Prediction of the diabetes

Yuxin Cui

3 Feb 2023

Introduction

In this project, I aim to predict the outcome of the diabetes. The dataset is available from https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database?resource=download. According to the data source, this dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases, and all patients are females at least 21 years old. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. The dataset contains several clinicopathological predictors and one dependent (called "Outcome"). The predictor variables include "Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction" and "Age".

Data exploratory analysis will be performed in order to understand the nature of the dataset including variables, observations, distributions and sparsity. After the dataset is ready, models will be built and compared to predict the outcome of diabetes using the information available (predictors).

The accuracy of different models will be compared using RMSE. And the model with the lowest RMSE will be considered the best.

Methods

In this section, the libraries, data import and exploration, and modelling will be described.

Libraries

To make them tidy, all libraries are loaded here unless stated elsewhere.

```r
if(!require(librarian)) install.packages("librarian", repos =
"http://cran.us.r-project.org")
library(librarian)

librarian::shelf("DataExplorer")
librarian::shelf("tidyverse")
librarian::shelf("caret")
librarian::shelf("data.table")
librarian::shelf("PerformanceAnalytics")
librarian::shelf("corrplot")
librarian::shelf("janitor")
```

Data

Data loading

The dataset has been downloaded first from the link in the introduction.

```
edx <- fread ("diabetes.csv")
```

Data exploration

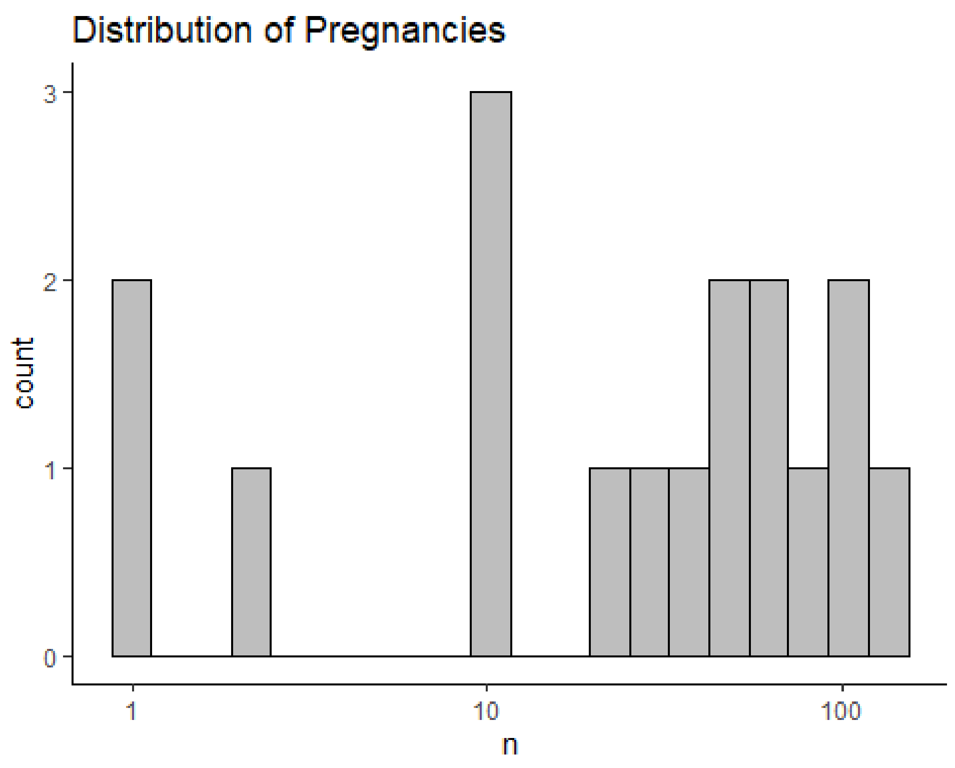
Summary

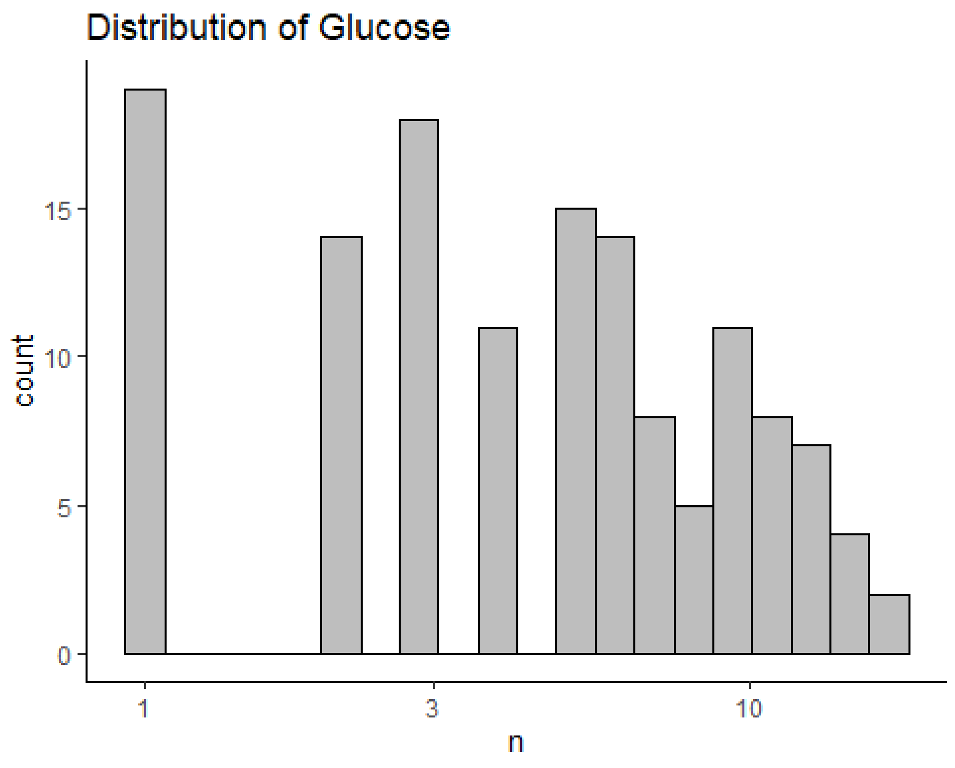
```
str(edx)

## Classes 'data.table' and 'data.frame':   768 obs. of  9 variables:
##  $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
##  $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3
30.5 0 ...
##  $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
##  $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>

names(edx)

## [1] "Pregnancies"              "Glucose"
## [3] "BloodPressure"            "SkinThickness"
## [5] "Insulin"                  "BMI"
## [7] "DiabetesPedigreeFunction" "Age"
## [9] "Outcome"

dim(edx)

## [1] 768   9

summary(edx)

##   Pregnancies        Glucose      BloodPressure    SkinThickness
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
##  Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
##  Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
##  3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
##     Insulin           BMI        DiabetesPedigreeFunction      Age
##  Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
##  1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
##  Median : 30.5   Median :32.00   Median :0.3725           Median :29.00
##  Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24
##  3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
##  Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
##     Outcome
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.349
##  3rd Qu.:1.000
##  Max.   :1.000
```

The dataset contains 9 variables and 768 observations. The data types of “BMI” and “DiabetesPedigreeFunction” are numeric, while others are integer.

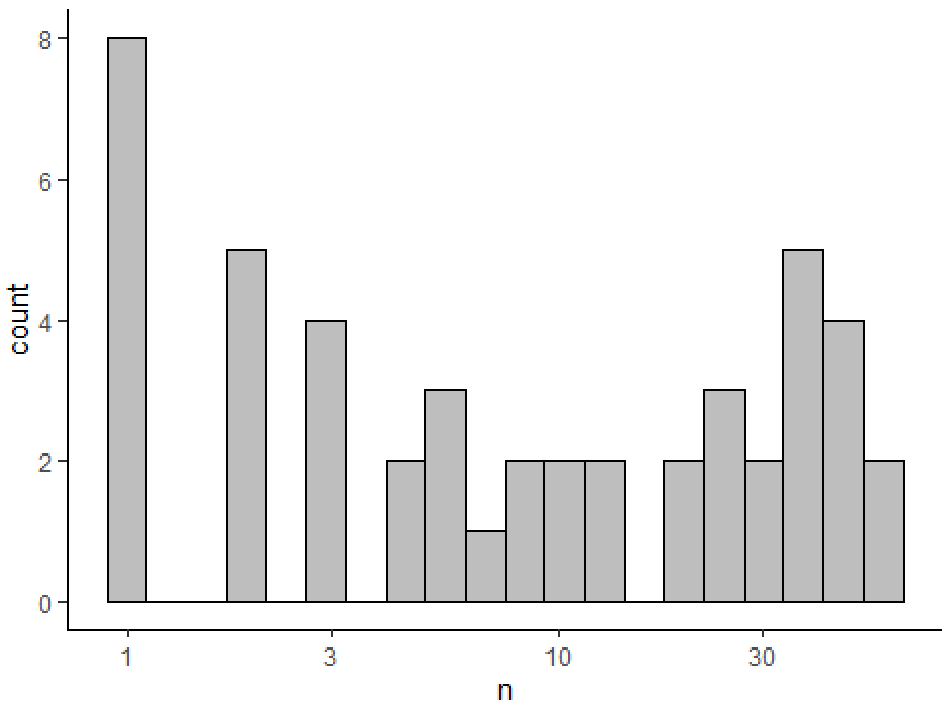
Distribution of each predictor variable

```r
for (i in c("Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
"Insulin",          "BMI", "DiabetesPedigreeFunction","Age")){
  p <- edx %>%
    count(.data[[i]]) %>% ggplot(aes(n))+
  geom_histogram(bins = 20 , color = "black", fill = "gray")+
  scale_x_log10()+
  ggtitle(paste0("Distribution of ", i)) +
    theme_classic()
  print(p)
}
```

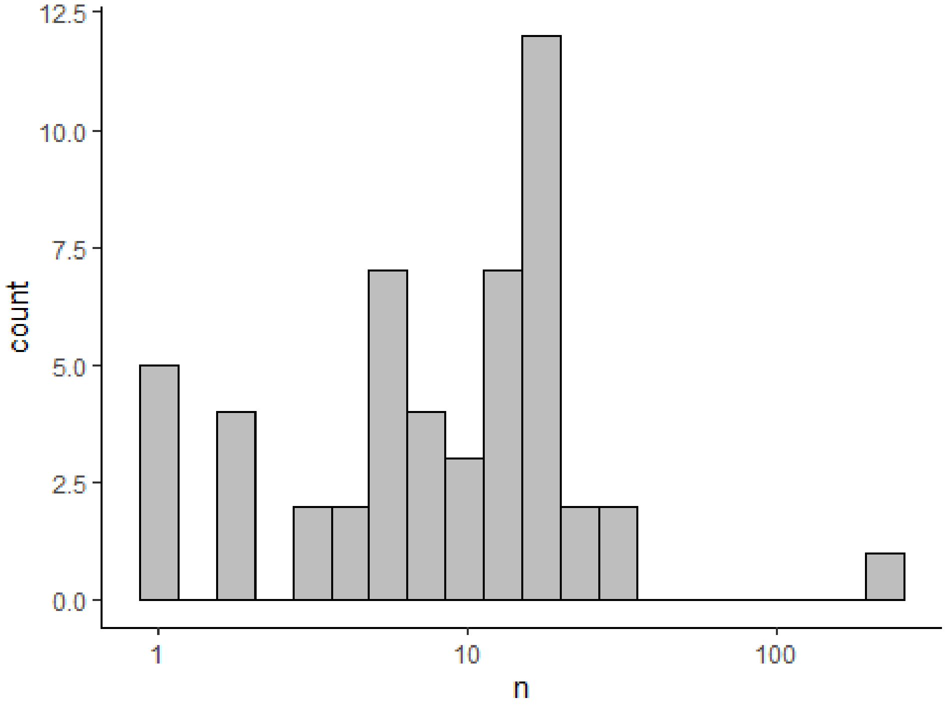

Distribution of Pregnancies
count
3
2
1
0
1
10
100
n


Distribution of Glucose
count
15
10
5
0
1
3
10
n

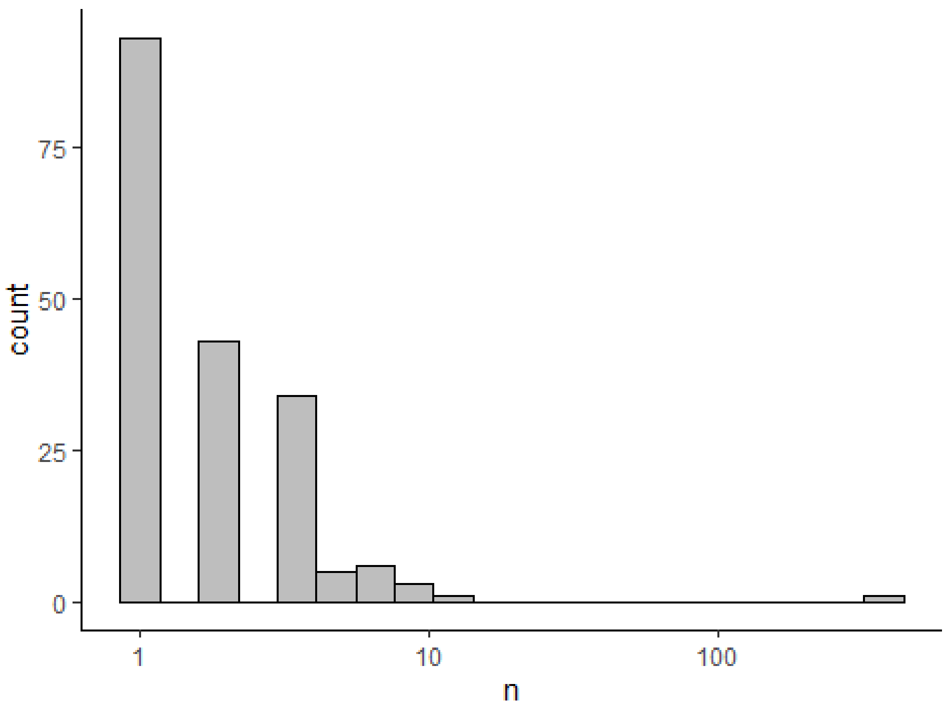
Distribution of BloodPressure



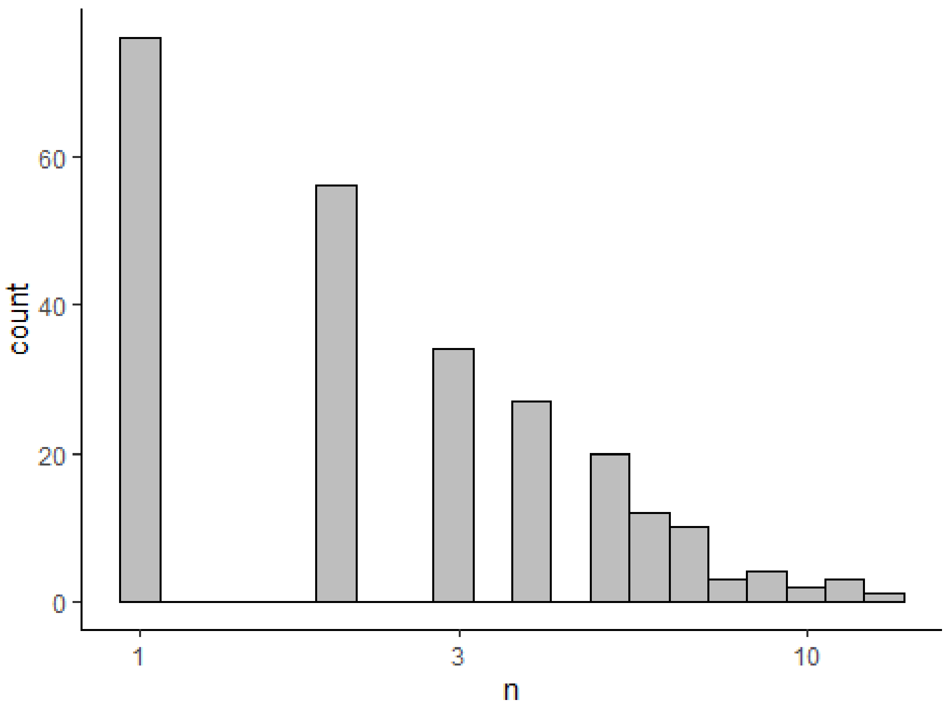
Distribution of SkinThickness

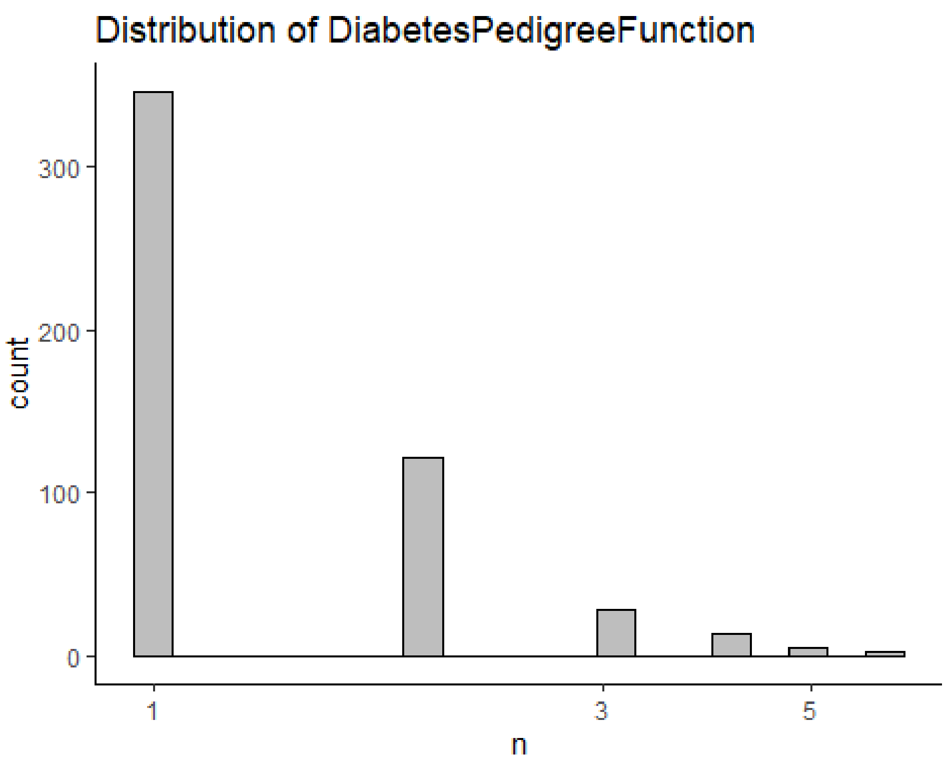


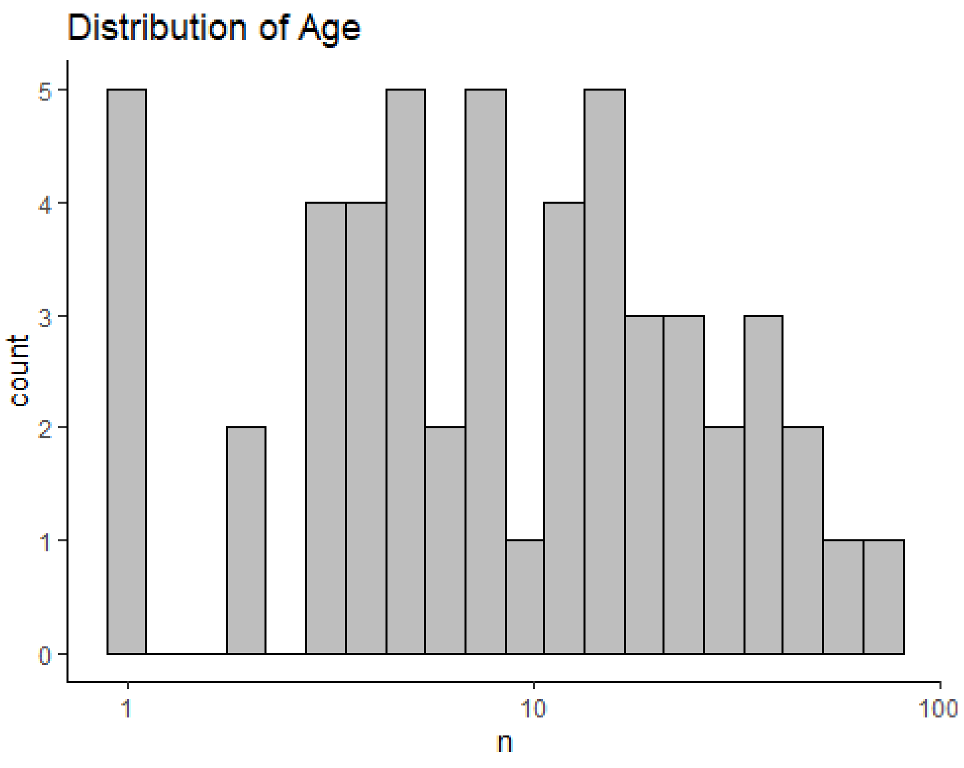
Distribution of Insulin



Distribution of BMI



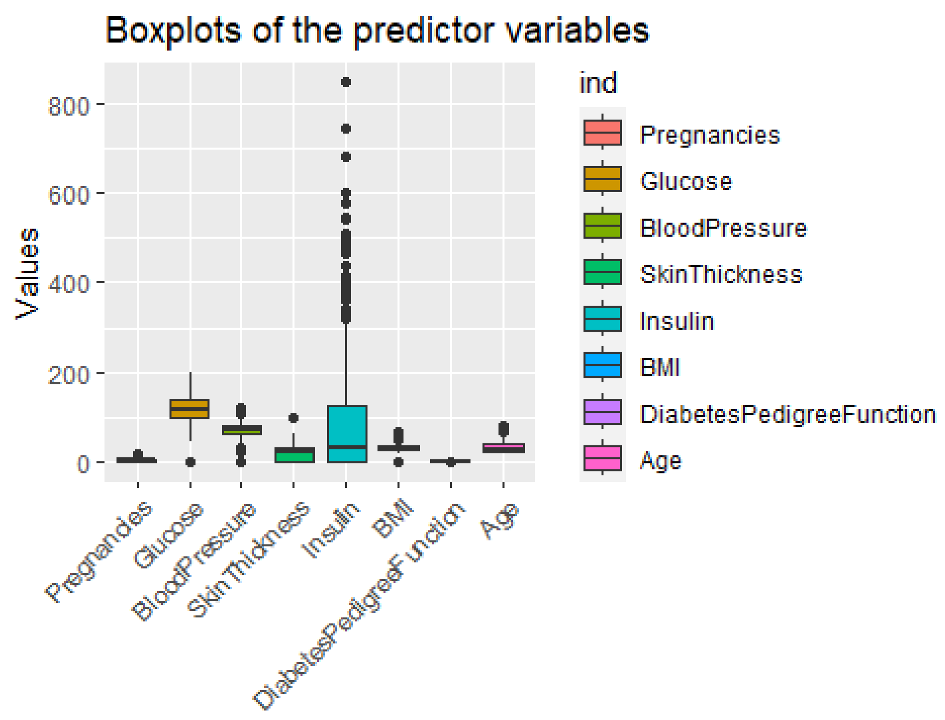




The data distribution tells us all predictor variables are varied. The effects of these variables on the diabetes outcome should be evaluated.

box plots of the variables

```
ggplot(stack(edx[, 1:8]), aes(x = ind, y = values, fill = ind)) +
  geom_boxplot() +
  labs(title = "Boxplots of the predictor variables") +
  labs(x = "", y = "Values") +
  theme(axis.text.x = element_text(
    angle = 45,
    hjust = 1,
    vjust = 1
  ))
```

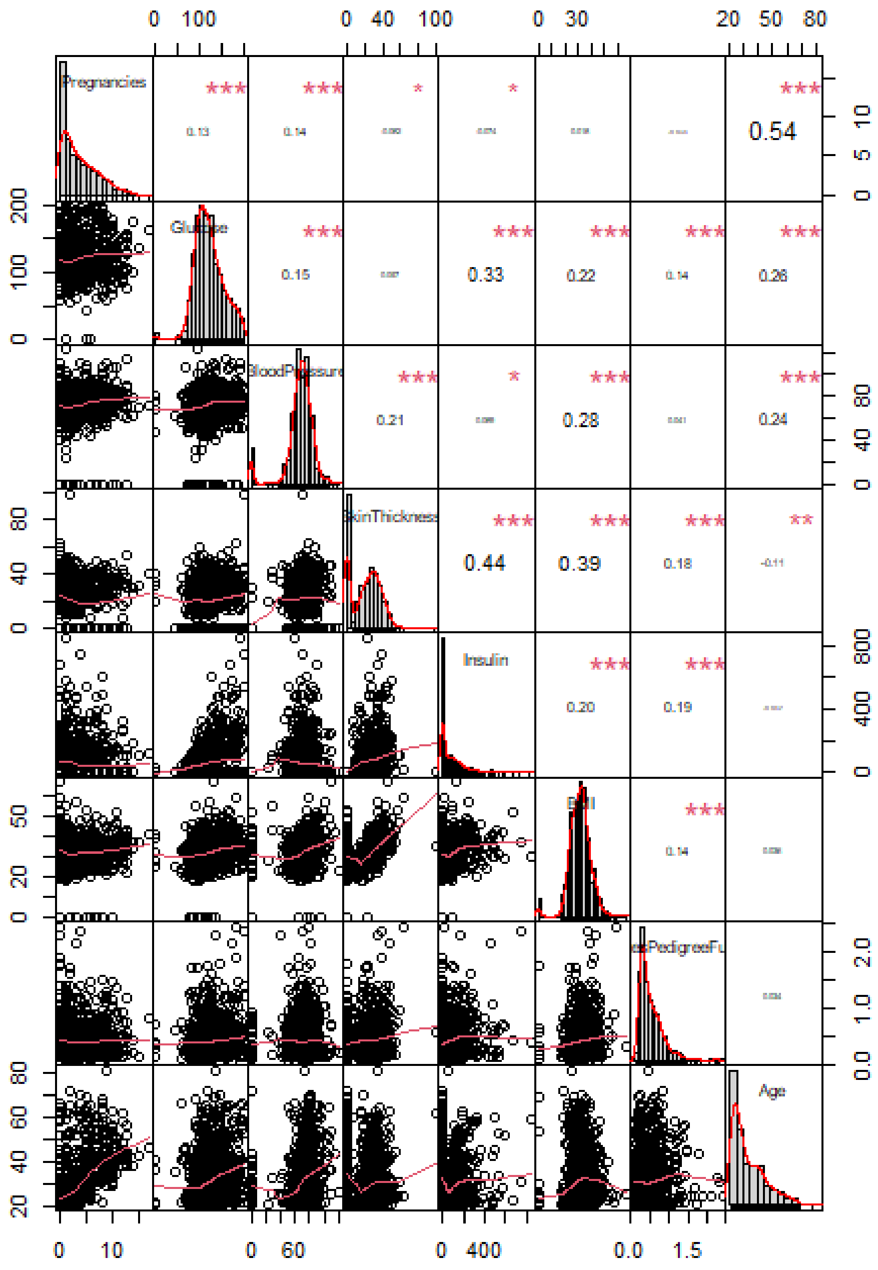


The boxplots of the dataset show insulin has relatively wide variances with more outliers.

Correlation matrix

In the following correlation matrix plot, the distribution of each variable is shown on the diagonal. On the bottom of the diagonal, the bivariate scatter plots with a fitted line are displayed. On the top of the diagonal, the value of the correlation plus the significance level as stars. Each significance level is associated with a symbol: p-values(0, 0.001, 0.01, 0.05, 0.1, 1) <=> symbols(“”, “”, “”, “.”, ” “).

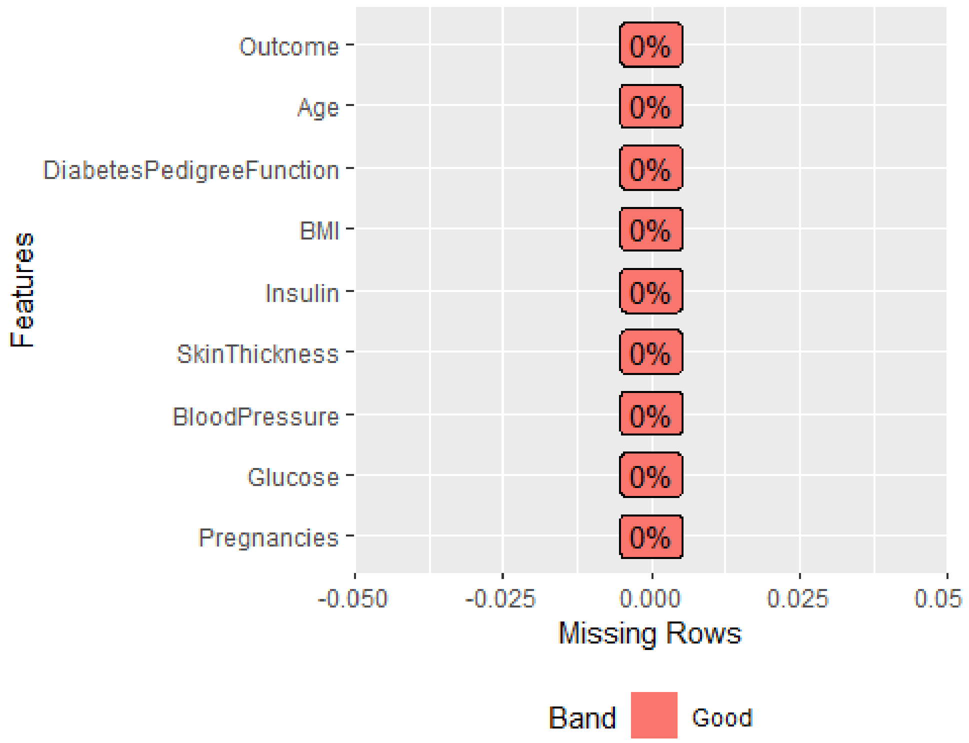
```
chart.Correlation(edx[, 1:8], histogram=TRUE, pch=19)
```



This correlation matrix indicates there is a positive correlation between Pregnancies and Age.

Chech missing values

```
plot_missing(edx)
```



There is no missing value showed in the plot.

Modelling and prediction

Data participation before modelling

I first split the dataset into two parts: train_set and test_set. And the split percentage is 80%, which gives a ratio of train_set:test_set to 4:1.

```
# A method from caret
set.seed(123)

inTrain = createDataPartition(y = edx$Outcome, p = .80, list = FALSE)
train_set = edx[inTrain,]
test_set = edx[-inTrain,]
```

RMSE calculation function

Root Mean Square Error (RMSE) is used to measure the error of a model in predicting quantitative data. The RMSE was calculated to represent the error loss between the predicted ratings derived from applying the algorithm and actual ratings in the test set. Just assume there are n observations y_i and an estimator that estimates the prediction values $\hat{y}_i$ The equation of RMSE is

$$RMSE = \sqrt{\frac{1}{n}\sum(\hat{y}_i - y_i)^2)}$$

RMSE indicates the accuracy. The lower the RMSE, the better the accuracy of a model and its prediction.

```r
RMSE <- function(true_value, predicted_value){
  sqrt(mean((true_value - predicted_value)^2, na.rm = TRUE))
}
```

Prepare control parameters for model training

More details of this pre-modelling setting can be found on https://rdrr.io/cran/caret/man/trainControl.html.

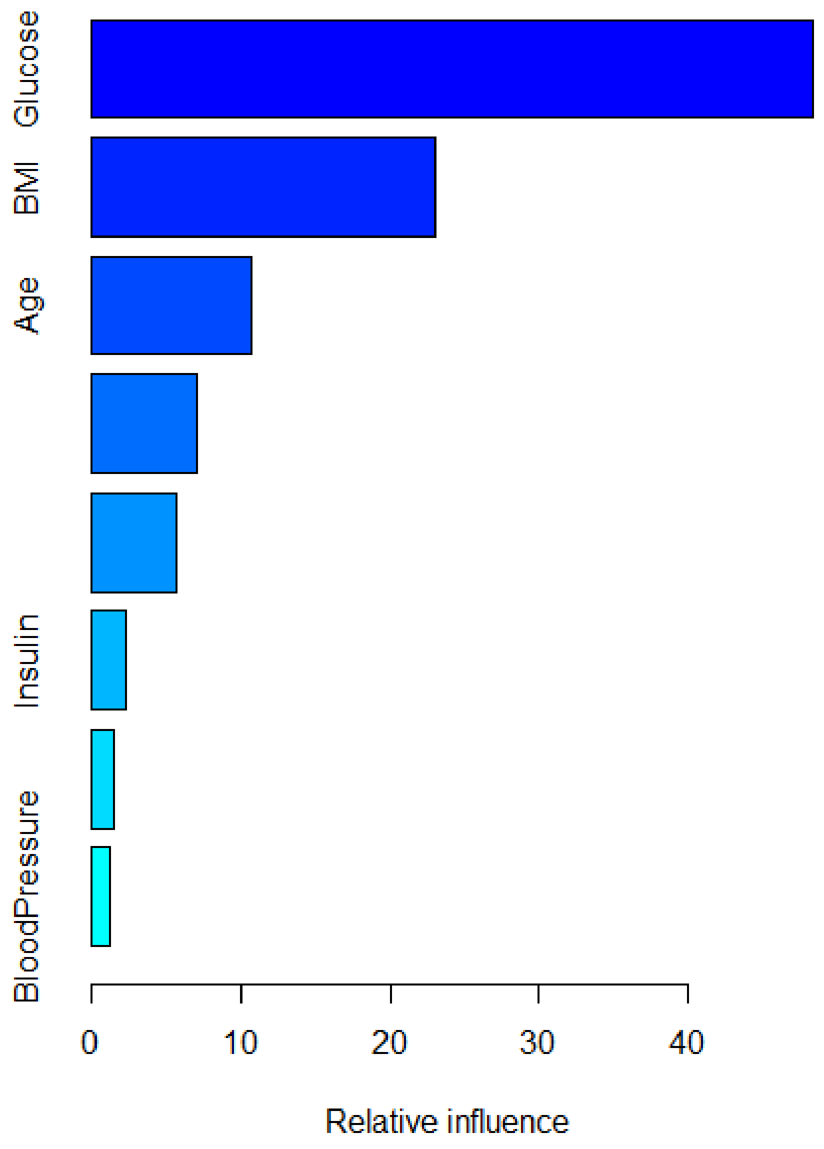
```r
trControl <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 10)
```

Model 1: GBM

The model GBM (Generalized Boosted Regression Modeling) is a forward learning method, which builds an ensemble of shallow trees in sequence with each tree learning and improving on the previous one.

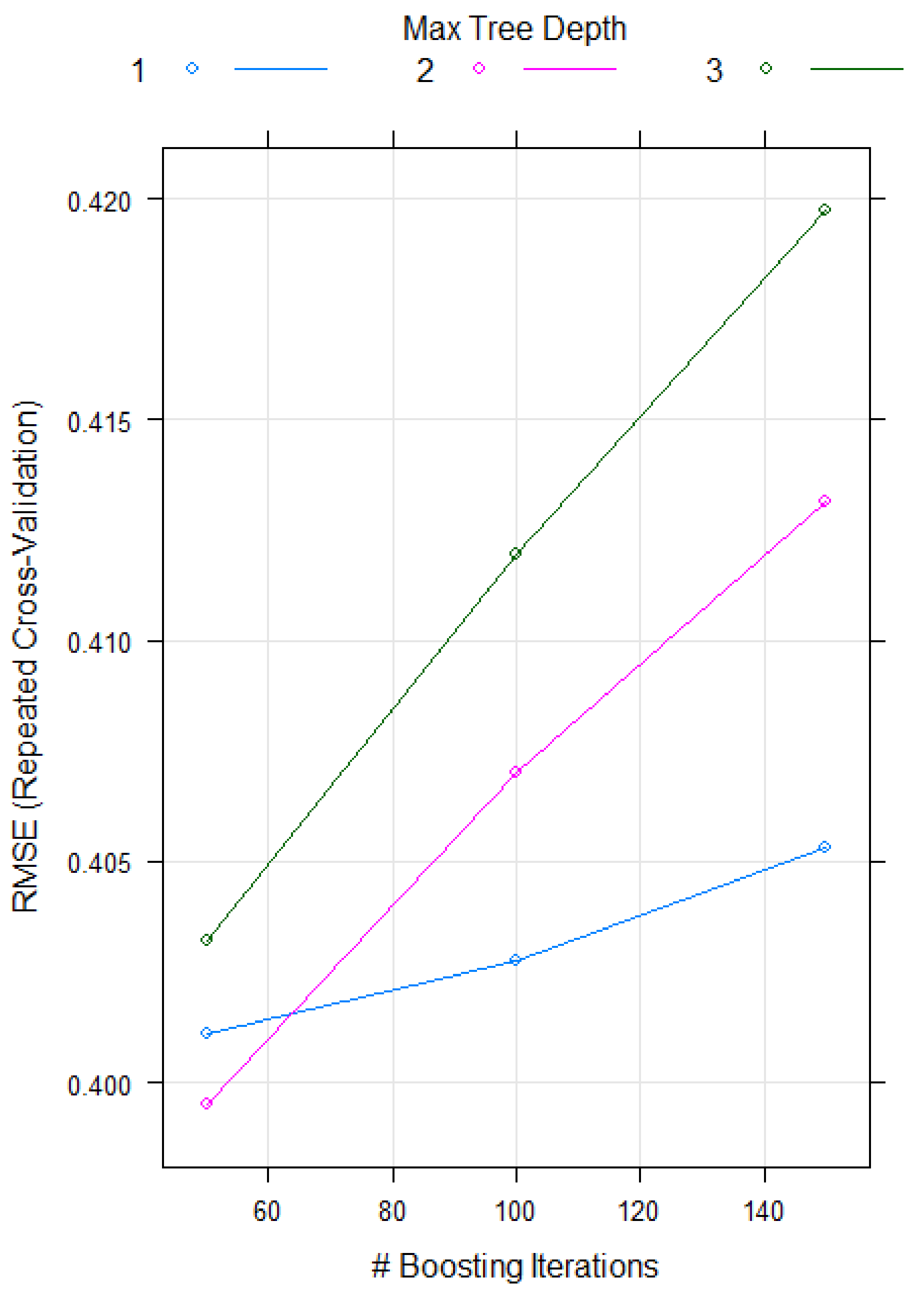
```r
set.seed(123)
fit_gbm <- train(Outcome ~ ., data = train_set,
                 method = "gbm",
                 trControl = trControl,
                 verbose = FALSE)

## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to
do
## classification? If so, use a 2 level factor as your outcome column.

fit_gbm

## Stochastic Gradient Boosting
##
## 615 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 553, 553, 554, 554, 553, 554, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  RMSE       Rsquared   MAE
##   1                   50      0.4011011  0.3008893  0.3318313
##   1                  100      0.4027712  0.2968410  0.3272734
##   1                  150      0.4053078  0.2889662  0.3286117
##   2                   50      0.3995051  0.3059050  0.3175466
##   2                  100      0.4070231  0.2856793  0.3203404
##   2                  150      0.4131445  0.2705103  0.3243075
##   3                   50      0.4031971  0.2968875  0.3149947
##   3                  100      0.4119291  0.2743958  0.3199147
##   3                  150      0.4197355  0.2553353  0.3258809
## 
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## 
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  2, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
summary(fit_gbm)
```



```
##                                               var   rel.inf
## Glucose                                   Glucose 48.377196
## BMI                                           BMI 23.070730
## Age                                           Age 10.745276
## DiabetesPedigreeFunction DiabetesPedigreeFunction  7.006629
## Pregnancies                           Pregnancies  5.667199
## Insulin                                   Insulin  2.341435
## SkinThickness                       SkinThickness  1.555970
## BloodPressure                       BloodPressure  1.235565
```

```
plot(fit_gbm)
```



```
# predict and RMSE
predictions <- predict(fit_gbm, test_set)
RMSE_gbm <- RMSE(predictions, test_set$Outcome)

# A result table is created to collect the RMSE results from different
modelling
rmse_results <- data.frame(Method = "gbm", RMSE = RMSE_gbm) %>%  print()
```

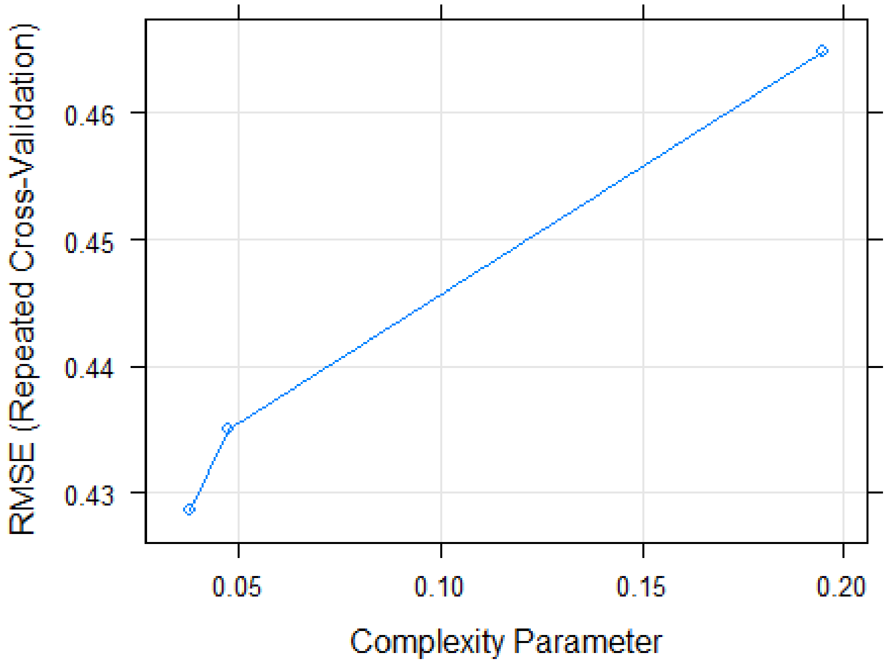
```
##   Method      RMSE
## 1    gbm 0.3899336
```

The result of gbm modelling suggests that the first three most important predictors are: Glucose, BMI and Age, while the BloodPressure is the least important one.

Model 2: CART

CART (Classification and Regression Trees): can be used for both classification and regression problems. CART is a decision tree algorithm. In the decision tree, each fork is split into a predictor variable and each node has a prediction for the target variable at the end. The prediction process is: Obtain the best split point, identify the new best split point, split the input by the split point, repeated splitting until a stopping criterion is met. in CART training, the complexity parameter (cp) is used as a penalty to the tree for over fitting of the data.

```
set.seed(123)
fit_cart <- train(Outcome~., data = train_set, method = "rpart", trControl =
trControl)

## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to
do
## classification? If so, use a 2 level factor as your outcome column.

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
trainInfo, :
## There were missing values in resampled performance measures.

fit_cart

## CART
##
## 615 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 553, 553, 554, 554, 553, 554, ...
## Resampling results across tuning parameters:
##
##   cp          RMSE       Rsquared   MAE
##   0.03819025  0.4286521  0.2121052  0.3510196
##   0.04789330  0.4349559  0.1897258  0.3629044
##   0.19470973  0.4648180  0.1104600  0.4150740
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.03819025.

plot(fit_cart)
```



```
# predict and RMSE
predictions <- predict(fit_cart, test_set)
RMSE_cart <- RMSE(predictions, test_set$Outcome)

# The result table is expanded to collect the RMSE results from different
modelling
rmse_results <- bind_rows(rmse_results, tibble(Method="cart",  RMSE =
RMSE_cart)) %>%
  arrange(RMSE) %>%
  print()

##   Method      RMSE
## 1    gbm 0.3899336
## 2   cart 0.4449477
```

For this dataset, when cp = 0.04196737, the lowest RMSE is achieved. With the increase of cp, the prediction accuracy starts to drop.

Model 3: KNN

The k-nearest neighbors classifier (kNN) is a non-parametric supervised machine learning algorithm. It classifies a new data point into its proximate neighbours' classes. kNN is used for classification and regression tasks.

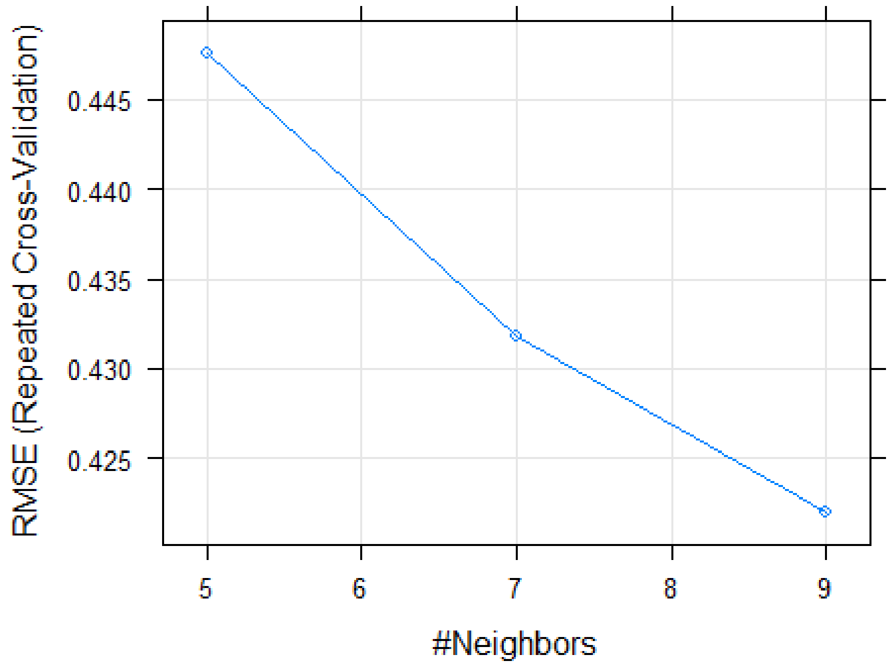
```
set.seed(123)
fit_knn <- train(Outcome~., data = train_set, method = "knn", trControl =
trControl)
```

```
## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to
do
## classification? If so, use a 2 level factor as your outcome column.
```

```
fit_knn
```

```
## k-Nearest Neighbors
##
## 615 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 553, 553, 554, 554, 553, 554, ...
## Resampling results across tuning parameters:
##
##   k  RMSE       Rsquared   MAE
##   5  0.4475821  0.1840645  0.3370175
##   7  0.4317477  0.2146194  0.3328586
##   9  0.4219886  0.2369973  0.3309451
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 9.
```

```
plot(fit_knn)
```



```
# predict and RMSE
predictions <- predict(fit_knn, test_set)
RMSE_knn <- RMSE(predictions, test_set$Outcome)

# The result table is expanded to collect the RMSE results from different
modelling
rmse_results <- bind_rows(rmse_results, tibble(Method="knn",  RMSE =
RMSE_knn)) %>%
  arrange(RMSE) %>%
  print()

##   Method      RMSE
## 1    gbm 0.3899336
## 2    knn 0.4180623
## 3   cart 0.4449477
```

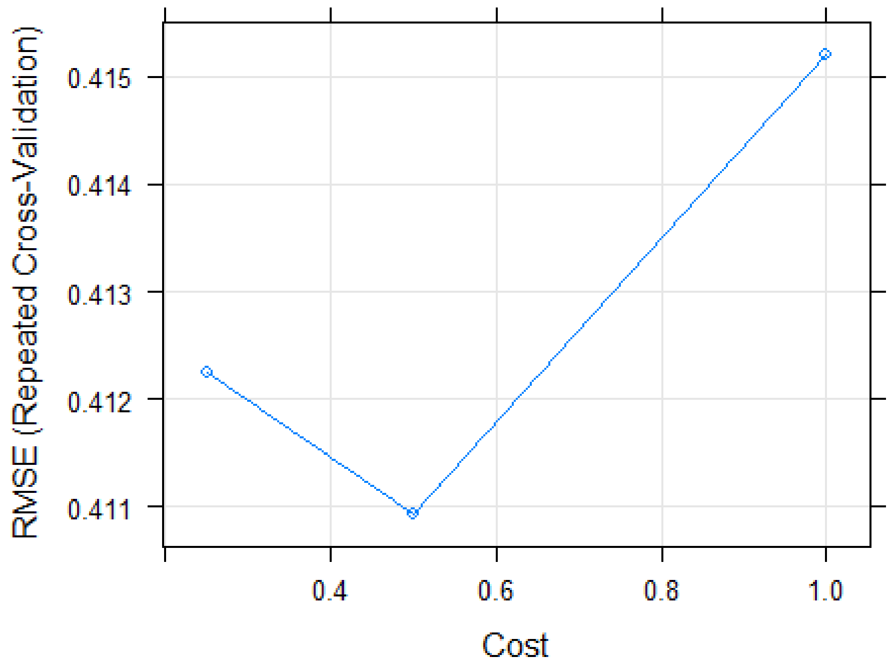
In this algorithm, the modelling based on k =9 obtained the best prediction accuracy.

Model 4: SVM

The SVM (support vector machine) algorithm is a supervised machine learning model. It tries to identify a hyperplane with the maximum margin to separate an N-dimensional space of the data points. It can be used for both classification and regression problems. Radial SVM (svmRadial) Implements a radial SVM using the general svm function. There are two tuning parameters: the sigma parameter defines how far the influence of a single training example reaches, while "C" (cost) parameter is a penalty parameter of the error term.

```
set.seed(123)
fit_svm <- train(Outcome~., data = train_set, method = "svmRadial", trControl
= trControl)

## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to
do
## classification? If so, use a 2 level factor as your outcome column.

fit_svm

## Support Vector Machines with Radial Basis Function Kernel
##
## 615 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 553, 553, 554, 554, 553, 554, ...
## Resampling results across tuning parameters:
##
##   C     RMSE       Rsquared   MAE
##   0.25  0.4122485  0.2991443  0.2957362
##   0.50  0.4109326  0.3017789  0.2892356
##   1.00  0.4152015  0.2911628  0.2894587
##
## Tuning parameter 'sigma' was held constant at a value of 0.1394104
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1394104 and C = 0.5.
```

```
plot(fit_svm)
```



```
# predict and RMSE
predictions <- predict(fit_svm, test_set)
RMSE_svm <- RMSE(predictions, test_set$Outcome)

# The result table is expanded to collect the RMSE results from different
modelling
rmse_results <- bind_rows(rmse_results, tibble(Method="svm",  RMSE =
RMSE_svm)) %>%
  arrange(RMSE) %>%
  print()
```

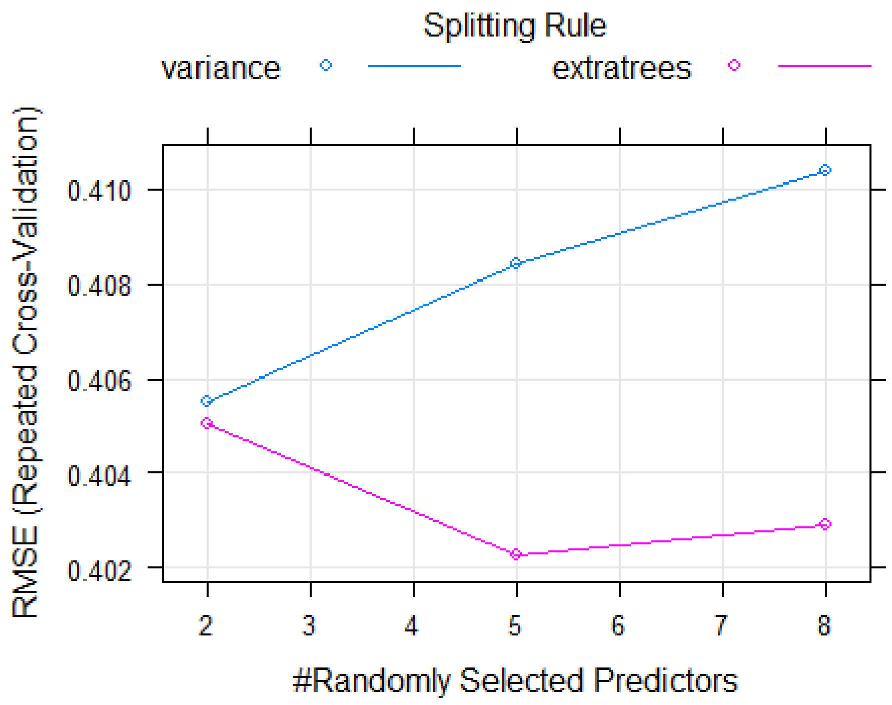
```
##   Method      RMSE
## 1    gbm 0.3899336
## 2    svm 0.4079607
## 3    knn 0.4180623
## 4   cart 0.4449477
```

Following the svm training, the best prediction can be seen when two tuning parameters are: sigma = 0.1350768 and C = 0.25.

Model 5: random forest

The random forest is a supervised learning algorithm that randomly creates and merges multiple decision trees into one "forest". It can also be used to solve regression and classification problems. The method "ranger" is considered a faster implementation of the random forest.

```
set.seed(123)
fit_rf <- train(Outcome~., data = train_set, method = "ranger", trControl =
trControl)

## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to
do
## classification? If so, use a 2 level factor as your outcome column.

fit_rf

## Random Forest
##
## 615 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 553, 553, 554, 554, 553, 554, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   RMSE       Rsquared   MAE
##   2     variance    0.4055056  0.2854353  0.3294908
##   2     extratrees  0.4050580  0.2929739  0.3468654
##   5     variance    0.4084298  0.2795069  0.3196585
##   5     extratrees  0.4022782  0.2961590  0.3315483
##   8     variance    0.4103705  0.2749477  0.3184708
##   8     extratrees  0.4029134  0.2937158  0.3267346
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 5, splitrule = extratrees
##  and min.node.size = 5.

plot(fit_rf)
```



```
# predict and RMSE
predictions <- predict(fit_rf, test_set)
RMSE_rf <- RMSE(predictions, test_set$Outcome)

# The result table is expanded to collect the RMSE results from different
modelling
rmse_results <- bind_rows(rmse_results, tibble(Method="rf",  RMSE = RMSE_rf))
%>%
  arrange(RMSE)

rmse_results

##   Method      RMSE
## 1     rf 0.3864367
## 2    gbm 0.3899336
## 3    svm 0.4079607
## 4    knn 0.4180623
## 5   cart 0.4449477
```

The best prediction outcome was obtained when “mtry” (the number of features, randomly sampled, to split at each node)is 5, the split rule is “extratrees” and the minimum node size is 5.

Comparison of the Caret models

The Caret modelling results can also be compared after collecting resamples. There are three metrics to compare: RMSE, MAE and Rsquared. Just assume there are n observations

y_i and an estimator that estimates the prediction values $\widehat{y_i}$, MAE is the Mean of Absolute value of Errors. The equation of MAE is:

$$MAE = \frac{\sum_{i=1}^{n}(\bar{Y} - Y_i)}{n}$$

Another metric is Rsquared (R^2). The R^2 is equal to

$$R^2 = 1 - SSE/TSS$$

, where SSE is the sum of squared errors:

$$SSE = \sum_{i=1}^{n}(y_i - \widehat{y_i})^2$$

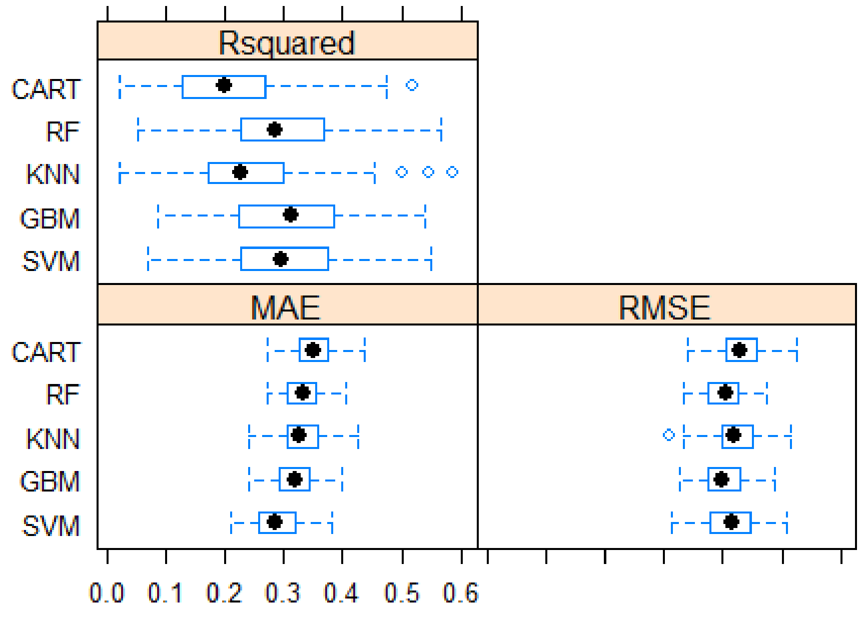
. The TSS is the total sum of squares and is equal to

$$TSS = \sum_{i=1}^{n}(y_i - \bar{Y})^2$$

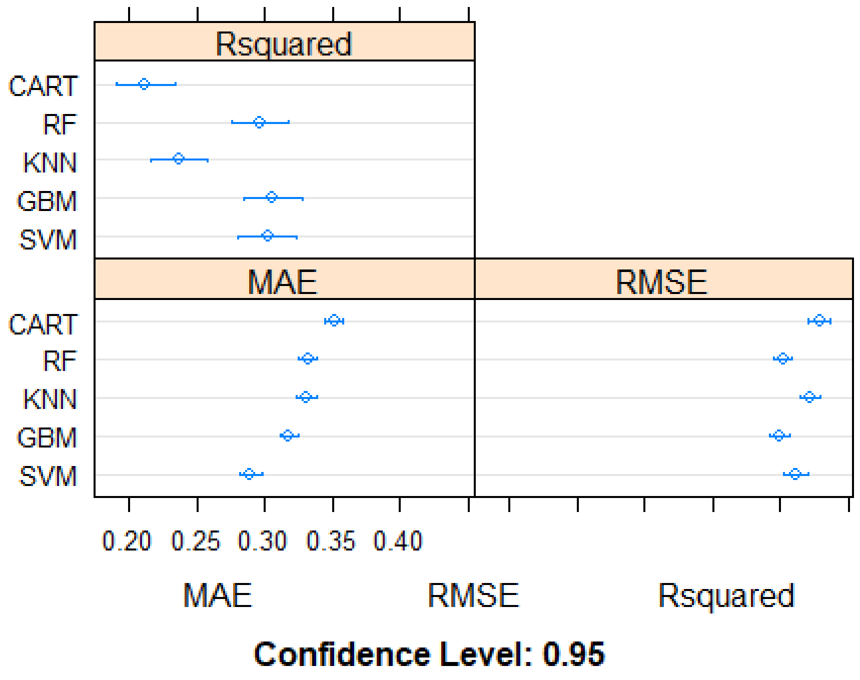
, where \${Y} = \$. R^2 is conveniently scaled between 0 and 1.

```
results <- resamples(list(GBM=fit_gbm, SVM=fit_svm, CART=fit_cart,
KNN=fit_knn, RF=fit_rf))

summary(results)

##
## Call:
## summary.resamples(object = results)
##
## Models: GBM, SVM, CART, KNN, RF
## Number of resamples: 100
##
## MAE
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## GBM  0.2415142 0.2913688 0.3201982 0.3175466 0.3417303 0.3968085    0
## SVM  0.2110458 0.2598750 0.2856988 0.2892356 0.3185724 0.3804225    0
## CART 0.2727745 0.3261986 0.3509138 0.3510196 0.3744051 0.4355170    0
## KNN  0.2401434 0.3041894 0.3252688 0.3309451 0.3585258 0.4244080    0
## RF   0.2713038 0.3066773 0.3335817 0.3315483 0.3538987 0.4047333    0
##
## RMSE
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## GBM  0.3275189 0.3738728 0.3972457 0.3995051 0.4281832 0.4881543    0
## SVM  0.3125272 0.3788885 0.4155617 0.4109326 0.4455338 0.5091192    0
## CART 0.3409808 0.4048334 0.4279769 0.4286521 0.4571044 0.5251681    0
## KNN  0.3104448 0.3973830 0.4208978 0.4219886 0.4502572 0.5148144    0
## RF   0.3331914 0.3762990 0.4039707 0.4022782 0.4266171 0.4739435    0
##
```

```
## Rsquared
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## GBM  0.08655461 0.2226870 0.3130729 0.3059050 0.3847234 0.5403120    0
## SVM  0.06807335 0.2283190 0.2940238 0.3017789 0.3715438 0.5497518    0
## CART 0.02179246 0.1286407 0.1994646 0.2121052 0.2673542 0.5188201    0
## KNN  0.02221593 0.1719299 0.2279687 0.2369973 0.2978566 0.5868168    0
## RF   0.05352569 0.2299446 0.2836731 0.2961590 0.3665497 0.5665901    0

bwplot(results)
```



```
dotplot(results)
```



AutoML method

In this method, a potentially better model could be found by running the H2O's AutoML tool and setting up the number of models (max_models). The documents of this tool can be found at https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html. However, the running speed could be slow.

```
pkgs <- c("RCurl","jsonlite")
for (pkg in pkgs) {
  if (! (pkg %in% rownames(installed.packages()))) { install.packages(pkg) }
}
install.packages("h2o", type="source", repos=(c("http://h2o-
release.s3.amazonaws.com/h2o/latest_stable_R")))

## Warning in install.packages("h2o", type = "source", repos =
## (c("http://h2o-release.s3.amazonaws.com/h2o/latest_stable_R"))):
installation of
## package 'h2o' had non-zero exit status

library(h2o)

##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
```

```
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------

##
## Attaching package: 'h2o'

## The following objects are masked from 'package:data.table':
##
##     hour, month, week, year

## The following objects are masked from 'package:stats':
##
##     cor, sd, var

## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc

# Start the H2O cluster (locally)
h2o.init()

##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 56 minutes
##     H2O cluster timezone:       Europe/London
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.38.0.4
##     H2O cluster version age:    29 days
##     H2O cluster name:           H2O_started_from_R_Yuxin_Cui_tbw461
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   15.88 GB
##     H2O cluster total cores:    16
##     H2O cluster allowed cores:  16
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.2 (2021-11-01)

# import datasets
train_set2 <- as.h2o(train_set)
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%

test_set2 <- as.h2o(test_set)

##   |
|                                                                      |   0%
|
|======================================================================| 100%

# Identify predictors and response
y <- "Outcome"
x <- setdiff(names(train_set2), y)

# Run AutoML for mutliple base models
aml <- h2o.automl(x = x, y = y,
                  training_frame = train_set2,
                  max_models = 5,
                  seed = 1)

##   |
|                                                                      |   0%
|
|======================================================================| 100%
##
## 22:19:22.664: AutoML: XGBoost is not available; skipping it.
## 22:19:22.664: _response param, We have detected that your response column
has only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before
training.
## 22:19:22.690: _response param, We have detected that your response column
has only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before
training.
## 22:19:22.805: _response param, We have detected that your response column
has only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before
training.
## 22:19:23.29: _response param, We have detected that your response column
has only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before
training.
## 22:19:23.113: _response param, We have detected that your response column
has only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before
training.
## 22:19:23.197: _response param, We have detected that your response column
has only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before
```

```
training.
## 22:19:23.324: _response param, We have detected that your response column
has only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before
training.
```

```r
# View the AutoML Leaderboard
lb <- aml@leaderboard
print(lb, n = nrow(lb))  # Print all rows instead of default (6 rows)
```

```
##                                                    model_id      rmse
mse
## 1                          GBM_1_AutoML_7_20230203_221922 0.4002068
0.1601655
## 2 StackedEnsemble_BestOfFamily_1_AutoML_7_20230203_221922 0.4004365
0.1603494
## 3    StackedEnsemble_AllModels_1_AutoML_7_20230203_221922 0.4022915
0.1618385
## 4                          GLM_1_AutoML_7_20230203_221922 0.4063917
0.1651542
## 5                          GBM_3_AutoML_7_20230203_221922 0.4109654
0.1688925
## 6                          DRF_1_AutoML_7_20230203_221922 0.4128595
0.1704530
## 7                          GBM_2_AutoML_7_20230203_221922 0.4136752
0.1711272
##         mae     rmsle mean_residual_deviance
## 1 0.3300960 0.2812784              0.1601655
## 2 0.3304881 0.2820641              0.1603494
## 3 0.3309506 0.2833101              0.1618385
## 4 0.3411176 0.2905898              0.1651542
## 5 0.3278199 0.2877575              0.1688925
## 6 0.3215393 0.2889685              0.1704530
## 7 0.3302996 0.2893490              0.1711272
##
## [7 rows x 6 columns]
```

```r
# The leader model is stored here
fit_aml <- aml@leader
fit_aml
```

```
## Model Details:
## ==============
##
## H2ORegressionModel: gbm
## Model ID:  GBM_1_AutoML_7_20230203_221922
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1              39                       39                4216         2
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1         4    2.74359          3          5     3.94872
```

```
##
##
## H2ORegressionMetrics: gbm
## ** Reported on training data. **
##
## MSE:  0.1420703
## RMSE:  0.3769222
## MAE:  0.3066164
## RMSLE:  0.2651549
## Mean Residual Deviance :  0.1420703
##
##
##
## H2ORegressionMetrics: gbm
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined
holdout predictions) **
##
## MSE:  0.1601655
## RMSE:  0.4002068
## MAE:  0.330096
## RMSLE:  0.2812784
## Mean Residual Deviance :  0.1601655
##
##
## Cross-Validation Metrics Summary:
##                             mean       sd cv_1_valid cv_2_valid cv_3_valid
## mae                     0.329821 0.009804   0.319375   0.333361   0.344875
## mean_residual_deviance 0.160191 0.007765   0.153452   0.163311   0.167751
## mse                     0.160191 0.007765   0.153452   0.163311   0.167751
## r2                      0.293574 0.038551   0.300732   0.323085   0.235570
## residual_deviance       0.160191 0.007765   0.153452   0.163311   0.167751
## rmse                    0.400144 0.009740   0.391729   0.404118   0.409574
## rmsle                   0.281318 0.007488   0.276295   0.276505   0.294104
##                         cv_4_valid cv_5_valid
## mae                       0.324525   0.326966
## mean_residual_deviance   0.150439   0.166001
## mse                       0.150439   0.166001
## r2                        0.330986   0.277496
## residual_deviance         0.150439   0.166001
## rmse                      0.387864   0.407432
## rmsle                     0.277834   0.281851
```

```
# retrieve the model performance
perf <- h2o.performance(fit_aml, test_set2)
perf
```

```
## H2ORegressionMetrics: gbm
##
## MSE:  0.160923
## RMSE:  0.4011521
## MAE:  0.3262554
## RMSLE:  0.2825789
## Mean Residual Deviance :  0.160923
```

```
RMSE_aml <- perf@metrics$RMSE

# The result table is expanded to collect the RMSE results from different
modelling
rmse_results <- bind_rows(rmse_results, tibble(Method="H2O’s AutoML",  RMSE =
RMSE_aml)) %>%
  arrange(RMSE)

DiffValue = c(NA, diff(rmse_results$RMSE))
rmse_results %>% mutate(Difference = DiffValue) %>%
  replace (is.na(.), "") %>%
   print()
```

```
##          Method      RMSE          Difference
## 1            rf 0.3864367
## 2           gbm 0.3899336 0.00349691856605416
## 3 H2O’s AutoML 0.4011521  0.0112184502060812
## 4           svm 0.4079607 0.00680858985512089
## 5           knn 0.4180623  0.0101016133057267
## 6          cart 0.4449477  0.0268853728984784
```

Results

I carried out a project to predict the outcome of diabetes by modelling using multiple ML algorithms. The prediction accuracies of all the selected algorithms were ranked according to the RMSE values. The results showed the random forest (ranger) model built through the caret framework outperformed others to provide the best accuracy. gbm is the second. And a gbm model built through H2) autoML performed as the third. The algorithm followers were svm, knn and cart in terms of their accuracy. It is noted that the accuracies produced by these models have no huge difference (0.05 top vs bottom), suggesting each algorithm possesses a certain prediction capability.

Conclusion

In this project, I demonstrate the outcome of diabetes can be predicted when data containing certain clinicopathological features are available. An optimum prediction mode can be built when multiple ML algorithms are used to train.

The dictation of the right algorithm will largely depend on the structure and complexity of the data. In this case, the random forest algorithm yields the lowest prediction RMSE. Although the model built through H2O autoMl was not ranked at the top, the advantage of H2O autoMl is obvious as algorithm candidates do not need to be chosen first. But the limitation of this sort of automated machine learning is time-consuming, particularly when a dataset is getting bigger and bigger. Also, manual ML modelling may allow more control than automated modelling when tuning is required.

In this project, I only trained limited prediction models based on the preliminary comparison. The prediction may be improved if more algorithms could be included in the future. Other ML frameworks such as Tensorflow, PyTorch, and scikit-learn may also be worthy to try although they are more python friendly. It may be worthy to consider excluding certain variables which may have little prediction potential according to expert advice, and this may be valuable to reduce computational time and even improve accuracy.