Final Project Report

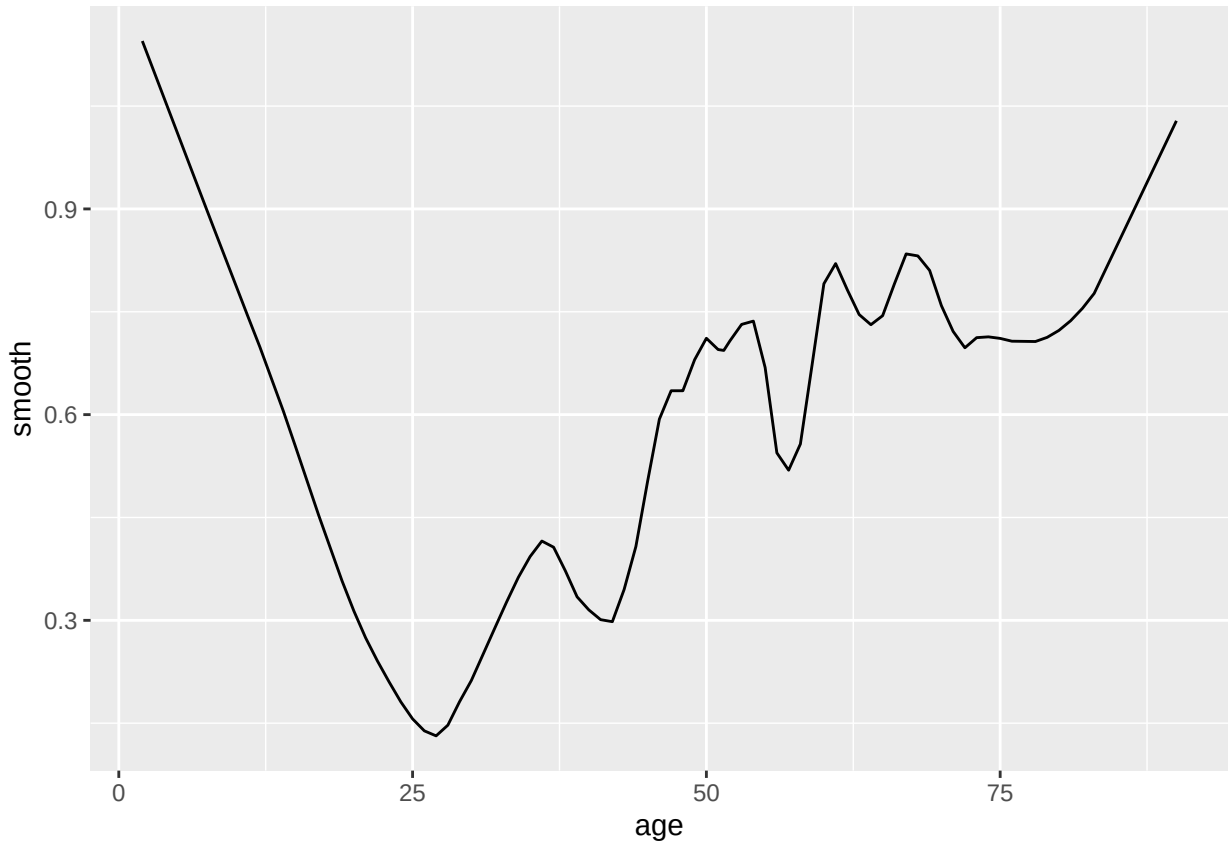
Carrie Cheng

2022-11-30

Introduction

Higher accuracy in disease prediction would largely help patients and doctors. In this project, we aim to analyze whether machine learning algorithms would outperform the traditional logistic regression in a classification problem under the context of predicting disease. We will look at the chronic kidney disease data set from the UCI machine learning repository, and investigate which model out of the traditional statistical model and machine learning models could best classify the development of chronic kidney disease.

The data set has 25 variables. One of these variables is the binary outcome variable recording whether the patient develops the kidney disease. The rest of the variables record the clinical information of the patients such as age, blood pressure, specific gravity, albumin, sugar, red blood cells, pus cell and bacteria. The data set contains both quantitative variables such as age, blood pressure, and sodium, and qualitative data such as albumin, sugar, hypertension. The data has missing values. To deal with the missing values, we impute the missing values with the mean of each variable excluding the missing values for quantitative variables, and the lowest or reference category for categorical variables. For example, for the variable hypertension, the missing values are replaced by no hypertension (htn = 0).



For the explanatory data analysis, we investigate the relationship between the variables and the outcome. From the plot, we see that age and the outcome might have a nonlinear relationship. Therefore, we calculate the pearson correlation to see the strength of linear relationship between each variable and the outcome.

```
##       pearson correlation with outcome     p values
## age                          0.21787598 1.098513e-05
## bp                           0.29136656 2.877686e-09
## sg                          -0.65970612 0.000000e+00
## al                           0.52337662 0.000000e+00
## su                           0.26625071 6.436108e-08
## rbc                          0.28870505 4.059777e-09
## pc                           0.35700856 1.814104e-13
## pcc                          0.23748477 1.557398e-06
## ba                           0.19087904 1.225307e-04
## bgr                          0.39005523 4.440892e-16
## bu                           0.37981834 3.552714e-15
## sc                           0.29697131 1.377874e-09
## sod                         -0.33828027 3.642864e-12
## pot                          0.07699492 1.242047e-01
## hemo                        -0.72217253 0.000000e+00
## pcv                         -0.68935489 0.000000e+00
## wbcc                         0.21059989 2.171006e-05
## rbcc                        -0.58231038 0.000000e+00
## htn                          0.57113322 0.000000e+00
## dm                           0.52126341 0.000000e+00
## cad                          0.24115247 1.059976e-06
## appet                       -0.35747752 1.678657e-13
```

```
## pe                          0.37010498 1.976197e-14
## ane                         0.31798693 7.537082e-11
## class                       1.00000000           NA
```

From the table, we see that the absolute values of the pearson correlation mostly do not exceed 0.5, which means that most of the variables appear to have a weak linear correlation with the outcome. However, we see that their corresponding p-values are small, which indicates that the variables are associated with the outcome. Therefore, we need to consider flexible models which could capture complex and nonlinear relationship between predictors and the outcome. Under the classification context, classification tree and random forests could help learning complicated information from the data. Specifically, random forests might perform more stable than a single classification tree because it bootstraps the data sample and take an average over differently trained trees.

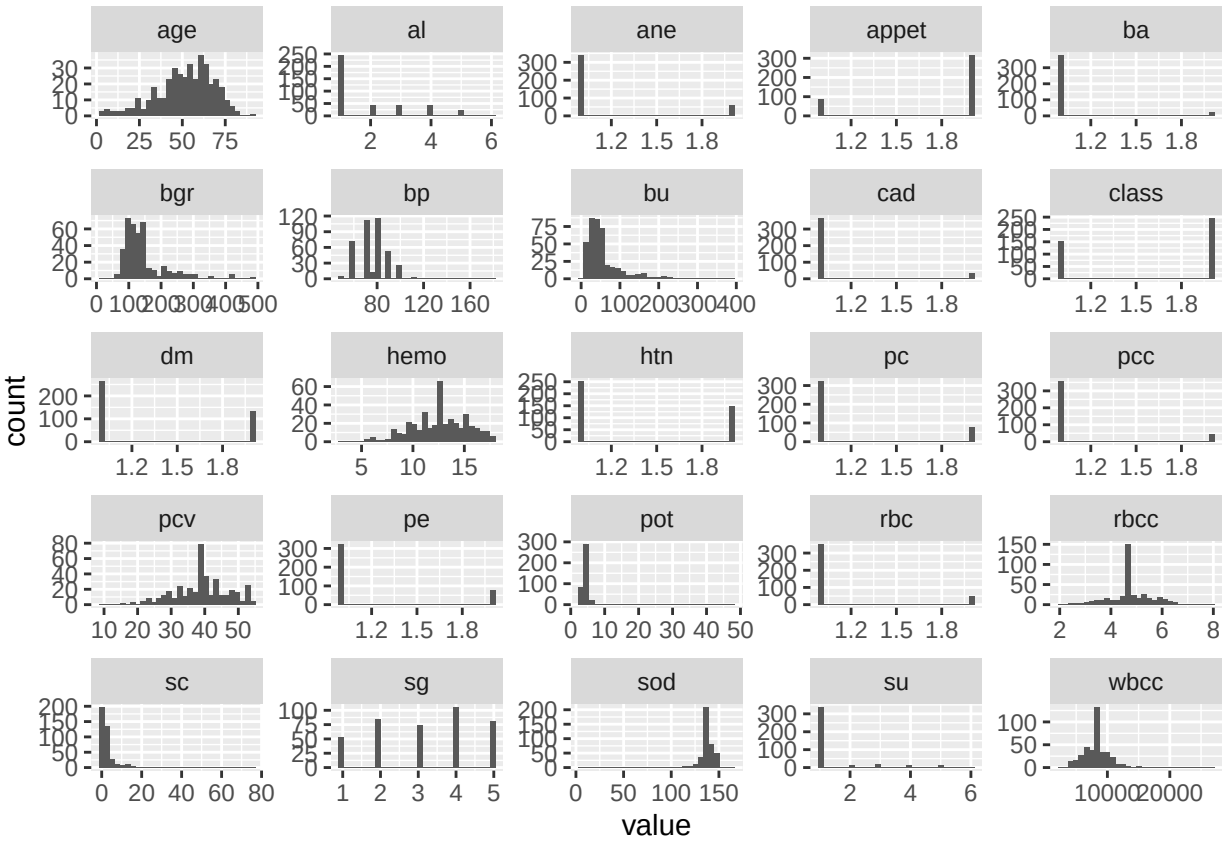
Furthermore, for a sanity check, we look at the distribution of the categorical variables. We see that each category of the categorical variables contains nonzero counts. Since the data set has both quantitative and qualitative variables, machine learning algorithms such as k-nearest neighbors and naive bayes and trees might help in adapting this because they could flexibly select and learn from each variable. Therefore, we are ready to train these models, and finally, we will also train a logistic regression model to provide a comparison. In addition, we will also apply parameter tuning using a 10-fold cross-validation to select the best tuned model.

Results

In order to investigate whether machine learning algorithms could capture more complex relationships between predictors and outcome, we will apply flexible models including classification trees, random forests, k-nearest neighbors, and naive bayes. First, we will train each algorithm with fixed parameters using only one train and test set for a reference purpose. We will also train a logistic regression in order to establish a comparison on the performance between the traditional statistical methods and the machine learning algorithms. For random forests, we set the number of trees trained to be 100, and for k-nearest neighbors, we set the number of neighborhood to be 5.

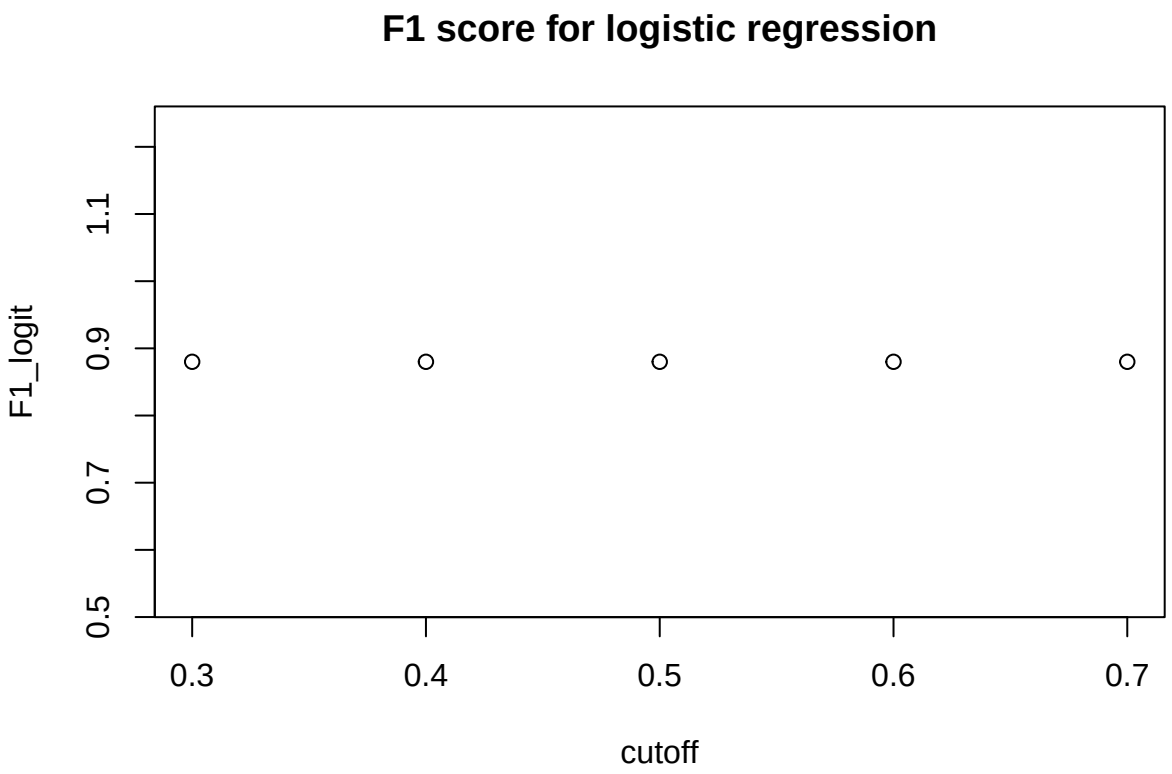
```
##               Accuracy Sensitivity Specificity
## metric_logit 0.9008264   0.9361702   0.8783784
## metric_rf    0.9752066   0.9361702   1.0000000
## metric_tree  0.9586777   0.9148936   0.9864865
## metric_nb    0.9090909   0.9574468   0.8783784
## metric_knn   0.6942149   0.8510638   0.5945946
```

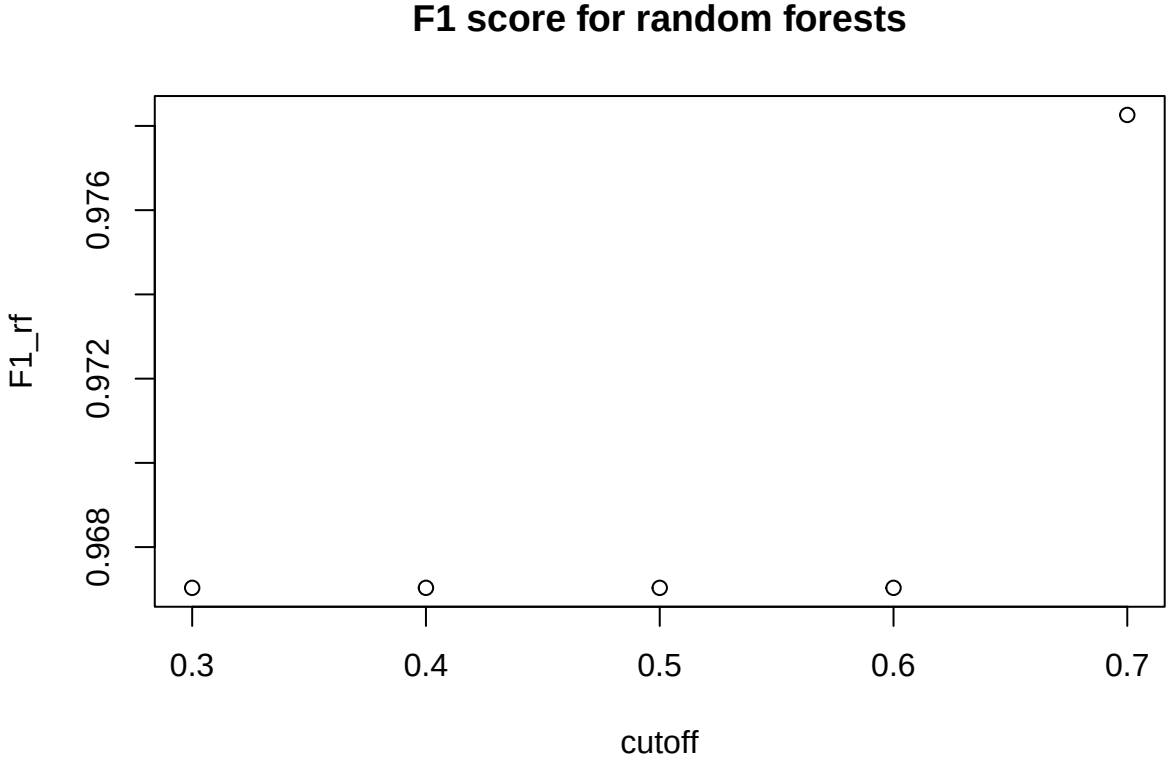
From the table, random forests and classification trees have higher accuracy than logistic regression, but k-nearest neighbors have lower accuracy than logistic regression, and the accuracy of naive bayes is close to that of logistic regression. However, we see that in terms of sensitivity, naive bayes has the highest value, followed by logistic regression and random forests, then classification trees and k-nearest neighbors. In addition, in terms of specificity, random forests and classification trees have the two highest values, followed by logistic regression and naive bayes, then k-nearest neighbors. In general, the sensitivity is higher than the specificity rate for three out of five algorithms. This might result from the fact that there is an imbalance between the observed outcome of having an event and not having an event. If we look at the distribution of the outcome, we see that the proportion of outcome = 1 is higher than that of outcome = 0.

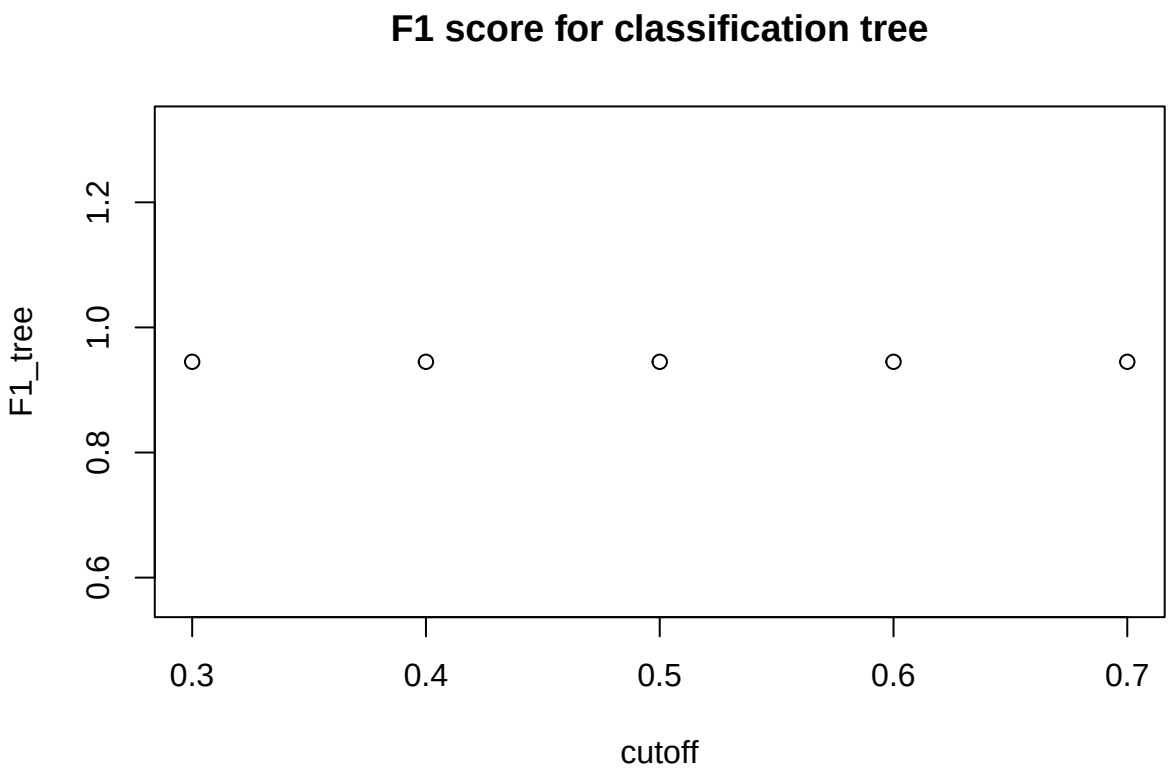


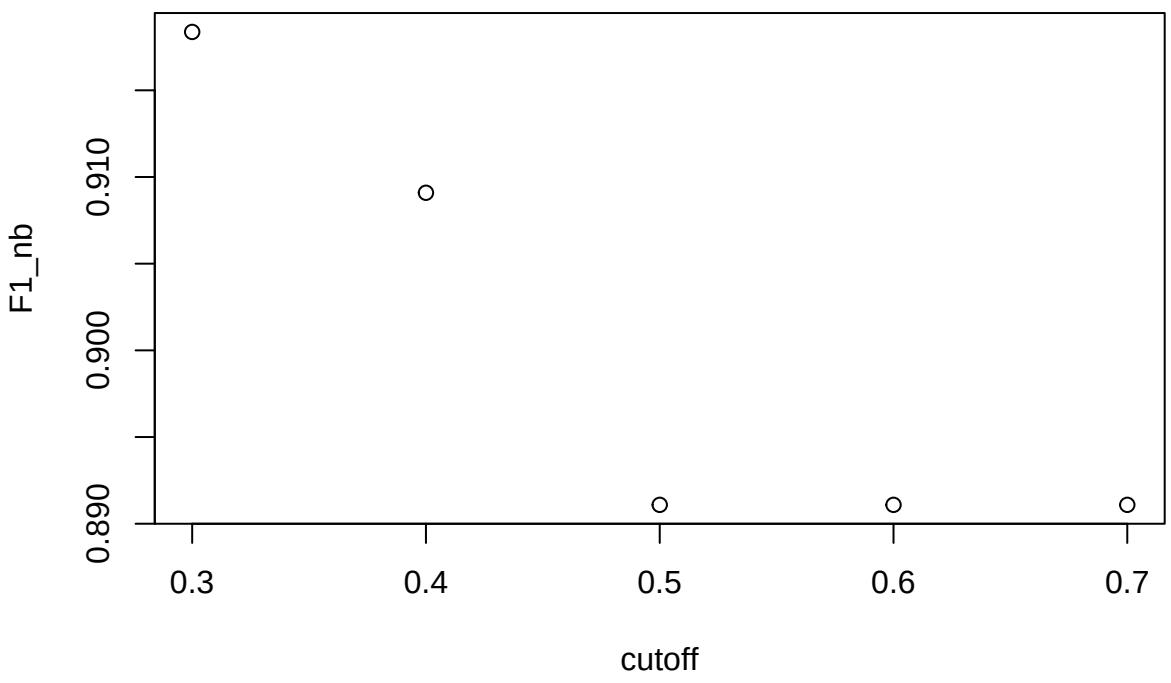
In general, random forests perform the best in terms of an aggregate of the metrics. We see that k-nearest neighbors performs the worst of the five algorithms in terms of each metric. This might due to the fact that the k-nearest neighbors simply take an average over its neighbors and might ignore some complex relationship between predictors and outcome and their variability while performing the predictions.

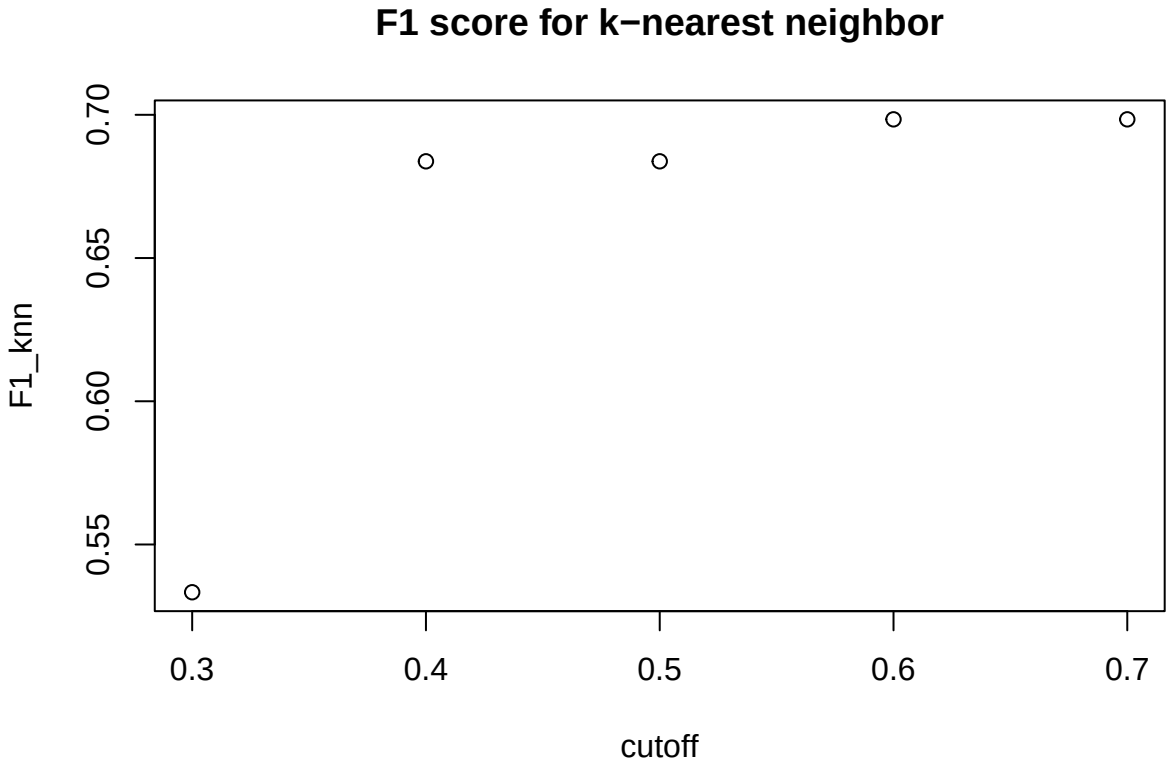
Now, let's look at their F1 score, which is a combination (a harmonic average) of sensitivity and specificity metrics. Let's plot their F1 score curves against five different cut-offs ranging from 0.3 to 0.7. We see that for logistic regression, and classification trees, different cut-offs do not generally influence the F1 score. For random forests, the cut-off values from 0.3 to 0.6 have the same F1 score, and the cut-off value of 0.7 achieves the highest F1 score for random forests. This situation is likely due to that random forests apply bootstrapping strategy, which is very likely to result in variability among individual trees and their predictions and results. For k-nearest neighbors, we see there is a sudden increase of F1 score from cut-off value of 0.3 to cut-off value of 0.4. For naive bayes, a smaller cut-off value has a higher F1 score, and the highest F1 score occurs when the cut-off value is 0.3.


F1 score for logistic regression
F1_logit
0.5
0.7
0.9
1.1
0.3
0.4
0.5
0.6
0.7
cutoff


F1 score for random forests
F1_rf
0.968
0.972
0.976
0.3
0.4
0.5
0.6
0.7
cutoff

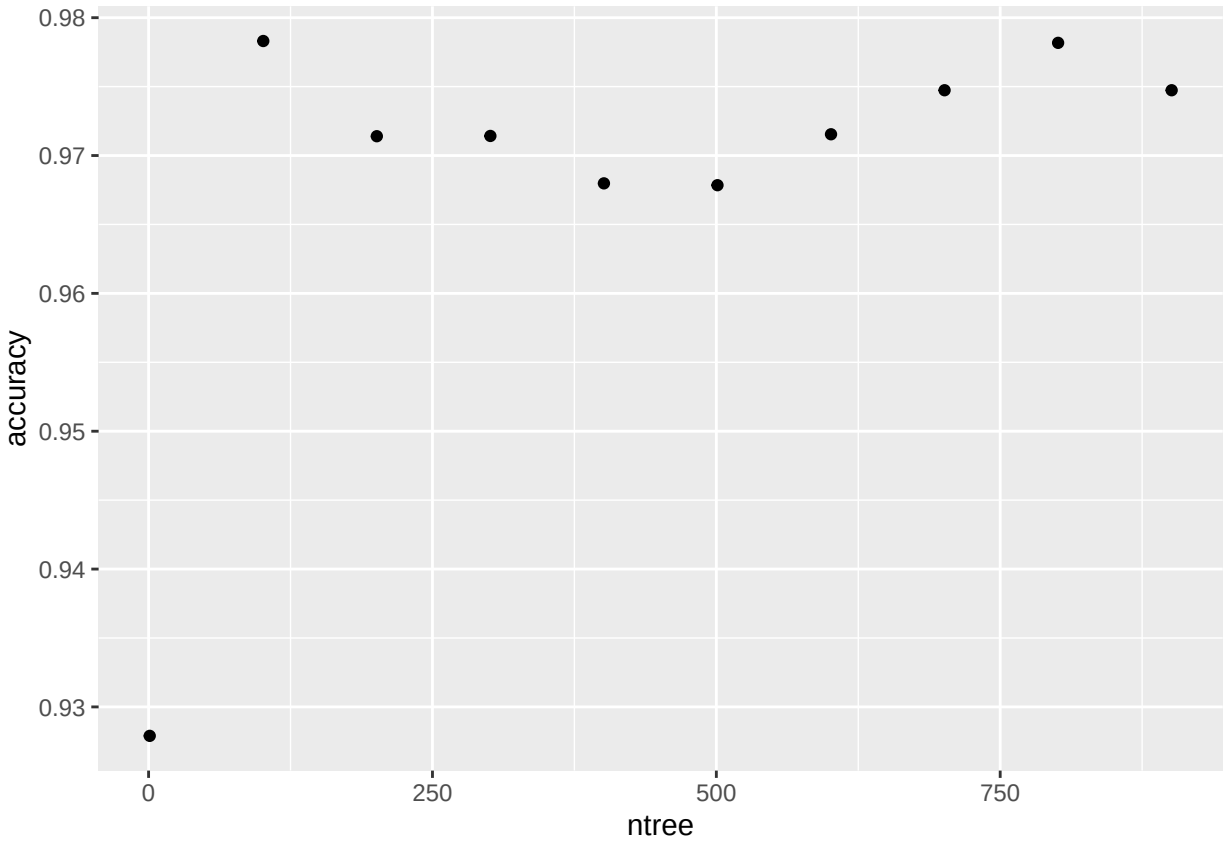

F1 score for classification tree
F1_tree
1.2
1.0
0.8
0.6
0.3
0.4
0.5
0.6
0.7
cutoff


F1 score for naive bayes
F1_nb
0.910
0.900
0.890
0.3
0.4
0.5
0.6
0.7
cutoff



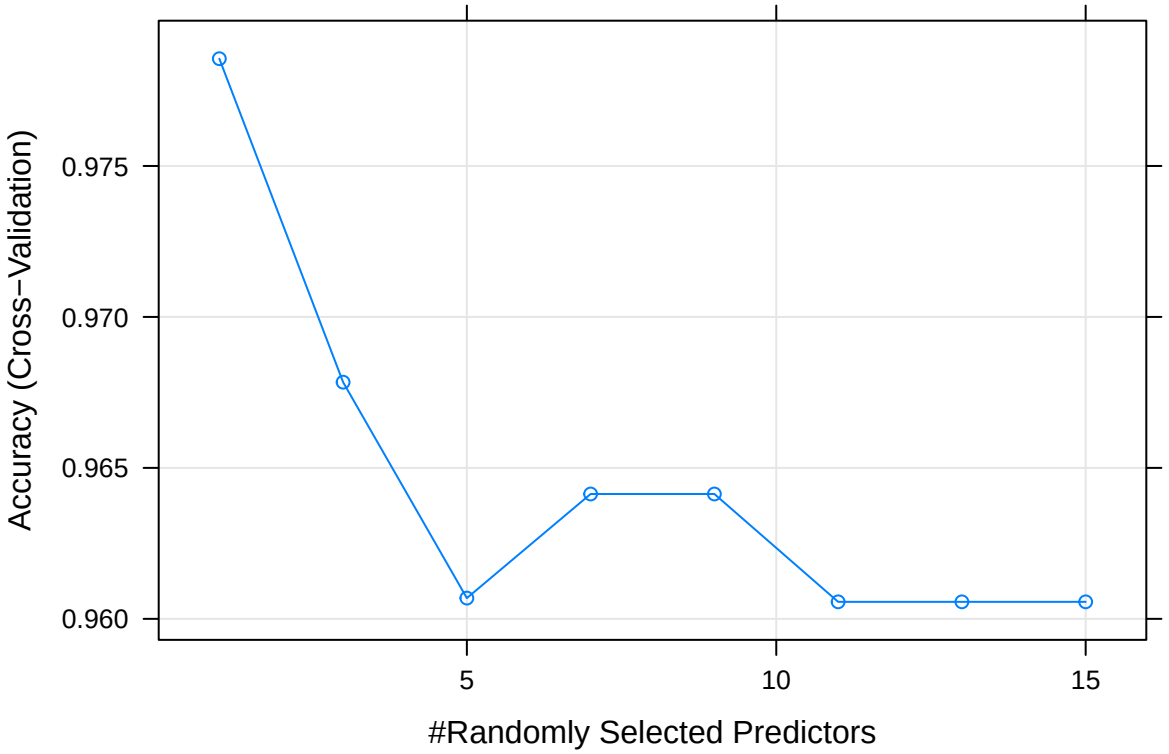
Nevertheless, these results are only based on a single cross-validation set with fixed parameters. However, the performance of these algorithms might differ with different parameters. Therefore, we will implement a common practice of parameter tuning using a 10-fold cross-validation on the train set and record the best tuned model. We will analyze whether parameter tuning improves the performance for each algorithm. Specifically, we will compare the tuning parameters with the accuracy. Since there is no tuning parameter for logistic regression, we will only train the model with a 10-fold cross-validation.

We will start with random forests. There are several parameters that could be tuned for random forests such as the number of variables randomly selected, the size of each node, and number of trees trained. Here, we will investigate the accuracy versus the number of trees trained. We will train a random forests model for 10 different numbers of trees with a 10-fold cross-validation and we keep the number of variables randomly selected for placing splits as three.



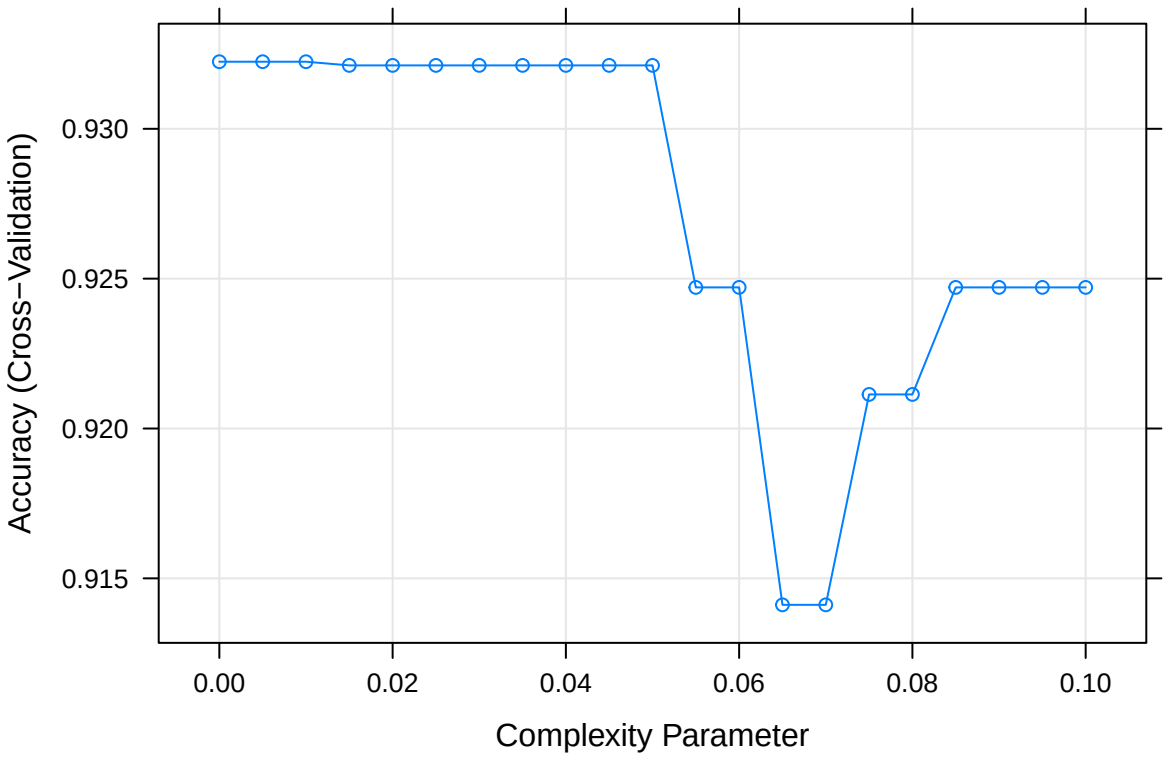
From the graph, we see that the accuracy is greatly boosted from training only one tree to 101 trees, and the accuracy stays approximately the same for even larger number of trees trained. This is a result from the bootstrapping and ensemble strategy for random forests. Since training one tree might produce great variability, bootstrapping the sample data, training trees repetitively, and taking averages across the individual trees might help the performance and reduce the variability of the outcome prediction. Therefore, having a large number of trees trained helps the accuracy of prediction.

Next, we will also look at the influence of number of variables randomly selected as predictors on the accuracy of prediction. In this tuning process, we trained random forests for 8 different number of variables randomly selected as predictors with a 10-fold cross-validation and plot them against the accuracy.



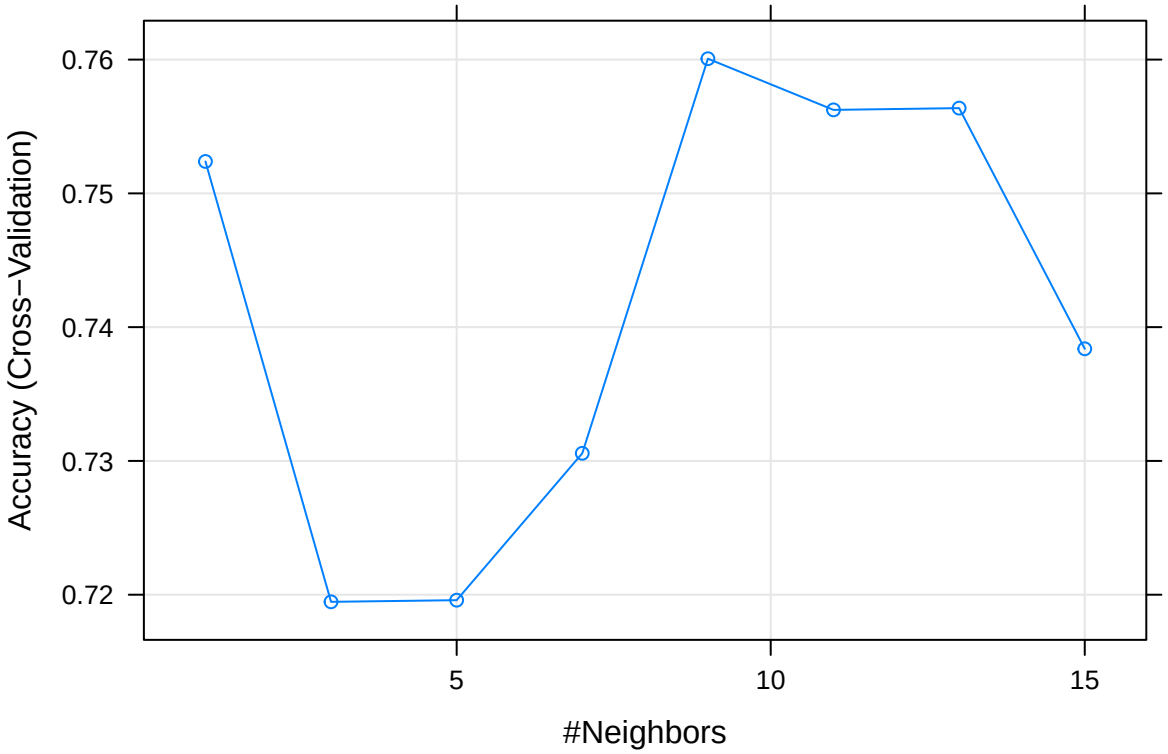
From the plot, we see that having three randomly selected predictors has the highest accuracy. However, the accuracy across different number of randomly selected predictors do not vary much and are generally close to each other. With contrast to number of trees trained, the accuracy of prediction is more sensitive to a small change in the number of randomly selected predictors. This is reasonable because different information will be learned by each individual tree when different variables are randomly selected as predictors, which would result in a variation of prediction accuracy.

Next, we will tune the classification tree using the complexity parameter ranging from 0 to 0.1 with a 10-fold cross-validation and plot them against the accuracy.



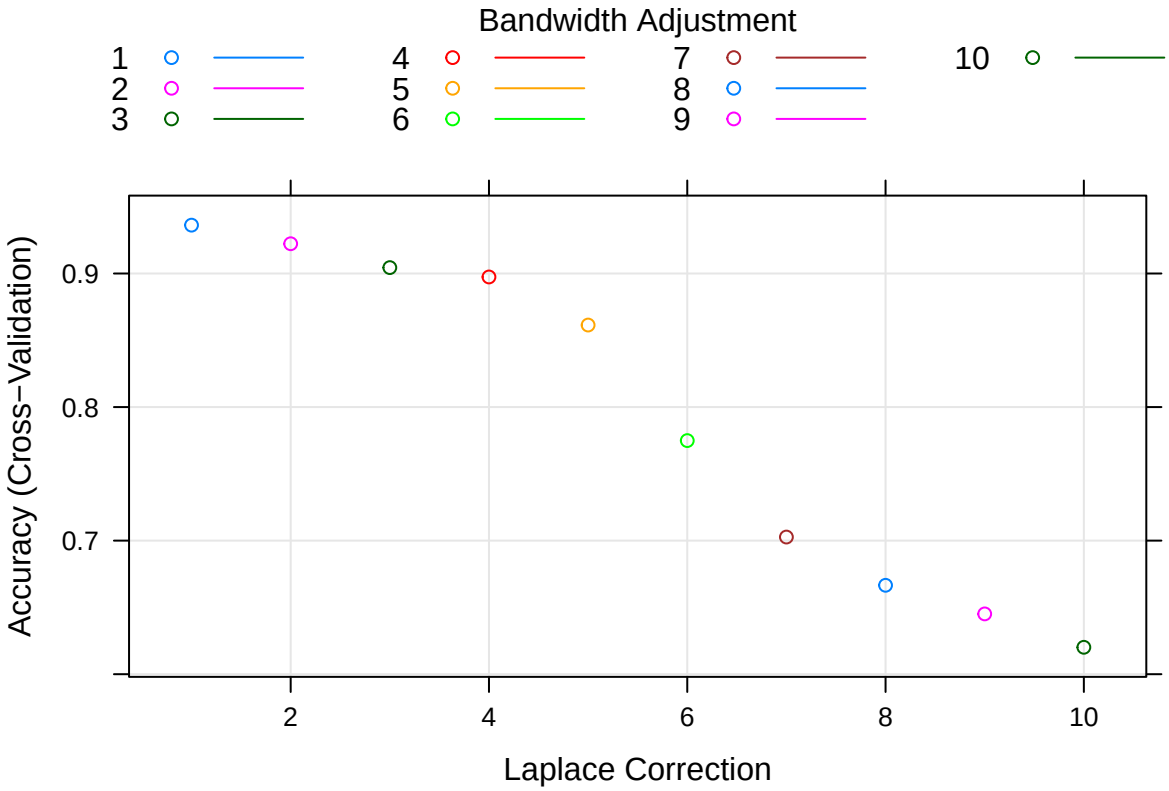
From the graph, we see that the highest accuracy occurs when the complexity parameter is 0.02. The accuracy generally decreases as complexity parameter becomes larger. The complexity parameter determines the minimum improvement on RSS for a new partition, so a larger complexity parameter will generally result in a smaller number of nodes. This explains the general decreasing pattern in accuracy with complexity parameter increasing.

We will tune the number of neighbors for k-nearest neighbors with a 10-fold cross-validation and plot them against the accuracy.



From the graph, we see that the highest accuracy occurs for one number of neighbor. This might be due to that the predictors in the data set have a wide and varying range. We also see that the overall accuracy of k-nearest neighbor is low even though the accuracy is higher when number of neighbor is one.

We will tune the bandwidth or flexibility of the kernel density and the laplace smoothing correction for naive bayes using a 10-fold cross-validation and plot them against the accuracy.



From the graph, we see that the accuracy is highest when bandwidth adjustment and laplace correction both equal to 1.

Finally, we train a logistic regression with a 10-fold cross-validation since there is no tuning parameter for logistic regression.

```
##                Accuracy
## random forest 0.9785532
## tree          0.9322386
## knn           0.7600666
## naive bayes   0.9361978
## logit         0.9504926
```

Now, let's compare the best tuned models. From the table, we see that the best tuned model for random forests has the highest accuracy, followed by the logistic regression. Classification tree and naive bayes have similar accuracy, and the k-nearest neighbors have the lowest accuracy.

Next, let's use the best tuned parameter for each algorithm to train the corresponding algorithm on the train set and evaluated it using the test set, and we will compare the performances of the best tuned models with the models using fixed parameters that we trained previously.

```
##                         Accuracy Sensitivity Specificity
## metric_logit           0.9008264   0.9361702   0.8783784
## metric_rf_bestmodel    0.9752066   0.9361702   1.0000000
## metric_tree_bestmodel 0.9586777   0.9148936   0.9864865
## metric_nb_bestmodel    0.9504132   0.9574468   0.9459459
## metric_knn_bestmodel   0.6611570   0.7872340   0.5810811
```

From the above table, we see that after training the best tuned model, random forests have the highest accuracy, followed by classification tree and naive bayes, then logistic regression, and k-nearest neighbors

have the lowest accuracy. Compared to the performance of each algorithm with fixed parameters, we see that random forests and classification tree have the same performance, and naive bayes increases its accuracy. However, k-nearest neighbors decreases accuracy by parameter tuning, and it is still the lowest in terms of accuracy, which underperforms logistic regression. In general, by tuning parameter, the machine learning algorithms random forests, naive bayes, and classification trees perform at least no worse than the traditional logistic regression.

Conclusion

In this project, we aim to compare the performance of nonparametric machine learning algorithms and parametric logistic regression under a classification problem for predicting diseases and try to determine which machine learning model performs the best. In particular, we used random forests, classification tree, k-nearest neighbors, and naive bayes to compare with the logistic regression. For each algorithm, we first split the data into one train set and one test set, and train the model using the train set. We compare the resulting accuracy, sensitivity, and specificity using the test set. Of the five models, random forests and classification tree have the highest accuracy. Naive bayes has the highest sensitivity, and random forests has the highest specificity. We also look at the F1 score with different cut-offs for each model. Of the five models, k-nearest neighbor, naive bayes and random forests appear to be influenced by the different cut-off values. These metrics will serve as a reference performance, and we next implement the strategy of parameter tuning with a 10-fold cross-validation for each of the machine learning algorithms on the train set. Specifically, we tune the number of trees trained and number of randomly selected predictors for random forests, complexity parameter for tree, number of neighbors for k-nearest neighbors, kernel bandwidth and laplace correction for naive bayes. For the logistic regression, we only apply a 10-fold cross-validation since there is no tuning parameter to provide a comparison with the machine learning algorithms. In general, we see that parameter tuning helps improve the performance of most of machine learning algorithms. After tuning the parameter, most machine learning algorithms except the k-nearest neighbors performs better than the logistic regression. Therefore, the nonparametric approaches using machine learning algorithms outperform the logistic regression for classifying the kidney disease in this data set, and the best predictive model in terms of accuracy is random forests.

Therefore, the analysis in this project was successful as we see that there was an improvement in accuracy in predicting the kidney disease using machine learning algorithms implemented in this project except k-nearest neighbors. However, because of the limitation of time, one should also consider the variance of the performance metrics including accuracy, sensitivity, and specificity by repeatedly splitting the data into test and train set and look at the variation among the performance metrics calculated for each data splitting. Another limitation of this project is that the project only considered the performance metrics of the machine learning algorithms, but did not analyze the time that each machine learning algorithm took to perform the training and prediction and compare it with time taken for training logistic regression.

Reference

UCI Machine Learning Repository: Chronic_kidney_disease Data Set, https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease.

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Irizarry, Rafael A. Introduction to Data Science: Data Analysis and Prediction Algorithms with R. CRC, 2020.

Appendix

```
library(farff)
library(Hmisc)
library(dplyr)
library(tidyverse)
library(caret)
library(ggplot2)
library(purrr)
library(reshape2)
library(plotROC)


data <- readARFF("chronic_kidney_disease.arff")
```

```
## Parse with reader=readr : chronic_kidney_disease.arff
```

```
## Warning: One or more parsing issues, call `problems()` on your data frame for details,
## e.g.:
##   dat <- vroom(...)
##   problems(dat)
```

```
## header: 0.009000; preproc: 0.002000; data: 0.051000; postproc: 0.000000; total: 0.062000
```

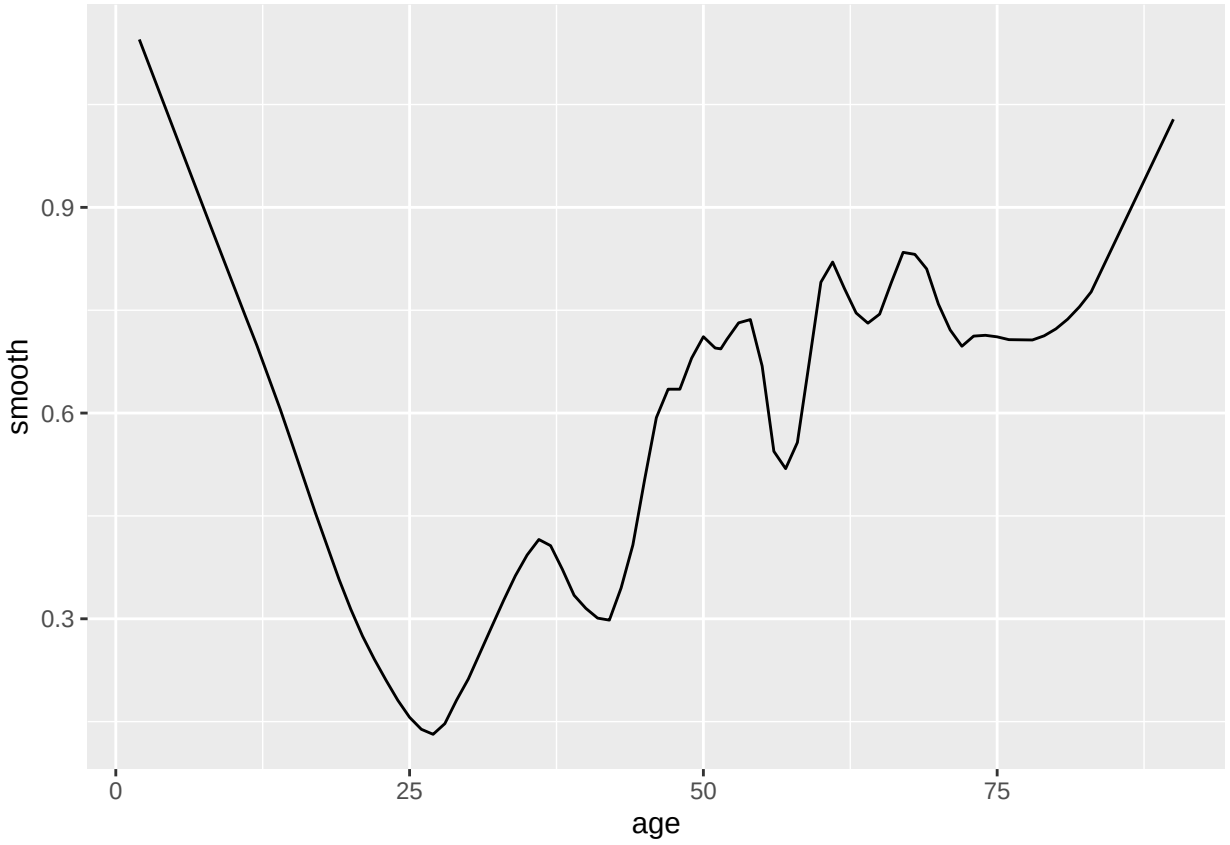
```
# data_remove <- na.omit(data)

#### let's first wrangle the data a little bit

data <- data %>% mutate(rbc = ifelse(rbc == "normal", 0, 1),
                        pc = ifelse(pc == "normal", 0, 1),
                        pcc = ifelse(pcc == "present", 1, 0),
                        ba = ifelse(ba == "present", 1, 0),
                        htn = ifelse(htn == "yes", 1, 0),
                        dm = ifelse(dm == "yes", 1, 0),
                        cad = ifelse(cad == "yes", 1, 0),
                        pe = ifelse(pe == "yes", 1, 0),
                        ane = ifelse(ane == "yes", 1, 0),
                        appet = ifelse(appet == "good", 1, 0),
                        class = ifelse(class == "ckd", 1, 0))

summary(data)
```

```
##       age              bp              sg         al         su
##  Min.   : 2.00   Min.   : 50.00   1.005:  7   0   :199   0   :290
##  1st Qu.:42.00   1st Qu.: 70.00   1.010: 84   1   : 44   1   : 13
##  Median :55.00   Median : 80.00   1.015: 75   2   : 43   2   : 18
##  Mean   :51.48   Mean   : 76.47   1.020:106   3   : 43   3   : 14
##  3rd Qu.:64.50   3rd Qu.: 80.00   1.025: 81   4   : 24   4   : 13
##  Max.   :90.00   Max.   :180.00   NA's : 47   5   :  1   5   :  3
##  NA's   :9       NA's   :12                   NA's: 46   NA's: 49
##       rbc              pc             pcc              ba
```

```
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000
##  Median :0.0000   Median :0.0000   Median :0.0000   Median :0.00000
##  Mean   :0.1895   Mean   :0.2269   Mean   :0.1061   Mean   :0.05556
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
##  NA's   :152      NA's   :65       NA's   :4        NA's   :4
##       bgr              bu               sc               sod
##  Min.   : 22   Min.   :  1.50   Min.   : 0.400   Min.   :  4.5
##  1st Qu.: 99   1st Qu.: 27.00   1st Qu.: 0.900   1st Qu.:135.0
##  Median :121   Median : 42.00   Median : 1.300   Median :138.0
##  Mean   :148   Mean   : 57.43   Mean   : 3.072   Mean   :137.5
##  3rd Qu.:163   3rd Qu.: 66.00   3rd Qu.: 2.800   3rd Qu.:142.0
##  Max.   :490   Max.   :391.00   Max.   :76.000   Max.   :163.0
##  NA's   :44    NA's   :19       NA's   :17       NA's   :87
##       pot              hemo             pcv              wbcc
##  Min.   : 2.500   Min.   : 3.10   Min.   : 9.00   Min.   : 2200
##  1st Qu.: 3.800   1st Qu.:10.30   1st Qu.:32.00   1st Qu.: 6500
##  Median : 4.400   Median :12.65   Median :40.00   Median : 8000
##  Mean   : 4.627   Mean   :12.53   Mean   :38.87   Mean   : 8414
##  3rd Qu.: 4.900   3rd Qu.:15.00   3rd Qu.:45.00   3rd Qu.: 9800
##  Max.   :47.000   Max.   :17.80   Max.   :54.00   Max.   :26400
##  NA's   :88       NA's   :52      NA's   :72      NA's   :108
##       rbcc             htn               dm               cad
##  Min.   :2.100   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:3.900   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000
##  Median :4.800   Median :0.0000   Median :0.0000   Median :0.00000
##  Mean   :4.707   Mean   :0.3693   Mean   :0.3444   Mean   :0.08586
##  3rd Qu.:5.400   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.00000
##  Max.   :8.000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
##  NA's   :131     NA's   :2        NA's   :8        NA's   :4
##      appet             pe               ane              class
##  Min.   :0.000   Min.   :0.000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:1.000   1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :1.000   Median :0.000   Median :0.0000   Median :1.0000
##  Mean   :0.794   Mean   :0.191   Mean   :0.1504   Mean   :0.6228
##  3rd Qu.:1.000   3rd Qu.:0.000   3rd Qu.:0.0000   3rd Qu.:1.0000
##  Max.   :1.000   Max.   :1.000   Max.   :1.0000   Max.   :1.0000
##  NA's   :2       NA's   :2       NA's   :1        NA's   :5
```

```
### we see that there are some missing values in the data set
### let's fill the NA values with column means for continuous outcome
### and fill the NA values with zero for binary outcome
### for the variable sg specifically, fill the NA values with the lowest category which is 1.005

# colnames(data)

cols <- c("age", "bp", "bgr", "bu", "sc", "sod", "pot", "hemo", "pcv", "wbcc", "rbcc")

for(i in 1:length(cols)) {
  col <- cols[i]
  data[, col][is.na(data[, col])] <- mean(data[, col], na.rm = TRUE)
}
```

```r
factor_cols <- c("al", "su", "rbc", "pc", "pcc", "ba", "htn", "dm", "cad", "appet", "pe", "ane", "class

for(i in 1:length(factor_cols)) {
  col <- factor_cols[i]
  data[, col][is.na(data[, col])] <- 0
  data[, col] <- as.factor(data[, col])
}

data$sg[is.na(data$sg)] <- 1.005
data$sg <- as.factor(data$sg)

fit <- loess(as.character(class) ~ age, data = data, span = 0.2)
data %>%
  mutate(smooth = fit$fitted) %>%
  ggplot(aes(age, smooth)) + geom_line()
```



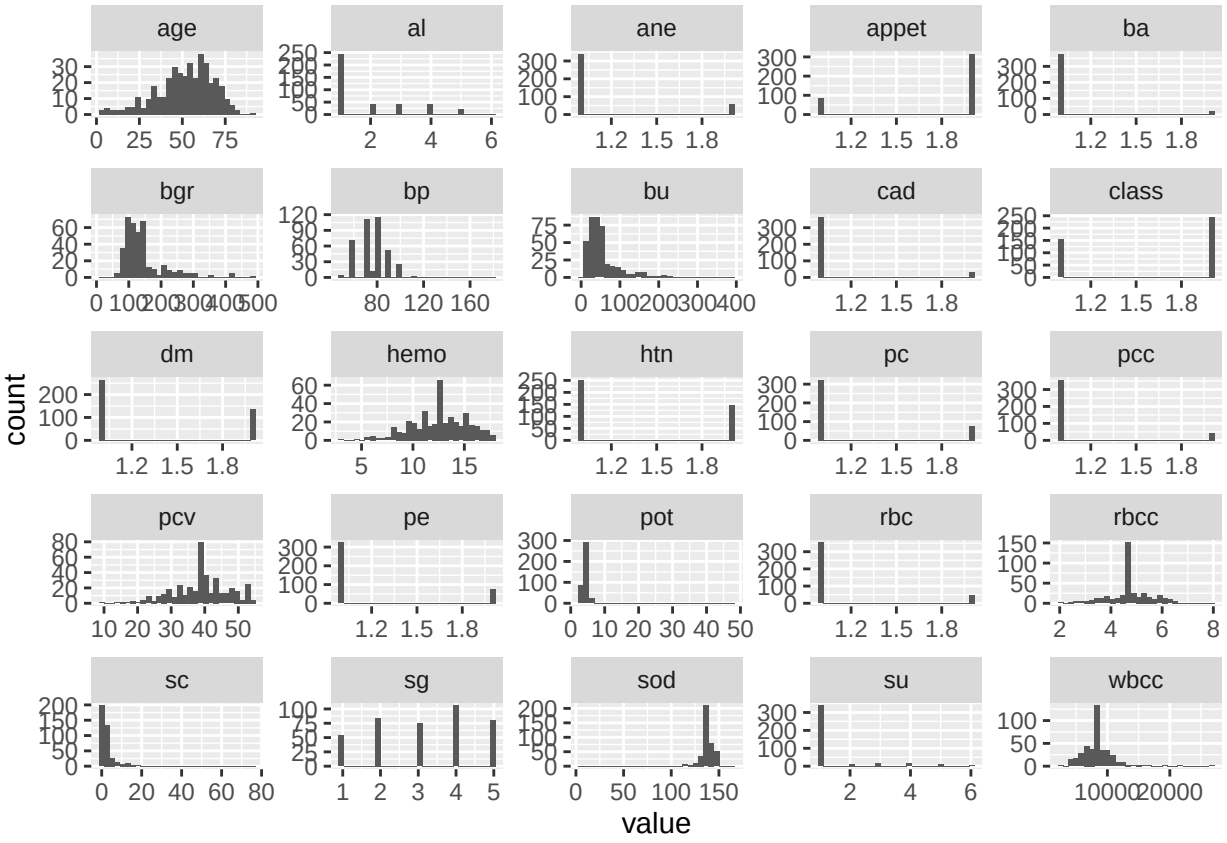

```r
### Let's look at the correlation among the variables
corr_mat <- rcorr(as.matrix(data), type = "pearson")

pvalues <- corr_mat$P

correlations <- corr_mat$r

pvalue_class <- as.matrix(pvalues[, "class"])
colnames(pvalue_class) <- "p values"
```

```r
corr_class <- as.matrix(correlations[, "class"])
colnames(corr_class) <- "pearson correlation with outcome"

class_corr_pvalue <- cbind(corr_class, pvalue_class)

class_corr_pvalue
```

```
##       pearson correlation with outcome     p values
## age                          0.21787598 1.098513e-05
## bp                           0.29136656 2.877686e-09
## sg                          -0.65970612 0.000000e+00
## al                           0.52337662 0.000000e+00
## su                           0.26625071 6.436108e-08
## rbc                          0.28870505 4.059777e-09
## pc                           0.35700856 1.814104e-13
## pcc                          0.23748477 1.557398e-06
## ba                           0.19087904 1.225307e-04
## bgr                          0.39005523 4.440892e-16
## bu                           0.37981834 3.552714e-15
## sc                           0.29697131 1.377874e-09
## sod                         -0.33828027 3.642864e-12
## pot                          0.07699492 1.242047e-01
## hemo                        -0.72217253 0.000000e+00
## pcv                         -0.68935489 0.000000e+00
## wbcc                         0.21059989 2.171006e-05
## rbcc                        -0.58231038 0.000000e+00
## htn                          0.57113322 0.000000e+00
## dm                           0.52126341 0.000000e+00
## cad                          0.24115247 1.059976e-06
## appet                       -0.35747752 1.678657e-13
## pe                           0.37010498 1.976197e-14
## ane                          0.31798693 7.537082e-11
## class                        1.00000000           NA
```

```r
data %>%
  mutate(class = as.numeric(class),
         sg = as.numeric(sg),
         al = as.numeric(al),
         su = as.numeric(su),
         rbc = as.numeric(rbc),
         pc = as.numeric(pc),
         pcc = as.numeric(pcc),
         ba = as.numeric(ba),
         htn = as.numeric(htn),
         dm = as.numeric(dm),
         cad = as.numeric(cad),
         appet = as.numeric(appet),
         pe = as.numeric(pe),
         ane = as.numeric(ane)) %>%
  pivot_longer(age:class) %>%
  ggplot(aes(value)) + geom_histogram() +
    facet_wrap(~ name, scales = "free")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
#### Let's start with logistics regression

set.seed(2022)
test_index <- createDataPartition(data$class, times = 1, p = 0.3, list = FALSE)

train_dta <- data[-test_index, ]
test_dta <- data[test_index, ]


logit_fit <- glm(as.factor(class) ~., data = train_dta, family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
y_hat_logit_raw <- predict(logit_fit, newdata = test_dta, type = "response")

y_hat_logit <- ifelse(y_hat_logit_raw < 0.5, 0, 1) %>% factor()
tab_logit <- table(y_hat_logit, test_dta$class)
conf_mat_logit <- confusionMatrix(y_hat_logit, test_dta$class)

##### random forest ########
library(randomForest)
```

```
rf_fit <- randomForest(as.factor(class) ~., data = train_dta, ntree = 100)

y_hat_rf <- predict(rf_fit, newdata = test_dta)


tab_rf <- table(y_hat_rf, test_dta$class)
conf_mat_rf <- confusionMatrix(y_hat_rf, test_dta$class)

##### classification trees #######
library(rpart)

tree_fit <- rpart(as.factor(class) ~., data = train_dta)

y_hat_tree <- predict(tree_fit, newdata = test_dta, type = "class")

tab_tree <- table(y_hat_tree, test_dta$class)
conf_mat_tree <- confusionMatrix(y_hat_tree, test_dta$class)


##### Naive bayes ############
library(naivebayes)

naive_fit <- naive_bayes(as.factor(class) ~., data = train_dta)
```

```
## Warning: naive_bayes(): Feature al - zero probabilities are present. Consider
## Laplace smoothing.

## Warning: naive_bayes(): Feature su - zero probabilities are present. Consider
## Laplace smoothing.

## Warning: naive_bayes(): Feature rbc - zero probabilities are present. Consider
## Laplace smoothing.

## Warning: naive_bayes(): Feature ba - zero probabilities are present. Consider
## Laplace smoothing.

## Warning: naive_bayes(): Feature cad - zero probabilities are present. Consider
## Laplace smoothing.

## Warning: naive_bayes(): Feature pe - zero probabilities are present. Consider
## Laplace smoothing.

## Warning: naive_bayes(): Feature ane - zero probabilities are present. Consider
## Laplace smoothing.
```

```
y_hat_naive <- predict(naive_fit, newdata = test_dta)
```

```
## Warning: predict.naive_bayes(): more features in the newdata are provided as
## there are probability tables in the object. Calculation is performed based on
## features to be found in the tables.
```

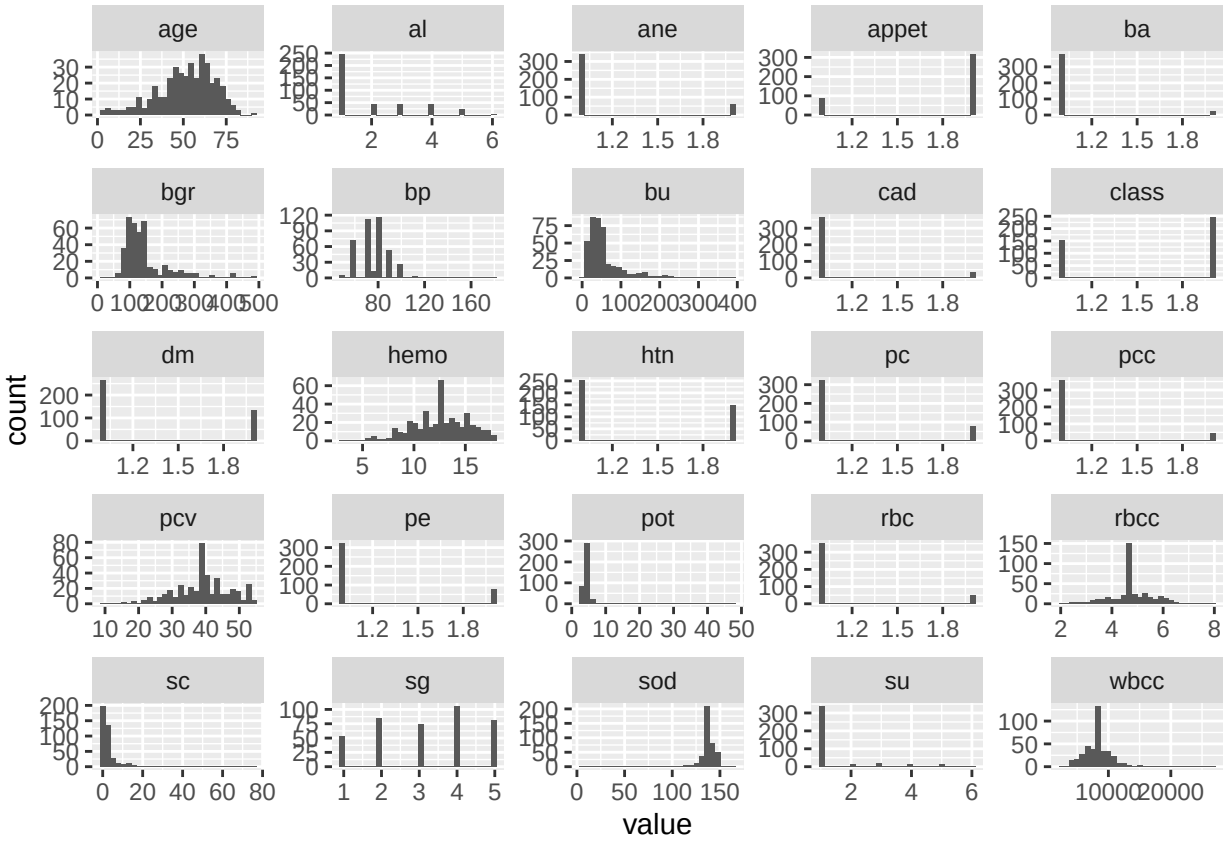
```r
tab_nb <- table(y_hat_naive, test_dta$class)
conf_mat_nb <- confusionMatrix(y_hat_naive, test_dta$class)


###### knn ############

knn_fit <- knn3(train_dta[,-25],train_dta$class, k = 5)
y_hat_knn <- predict(knn_fit, newdata = test_dta[, -25], type = 'class')

tab_knn <- table(y_hat_knn, test_dta$class)
conf_mat_knn <- confusionMatrix(y_hat_knn, test_dta$class)

##### Table to compare accuracy, sensitivity and specificity

metric_logit <- c(conf_mat_logit$overall[["Accuracy"]],
                  conf_mat_logit$byClass[c("Sensitivity","Specificity")])
metric_rf <- c(conf_mat_rf$overall[["Accuracy"]],
               conf_mat_rf$byClass[c("Sensitivity","Specificity")])
metric_tree <- c(conf_mat_tree$overall[["Accuracy"]],
                 conf_mat_tree$byClass[c("Sensitivity","Specificity")])
metric_nb <- c(conf_mat_nb$overall[["Accuracy"]],
               conf_mat_nb$byClass[c("Sensitivity","Specificity")])
metric_knn <- c(conf_mat_knn$overall[["Accuracy"]],
                conf_mat_knn$byClass[c("Sensitivity","Specificity")])

table_compare <- rbind(metric_logit, metric_rf, metric_tree, metric_nb, metric_knn)

colnames(table_compare) <- c("Accuracy", "Sensitivity", "Specificity")

table_compare
```

```
##               Accuracy Sensitivity Specificity
## metric_logit 0.9008264   0.9361702   0.8783784
## metric_rf    0.9752066   0.9361702   1.0000000
## metric_tree  0.9586777   0.9148936   0.9864865
## metric_nb    0.9090909   0.9574468   0.8783784
## metric_knn   0.6942149   0.8510638   0.5945946
```

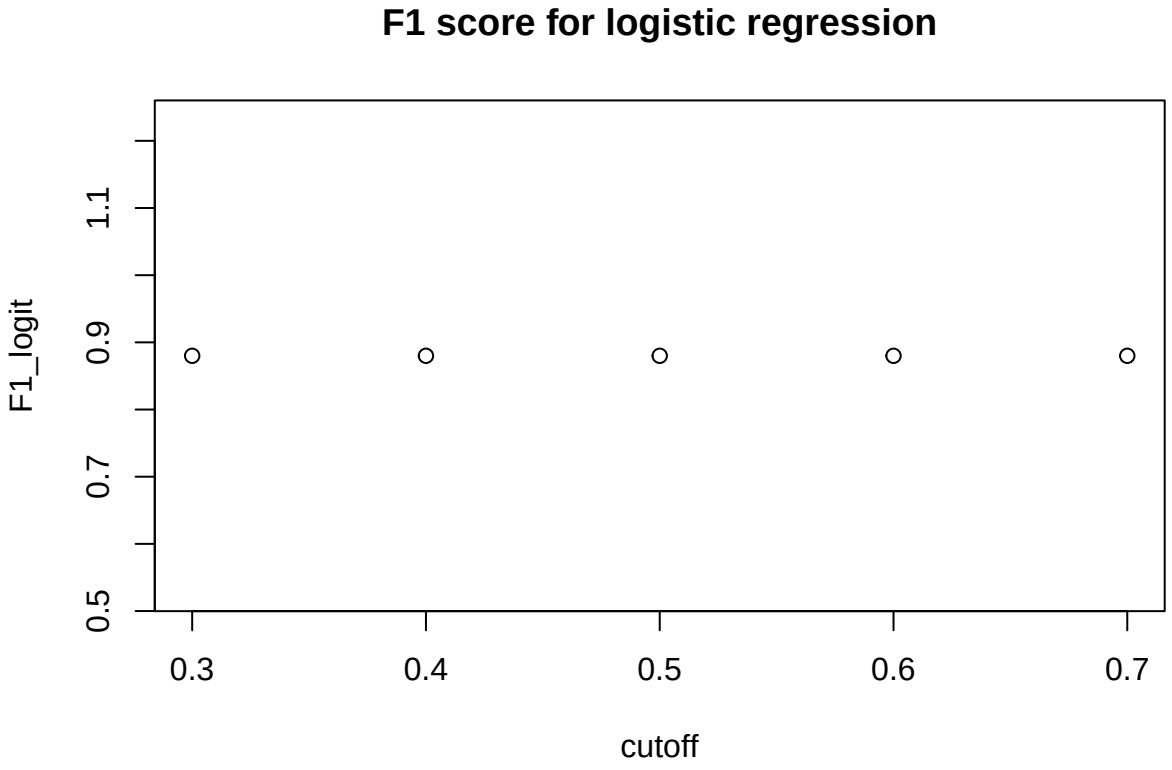
```r
data %>%
  mutate(class = as.numeric(class),
         sg = as.numeric(sg),
         al = as.numeric(al),
         su = as.numeric(su),
         rbc = as.numeric(rbc),
         pc = as.numeric(pc),
         pcc = as.numeric(pcc),
         ba = as.numeric(ba),
         htn = as.numeric(htn),
         dm = as.numeric(dm),
         cad = as.numeric(cad),
         appet = as.numeric(appet),
         pe = as.numeric(pe),
         ane = as.numeric(ane)) %>%
```

```
  pivot_longer(age:class) %>%
  ggplot(aes(value)) + geom_histogram() +
    facet_wrap(~ name, scales = "free")
```

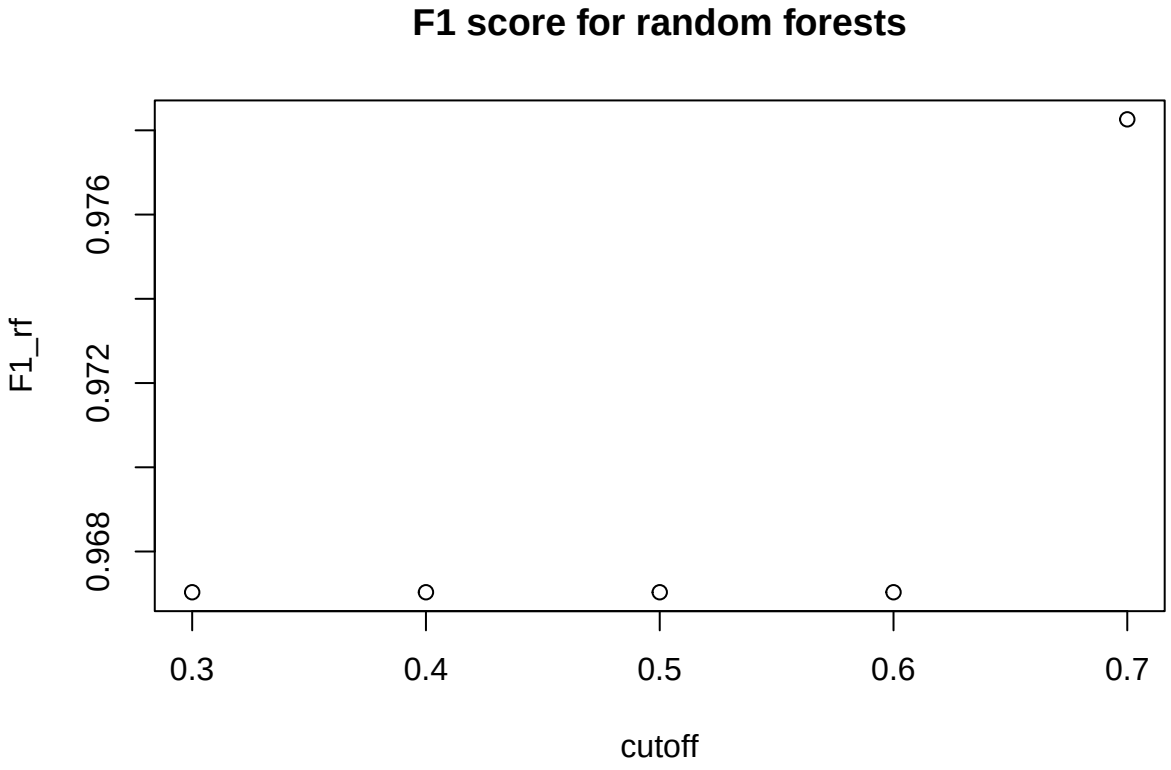
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



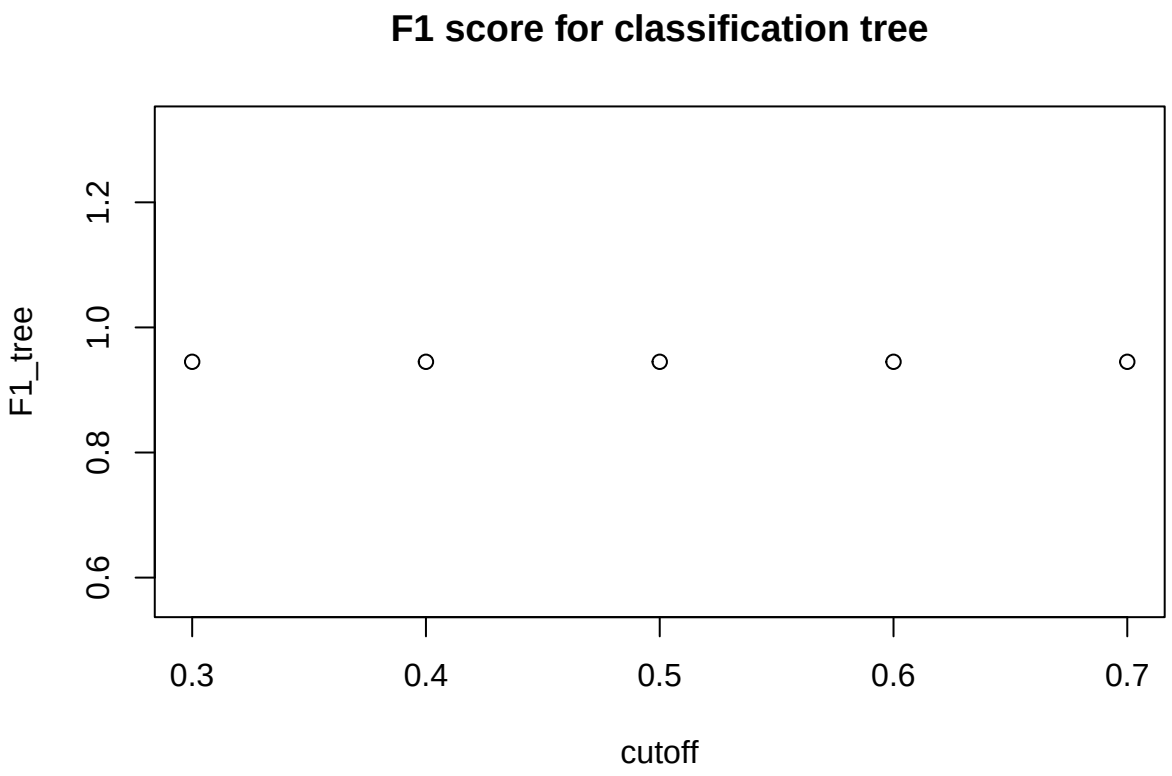

```
cutoff <- seq(0.3, 0.7, by = 0.1)
F1_logit <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(y_hat_logit_raw > x, 1, 0) %>% factor()
  F_meas(data = y_hat, reference = test_dta$class)
})
plot(cutoff, F1_logit, main = "F1 score for logistic regression")
```



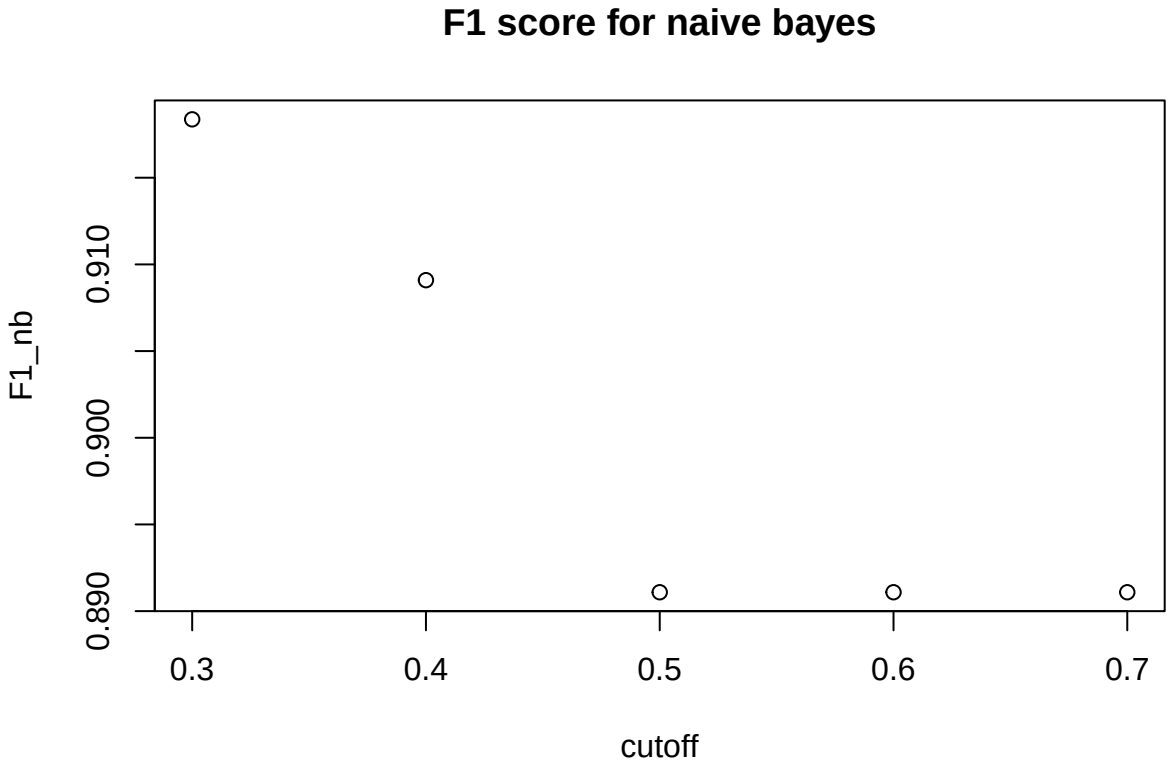

```
y_hat_rf_raw <- predict(rf_fit, newdata = test_dta, type = "prob")[, 2]
F1_rf <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(y_hat_rf_raw > x, 1, 0) %>% factor()
  F_meas(data = y_hat, reference = test_dta$class)
})
plot(cutoff, F1_rf, main = "F1 score for random forests")
```



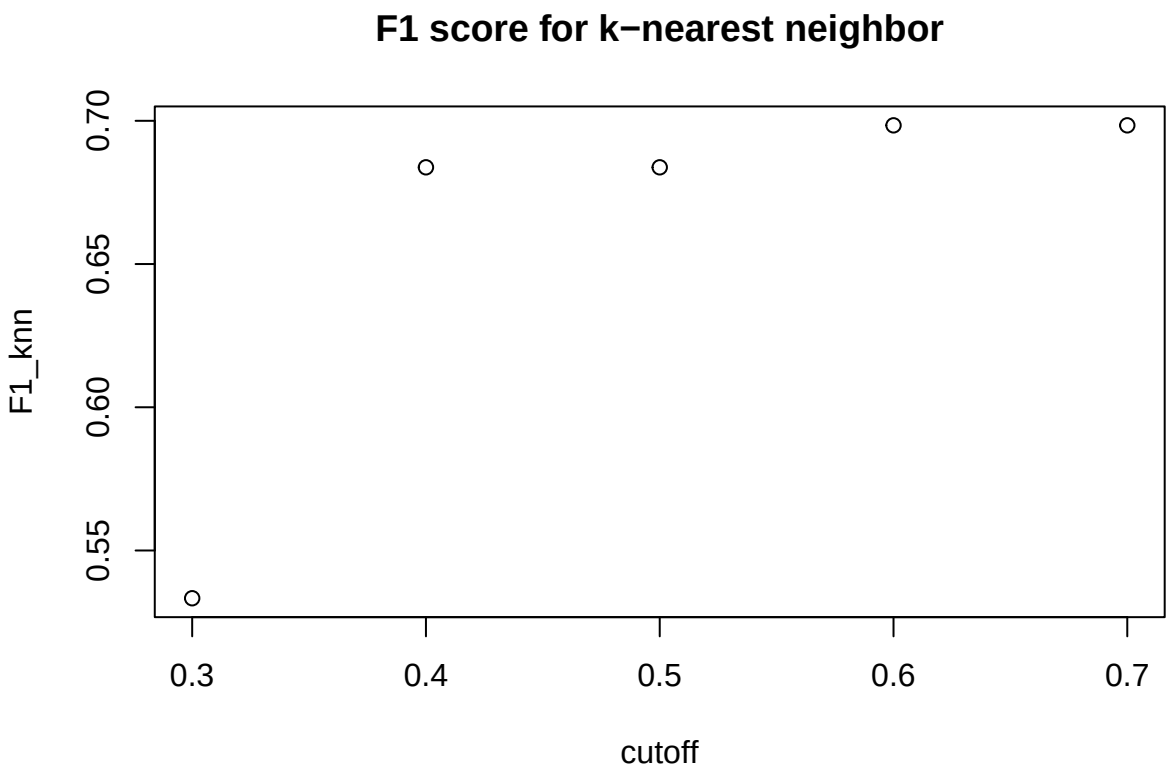

```
y_hat_tree_raw <- predict(tree_fit, newdata = test_dta, type = "prob")[, 2]
F1_tree <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(y_hat_tree_raw > x, 1, 0) %>% factor()
  F_meas(data = y_hat, reference = test_dta$class)
})
plot(cutoff, F1_tree, main = "F1 score for classification tree")
```




```
y_hat_nb_raw <- predict(naive_fit, newdata = test_dta[, -25], type = "prob")[, 2]
F1_nb <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(y_hat_nb_raw > x, 1, 0) %>% factor()
  F_meas(data = y_hat, reference = test_dta$class)
})
plot(cutoff, F1_nb, main = "F1 score for naive bayes")
```




```
y_hat_knn_raw <- predict(knn_fit, newdata = test_dta[, -25], type = 'prob')[, 2]
F1_knn <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(y_hat_knn_raw > x, 1, 0) %>% factor()
  F_meas(data = y_hat, reference = test_dta$class)
})
plot(cutoff, F1_knn, main = "F1 score for k-nearest neighbor")
```

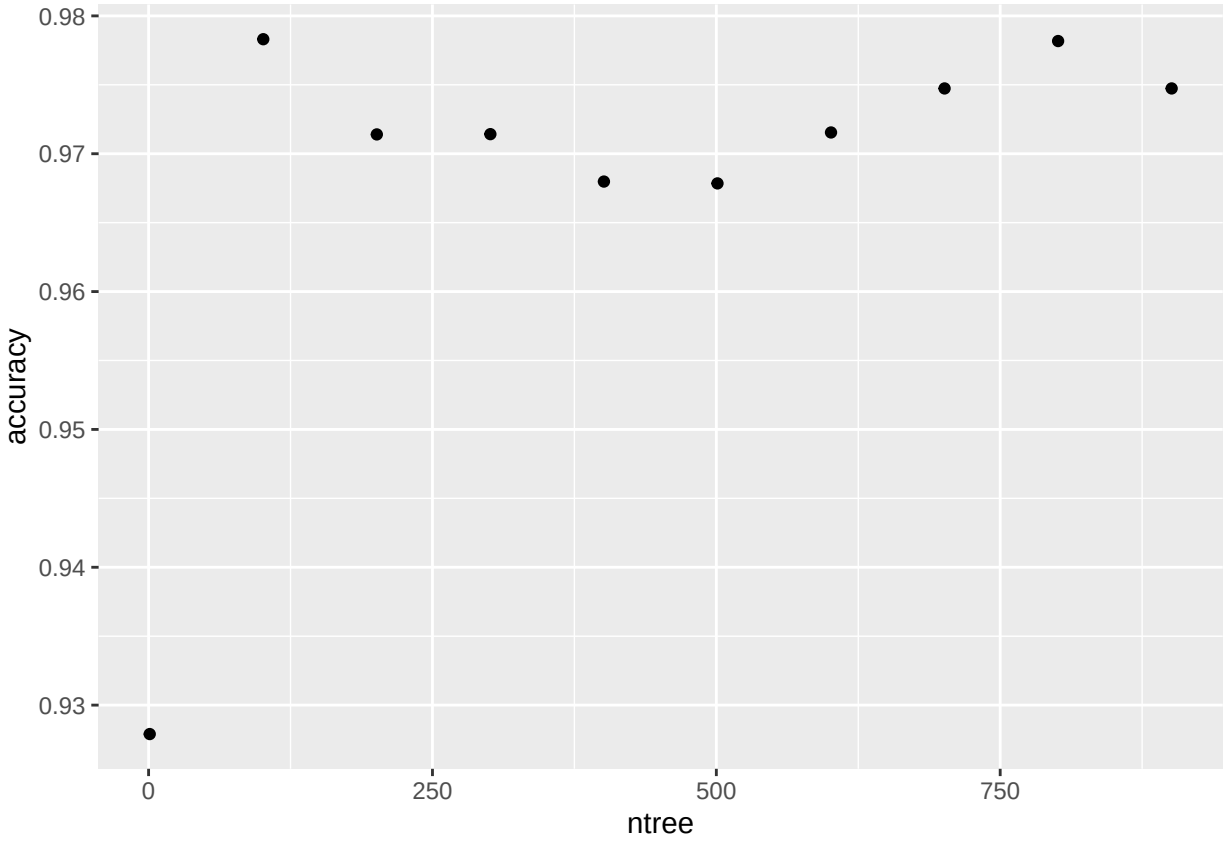



```
table_F1 <- rbind(F1_logit, F1_rf, F1_tree, F1_nb, F1_knn)
rownames(table_F1) <- c("F1 score logit",
                        "F1 score random forests",
                        "F1 score classification tree",
                        "F1 score naive bayes",
                        "F1 score k-nearest neighbor")

colnames(table_F1) <- c(0.3, 0.4, 0.5, 0.6, 0.7)

table_F1
```

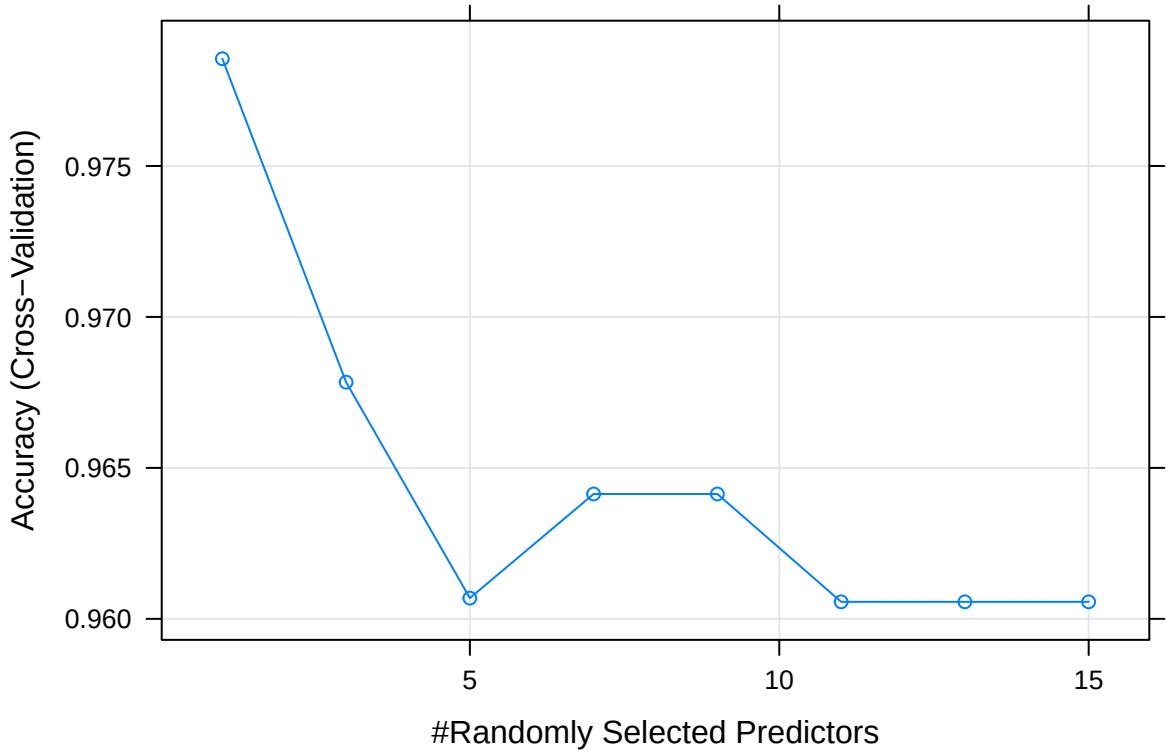
```
##                                 0.3       0.4       0.5       0.6       0.7
## F1 score logit              0.8800000 0.8800000 0.8800000 0.8800000 0.8800000
## F1 score random forests     0.9670330 0.9670330 0.9670330 0.9670330 0.9782609
## F1 score classification tree 0.9450549 0.9450549 0.9450549 0.9450549 0.9450549
## F1 score naive bayes        0.9183673 0.9090909 0.8910891 0.8910891 0.8910891
## F1 score k-nearest neighbor  0.5333333 0.6837607 0.6837607 0.6984127 0.6984127
```

```
##### 10-fold cv #######

set.seed(2022)

###### random forest ######
ntree <- seq(1, 1000, by = 100)
accuracy <- sapply(ntree, function(n){
  train(as.factor(class) ~ ., method = "rf", data = train_dta,
              tuneGrid = data.frame(mtry = 3),
              ntree = n, trControl = trainControl(method = "cv", number = 10))$results$Accuracy
})
```

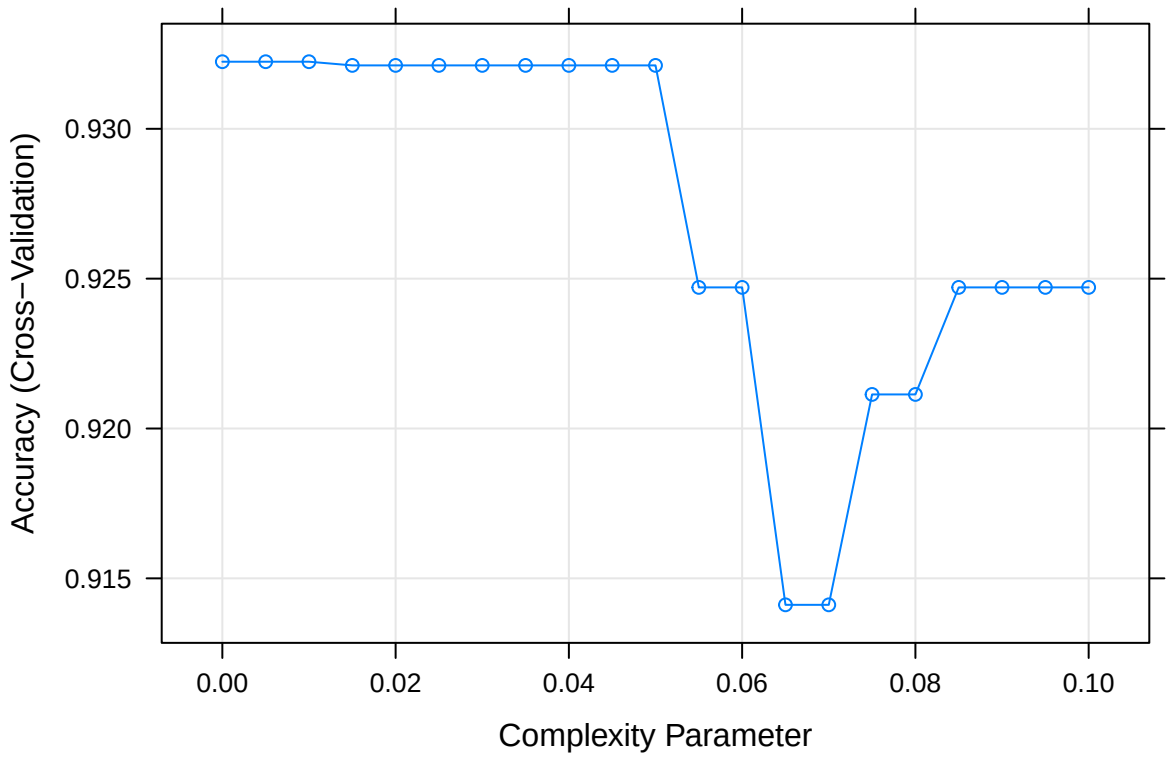
```
qplot(ntree, accuracy)
```



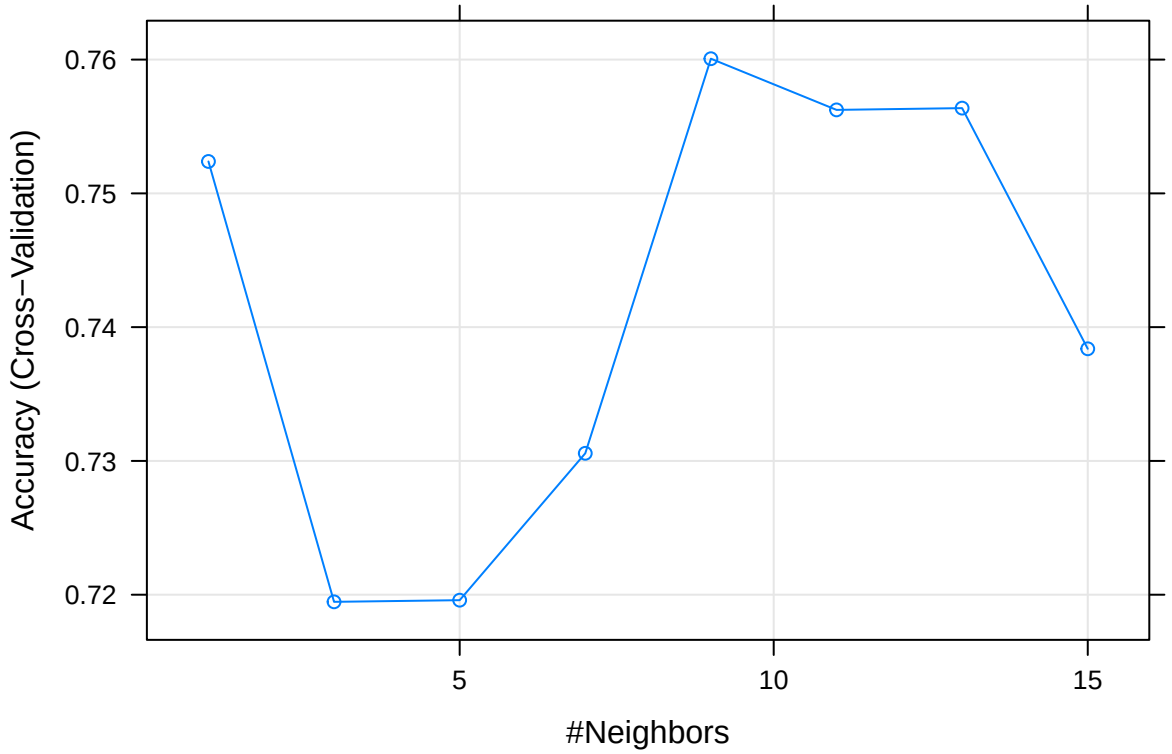

```
###### random forest tuning ########

train_rf <- train(as.factor(class) ~ ., method = "rf", data = train_dta,
                  tuneGrid = data.frame(mtry = seq(1, 15, by = 2)),
                  nodesize = 10, trControl = trainControl(method = "cv", number = 10))

plot(train_rf)
```



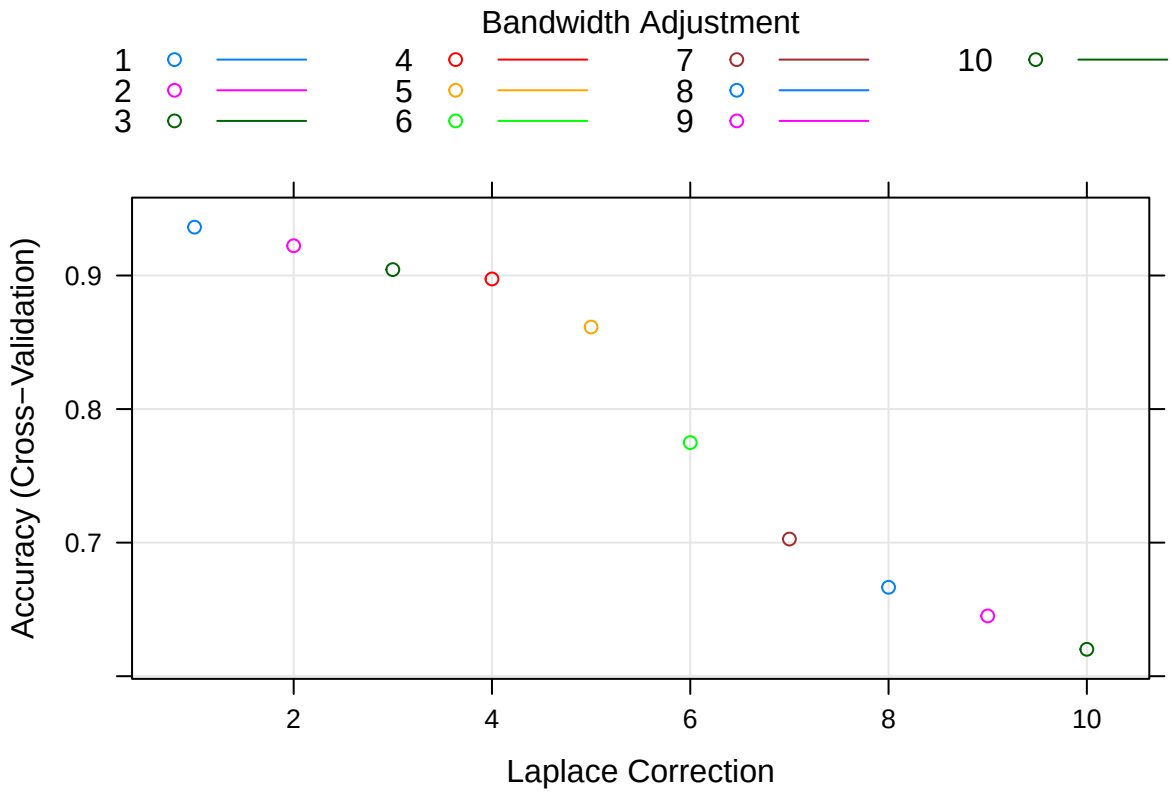

```
rf_bestTune <- train_rf$bestTune

rf_result <- train_rf$results

######## trees ############

train_rpart <- train(as.factor(class) ~ .,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0.0, 0.1, by = 0.005)),
                     data = train_dta, trControl = trainControl(method = "cv", number = 10))
plot(train_rpart)
```




```
tree_bestTune <- train_rpart$bestTune

tree_result <- train_rpart$results

########## knn ############

train_knn <- train(as.factor(class) ~ ., method = "knn", data = train_dta,
                   tuneGrid = data.frame(k = seq(1, 15, by = 2)), trControl = trainControl(method = "cv

plot(train_knn)
```




```
knn_bestTune <- train_knn$bestTune

knn_result <- train_knn$results

######## naive bayes #############

train_nb <- train(as.factor(class) ~ ., method = "naive_bayes", data = train_dta,
                  tuneGrid = data.frame(laplace = 1:10, usekernel = TRUE, adjust = 1:10),
                  trControl = trainControl(method = "cv", number = 10))

plot(train_nb)
```




```
nb_bestTune <- train_nb$bestTune

nb_result <- train_nb$results

########## logistic regression #########

train_logit <- train(as.factor(class) ~ ., method = "glm", family = "binomial", data = train_dta,
                     trControl = trainControl(method = "cv", number = 10))
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
logit_result <- train_logit$results

######## table comparing accuracy of best tuned #################

table_10cv <- rbind(max(as.vector(rf_result$Accuracy)), max(as.vector(tree_result$Accuracy)),
                    max(as.vector(knn_result$Accuracy)), max(as.vector(nb_result$Accuracy)),
                    max(as.vector(logit_result$Accuracy)))
colnames(table_10cv) <- "Accuracy"
rownames(table_10cv) <- c("random forest", "tree", "knn", "naive bayes", "logit")

as.data.frame(table_10cv)
```

```
##                Accuracy
## random forest 0.9785532
## tree          0.9322386
## knn           0.7600666
## naive bayes   0.9361978
## logit         0.9504926
```

```
##### predict using the best tuned model #####

fit_rf_bestmodel <- randomForest(as.factor(class) ~., data = train_dta, mtry = rf_bestTune$mtry, nodesi

y_hat_rf_bestmodel <- predict(fit_rf_bestmodel, newdata = test_dta)
conf_mat_rf_bestmodel <- confusionMatrix(y_hat_rf_bestmodel, test_dta$class)

fit_tree_bestmodel <- rpart(as.factor(class) ~., data = train_dta, control = rpart.control(cp = tree_bes
y_hat_tree_bestmodel <- predict(fit_tree_bestmodel, newdata = test_dta, type = "class")
conf_mat_tree_bestmodel <- confusionMatrix(y_hat_tree_bestmodel, test_dta$class)

fit_naive_bestmodel <- naive_bayes(as.factor(class) ~., data = train_dta,
                                   laplace = nb_bestTune$laplace,
                                   usekernel = nb_bestTune$usekernel,
                                   adjust = nb_bestTune$adjust)

y_hat_naive_bestmodel <- predict(fit_naive_bestmodel, newdata = test_dta)
```

```
## Warning: predict.naive_bayes(): more features in the newdata are provided as
## there are probability tables in the object. Calculation is performed based on
## features to be found in the tables.
```

```
conf_mat_nb_bestmodel <- confusionMatrix(y_hat_naive_bestmodel, test_dta$class)

fit_knn_bestmodel <- knn3(train_dta[,-25], train_dta$class, k = knn_bestTune$k)
y_hat_knn_bestmodel <- predict(fit_knn_bestmodel, newdata = test_dta[, -25], type = 'class')
conf_mat_knn_bestmodel <- confusionMatrix(y_hat_knn_bestmodel, test_dta$class)

metric_rf_bestmodel <- c(conf_mat_rf_bestmodel$overall[["Accuracy"]],
               conf_mat_rf_bestmodel$byClass[c("Sensitivity","Specificity")])
metric_tree_bestmodel <- c(conf_mat_tree_bestmodel$overall[["Accuracy"]],
                 conf_mat_tree_bestmodel$byClass[c("Sensitivity","Specificity")])
metric_nb_bestmodel <- c(conf_mat_nb_bestmodel$overall[["Accuracy"]],
               conf_mat_nb_bestmodel$byClass[c("Sensitivity","Specificity")])
metric_knn_bestmodel <- c(conf_mat_knn_bestmodel$overall[["Accuracy"]],
                conf_mat_knn_bestmodel$byClass[c("Sensitivity","Specificity")])

table_compare_bestmodel <- rbind(metric_logit,
                                 metric_rf_bestmodel,
                                 metric_tree_bestmodel,
                                 metric_nb_bestmodel,
                                 metric_knn_bestmodel)

colnames(table_compare_bestmodel) <- c("Accuracy", "Sensitivity", "Specificity")

table_compare_bestmodel
```

```
##                         Accuracy Sensitivity Specificity
## metric_logit           0.9008264   0.9361702   0.8783784
## metric_rf_bestmodel    0.9752066   0.9361702   1.0000000
## metric_tree_bestmodel 0.9586777   0.9148936   0.9864865
## metric_nb_bestmodel    0.9504132   0.9574468   0.9459459
## metric_knn_bestmodel   0.6611570   0.7872340   0.5810811
```