DSA4211 High-Dimensional Statistical Analysis

AY 2021/2022 Semester 1

Project Report

TAN YEE JET
(A0200699Y)

Contents

Executive summary

In this project, our goal is purely prediction. I started off by checking the data for any issues and cleaning them. I found that most train and test data are clean except for two columns – `X55` and `X40`, which are found to have large number of null values and high correlation respectively. Consequently, they were removed from further training and testing. I also did a simple data exploration by checking the training data's distribution. By performing normality testing on each column of the training data using Shapiro-Wilk normality test, I found most columns to be normally distributed except for `X2`, `X22`, `X29`, `X42`, `X48` and `X83`, which upon visual inspection on the their distribution plots, are slightly skewed.

I went on to do a 80 : 20 split on the original training data to give a new 800 rows for training and 200 for validation. For each model, I used the training set for fitting and / or cross-validation, and the validation set for evaluation. The evaluation metrics I used are R^2 and mean squared error (MSE). Although maximizing R^2 is logically equivalent to minimizing MSE, I still put R^2 in as it is the actual evaluation metric used for project scoring.

Moving on to model fitting, I first used the mean of training set `Y` as my baseline prediction. From there, I went on to use as many methods as I can from the lectures within my time limit. The list of methods used include linear regression, stepwise regression, sure independence screening (SIS), iterate SIS (ISIS), ridge regression, lasso, boosting, neural network, principal components analysis (PCA), principal components regression (PCR), partial least squares (PLS) and ensembles of PCA with boosting and neural network respectively. In each model, I started off by fitting `Y` against all other predictors (except the ones dropped from data cleaning – `X55`, `X40`) with the aim of finding a smaller subset of predictors (if applicable) using certain criterias that I determine empirically. The same concept applies when I used principal components (PC) for other models. Instead of doing subset selection on the original predictors, I did it on the PCs.

This results in the list of models in the result section. While it only shows the validation R^2 and predictors used, more detailed information such as predictor selection and their selection criteria can be found in each model's individual section. This is to help keep track of the best predictors from each method. In the end, the best performing model comes from using boosting, with the highest R^2 of 0.615.

One of the things I learnt from this project is be fast when it comes to finding the best model. There are too many methods out there for us to ponder one-by-one before settling down on a handfull. All we have to do is get all reasonable models, run them, record down the results and let the scores do the talking.

One of the things that I can improve from this project is to write better automation when it comes to fitting different models. Currently, I am manually changing a lot of the parameters on the same piece of code when I try different things. Hence, one way is to use pipelining libraries to automate the entire process. All I have to do then, is to list down the different parameters to try and feed the list to the pipeline.

Helper functions

I defined a few helper functions to calculate MSE and R^2. For data exploration, I included a function to plot data distribution across specified columns.

```
# MSE score
mse_score <- function(Yhat, Y) {
  mean((Yhat - Y) ^ 2)
}

# R2 score
r2_score <- function(Yhat, Y) {
  1 - sum((Y - Yhat) ^ 2) / sum((Y - mean(Y)) ^ 2)
}

# plot data distribution
plot_distribution.continuous <- function(df, vars) {
  df %>%
    dplyr::select(all_of(vars)) %>%
    gather() %>%
    ggplot(aes(value)) +
    facet_wrap(~ key, scales = "free") +
    geom_histogram() +
    theme_classic() +
    theme(
      strip.background = element_blank(),
      axis.title = element_blank()
    )
}
```

Raw data

These are the raw data provided for the project.

```
# raw data
train.full <- read_csv('train-xy.csv')
test.full <- read_csv('test-x.csv')
```

Empirical analysis

Check for null values

In the first section, I checked the full train and test data for null values. I found that column `X55` contains 813 and 7987 null values in the train and test set respectively. This amounts to about 80% of null values in both train and test set. Therefore, I dropped `X55` in subsequent processes.

```
# null values
num_na <- function(x) {sum(is.na(x))}
train.num_na <- sapply(train.full, num_na)
test.num_na <- sapply(test.full, num_na)
train.num_na[which(train.num_na > 0)]  # X55 has 813 null values
```

```
## X55
## 813
```

```
test.num_na[which(test.num_na > 0)]  # X55 has 7987 null values
```

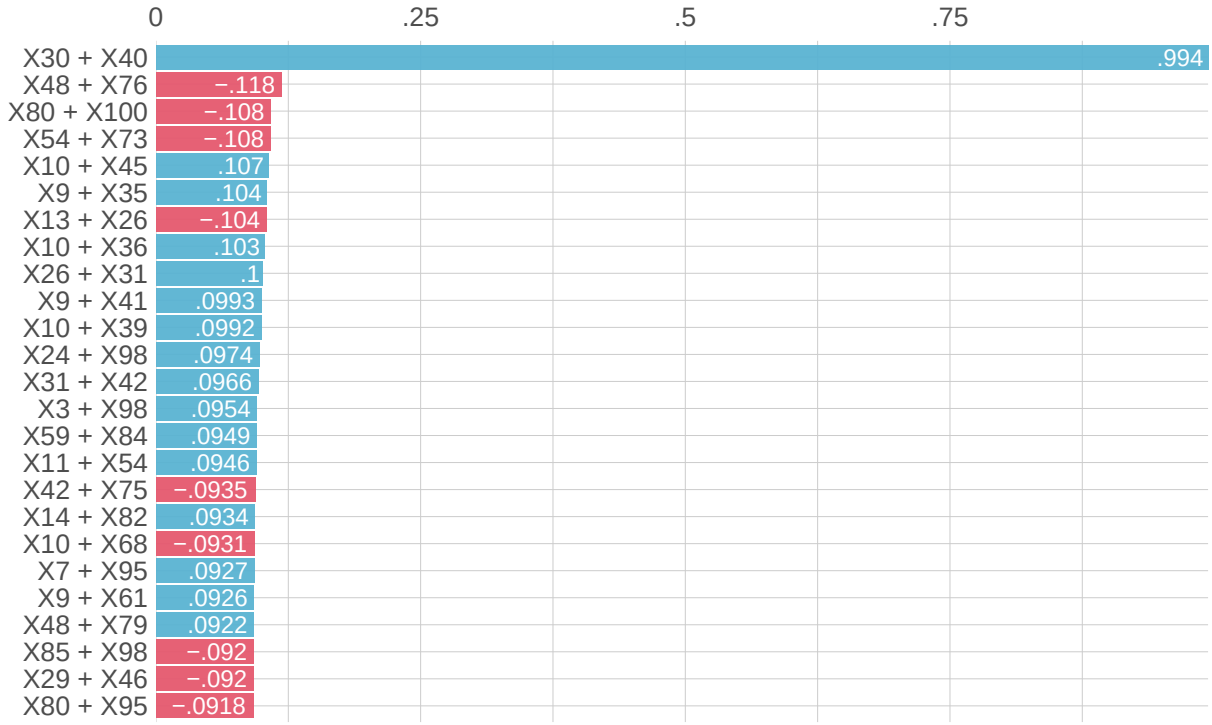
```
##  X55
## 7987
```

Check for correlation

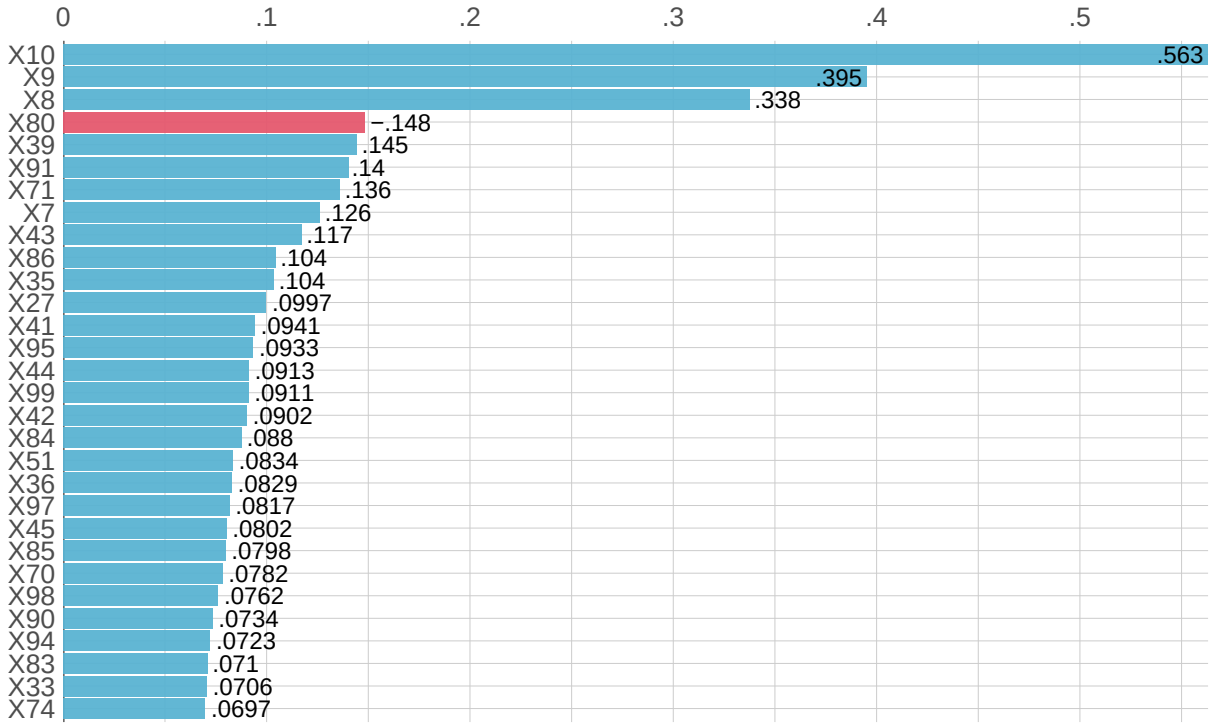
In the second section, I checked for the correlation among predictors to tackle the issue of multicollinearity. I found that `X40` and `X30` are highly correlated, with correlation of 0.994. This means that one of the features have to be dropped. To decide which one to drop, I trained a linear model using all predictors and calculating VIF for each predictor. It turned out that `X40` has the higher VIF value, hence it is the one I dropped. I verified that there are no further correlation among predictors by training a second linear model without `X40`, and this time all VIF values were small enough.

I also checked the predictors which are most correlated to `Y` to give me an idea of which predictors might be most useful. The top 3 most predictors most correlated to `Y` are `X10`, `X9` and `X8`.

```
# top correlated variables
(cor.train.full <- corr_cross(train.full %>% select(-X55, -Y)))
```




```
# top correlated variables w/ Y
(cor.Y <- corr_var(train.full %>% select(-X55), Y))
```




```
# VIF
lm1 <- lm(Y ~ ., train.full %>% select(-X55))
lm1.vif <- vif(lm1)
sort(lm1.vif, decreasing = T)  # X40, X30 collinear
```

```
##       X40       X30        X9       X10        X8       X98       X90       X75
## 91.150238 90.758859  1.179566  1.170184  1.167768  1.161964  1.155376  1.147043
##       X35      X100       X13       X28       X68       X80       X95       X47
##  1.142931  1.141680  1.139218  1.138634  1.136837  1.134784  1.132666  1.132573
##        X5       X74       X25        X7       X29       X82       X54       X42
##  1.131376  1.130752  1.128918  1.127314  1.124561  1.123593  1.123525  1.123061
##       X24        X4       X71       X67       X64       X36       X11       X51
##  1.122399  1.121727  1.121674  1.121412  1.121350  1.120725  1.119806  1.119259
##       X79       X32       X76       X62       X31       X63       X45       X84
##  1.119190  1.118694  1.118367  1.117895  1.117860  1.117711  1.117162  1.116767
##       X41       X14        X6       X39       X23        X3       X78       X94
##  1.116735  1.116541  1.116244  1.115712  1.115547  1.115336  1.114964  1.114778
##       X70       X73       X59       X87       X48       X77       X83       X26
##  1.114766  1.114128  1.113654  1.113527  1.113524  1.112820  1.112333  1.112219
##       X12       X46        X2       X97       X85       X72       X20       X50
##  1.110872  1.110791  1.110204  1.110045  1.110028  1.109571  1.107325  1.107254
```

```
##       X33       X19       X91       X65       X81       X61       X37       X52
##  1.106585  1.106155  1.105496  1.105249  1.104843  1.104747  1.104165  1.101030
##       X99        X1       X60       X92       X27       X15       X22       X57
##  1.100832  1.100234  1.099845  1.099638  1.099627  1.098045  1.097700  1.097683
##       X38       X86       X18       X43       X44       X21       X16       X93
##  1.097488  1.096088  1.095638  1.095540  1.094506  1.093840  1.093601  1.092688
##       X89       X96       X49       X34       X17       X88       X56       X53
##  1.091769  1.091315  1.089734  1.086455  1.085030  1.084618  1.082594  1.080886
##       X66       X58       X69
##  1.076005  1.071287  1.065430
```

```
lm2 <- lm(Y ~ . -X40, train.full %>% select(-X55))
lm2.vif <- vif(lm2)
sort(lm2.vif, decreasing = T)  # no more collinear
```

```
##       X9       X8      X10      X98      X90      X75      X35     X100
## 1.178040 1.167371 1.166933 1.160845 1.149312 1.146852 1.141790 1.141536
##      X28      X13      X80      X47       X5      X74      X95      X68
## 1.138177 1.137126 1.133805 1.132564 1.130762 1.130671 1.129948 1.128441
##       X7      X25      X29      X82      X24      X54      X42       X4
## 1.127064 1.126148 1.124348 1.123514 1.122131 1.122042 1.121987 1.121717
##      X71      X64      X67      X11      X79      X32      X76      X63
## 1.121515 1.121116 1.120733 1.119712 1.119180 1.118518 1.118344 1.117693
##      X62      X31      X51      X45      X14      X41       X3      X36
## 1.116403 1.116378 1.116344 1.115973 1.115620 1.115209 1.114749 1.114222
##      X70      X23      X84      X48      X39      X26      X59      X46
## 1.114088 1.113862 1.113709 1.112584 1.112474 1.112055 1.111177 1.110736
##      X94       X2      X77      X78      X97       X6      X85      X12
## 1.110671 1.110186 1.110105 1.109954 1.109948 1.109892 1.109863 1.109219
##      X72      X73      X50      X20      X33      X87      X19      X83
## 1.109129 1.107442 1.107029 1.106633 1.106456 1.106276 1.105976 1.104794
##      X65      X61      X37      X81      X91      X52      X60      X99
## 1.104704 1.104514 1.104145 1.103210 1.102189 1.101027 1.099844 1.099721
##      X27      X92       X1      X15      X57      X38      X22      X43
## 1.098763 1.098538 1.098420 1.098037 1.097267 1.097240 1.096668 1.094751
##      X44      X86      X18      X21      X89      X93      X16      X96
## 1.094468 1.094122 1.093945 1.093759 1.091560 1.090365 1.089480 1.089282
##      X49      X30      X34      X17      X88      X56      X53      X66
## 1.089119 1.088148 1.086242 1.084877 1.083812 1.081496 1.080592 1.075154
##      X58      X69
## 1.071282 1.065093
```

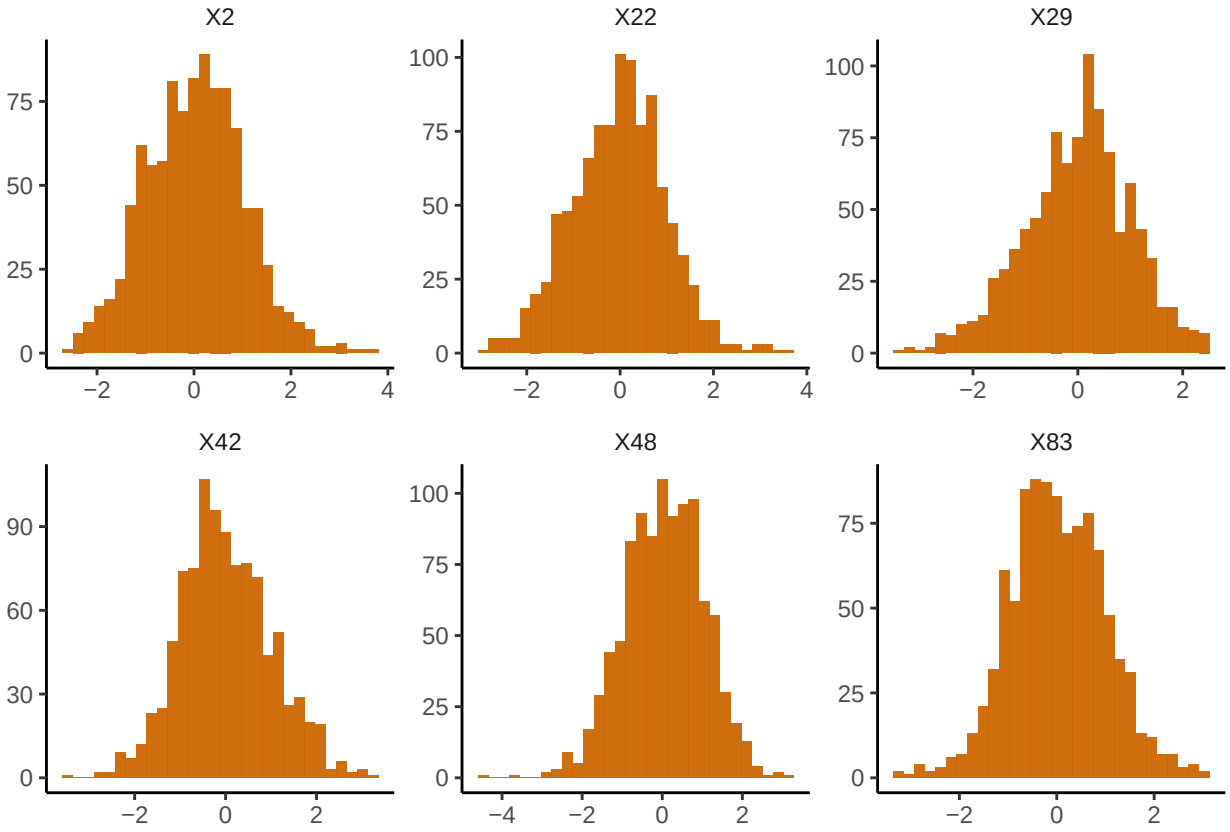
Check data distribution

Lastly, I checked the training data's distribution to give me a better idea of how they look like. I found most columns to be normally distributed using the Shapiro-Wilk normality test. `X2`, `X22`, `X29`, `X42`, `X48` and `X83` were the only ones which were identified as non-normal. Hence, I plotted them out to take a look. It seems that they were indeed skewed, but only slightly.

```
################
# distribution #
################
# check normality
lshap <- lapply(train.full %>% select(-X55), shapiro.test)
lres <- data.frame(t(sapply(lshap, `[`, c("statistic", "p.value"))))

# 6 non-normal features
(X.non_normal <- row.names(lres)[which(lres$p.value < 0.05)])
```

```
## [1] "X2"  "X22" "X29" "X42" "X48" "X83"
```

```
plot_distribution.continuous(train.full %>% select(-X55),
                             c("X2", "X22", "X29", "X42", "X48", "X83"))
```



Split data

In my first submission, I trained my models using a 90 : 10 split. However, I found that the validation set was not very representative of the test set due to the discrepency of my validation R^2 and the resulting test R^2. Hence, I increased its proportion. In this submission, I used a 80 : 20 split.

```
# train validation 80:20 split
set.seed(42)
train.idx <- sample(1:nrow(train.full), 0.8 * nrow(train.full))

# train
train <- train.full[train.idx,]

# validation
valid <- train.full[-train.idx,]
```

Models

Here are all the models I tried. All of them are adapted from lectures. Two of the methods that I have not tried are non-linear dimension reduction and hidden markov model (HMM). For each model, I will explain the training and validation process, and show the recorded results. Note that the codes do not represent all the models in the results because I got the different models by tweaking model parameters in the same code chunk. Additionally, they are not ran for outputs because some of them are computationally intensive. Note also that from here onwards, "all predictors" shall mean all original predictors except for `X55` and `X40`.

Baseline

I used the mean of training label as my baseline prediction. No model trained should be below the baseline validation score.

```
# mean training labels
valid.Yhat <- rep(mean(train$Y), length(valid$Y))
(baseline.valid.r2 <- r2_score(valid.Yhat, valid$Y))
(baseline.valid.mse <- mse_score(valid.Yhat, valid$Y))
(baseline.valid.r2 <- r2_score(valid.Yhat, valid$Y))
```

Model	Validation R2	Predictors Used	Predictors Selected	Selection Criteria
baseline	-0.002	NA	NA	NA

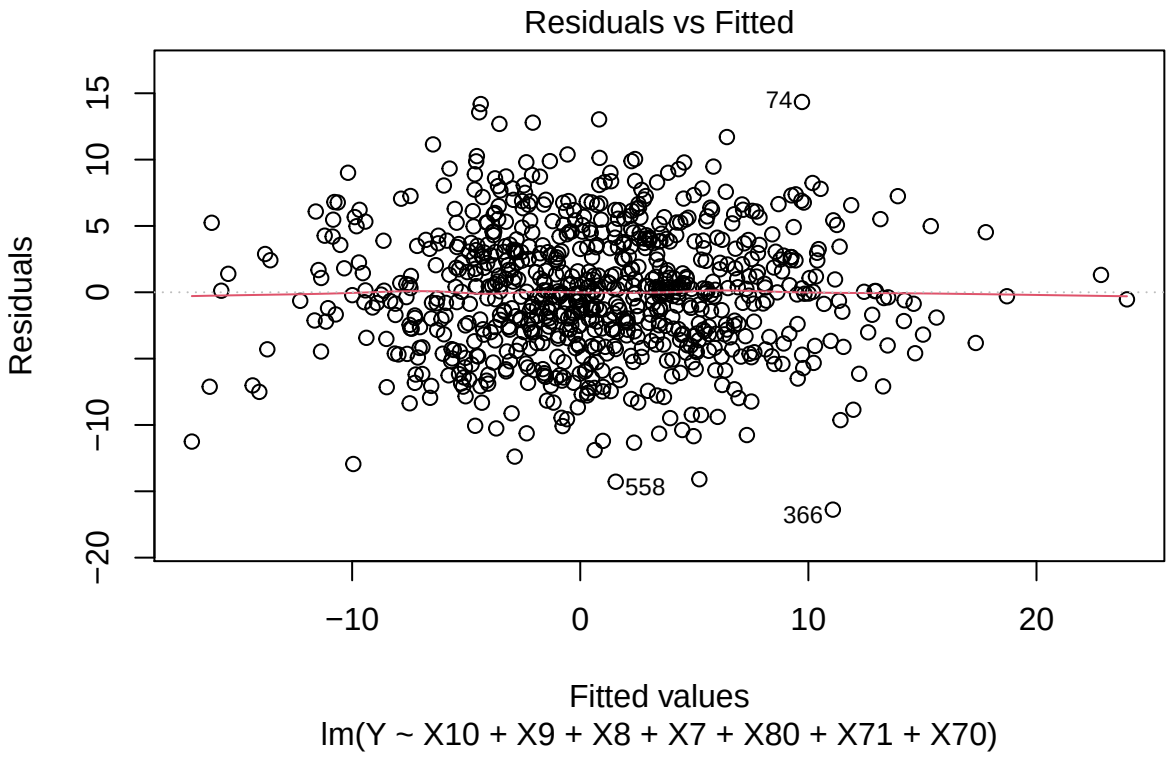
Linear regression

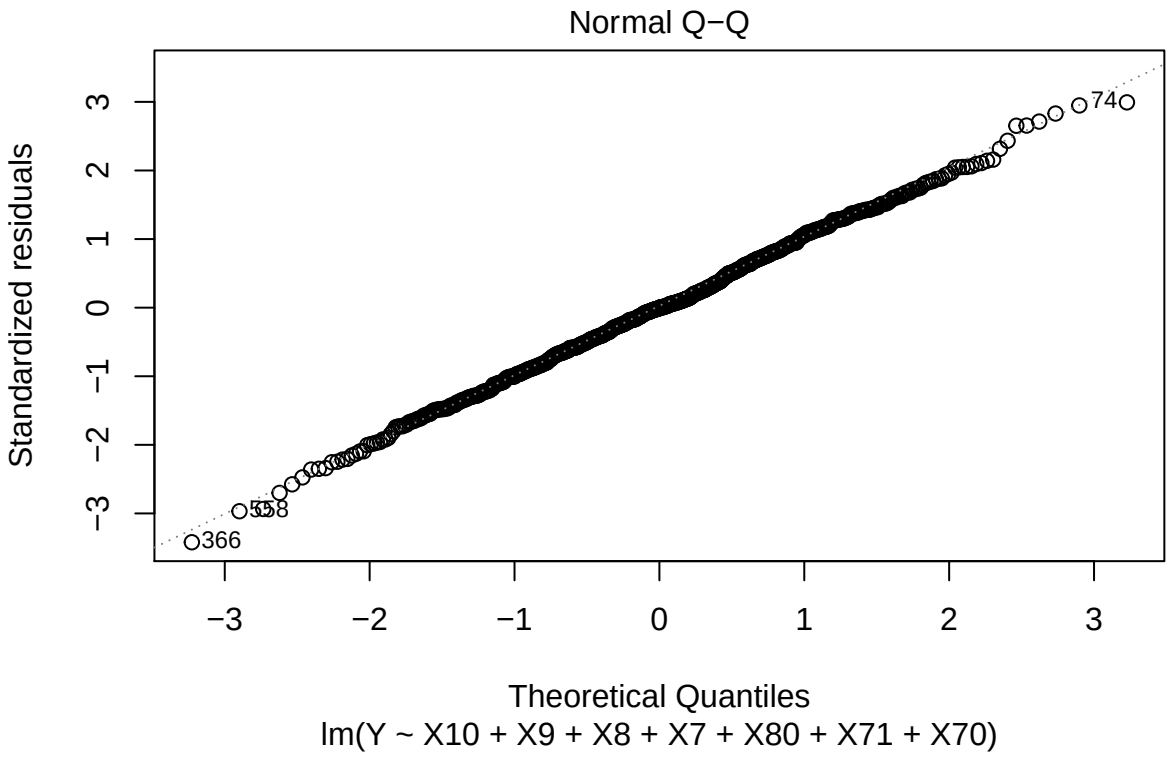
I started out with linear regression because it is one of the simplest and straightforward method to do feature selection with. First, I trained `lm1` on all predictors and selected a subset of significant predictors for `lm2`. Then, I trained `slm1` using stepwise regression and selected a subset of significant of predictors for `lm3`. I selected the predictors based on their p-values, and I only selected the ones where $p < 0.001$. This is because I want to choose the most significant predictors. In this case, `lm2` performed the best and its diagnostics are shown to prove that it is quite adequate.

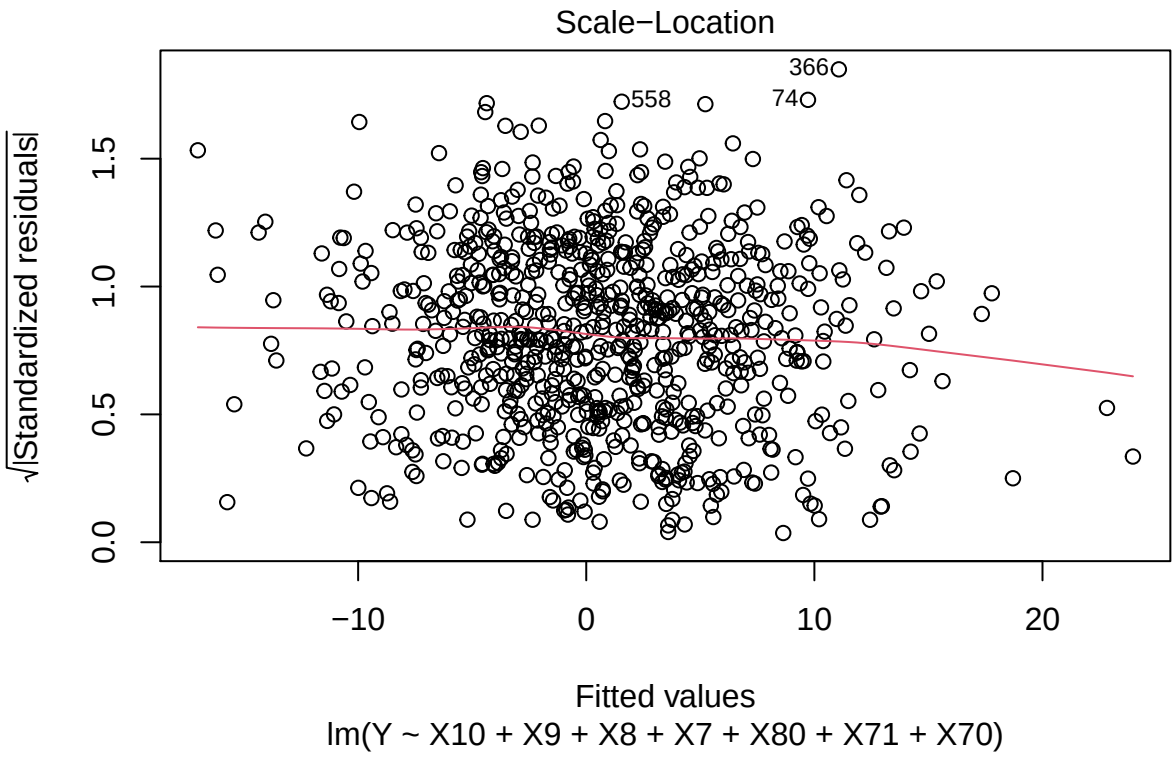
```
# all variables except X40 X55
lm1 <- lm(Y ~ ., train %>% select(-X40, -X55))
s.lm1 <- summary(lm1)
pval.lm1 <- s.lm1$coefficients[-1, 4]
names(sort(pval.lm1[which(pval.lm1 < 0.001)]))

# variables w/ p < 0.001 in lm1
```

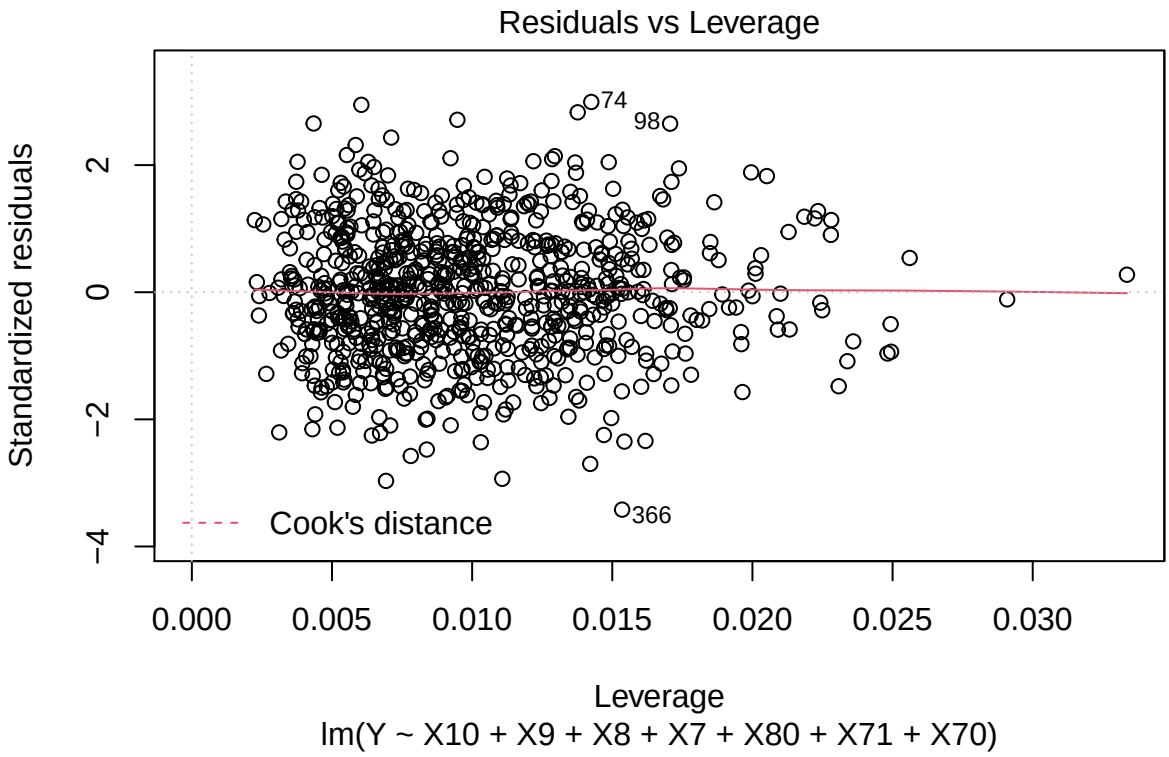
```r
lm2 <- lm(Y ~ X10 + X9 + X8 + X7 + X80 + X71 + X70, train)
s.lm2 <- summary(lm2)
pval.lm2 <- s.lm2$coefficients[-1, 4]
names(sort(pval.lm2[which(pval.lm2 < 0.001)]))

# stepwise regression w/ lm1
slm1 <- step(lm1, direction = "both", steps = 100000)
s.slm1 <- summary(slm1)
pval.slm1 <- s.slm1$coefficients[-1, 4]
names(sort(pval.slm1[which(pval.slm1 < 0.001)]))

# variables w/ p < 0.001 in slm1
lm3 <- lm(Y ~ X10 + X9 + X8 + X7 + X80 + X70 + X71 + X91, train)
s.lm3 <- summary(lm3)
pval.lm3 <- s.lm3$coefficients[-1, 4]
names(sort(pval.lm3[which(pval.lm3 < 0.001)]))

# validation
train.Yhat <- predict(lm3, train)
valid.Yhat <- predict(lm3, valid)
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)
mse_score(valid.Yhat, valid$Y)
```

```r
# diagnostics
lm2 <- lm(Y ~ X10 + X9 + X8 + X7 + X80 + X71 + X70, train)
plot(lm2)  # minor violation of assumption
```


Residuals vs Fitted
Residuals
Fitted values
lm(Y ~ X10 + X9 + X8 + X7 + X80 + X71 + X70)
74
558
366
15
10
5
0
−10
−20
−10
0
10
20


Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(Y ~ X10 + X9 + X8 + X7 + X80 + X71 + X70)
74
366
−3
−2
−1
0
1
2
3


Scale−Location
366
558
74
√|Standardized residuals|
0.0
0.5
1.0
1.5
−10
0
10
20
Fitted values
lm(Y ~ X10 + X9 + X8 + X7 + X80 + X71 + X70)



Model	Validation R2	Predictors Used	Predictors Selected
lm1	0.531	all except X55 X40	X10 X9 X8 X7 X80 X71 X70
lm2	0.598	X10 X9 X8 X7 X80 X71 X70	unchanged
slm1	0.559	all except X55 X40	X10 X9 X8 X7 X80 X70 X71 X91
lm3	0.597	X10 X9 X8 X7 X80 X70 X71 X91	unchanged

Model	Selection Criteria
lm1	$p < 0.001$
lm2	$p < 0.001$
slm1	$p < 0.001$
lm3	$p < 0.001$

Sure independence screening (SIS) and iterate SIS (ISIS)

I decided to try out SIS and ISIS for feature selection. In this case, I trained SIS `sis1` without regularization and ISIS `sis2`, which is just SIS with regularization. In this case, I got exactly the same score for both models. However, I have failed to select a subset of feature from both models as the predictors they suggested were simply too many. It is not meaningful to continue using their suggested predictors as I might just as well use the full set of predictors. Additionally, I was confident that smaller subsets like the ones in linear regression are possibly better.

```
# data matrix
train.mat <- model.matrix(Y ~ ., train %>% select(-X55, -X40))[,-1]
valid.mat <- model.matrix(Y ~ ., valid %>% select(-X55, -X40))[,-1]

# SIS without regularization
sis1 = SIS(train.mat, train$Y, family='gaussian', iter = TRUE)
sis1$sis.ix0

train.Yhat <- predict(sis1, newx = train.mat, type = 'response')
valid.Yhat <- predict(sis1, newx = valid.mat, type = 'response')
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)
mse_score(valid.Yhat, valid$Y)

# predictors w/ non-zero coefficients
sis1.coef = predict(sis1, type = "coefficients", s = bestlam)
rownames(sis1.coef)[which(sis1.coef != 0)]


# ISIS with regularization
sis2 = SIS(train.mat, train$Y, family = 'gaussian', tune = 'bic')
sis2$ix
train.Yhat <- predict(sis2, newx = train.mat, type = 'response')
valid.Yhat <- predict(sis2, newx = valid.mat, type = 'response')
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)
mse_score(valid.Yhat, valid$Y)

# predictors w/ non-zero coefficients
sis2.coef = predict(sis2, type = "coefficients", s = bestlam)
rownames(sis2.coef)[which(sis2.coef != 0)]
```

Model	Validation R2	Predictors Used	Predictors Selected	Selection Criteria
sis1	0.530	all except X55 X40	too many	NA
sis2	0.530	all except X55 X40	too many	NA

Ridge regression and lasso

I decided to try both ridge regression `ridge1` and lasso `lasso1` for feature selection. Note that I used cross-validation to select the best λ for both ridge regression and lasso. While ridge regression does not do feature selection by itself, I could still do it by selecting the predictors when its absolute coefficients are larger than a certain threshold. I varied the coefficient thresholds such that the features selected would not be too many. In this case, I only selected the predictors with absolute coefficients larger than 0.7 (trained in `ridge2`) and 0.5 (trained in `ridge3`). For the lasso, the model trained after feature selection `lasso2` is worse than the original model `lasso1`. In my results, the lasso method yields the best model `lasso1`. Its score, however is not too far away from the best ridge regression model `ridge3`.

```
# data matrix
train.mat <- model.matrix(Y ~ ., train %>% select(Y, X7, X8, X9, X10, X80))[,-1]
valid.mat <- model.matrix(Y ~ ., valid %>% select(Y, X7, X8, X9, X10, X80))[,-1]

#########
# lasso #
#########
# coefficient plot
grid <- 10^seq(10, -2, length = 100)
lasso1 <- glmnet(train.mat, train$Y, alpha = 1, lambda = grid)
plot(lasso1)

# cross validation to select best lambda
set.seed(42)
cv.out <- cv.glmnet(train.mat, train$Y, alpha = 1)
plot(cv.out)
(bestlam <- cv.out$lambda.min)

# validation
train.Yhat <- predict(lasso1, s = bestlam, newx = train.mat)
valid.Yhat <- predict(lasso1, s = bestlam, newx = valid.mat)
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)
mse_score(valid.Yhat, valid$Y)

# predictors w/ non-zero coefficients
lasso.coef = predict(lasso1, type = "coefficients", s = bestlam)
rownames(lasso.coef)[which(lasso.coef != 0)]


#########
# ridge #
```

```
#########
# coefficient plot
grid <- 10^seq(10, -2, length = 100)
ridge1 <- glmnet(train.mat, train$Y, alpha = 0, lambda = grid)
plot(ridge1)

# cross validation to select best lambda
set.seed(42)
cv.out <- cv.glmnet(train.mat, train$Y, alpha = 0)
plot(cv.out)
(bestlam <- cv.out$lambda.min)

# validation
train.Yhat <- predict(ridge1, s = bestlam, newx = train.mat)
valid.Yhat <- predict(ridge1, s = bestlam, newx = valid.mat)
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)
mse_score(valid.Yhat, valid$Y)

# predictors sorted by coefficient magnitude
ridge.coef = predict(ridge1, type = "coefficients", s = bestlam)
tibble(X = ridge.coef@Dimnames[[1]], coef = ridge.coef@x)
  %>% arrange(desc(abs(coef)))
```

Model	Validation R2	Predictors Used	Predictors Selected	Selection Criteria
ridge1	0.549	all except X55 X40	X7 X8 X9 X10 X80	abs(coefficient) > 0.7
ridge1	0.549	all except X55 X40	X7 X8 X9 X10 X80 X71 X70 X91	abs(coefficient) > 0.5
ridge2	0.587	X7 X8 X9 X10 X80	unchanged	abs(coefficient) > 0.8
ridge3	0.598	X7 X8 X9 X10 X80 X71 X70 X91	X7 X8 X9 X10 X80	abs(coefficient) > 0.8
lasso1	0.603	all except X55 X40	X7 X8 X9 X10 X80	coefficient non-zero
lasso2	0.588	X7 X8 X9 X10 X80	unchanged	coefficient non-zero

Boosting

I decided to skip decision trees and random forest because theoretically speaking, it makes sense that the boosting algorithm is better and will yield a better model. Hence, I did not want to waste time on other tree methods. Note that I used grid search to select the best number of trees, depth and shrinkage value for boosting. I performed feature selection on boosting using the relative influence of each predictor on each boosting tree. Similar to feature selection with ridge regression, I varied the relative influence thresholds such that the features selected would not be too many. In this case, I trained `boost1` with all predictors and selected three subsets of features with thresholds 14 (trained in `boost2`), 2 (trained in `boost3`) and 1 (trained in `boost4`). With this, the best model appears to be `boost4`. For reference, I also attached the hyperparameter grid I used for search as well as the best hyperparameters for each model.

```
# create hyper-parameter grid
(grid <- expand.grid(depth = seq(1, 6, 1), shrinkage = 10^seq(-3, -1, 1)))
best.n <- rep(NA, nrow(grid))
train.mse <- rep(NA, nrow(grid))
valid.mse <- rep(NA, nrow(grid))
train.r2 <- rep(NA, nrow(grid))
valid.r2 <- rep(NA, nrow(grid))

# try different subsets of predictors
train.tmp <- train %>% select(Y, X10, X9, X91, X8, X80, X86, X70, X7, X71)
valid.tmp <- valid %>% select(Y, X10, X9, X91, X8, X80, X86, X70, X7, X71)

# grid search for best no. of trees, depth, shrinkage value
for (i in 1:nrow(grid)) {
  print(paste("Iteration", i))

  set.seed(42)
  boost <- gbm(
      Y ~ .,
      data = train.tmp,
      distribution = "gaussian",
      n.trees = 15000,
      shrinkage = grid$shrinkage[i],
      interaction.depth = grid$depth[i]
  )

  tmp.best.n <- gbm.perf(boost, method = "OOB")
  train.Yhat <- predict(boost, newdata = train, n.trees = tmp.best.n)
  valid.Yhat <- predict(boost, newdata = valid, n.trees = tmp.best.n)

  best.n[i] <- tmp.best.n
```

```
  train.mse[i] <- mse_score(train.Yhat, train$Y)
  valid.mse[i] <- mse_score(valid.Yhat, valid$Y)
  train.r2[i] <- r2_score(train.Yhat, train$Y)
  valid.r2[i] <- r2_score(valid.Yhat, valid$Y)

  print(paste(best.n[i], train.mse[i], valid.mse[i], train.r2[i], valid.r2[i]))
}

# store results
(boost.res <- cbind(grid, best.n, train.mse, valid.mse, train.r2, valid.r2))
boost.res %>% write_csv('boost4.csv')
boost.res <- read_csv('boost4.csv')

# best results by validation MSE
(best.boost <- boost.res[which.min(boost.res$valid.mse),])
best.shrinkage <- best.boost$shrinkage
best.depth <- best.boost$depth

# best model
set.seed(42)
boost <- gbm(
    Y ~ .,
    data = train.tmp,
    distribution = "gaussian",
    n.trees = 15000,
    shrinkage = best.shrinkage,
    interaction.depth = best.depth
)
(best.n <- gbm.perf(boost, method = "OOB", plot.it = F))
train.Yhat <- predict(boost, newdata = train.tmp, n.trees = best.n)
valid.Yhat <- predict(boost, newdata = valid.tmp, n.trees = best.n)
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)
mse_score(valid.Yhat, valid$Y)

# feature importance
summary(boost)
```

```
# hyper-parameter grid
(grid <- expand.grid(depth = seq(1, 6, 1), shrinkage = 10^seq(-3, -1, 1)))
```

```
##    depth shrinkage
## 1      1     0.001
## 2      2     0.001
```

```
## 3      3     0.001
## 4      4     0.001
## 5      5     0.001
## 6      6     0.001
## 7      1     0.010
## 8      2     0.010
## 9      3     0.010
## 10     4     0.010
## 11     5     0.010
## 12     6     0.010
## 13     1     0.100
## 14     2     0.100
## 15     3     0.100
## 16     4     0.100
## 17     5     0.100
## 18     6     0.100
```

```
# best hyper-parameters for each tree
boost.res1 <- read_csv('boost1.csv')
boost.res2 <- read_csv('boost2-2.csv')
boost.res3 <- read_csv('boost3.csv')
boost.res4 <- read_csv('boost4.csv')
rbind(
  boost.res1[which.min(boost.res1$valid.mse),],
  boost.res2[which.min(boost.res2$valid.mse),],
  boost.res3[which.min(boost.res3$valid.mse),],
  boost.res4[which.min(boost.res4$valid.mse),]
)
```

```
## # A tibble: 4 x 7
##   depth shrinkage best.n train.mse valid.mse train.r2 valid.r2
##   <dbl>     <dbl>  <dbl>     <dbl>     <dbl>    <dbl>    <dbl>
## 1     2     0.001   6164      19.2      20.6    0.668    0.596
## 2     2     0.001   6400      19.3      20.2    0.667    0.603
## 3     1     0.1      246      22.7      21.9    0.609    0.570
## 4     1     0.1      253      18.0      19.6    0.689    0.615
```

Model	Validation R2	Predictors Used	Predictors Selected
boost1	0.596	all except X55 X40	X10 X9 X8 X80 X7 X70 X71 X91 X86 X39 X95
boost1	0.596	all except X55 X40	X10 X9 X8

Model	Validation R2	Predictors Used	Predictors Selected
boost2	0.601	X10 X9 X8 X80 X7 X70 X71 X91 X86 X39 X95	X10 X9 X91 X8 X80 X86 X70 X7 X71
boost3	0.570	X10, X9, X8	unchanged
boost4	0.615	X10 X9 X91 X8 X80 X86 X70 X7 X71	unchanged

Model	Selection Criteria
boost1	rel.inf > 1
boost1	rel.inf > 10
boost2	rel.inf > 7.5
boost3	rel.inf > 30
boost4	rel.inf > 9

Neural network

I decided to try a simple neural network – a multi-layer perceptron just to see how it would perform. I was not too optimistic about it as I think the data points are too few. Note that I used grid search to select the best hidden layer(s) and regularization penalty. For reference, I attached the hyperparameter grid I used for search as well as the best hyperparameters used to train the final neural network. In this case, the final model consists of a regularization penalty of 0.01 and one hidden layer with size 512.

```
# export and read for python
train %>% write_csv('train.csv')
valid %>% write_csv('valid.csv')
```

```
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# prepare data
train = pd.read_csv('train.csv')
valid = pd.read_csv('valid.csv')

xtr, ytr = train.drop(columns=['Y', 'X40', 'X55']).values, train['Y'].values
xvl, yvl = valid.drop(columns=['Y', 'X40', 'X55']).values, valid['Y'].values

# prepare ml pipeline
pipeline = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('nn', MLPRegressor(max_iter=2000, early_stopping=True, solver='sgd',
    momentum=0.7, learning_rate='adaptive'))
])

# create hyper-parameter grid
grid = {
    'nn__hidden_layer_sizes': [(4096), (2048), (1024), (512), (1024, 1024)],
    'nn__alpha': [0.00001, 0.0001, 0.001, 0.01]
}

# grid search for best hidden layers, learning rate
search = GridSearchCV(
```

```
    pipeline,
    param_grid=grid,
    verbose=3,
    scoring='r2',
    cv=3
)

# fit neural network
search.fit(xtr, ytr)

# grid search result
print(search.best_estimator_)

# save model
# pickle.dump(search, open('nn1.sav', 'wb'))

# load model
search = pickle.load(open('nn1.sav', 'rb'))

# validation
yhattr = search.predict(xtr)
yhatvl = search.predict(xvl)
print(r2_score(ytr, yhattr))
print(r2_score(yvl, yhatvl))
print(mean_squared_error(yvl, yhatvl))
```

```
import pickle
from pprint import pprint

# saved model
search = pickle.load(open('nn1.sav', 'rb'))

# hyper-parameter grid
grid = {
    'nn__hidden_layer_sizes': [(4096), (2048), (1024), (512), (1024, 1024)],
    'nn__alpha': [0.00001, 0.0001, 0.001, 0.01]
}
pprint(grid)

# model's best parameter
```

```
## {'nn__alpha': [1e-05, 0.0001, 0.001, 0.01],
##  'nn__hidden_layer_sizes': [4096, 2048, 1024, 512, (1024, 1024)]}
```

```
print(search.best_estimator_)
```

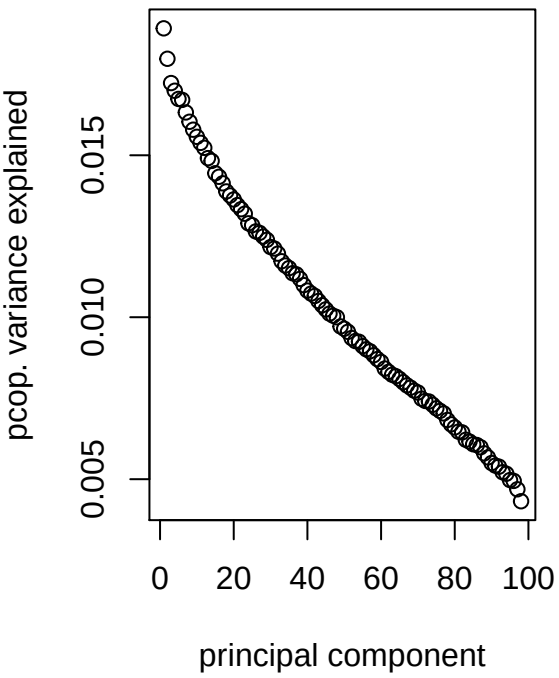
```
## Pipeline(steps=[('scaler', StandardScaler()),
##                 ('nn',
##                  MLPRegressor(alpha=0.01, early_stopping=True,
##                               hidden_layer_sizes=512, learning_rate='adaptive',
##                               max_iter=2000, momentum=0.7, solver='sgd'))])
```

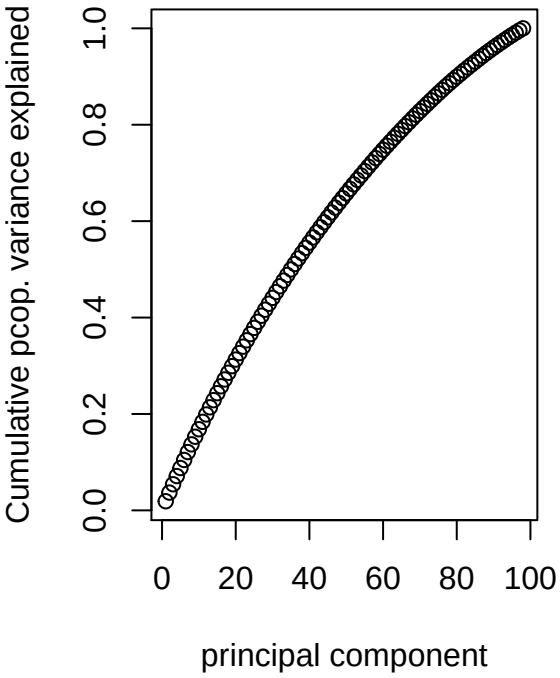
Model	Validation R2	Predictors Used	Predictors Selected	Selection Criteria
nn1	0.528	all except X55 X40	NA	NA

Principal components analysis (PCA)

I decided to try PCA to see if I can use the fitted principal components (PC) to better the results. However, it was a bit disappointing since the decrease in proportion varianced explained by the PC is too slow, although there is a weak elbow in the scree plot after the third PC. Nevertheless, I went on to try several model using the PCs obtained.

```
# pca
pca1 <- prcomp(x = train %>% select(-Y, -X40, -X55), scale = TRUE)

# variance
pc.var = (pca1$sdev) ^ 2

# % of variance
perc.var.explained = pc.var / sum(pc.var)

# plot
par(mfrow = c(1, 2))
plot(perc.var.explained,
     xlab = "principal component",
     ylab = "pcop. variance explained",
     type = 'b')
plot(
  cumsum(perc.var.explained),
  xlab = "principal component",
  ylab = "Cumulative pcop. variance explained",
  type = 'b'
)
```


pcop. variance explained
0.005
0.010
0.015
0
20
40
60
80
100
principal component


Cumulative pcop. variance explained
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
principal component

Principal components regression (PCR) and Partial least squares (PLS)

The first two models I tried with PCs are PCR and PLS. In this case, I yielded the same result for both PCR `pcr1` and PLS `pls1`. The best number of components are however, vastly different. PCR gave the best number of components to be 98 whereas PLS gave 10. They are attached below as reference.

```
#######
# PCR #
#######
set.seed(42)
pcr1 = pcr(
  Y ~ .,
  data = train %>% select(-X40, -X55),
  scale = TRUE,
  validation = "CV"
)
summary(pcr1)
validationplot(pcr1, val.type = "MSEP")

# get best no. of components
msep <- MSEP(pcr1)
which.min(msep$val[1,1,])
df <- data.frame(adj.cv = msep$val[1,1,])
tibble(comp = rownames(df), adj.cv = df$adj.cv) %>% arrange(adj.cv)

# validation
train.Yhat = predict(pcr1, train, ncomp = 20)
valid.Yhat = predict(pcr1, valid, ncomp = 20)
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)
mse_score(valid.Yhat, valid$Y)

#######
# PLS #
#######
set.seed(42)
pls1 = plsr(
  Y ~ .,
  data = train %>% select(-X40, -X55),
  scale = TRUE,
  validation = "CV"
)
summary(pls1)
```

```
validationplot(pls1, val.type = "MSEP")

# get best no. of components
msep <- MSEP(pls1)
which.min(msep$val[1,1,])
df <- data.frame(adj.cv = msep$val[1,1,])
tibble(comp = rownames(df), adj.cv = df$adj.cv) %>% arrange(adj.cv)

# validation
train.Yhat = predict(pls1, train, ncomp = 10)
valid.Yhat = predict(pls1, valid, ncomp = 10)
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)
mse_score(valid.Yhat, valid$Y)
```

```
# PCR
set.seed(42)
pcr1 = pcr(
  Y ~ .,
  data = train %>% select(-X40, -X55),
  scale = TRUE,
  validation = "CV"
)

# get best no. of components
msep <- MSEP(pcr1)
which.min(msep$val[1,1,])
```

```
## 98 comps
##       99
```

```
# PLS
set.seed(42)
pls1 = plsr(
  Y ~ .,
  data = train %>% select(-X40, -X55),
  scale = TRUE,
  validation = "CV"
)

# get best no. of components
msep <- MSEP(pls1)
which.min(msep$val[1,1,])
```

```
## 10 comps
##       11
```

Model	Validation R2	Predictors Used	Predictors Selected	Selection Criteria
pcr1	0.531	all except X55 X40	NA	NA
pls1	0.531	all except X55 X40	NA	NA

Principal components analysis + boosting

Since boosting gave us the best result so far with `boost4`, I decided to fit another set of boosted trees with the exact same procedure as the previous boosting section, except with PCs. However, the result was not as good.

```
# get PCs
train.pc <- data.frame(cbind(Y = train$Y, pca1$x))
valid.pc <- data.frame(cbind(
  Y = valid$Y, predict(pca1, newdata = valid %>% select(-Y, -X40, -X55))
))

# create hyper-parameter grid
(grid <- expand.grid(depth = seq(1, 6, 1), shrinkage = 10^seq(-3, -1, 1)))
best.n <- rep(NA, nrow(grid))
train.mse <- rep(NA, nrow(grid))
valid.mse <- rep(NA, nrow(grid))
train.r2 <- rep(NA, nrow(grid))
valid.r2 <- rep(NA, nrow(grid))

# try different subsets of PCs
train.tmp <- train.pc %>% select(Y, PC1)
valid.tmp <- valid.pc %>% select(Y, PC1)

# grid search for best no. of trees, depth, shrinkage value
for (i in 1:nrow(grid)) {
  print(paste("Iteration", i))

  set.seed(42)
  boost <- gbm(
      Y ~ .,
      data = train.tmp,
      distribution = "gaussian",
      n.trees = 25000,
      shrinkage = grid$shrinkage[i],
      interaction.depth = grid$depth[i]
  )

  tmp.best.n <- gbm.perf(boost, method = "OOB")
  train.Yhat <- predict(boost, newdata = train.tmp, n.trees = tmp.best.n)
  valid.Yhat <- predict(boost, newdata = valid.tmp, n.trees = tmp.best.n)

  best.n[i] <- tmp.best.n
  train.mse[i] <- mse_score(train.Yhat, train$Y)
  valid.mse[i] <- mse_score(valid.Yhat, valid$Y)
```

```
  train.r2[i] <- r2_score(train.Yhat, train$Y)
  valid.r2[i] <- r2_score(valid.Yhat, valid$Y)

  print(paste(best.n[i], train.mse[i], valid.mse[i], train.r2[i], valid.r2[i]))
}

# store results
(boost.res <- cbind(grid, best.n, train.mse, valid.mse, train.r2, valid.r2))
boost.res %>% write_csv('pcboost4.csv')
boost.res <- read_csv('pcboost4.csv')

# best results by validation MSE
(best.boost <- boost.res[which.min(boost.res$valid.mse),])
best.shrinkage <- best.boost$shrinkage
best.depth <- best.boost$depth

# best model
set.seed(42)
boost <- gbm(
    Y ~ .,
    data = train.tmp,
    distribution = "gaussian",
    n.trees = 25000,
    shrinkage = best.shrinkage,
    interaction.depth = best.depth
)
(best.n <- gbm.perf(boost, method = "OOB", plot.it = F))
train.Yhat <- predict(boost, newdata = train.tmp, n.trees = best.n)
valid.Yhat <- predict(boost, newdata = valid.tmp, n.trees = best.n)
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)
mse_score(valid.Yhat, valid$Y)

# feature importance
summary(boost)
```

Model	Validation R2	Predictors Used	Predictors Selected
pcb1	0.502	all PC	PC1
pcb1	0.502	all PC	PC1 PC2 PC3
pcb1	0.502	all PC	PC1 PC2 PC3 PC76 *
pcb2	0.304	PC1	NA
pcb3	0.291	PC1 PC2 PC3	unchanged
pcb4	0.418	PC1 PC2 PC3 PC76 *	PC1

Model	Validation R2	Predictors Used	Predictors Selected
pcb4	0.418	PC1 PC2 PC3 PC76 *	PC1 PC2

Model	Selection Criteria
pcb1	rel.inf > 14
pcb1	rel.inf > 2
pcb1	rel.inf > 1
pcb2	NA
pcb3	rel.inf > 23
pcb4	rel.inf > 11
pcb4	rel.inf > 5

Note:

PC1 PC2 PC3 PC76 * = PC1 PC2 PC3 PC76 PC18 PC6 PC89 PC38 PC27 PC19 PC77 PC64 PC13 PC97 PC73 PC82 PC44 PC86 PC67 PC80 PC41 PC45 PC12 PC52 PC65 PC35 PC68 PC92 PC51

Principal components analysis + neural network

I decided to give neural network another go, but this time using PCs. Once again, I used the exact same procedure as the previous neural network section. However, the result was about the same.

```
# export and read for python
train.pc <- data.frame(cbind(Y = train$Y, pca1$x))
valid.pc <- data.frame(cbind(
  Y = valid$Y, predict(pca1, newdata = valid %>% select(-Y, -X40, -X55))
))
train.pc %>% write_csv('train-pc.csv')
valid.pc %>% write_csv('valid-pc.csv')
```

```
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# prepare data
train_pc = pd.read_csv('train-pc.csv')
valid_pc = pd.read_csv('valid-pc.csv')

xtr, ytr = train_pc.drop(columns=['Y']).values, train_pc['Y'].values
xvl, yvl = valid_pc.drop(columns=['Y']).values, valid_pc['Y'].values

# prepare ml pipeline
pipeline = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('nn', MLPRegressor(max_iter=2000, early_stopping=True, solver='sgd',
    momentum=0.7, learning_rate='adaptive'))
])

# create hyper-parameter grid
grid = {
    'nn__hidden_layer_sizes': [(4096), (2048), (1024), (512), (1024, 1024)],
    'nn__alpha': [0.00001, 0.0001, 0.001, 0.01]
}
```

```
# grid search for best hidden layers, learning rate
search = GridSearchCV(
    pipeline,
    param_grid=grid,
    verbose=3,
    scoring='r2',
    cv=3
)

# fit neural network
search.fit(xtr, ytr)

# grid search result
print(search.best_estimator_)

# save model
pickle.dump(search, open('pcnn1.sav', 'wb'))

# load model
search = pickle.load(open('pcnn1.sav', 'rb'))

# validation
yhattr = search.predict(xtr)
yhatvl = search.predict(xvl)
print(r2_score(ytr, yhattr))
print(r2_score(yvl, yhatvl))
print(mean_squared_error(yvl, yhatvl))
```

Model	Validation R2	Predictors Used	Predictors Selected	Selection Criteria
pcnn1	0.525	all PC	NA	NA

Results

Here, I combine the results of all models trained. The model with best validation R^2 is `boost4`. Hence, it will be my final model.

Model	Validation R2	Predictors Used
baseline	-0.002	NA
lm1	0.531	all except X55 X40
lm2	0.598	X10 X9 X8 X7 X80 X71 X70
slm1	0.559	all except X55 X40
lm3	0.597	X10 X9 X8 X7 X80 X70 X71 X91
sis1	0.530	all except X55 X40
sis2	0.530	all except X55 X40
ridge1	0.549	all except X55 X40
ridge2	0.587	X7 X8 X9 X10 X80
ridge3	0.598	X7 X8 X9 X10 X80 X71 X70 X91
lasso1	0.603	all except X55 X40
lasso2	0.588	X7 X8 X9 X10 X80
boost1	0.596	all except X55 X40
boost2	0.596	X10 X9 X8 X80 X7 X70 X71 X91 X86 X39 X95
boost3	0.570	X10, X9, X8
boost4	0.615	X10 X9 X91 X8 X80 X86 X70 X7 X71
nn1	0.528	all except X55 X40
pca1	NA	all except X55 X40
pcr1	0.531	all except X55 X40
pls1	0.531	all except X55 X40
pcboost1	0.502	all PC
pcboost2	0.304	PC1
pcboost3	0.291	PC1 PC2 PC3
pcboost4	0.418	PC1 PC2 PC3 PC76 *
pcnn1	0.525	all PC

Note:

PC1 PC2 PC3 PC76 * = PC1 PC2 PC3 PC76 PC18 PC6 PC89 PC38 PC27 PC19 PC77 PC64 PC13 PC97 PC73 PC82 PC44 PC86 PC67 PC80 PC41 PC45 PC12 PC52 PC65 PC35 PC68 PC92 PC51

Submission

I retrained the best model `boost4` using the best hyper-parameters and best features previously saved with the full training data and sent it for submission.

```
# previously saved results
boost.res <- read_csv('boost4.csv')

# best results by validation MSE
(best.boost <- boost.res[which.min(boost.res$valid.mse),])
best.shrinkage <- best.boost$shrinkage
best.depth <- best.boost$depth

# full data
train.tmp <- train.full %>% select(Y, X10, X9, X91, X8, X80, X86, X70, X7, X71)
test.tmp <- test.full %>% select(X10, X9, X91, X8, X80, X86, X70, X7, X71)

# best model
set.seed(42)
boost <- gbm(
    Y ~ .,
    data = train.tmp,
    distribution = "gaussian",
    n.trees = 15000,
    shrinkage = best.shrinkage,
    interaction.depth = best.depth
)
(best.n <- gbm.perf(boost, method = "OOB", plot.it = F))
train.Yhat <- predict(boost, newdata = train.tmp, n.trees = best.n)
test.Yhat <- predict(boost, newdata = test.tmp, n.trees = best.n)
valid.Yhat <- predict(boost, newdata = valid.tmp, n.trees = best.n)
r2_score(train.Yhat, train$Y)
r2_score(valid.Yhat, valid$Y)

# output submission file
tibble(Y = test.Yhat) %>% write_csv('A0200699Y.csv')
```