has in the minimization problem. If $\alpha = 0$, the problem is reduced to ridge regression, and if $\alpha = 1$ it is purely Lasso regression. The two values were found using a repeated 5-fold CV grid search loop. Leave-one-out CV could have provided better estimates. Initial exploration was conducted to refine the grid search. Inside the loop, continuous predictors were normalized and the response was centered. For each fold, the elastic net with all possible combinations of 20 values of $\lambda \in [10^{-1}, 10]$ and 100 values of $\alpha \in (0, 1]$ was calculated, and the root-mean-square-error estimate ($\widehat{RMSE}$) found. The result is a heatmap, a zoomed version of which seen in Figure 2. Light colors correspond to low $\widehat{RMSE}$ and thus the better option. The lowest $\widehat{RMSE}$ is found at $\lambda = 2$ and $\alpha = 0.6$. In general, CV tends to select too complex models, and often the one-standard-error rule is used, however, this method was not used here.

2.4 Extreme Gradient Boosting Model

A new, popular method that could be used to solve this problem is Extreme Gradient Boosting Model (XGBoost), which is a fast implementation of boosted trees utilizing gradient descent. In contrast to

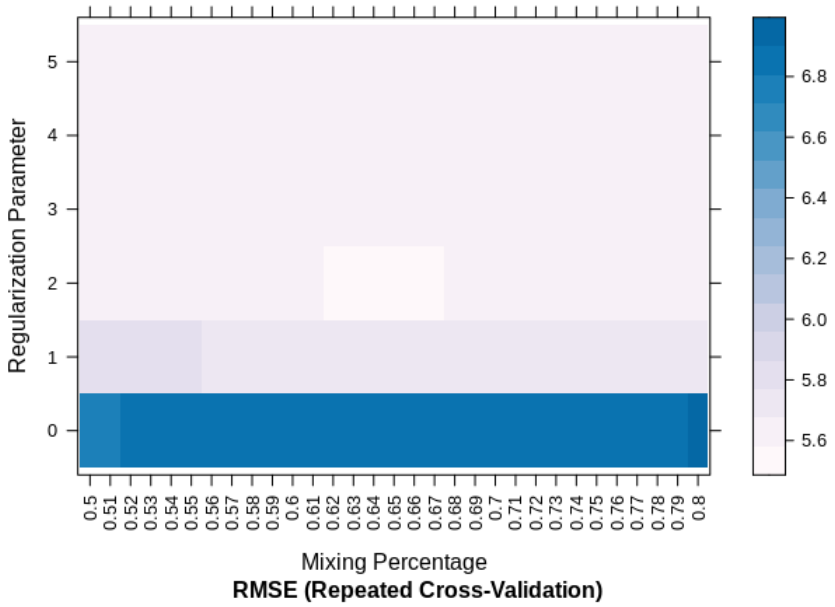

Figure 2: Zoomed heatmap of RMSE-estimates versus λ and α values.

Boosted Trees, XGBoost improves on one of the major inefficiencies of gradient boosted trees, namely considering the potential loss for all possible splits to create a new branch. This is especially fruitful if dealing with high dimensional data. XGBoost solves this by looking at the distribution of predictors in a leaf and uses this information to reduce the search space of possible feature splits. A drawback of the model is the fact that 6 different hyperparameters need to be estimated for optimal performance, and using grid search can take a very long time.

This model was trained in the same way as elastic net, using repeated 5-fold CV. Using XGboost we get a much better performance than the elastic net. In the Appendix, Figure 3 shows predicted values versus the true response for elastic net and XGboost. XGboost has a better fit, but overfitting is a worry.

3 Model Validation and Results

Although XGBoost shows very promising results ($\widehat{RMSE} = 2.940534$), the results shown here were obtained using elastic net, simply because XGboost is not in the curriculum of the course. Ideally, model assessment should happen on an independent test set not used for model building. Since the data set for this assignment contains very few observations, we opted for evaluating the model using a CV-loop on the full data set after having estimated parameters for the model. The final result is $\widehat{RMSE} = 5.141456$.

4 Appendix

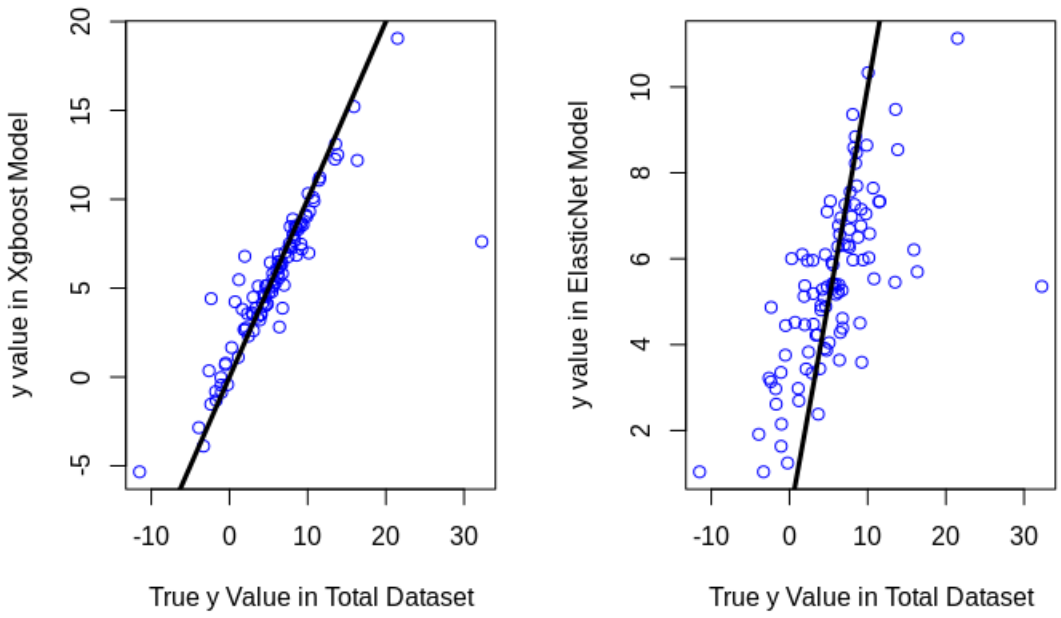

Figure 3: Predictions versus true values for XGboost (right) and elastic net (left).

4.1 Code for elastic net and XGboost (R)

```
library(glmnet)
library(caret)
library(elasticnet)
library(tidyverse)

x_new <- read.csv('case1Data_Xnew.csv')
data <- readxl::read_excel('case1Data.xlsx')
data$C_1 <- as.factor(data$`C_ 1`)
data$C_2 <- as.factor(data$`C_ 2`)
data$C_3 <- as.factor(data$`C_ 3`)
data$C_4 <- as.factor(data$`C_ 4`)
data$C_5 <- as.factor(data$`C_ 5`)

data_new <- data[,-c(97:101)]
dmy <- dummyVars(" ~ .", data = data_new)
data_new_dummy_ini <- data.frame(predict(dmy, newdata = data_new))


dmy_new <- dummyVars(" ~ .", data = x_new)
data_new_dummy_sub <- data.frame(predict(dmy_new, newdata = x_new))
```

```r
set.seed(1234)

indexes <- createDataPartition(data_new_dummy_ini$y,
                               times = 1,
                               p = 0.8,
                               list = FALSE)

train_data <- data_new_dummy_ini[indexes,]
test_data <- data_new_dummy_ini[-indexes,]

fitControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           ## repeated ten times
                           repeats = 5)


train.control <- trainControl(method = "repeatedcv",
                              number = 5,
                              repeats = 10,
                              search = "grid")



tune.grid <- expand.grid(alpha = seq(0.5,0.8,0.01),
                         lambda = seq(0,3,1))




caret_cv_reg <- train(y ~ .,
                  data = train_data,
                  method = "glmnet",
                  tuneGrid = tune.grid,
                  trControl = train.control)



fitControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 10,
                           verboseIter = TRUE)

tune_grid_xgboost <- expand.grid(eta = c(0.0205),
                         nrounds = c(460),
                         max_depth = c(2),
                         min_child_weight = c(4),
                         colsample_bytree = c(0.67),
                         gamma = c(3.4),
                         subsample = c(0.5))


caret_cv_xgboost_foc <- train(y ~ .,
                  data = train_data,
```

```
method = "xgbTree",
tuneGrid = tune_grid_xgboost,
trControl = fitControl)
```