Final

Team 10

5/2/2021

Introduction

The data source we use for this final project comes from the UCI machine learning repository. It contains information regarding the red and white variants of the Portuguese "Vinho Verde" wine. The dataset has 1599 observations and 12 variables, which are the fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality. The fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol are independent variables and are continuous. Quality is the response variable and is measured based on a score of 0 to 10. Later, we re-categorized quality to a binary variable called qual. Qual is considered good if the quality score is greater than 5, otherwise is considered poor.

By doing this project, we hope to classify the quality of each observation into either good or poor based on their performance on the physicochemical tests.

Exploratory analysis

In total 855 wines were classified as "Good" quality and 744 as "Poor" quality. The average values for the 11 features for wines of good and poor quality was shown in Table 1. Fixed acidity, volatile acidity, citric acid, chlorides, free sulfur dioxide, total sulfur dioxide, density, sulphates and alcohol were significantly associated with the wine quality (P-values for t-tests < 0.05), which suggests important predictors.

We also built the density plots to explore the distribution of the 11 continuous variables over "Poor" and "Good" quality of wine (Figure 1). The plots showed that wine with good and poor quality did not differ for PH and residual sugar, while different types of wine differs in other variables, which was consistent with the t-test results.

```
# Read in data
wine <- read.csv("./winequality-red.csv", stringsAsFactors = FALSE) %>%
  janitor::clean_names() %>%
  na.omit() %>%
  mutate(qual = case_when(quality > 5 ~ "good", quality <= 5 ~ "poor")) %>%
  mutate(qual = as.factor(qual)) %>%
  dplyr::select(-quality)

theme1 <- transparentTheme(trans = .4)
trellis.par.set(theme1)
featurePlot(x = wine[, 1:11],
            y = wine$qual,
            scales = list(x = list(relation = "free"),
                          y = list(relation = "free")),
            plot = "density", pch = "|",
            auto.key = list(columns = 2))
```

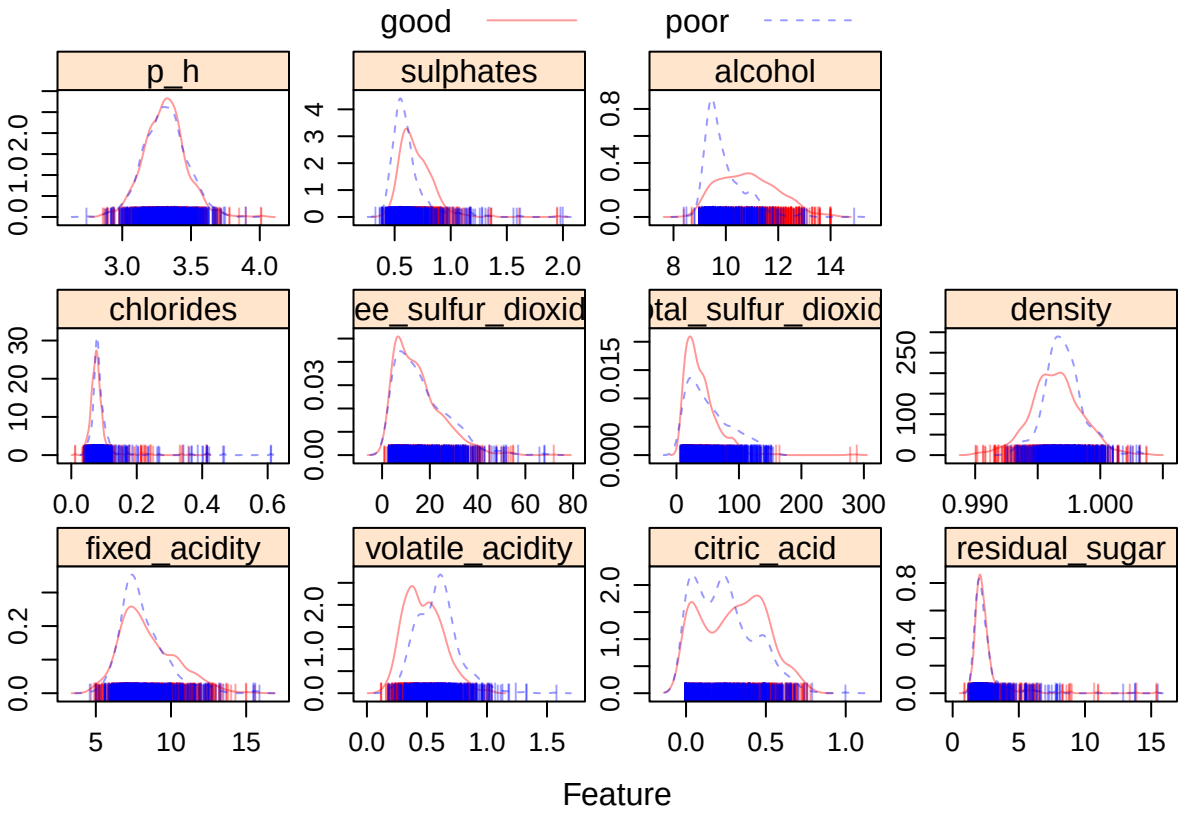


Figure 1. Descriptive plots between wine quality and predictive features.

Table 1. Basic characteristics of wines over good and poor quality.

```
# Create a variable list which we want in Table 1
listVars <- c("fixed_acidity", "volatile_acidity","citric_acid", "residual_sugar", "chlorides", "free_su

tab1 <- CreateTableOne(vars = listVars, strata = 'qual', data = wine)

tab1
```

```
##                                    Stratified by qual
##                                     good           poor           p      test
##   n                                   855            744
##   fixed_acidity (mean (SD))          8.47 (1.86)    8.14 (1.57)  <0.001
##   volatile_acidity (mean (SD))       0.47 (0.16)    0.59 (0.18)  <0.001
##   citric_acid (mean (SD))            0.30 (0.20)    0.24 (0.18)  <0.001
##   residual_sugar (mean (SD))         2.54 (1.42)    2.54 (1.39)   0.931
##   chlorides (mean (SD))              0.08 (0.04)    0.09 (0.06)  <0.001
##   free_sulfur_dioxide (mean (SD))   15.27 (10.04)  16.57 (10.89)  0.014
##   total_sulfur_dioxide (mean (SD)) 39.35 (27.25)  54.65 (36.72) <0.001
##   density (mean (SD))                1.00 (0.00)    1.00 (0.00)  <0.001
##   p_h (mean (SD))                    3.31 (0.15)    3.31 (0.15)   0.896
##   sulphates (mean (SD))              0.69 (0.16)    0.62 (0.18)  <0.001
##   alcohol (mean (SD))               10.86 (1.11)    9.93 (0.76)  <0.001
```

Model Building

We randomly selected 70% of the observations as the training data and the rest as the test data. All the 11 predictors were included into analysis. We performed linear methods, non-linear methods, the tree method and SVM to predict the classification of wine quality. For linear methods, we trained (penalized) logistic regression model and linear discriminant analysis (LDA). The assumptions for logistic regression includes observations being independent of each other and the linearity of independent variables and log odds. LDA and QDA assumes normally distributed features, that is, predictor variables are normally distributed for both "good" and "poor" quality of wine. For nonlinear models, we performed generalized additive model (GAM), multivariate adaptive regression splines (MARS), KNN model and quadratic discriminant analysis (QDA). For tree models, we conducted classification tree and random forest model. SVM with linear and radial kernels were also performed. We calculated the ROC and accuracy for model selection, and also investigated the variable importance. 10-fold cross-validation (CV) were used for all model buildings.

```
### Data Partition
set.seed(1)
indexTrain <- createDataPartition(y = wine$qual, p = 0.7, list = FALSE)
trainData <- wine[indexTrain, ]
testData <- wine[-indexTrain, ]
```

Linear models The multiple logistic regression showed that among the 11 predictors, volatile acidity, citric acid, free sulfur dioxide, total sulfur dioxide, sulphates and alcohol were significantly associated with wine quality (P-values < 0.05), explaining 25.1% of the total variance in wine quality. When applying this model to the test data, the accuracy is 0.75 (95%CI: 0.71-0.79) and the ROC is 0.818, which suggests relatively good fit for the data. When performing the penalized logistic regression, we found that when maximizing the ROC, the best tuning parameter was alpha=1 and lambda=0.00086, the accuracy was 0.75 (95%CI: 0.71-0.79) and the ROC was also 0.818. Since lambda was close to zero and the ROC was the same as the full logistic regression model, the penalization was relatively small, which suggested that the full logistic regression model was simple enough for classification.

However, since logistic regression requires there to be little or no multicollinearity among the independent variables, the model may be disturbed by collinearity between the 11 predictors, if there was any. As for LDA, when applying the model to the test data, the ROC was 0.819 and the accuracy was 0.762 (95%CI: 0.72-0.80). The most important variables in predicting wine quality were alcohol, volatile acidity and sulphates. Compared to the logistic regression models, LDA is more helpful when the sample size is small or when the classes are well separated, under the condition that normal assumptions are met.

```
### Logistic regression
# Using caret
ctrl <- trainControl(method = "cv", number = 10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)
set.seed(1)
model.glm <- train(x = trainData %>% dplyr::select(-qual),
                   y = trainData$qual,
                   method = "glm",
                   metric = "ROC",
                   trControl = ctrl)

# Checking the significance of predictors
summary(model.glm)
```

```
##
```

```
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2944  -0.8157  -0.3045   0.8380   3.3331
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -12.336077  94.275480  -0.131  0.89589
## fixed_acidity         -0.128817   0.115195  -1.118  0.26346
## volatile_acidity       3.766894   0.605226   6.224 4.85e-10 ***
## citric_acid            1.971704   0.693754   2.842  0.00448 **
## residual_sugar        -0.018813   0.067682  -0.278  0.78104
## chlorides              2.207284   1.825163   1.209  0.22652
## free_sulfur_dioxide   -0.022291   0.010162  -2.194  0.02826 *
## total_sulfur_dioxide   0.017924   0.003574   5.015 5.30e-07 ***
## density               19.301566  96.123157   0.201  0.84085
## p_h                    0.573935   0.842937   0.681  0.49595
## sulphates             -2.260253   0.513122  -4.405 1.06e-05 ***
## alcohol               -0.921104   0.126440  -7.285 3.22e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1547.2  on 1119  degrees of freedom
## Residual deviance: 1158.3  on 1108  degrees of freedom
## AIC: 1182.3
##
## Number of Fisher Scoring iterations: 4
```

```
# Building confusion matrix
test.pred.prob <- predict(model.glm, newdata = testData,
                          type = "prob")
test.pred <- rep("good", length(test.pred.prob$good))
test.pred[test.pred.prob$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred),
                reference = testData$qual,
                positive = "good")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good poor
##       good  194   58
##       poor   62  165
##
##                Accuracy : 0.7495
##                  95% CI : (0.7082, 0.7877)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
```

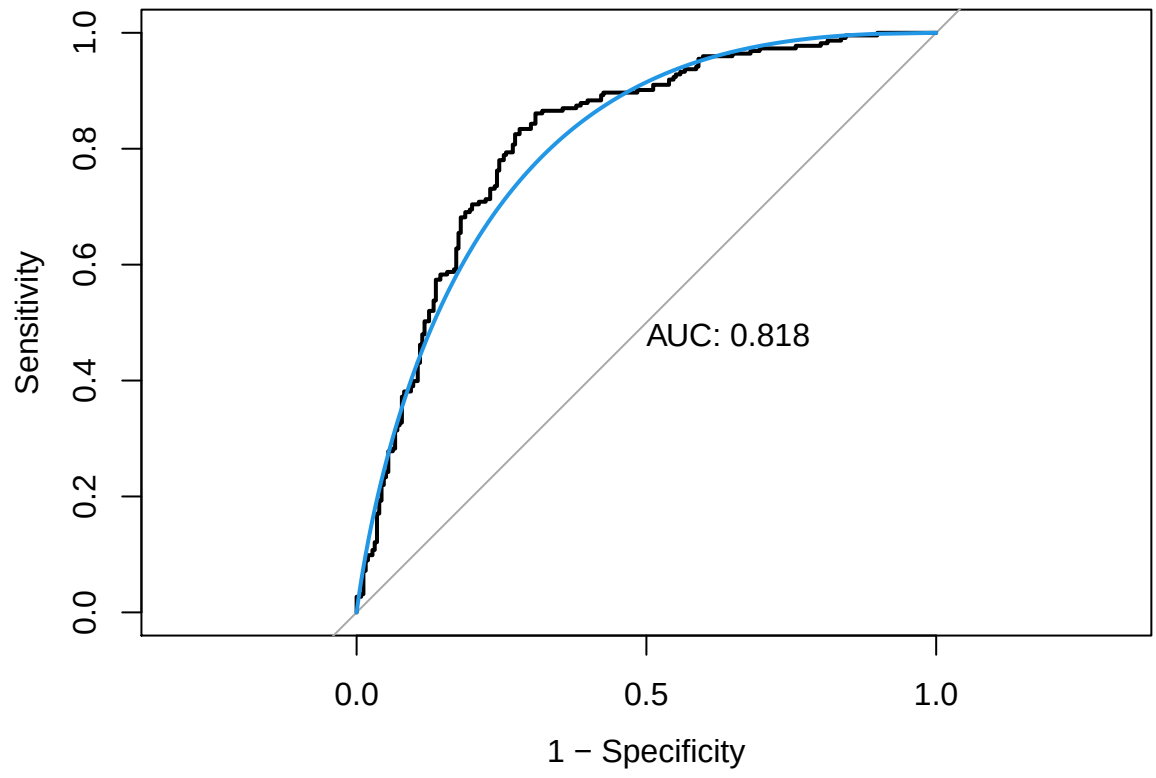
```
##                  Kappa : 0.4971
##
##  Mcnemar's Test P-Value : 0.7842
##
##             Sensitivity : 0.7578
##             Specificity : 0.7399
##          Pos Pred Value : 0.7698
##          Neg Pred Value : 0.7269
##              Prevalence : 0.5344
##          Detection Rate : 0.4050
##    Detection Prevalence : 0.5261
##       Balanced Accuracy : 0.7489
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.glm <- roc(testData$qual, test.pred.prob$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```
plot(roc.glm, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.glm), col = 4, add = TRUE)
```

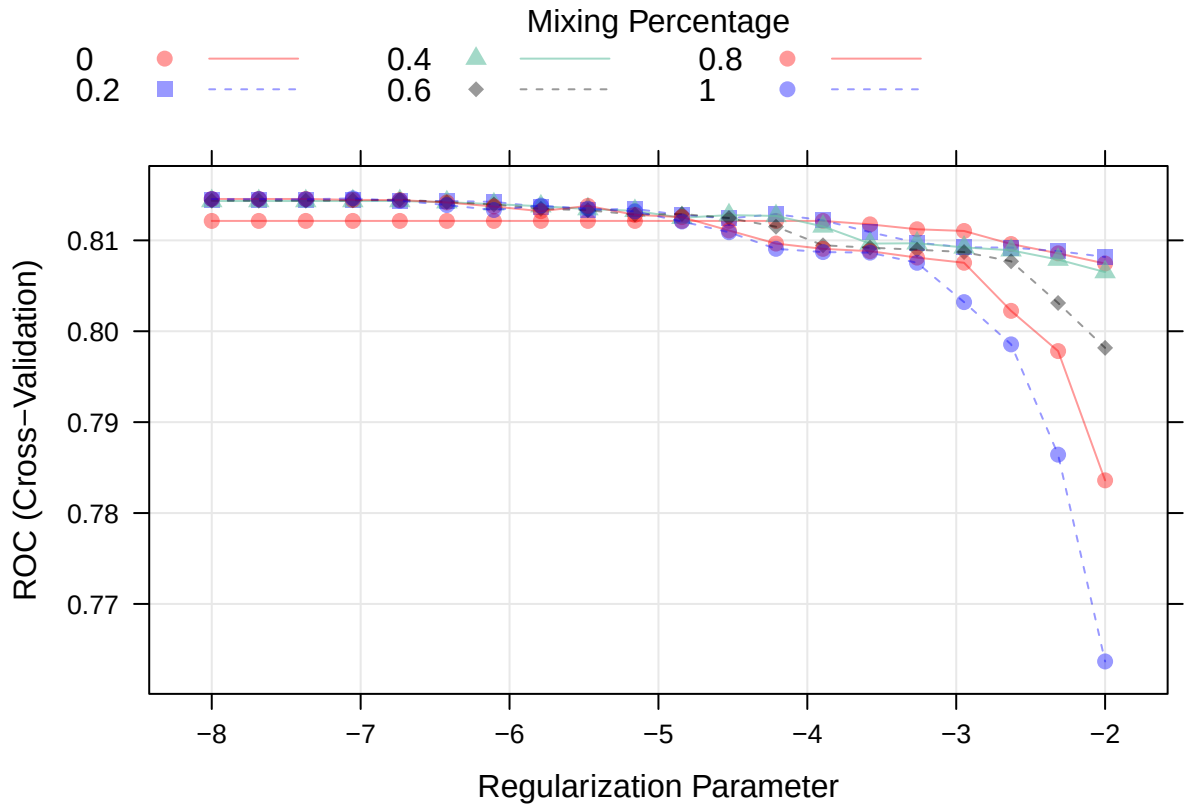


```r
## Test error and train error
error.test.glm <- mean(testData$qual != test.pred)

train.pred.prob.glm <- predict(model.glm, newdata = trainData,
                               type = "prob")
train.pred.glm <- rep("good", length(train.pred.prob.glm$good))
train.pred.glm[train.pred.prob.glm$good < 0.5] <- "poor"
error.trian.glm <- mean(trainData$qual != train.pred.glm)
```

```r
### Penalized logistic regression
glmnGrid <- expand.grid(.alpha = seq(0, 1, length = 6),
                        .lambda = exp(seq(-8, -2, length = 20)))
set.seed(1)
model.glmn <- train(x = trainData %>% dplyr::select(-qual),
                    y = trainData$qual,
                    method = "glmnet",
                    tuneGrid = glmnGrid,
                    metric = "ROC",
                    trControl = ctrl)

plot(model.glmn, xTrans = function(x) log(x))
```




```r
# select the best tune
model.glmn$bestTune
```

```
##     alpha       lambda
## 104     1 0.0008651293
```

```
# Building confusion matrix
test.pred.prob2 <- predict(model.glmn, newdata = testData,
                           type = "prob")
test.pred2 <- rep("good", length(test.pred.prob2$good))
test.pred2[test.pred.prob2$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred2),
                reference = testData$qual,
                positive = "good")
```

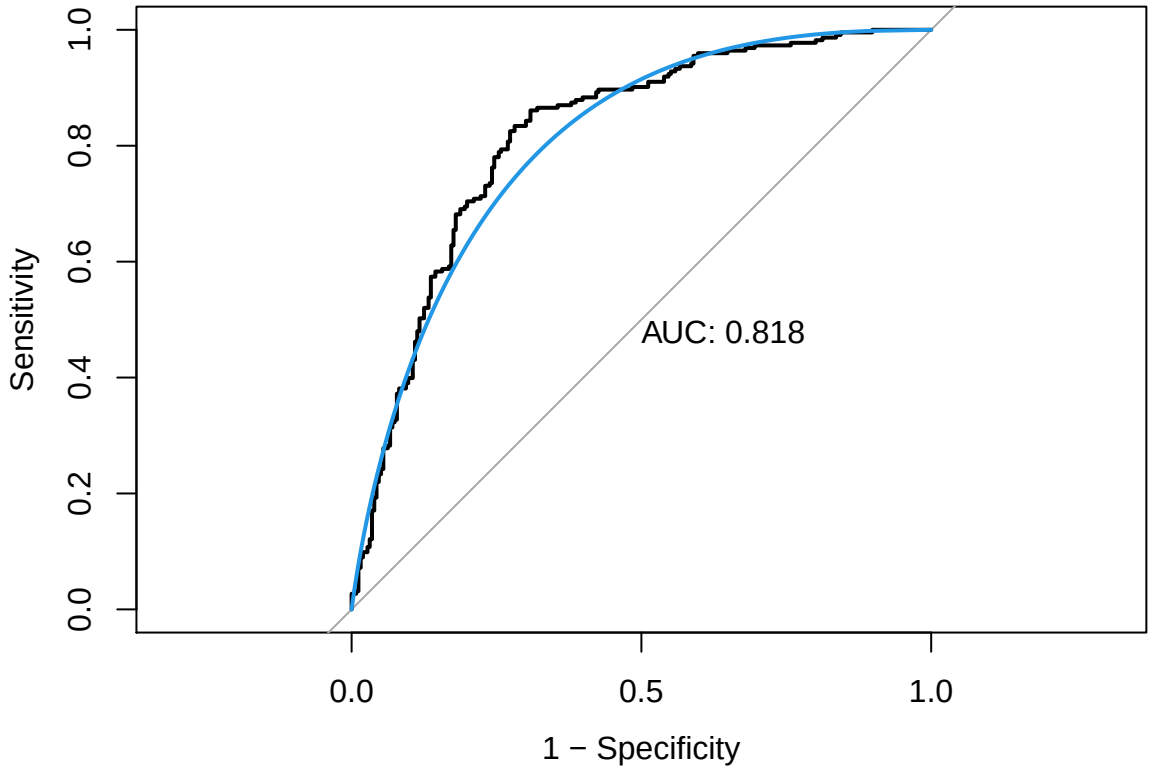
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good poor
##       good  196   58
##       poor   60  165
##
##                Accuracy : 0.7537
##                  95% CI : (0.7125, 0.7916)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5052
##
##  Mcnemar's Test P-Value : 0.9267
##
##             Sensitivity : 0.7656
##             Specificity : 0.7399
##          Pos Pred Value : 0.7717
##          Neg Pred Value : 0.7333
##              Prevalence : 0.5344
##          Detection Rate : 0.4092
##    Detection Prevalence : 0.5303
##       Balanced Accuracy : 0.7528
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.glmn <- roc(testData$qual, test.pred.prob2$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```
plot(roc.glm, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.glm), col = 4, add = TRUE)
```




```
## Test error and train error
error.test.glmn <- mean(testData$qual != test.pred2)

train.pred.prob.glmn <- predict(model.glmn, newdata = trainData,
                            type = "prob")
train.pred.glmn <- rep("good", length(train.pred.prob.glmn$good))
train.pred.glmn[train.pred.prob.glmn$good < 0.5] <- "poor"
error.trian.glmn <- mean(trainData$qual != train.pred.glmn)
```

```
### LDA
set.seed(1)
model.lda <- train(x = trainData %>% dplyr::select(-qual),
                   y = trainData$qual,
                   method = "lda",
                   metric = "ROC",
                   trControl = ctrl)

# Building confusion matrix
test.pred.prob5 <- predict(model.lda, newdata = testData,
                            type = "prob")
test.pred5 <- rep("good", length(test.pred.prob5$good))
test.pred5[test.pred.prob5$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred5),
                reference = testData$qual,
                positive = "good")
```

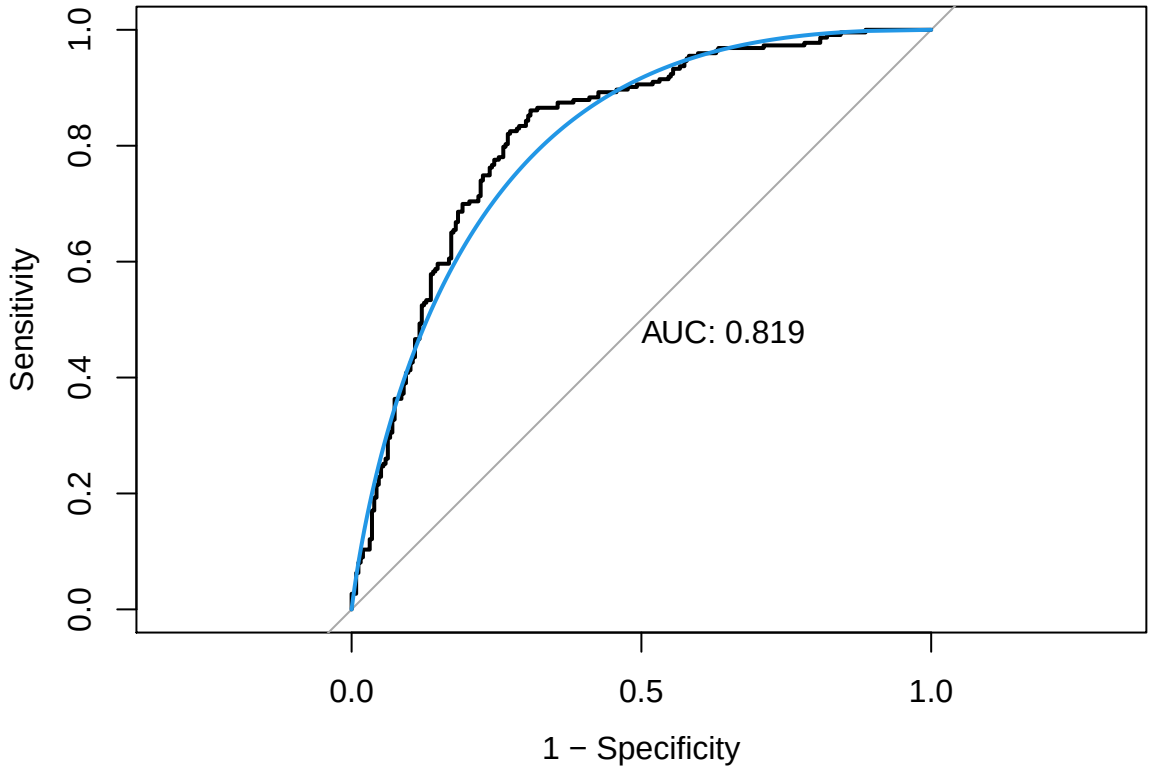
```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction good poor
##       good  195   53
##       poor   61  170
##
##                Accuracy : 0.762
##                  95% CI : (0.7213, 0.7995)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5228
##
##  Mcnemar's Test P-Value : 0.5121
##
##             Sensitivity : 0.7617
##             Specificity : 0.7623
##          Pos Pred Value : 0.7863
##          Neg Pred Value : 0.7359
##              Prevalence : 0.5344
##          Detection Rate : 0.4071
##    Detection Prevalence : 0.5177
##       Balanced Accuracy : 0.7620
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.lda <- roc(testData$qual, test.pred.prob5$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```
plot(roc.lda, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.lda), col = 4, add = TRUE)
```




```
## Calculate the test error
lda.pred = predict(model.lda, newdata = testData, type = "raw")

error.test.lda <- mean(testData$qual != lda.pred)

## train error
lda.pred.train = predict(model.lda, newdata = trainData, type = "raw")

error.train.lda <- mean(trainData$qual != lda.pred.train)

# variable importance
varImp(model.lda)
```

```
## ROC curve variable importance
##
##                       Importance
## alcohol                  100.000
## volatile_acidity          70.868
## sulphates                 70.457
## total_sulfur_dioxide      53.496
## chlorides                 41.809
## density                   38.425
## citric_acid               29.818
## free_sulfur_dioxide       18.524
## fixed_acidity             16.509
```

```
## residual_sugar            1.543
## p_h                       0.000
```

Nonlinear models In the GAM model, only the degree of freedom for volatile acidity was equal to 1, suggesting linear association, while smoothing spline was applied for all other 10 variables. The results showed that alcohol, citric acid, residual sugar, sulphates, fixed acidity, volatile acidity, chlorides and total sulfur dioxide were significant predictors (P-values < 0.05). In total, these variables explained 39.1% of the total variance in wine quality. The confusion matrix using the test data showed that the accuracy for GAM was 0.76 (95%CI: 0.72-0.80) and the ROC was 0.829. The MARS model showed that when maximizing the ROC, we included 5 terms out of 11 predictors, with nprune equal to 5 and degree of 2. In total, these predictors and hinge functions explained 32.2% of the total variance. According to the MARS output, the 3 most important predictors were total sulfur dioxide, alcohol and sulphates. When applying the MARS model to the test data, the accuracy is 0.75 (95%CI: 0.72, 0.80) and the ROC is 0.823. We also performed the KNN model for classification. When k was equal to 22, the ROC was maximized. The accuracy for KNN model was 0.63 (95%CI: 0.59-0.68) and the ROC was 0.672. The QDA model showed that ROC was 0.784 and the accuracy was 0.71 (95%CI: 0.66-0.75). The most important variables in predicting wine quality are alcohol, volatile acidity and sulphates.

The advantage of GAM and MARS is that both two models are nonparametric models and able to deal with highly complex nonlinear relationship. Specifically, MARS model can include potential interaction effects into the model. However, because of the model complexity, time-consuming computation and the high propensity of overfitting are the limitations for the two models. As for the KNN model, when k was large, the prediction may not be accurate.

```
### GAM
set.seed(1)
model.gam <- train(x = trainData %>% dplyr::select(-qual),
                   y = trainData$qual,
                    method = "gam",
                    metric = "ROC",
                    trControl = ctrl)


model.gam$finalModel
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## .outcome ~ s(free_sulfur_dioxide) + s(alcohol) + s(citric_acid) +
##     s(residual_sugar) + s(p_h) + s(sulphates) + s(fixed_acidity) +
##     s(volatile_acidity) + s(chlorides) + s(total_sulfur_dioxide) +
##     s(density)
##
## Estimated degrees of freedom:
## 4.232 6.021 1.804 7.709 0.478 3.774 3.492
## 1.000 5.916 6.421 7.333  total = 49.18
##
## UBRE score: -0.02749885
```

```
summary(model.gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## .outcome ~ s(free_sulfur_dioxide) + s(alcohol) + s(citric_acid) +
##     s(residual_sugar) + s(p_h) + s(sulphates) + s(fixed_acidity) +
##     s(volatile_acidity) + s(chlorides) + s(total_sulfur_dioxide) +
##     s(density)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.26628    0.09157  -2.908  0.00364 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                            edf Ref.df Chi.sq  p-value
## s(free_sulfur_dioxide)  4.2319      9  8.024 0.077961 .
## s(alcohol)              6.0214      9 47.408  < 2e-16 ***
## s(citric_acid)          1.8039      9 10.876 0.000814 ***
## s(residual_sugar)       7.7091      9 17.691 0.015860 *
## s(p_h)                  0.4784      9  0.783 0.164933
## s(sulphates)            3.7737      9 56.546  < 2e-16 ***
## s(fixed_acidity)        3.4917      9 13.818 0.000877 ***
## s(volatile_acidity)     1.0000      9 26.986  < 2e-16 ***
## s(chlorides)            5.9163      9 14.507 0.013552 *
## s(total_sulfur_dioxide) 6.4206      9 29.741 1.22e-05 ***
## s(density)              7.3330      9 12.897 0.064167 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.391   Deviance explained =   36%
## UBRE = -0.027499  Scale est. = 1         n = 1120
```

```r
# Building confusion matrix
test.pred.prob3 <- predict(model.gam, newdata = testData,
                           type = "prob")
test.pred3 <- rep("good", length(test.pred.prob3$good))
test.pred3[test.pred.prob3$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred3),
                reference = testData$qual,
                positive = "good")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good poor
##       good  200   59
##       poor   56  164
##
##                Accuracy : 0.7599
##                  95% CI : (0.7191, 0.7975)
```

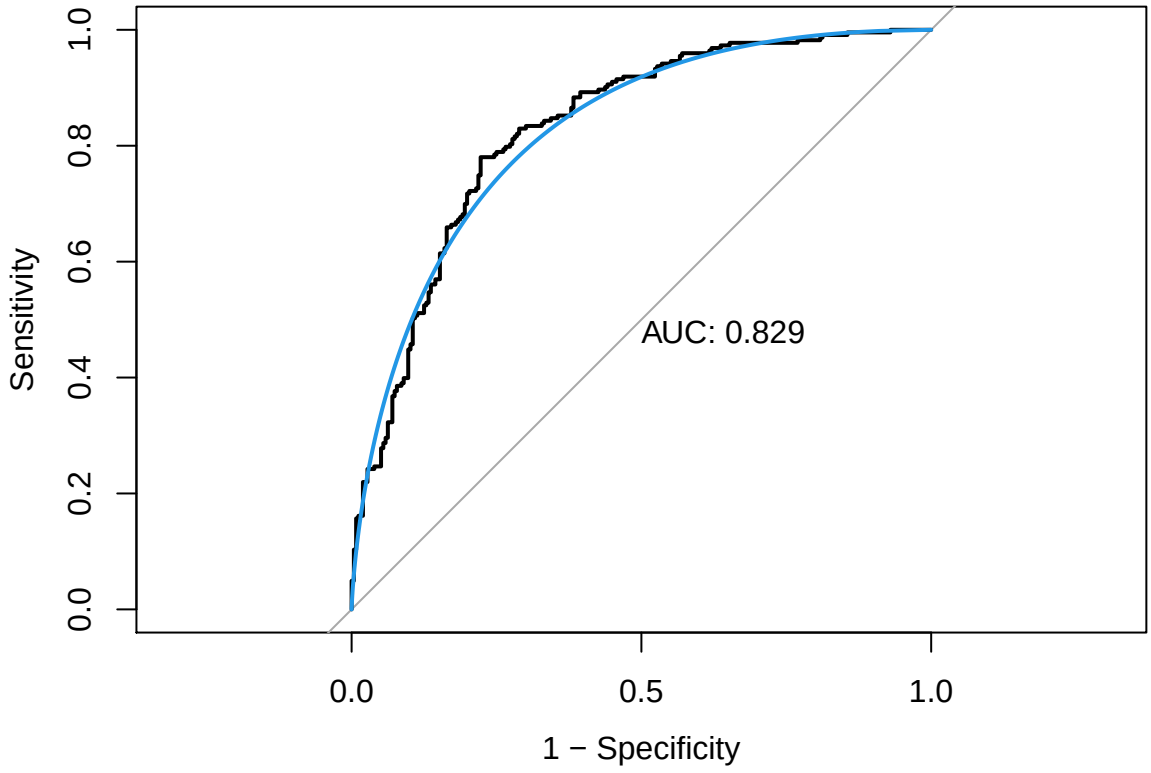
```
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5171
##
##  Mcnemar's Test P-Value : 0.8521
##
##             Sensitivity : 0.7812
##             Specificity : 0.7354
##          Pos Pred Value : 0.7722
##          Neg Pred Value : 0.7455
##              Prevalence : 0.5344
##          Detection Rate : 0.4175
##    Detection Prevalence : 0.5407
##       Balanced Accuracy : 0.7583
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.gam <- roc(testData$qual, test.pred.prob3$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```
plot(roc.gam, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.gam), col = 4, add = TRUE)
```




```
## Test error and train error
error.test.gam <- mean(testData$qual != test.pred3)

train.pred.prob.gam <- predict(model.gam, newdata = trainData,
                               type = "prob")
train.pred.gam <- rep("good", length(train.pred.prob.gam$good))
train.pred.gam[train.pred.prob.gam$good < 0.5] <- "poor"
error.trian.gam <- mean(trainData$qual != train.pred.gam)
```

```
### MARS
set.seed(1)
model.mars <- train(x = trainData %>% dplyr::select(-qual),
                    y = trainData$qual,
                    method = "earth",
                    tuneGrid = expand.grid(degree = 1:3,
                                           nprune = 2:23),
                    metric = "ROC",
                    trControl = ctrl)
```

```
## Loading required package: earth

## Loading required package: Formula

## Loading required package: plotmo
```

```
## Loading required package: plotrix
```

```
## Loading required package: TeachingDemos
```

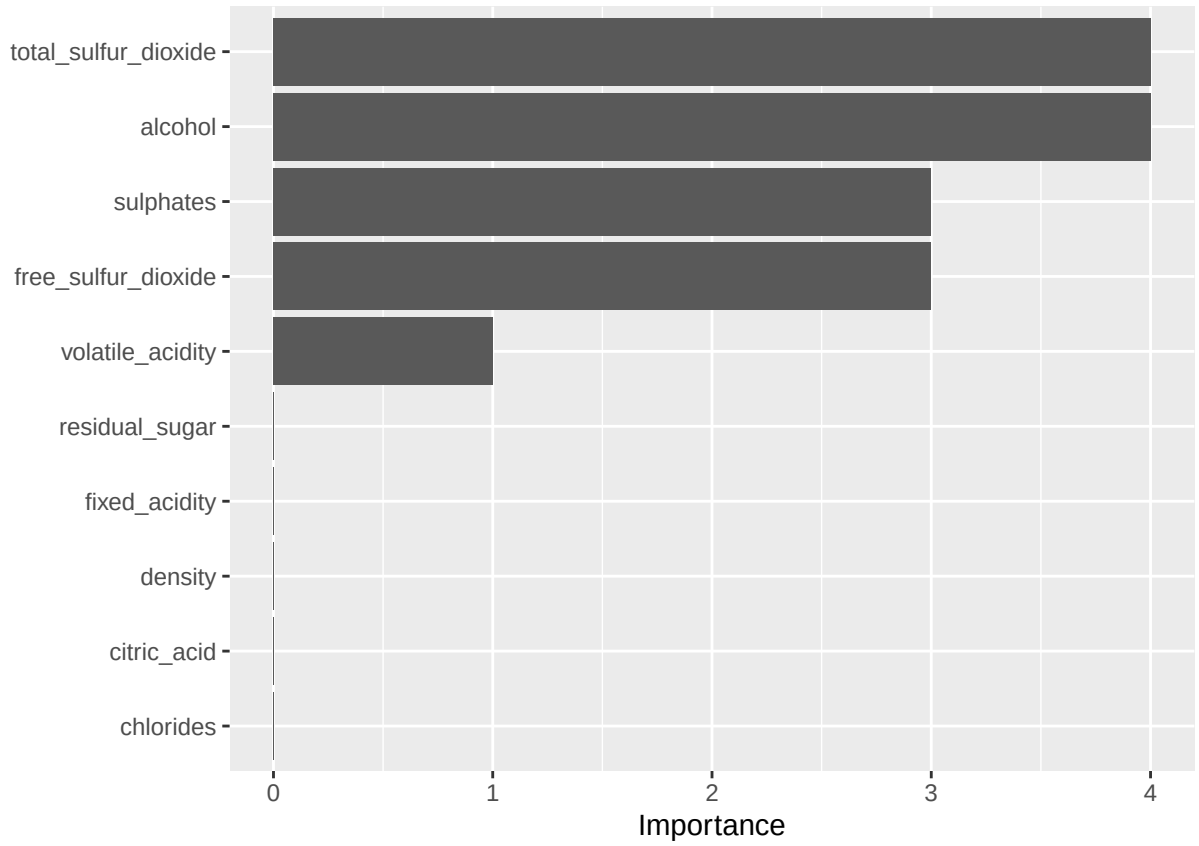
```
model.mars$bestTune
```

```
##    nprune degree
## 26      5      2
```

```
model.mars$finalModel
```

```
## GLM (family binomial, link logit):
##  nulldev   df       dev   df   devratio     AIC iters converged
##  1547.21 1119   1118.07 1115      0.277    1128     5         1
##
## Earth selected 5 of 22 terms, and 5 of 11 predictors (nprune=5)
## Termination condition: Reached nk 23
## Importance: total_sulfur_dioxide, alcohol, free_sulfur_dioxide, sulphates, ...
## Number of terms at each degree of interaction: 1 2 2
## Earth GCV 0.1721529    RSS 189.0425    GRSq 0.3092675    RSq 0.3215578
```

```
vip(model.mars$finalModel)
```



```r
# Building confusion matrix
test.pred.prob4 <- predict(model.mars, newdata = testData,
                           type = "prob")
test.pred4 <- rep("good", length(test.pred.prob4$good))
test.pred4[test.pred.prob4$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred4),
                reference = testData$qual,
                positive = "good")
```

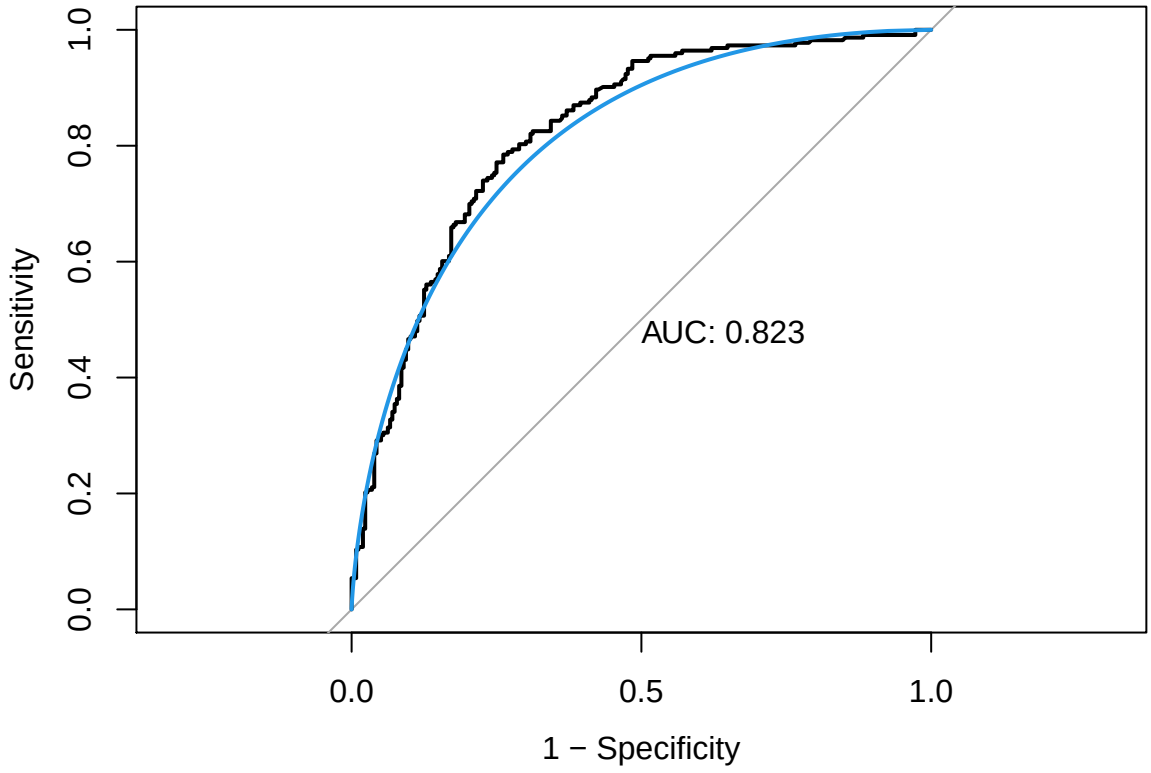
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good poor
##       good  201   63
##       poor   55  160
##
##                Accuracy : 0.7537
##                  95% CI : (0.7125, 0.7916)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5038
##
##  Mcnemar's Test P-Value : 0.5193
##
##             Sensitivity : 0.7852
##             Specificity : 0.7175
##          Pos Pred Value : 0.7614
##          Neg Pred Value : 0.7442
##              Prevalence : 0.5344
##          Detection Rate : 0.4196
##    Detection Prevalence : 0.5511
##       Balanced Accuracy : 0.7513
##
##        'Positive' Class : good
##
```

```r
# Plot the test ROC
roc.mars <- roc(testData$qual, test.pred.prob4$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```r
plot(roc.mars, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.mars), col = 4, add = TRUE)
```



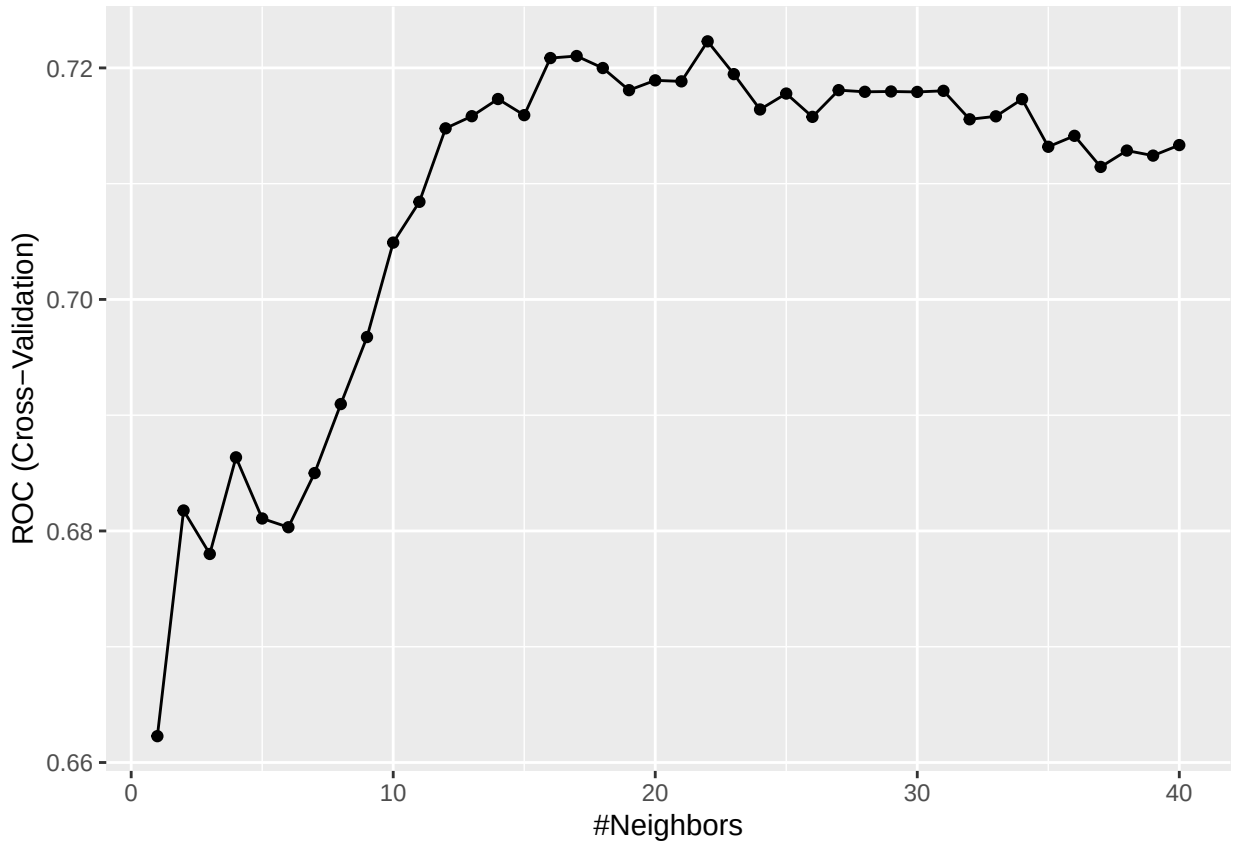

```
## Test error and train error
error.test.mars <- mean(testData$qual != test.pred4)

train.pred.prob.mars <- predict(model.mars, newdata = trainData,
                                type = "prob")
train.pred.mars <- rep("good", length(train.pred.prob.mars$good))
train.pred.mars[train.pred.prob.mars$good < 0.5] <- "poor"
error.trian.mars <- mean(trainData$qual != train.pred.mars)
```

```
### KNN
kGrid <- expand.grid(k = seq(from = 1, to = 40, by = 1))

set.seed(1)
fit.knn <- train(qual ~ .,
                 data = trainData,
                 method = "knn",
                 metric = "ROC",
                 trControl = ctrl,
                 tuneGrid = kGrid)

ggplot(fit.knn)
```




```
## The best TuneGrid
fit.knn$bestTune
```

```
##     k
## 22 22
```

```
# Building confusion matrix
test.pred.prob7 <- predict(fit.knn, newdata = testData,
                           type = "prob")
test.pred7 <- rep("good", length(test.pred.prob7$good))
test.pred7[test.pred.prob7$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred7),
                reference = testData$qual,
                positive = "good")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good poor
##       good  175   95
##       poor   81  128
##
##                Accuracy : 0.6326
##                  95% CI : (0.5876, 0.6759)
```

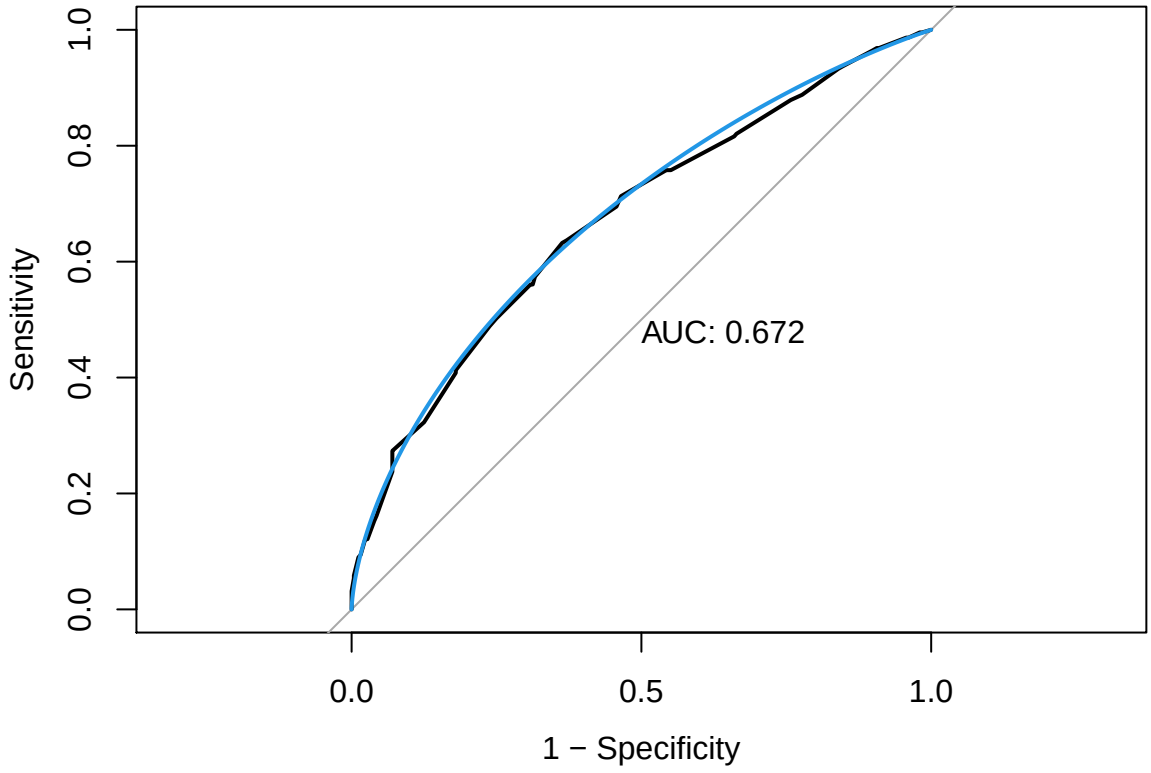
```
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : 8.897e-06
##
##                   Kappa : 0.2586
##
##  Mcnemar's Test P-Value : 0.3271
##
##             Sensitivity : 0.6836
##             Specificity : 0.5740
##          Pos Pred Value : 0.6481
##          Neg Pred Value : 0.6124
##              Prevalence : 0.5344
##          Detection Rate : 0.3653
##    Detection Prevalence : 0.5637
##       Balanced Accuracy : 0.6288
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.knn <- roc(testData$qual, test.pred.prob7$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```
plot(roc.knn, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.knn), col = 4, add = TRUE)
```




```
## Test error and train error
error.test.knn <- mean(testData$qual != test.pred7)

train.pred.prob.knn <- predict(fit.knn, newdata = trainData,
                               type = "prob")
train.pred.knn <- rep("good", length(train.pred.prob.knn$good))
train.pred.knn[train.pred.prob.knn$good < 0.5] <- "poor"
error.trian.knn <- mean(trainData$qual != train.pred.knn)
```

```
### QDA
set.seed(1)
model.qda <- train(x = trainData %>% dplyr::select(-qual),
                   y = trainData$qual,
                    method = "qda",
                    metric = "ROC",
                    trControl = ctrl)

# Building confusion matrix
test.pred.prob6 <- predict(model.qda, newdata = testData,
                           type = "prob")
test.pred6 <- rep("good", length(test.pred.prob6$good))
test.pred6[test.pred.prob6$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred6),
                reference = testData$qual,
                positive = "good")
```

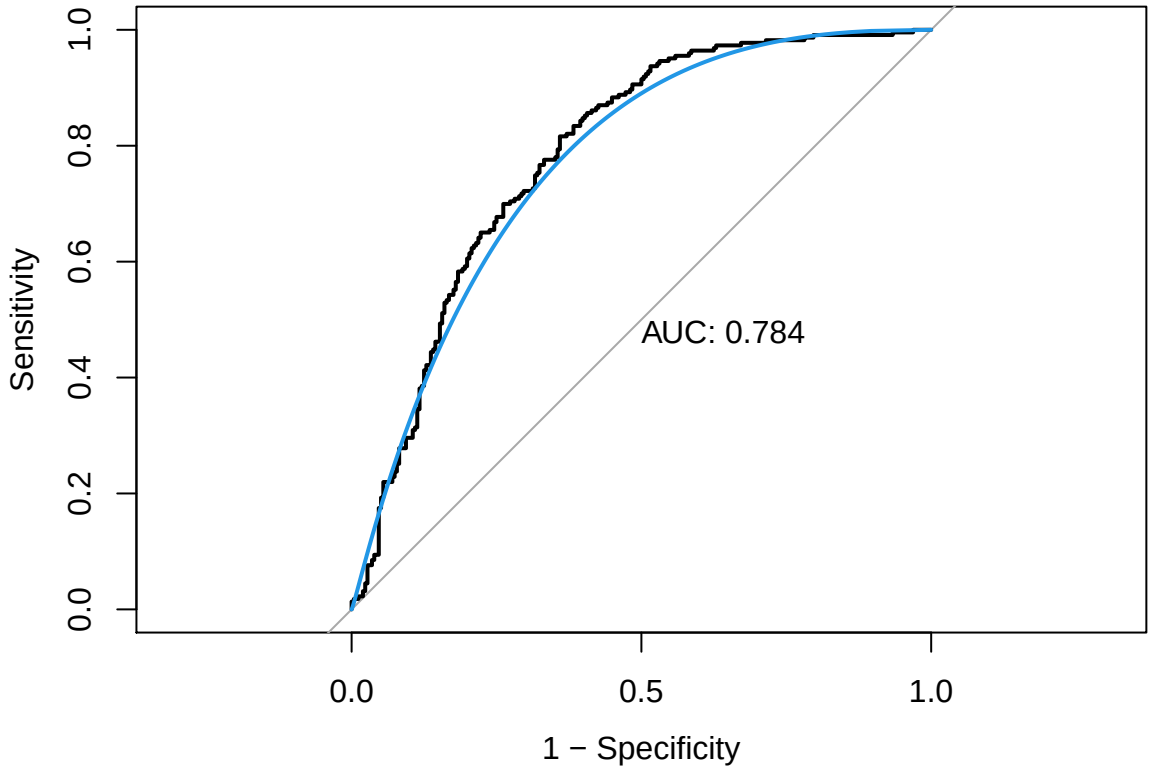
```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction good poor
##       good  204   88
##       poor   52  135
##
##                Accuracy : 0.7077
##                  95% CI : (0.6648, 0.7481)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : 6.88e-15
##
##                   Kappa : 0.4065
##
##  Mcnemar's Test P-Value : 0.003096
##
##             Sensitivity : 0.7969
##             Specificity : 0.6054
##          Pos Pred Value : 0.6986
##          Neg Pred Value : 0.7219
##              Prevalence : 0.5344
##          Detection Rate : 0.4259
##    Detection Prevalence : 0.6096
##       Balanced Accuracy : 0.7011
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.qda <- roc(testData$qual, test.pred.prob6$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```
plot(roc.qda, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.qda), col = 4, add = TRUE)
```




```
## Calculate the test error
qda.pred = predict(model.qda, newdata = testData, type = "raw")

error.test.qda <- mean(testData$qual != qda.pred)

## train error
qda.pred.train = predict(model.qda, newdata = trainData, type = "raw")

error.train.qda <- mean(trainData$qual != qda.pred.train)

# variable importance
varImp(model.qda)
```

```
## ROC curve variable importance
##
##                       Importance
## alcohol                  100.000
## volatile_acidity          70.868
## sulphates                 70.457
## total_sulfur_dioxide      53.496
## chlorides                 41.809
## density                   38.425
## citric_acid               29.818
## free_sulfur_dioxide       18.524
## fixed_acidity             16.509
```

```
## residual_sugar            1.543
## p_h                       0.000
```

Tree Methods

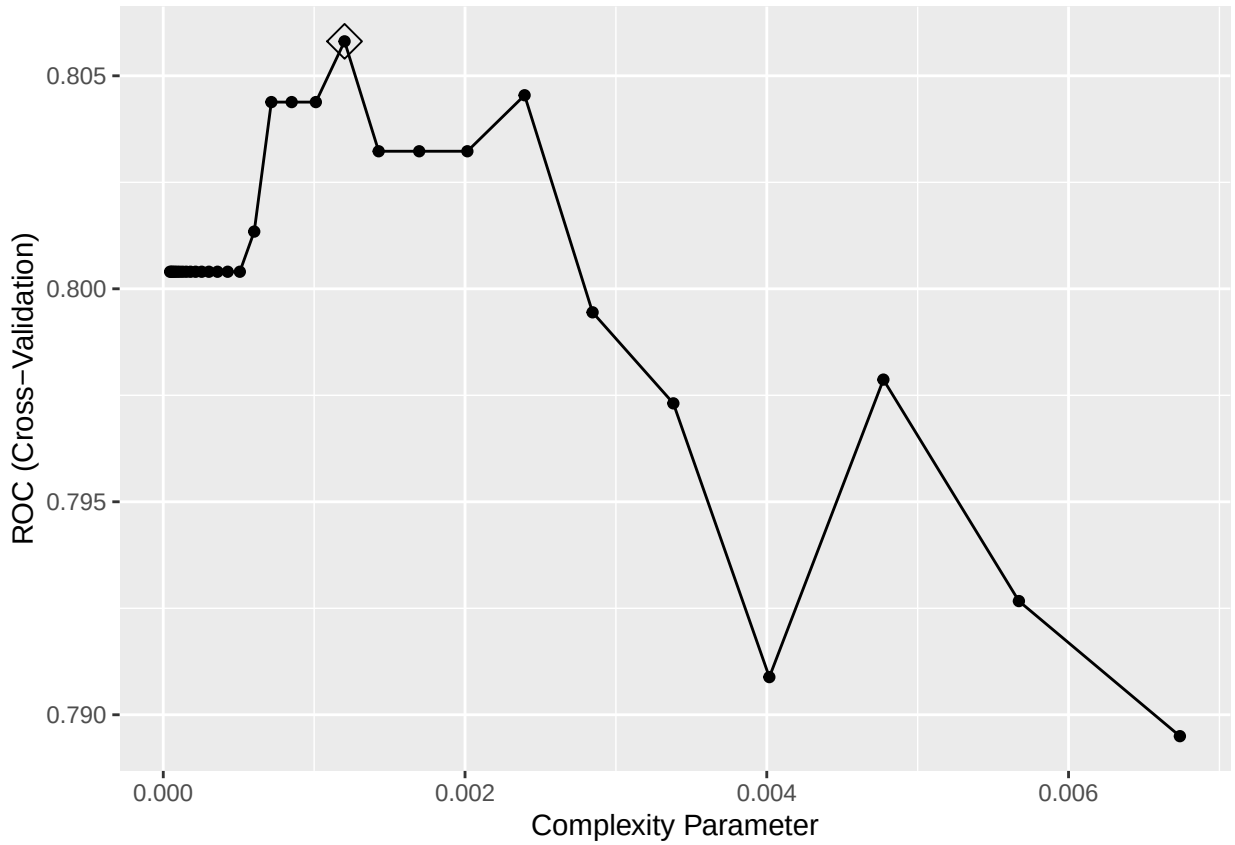
Based on the classification tree, the final tree size is 41 when maximizing the AUC. The test error rate is 0.24 and ROC is 0.809. The accuracy of this classification tree is 0.76 (95%CI: 0.72-0.80). We also conducted the random forest method to investigate the variable importance. As a result, alcohol is the most important variable, and followed by sulphates, volatile acidity, total sulfur dioxide, density, chlorides, fixed acidity, citric acid, free sulfur dioxide, and residual sugar. pH is the least important variable. For the random forest model, the test error rate is 0.163, the accuracy is 0.84 (95%CI: 0.80-0.87), and the ROC is 0.900.

One potential limitation for the tree methods is that they are sensitive to the change in data, that is, a small change in data may cause a large change of the classification tree.

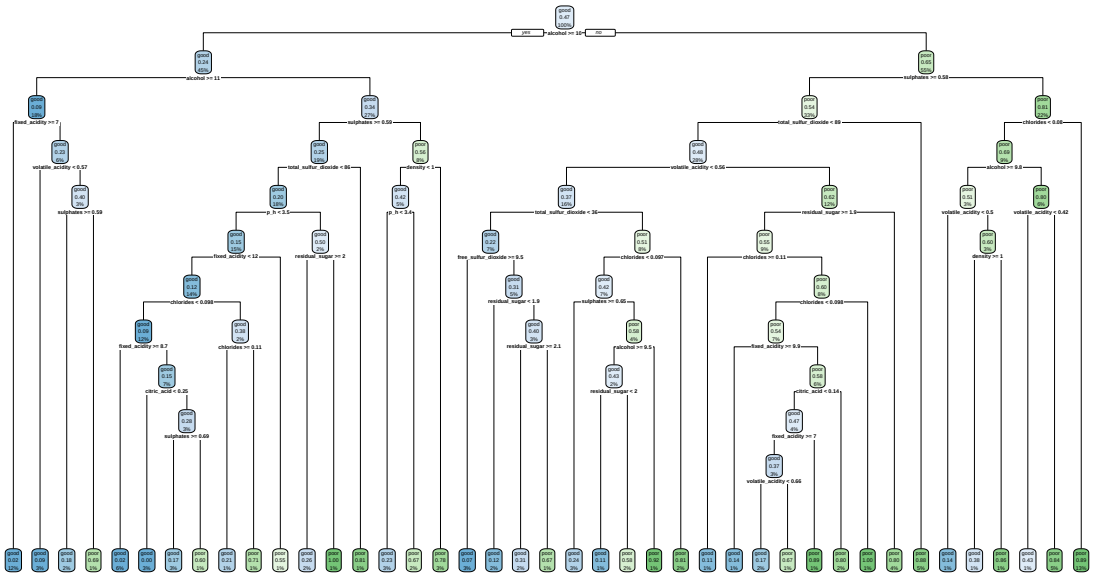
```
# Classification Tree
# Using caret
ctrl <- trainControl(method = "cv", number = 10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)

set.seed(1)
rpart_grid = data.frame(cp = exp(seq(-10,-5, len = 30)))
class.tree = train(qual~., trainData,
                   method = "rpart",
                   tuneGrid = rpart_grid,
                   trControl = ctrl,
                   metric = "ROC")

ggplot(class.tree, highlight = TRUE)
```




```
rpart.plot(class.tree$finalModel)
```



```r
## Calculate the test error
rpart.pred = predict(class.tree, newdata = testData, type = "raw")
test.error.classtree = mean(testData$qual != rpart.pred)

## Calculate the train error
rpart.pred_train = predict(class.tree, newdata = trainData, type = "raw")
train.error.classtree = mean(trainData$qual != rpart.pred_train)

# Building confusion matrix
test.pred.prob8 <- predict(class.tree, newdata = testData,
                           type = "prob")
test.pred8 <- rep("good", length(test.pred.prob8$good))
test.pred8[test.pred.prob8$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred8),
                reference = testData$qual,
                positive = "good")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good poor
##       good  194   54
##       poor   62  169
##
```

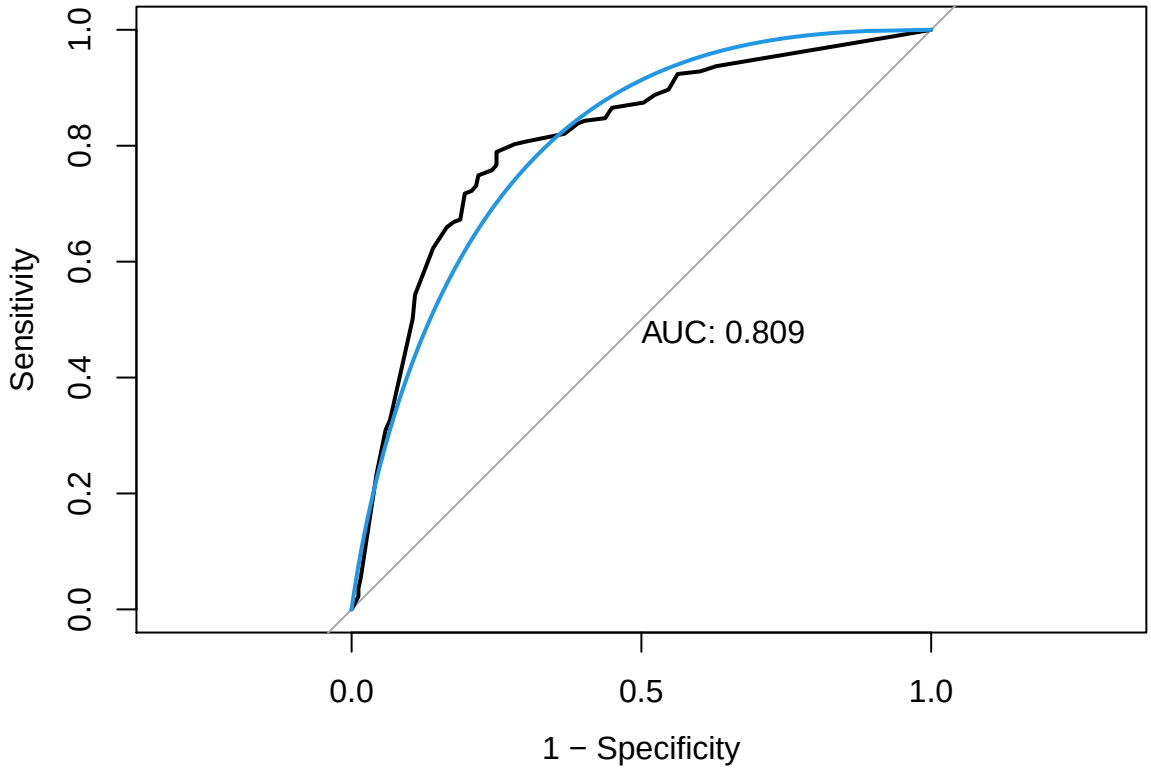
```
##                Accuracy : 0.7578
##                  95% CI : (0.7169, 0.7955)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5145
##
##  Mcnemar's Test P-Value : 0.5157
##
##             Sensitivity : 0.7578
##             Specificity : 0.7578
##          Pos Pred Value : 0.7823
##          Neg Pred Value : 0.7316
##              Prevalence : 0.5344
##          Detection Rate : 0.4050
##    Detection Prevalence : 0.5177
##       Balanced Accuracy : 0.7578
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.ctree <- roc(testData$qual, test.pred.prob8$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

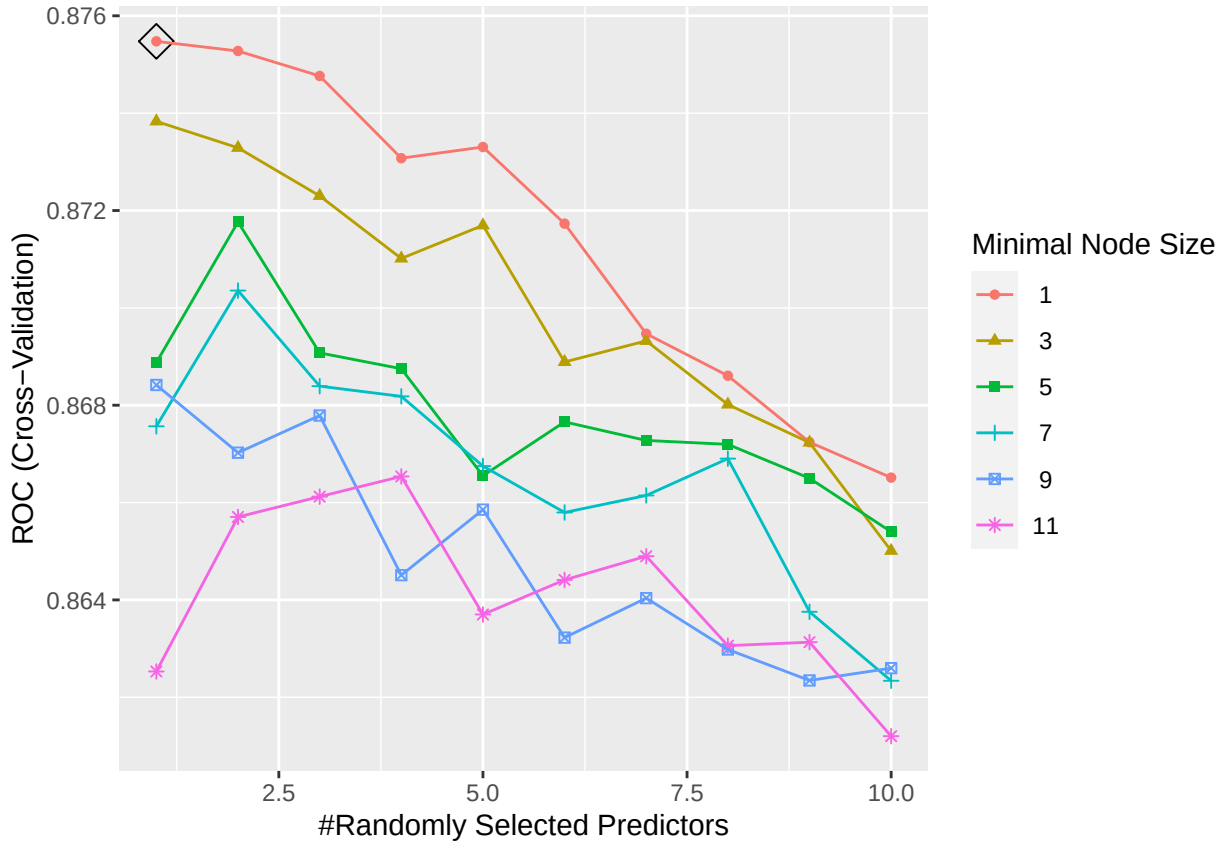
```
plot(roc.ctree, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.ctree), col = 4, add = TRUE)
```




```r
# Random forest and variable importance
ctrl <- trainControl(method = "cv", number = 10,
                     classProbs = TRUE,
                     summaryFunction = twoClassSummary)

rf.grid <- expand.grid(mtry = 1:10,
                       splitrule = "gini",
                       min.node.size = seq(from = 1, to = 12, by = 2))
set.seed(1)
rf.fit <- train(qual ~ . ,
                trainData,
                method = "ranger",
                tuneGrid = rf.grid,
                metric = "ROC",
                trControl = ctrl)

ggplot(rf.fit, highlight = TRUE)
```

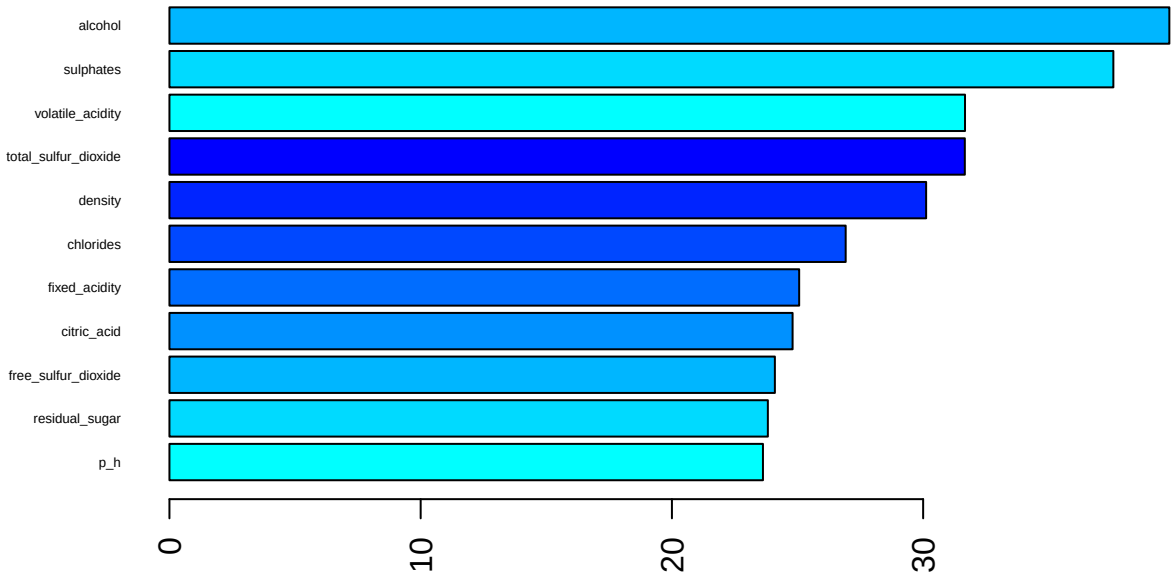



```
rf.fit$bestTune
```

```
##   mtry splitrule min.node.size
## 1    1      gini             1
```

```
# compute importance
set.seed(1)
random_forest <- ranger(qual ~ . ,
                        trainData,
                        mtry = 1,
                        splitrule = "gini",
                        min.node.size = 1,
                        importance = "permutation",
                        scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(random_forest), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.4,
        col = colorRampPalette(colors = c("cyan","blue"))(8))
```




```
# compute the test error rate
rf.pred <- predict(rf.fit, newdata = testData)
test.error.rf = mean(testData$qual != rf.pred)

# compute the train error rate
rf.pred_train <- predict(rf.fit, newdata = trainData)
train.error.rf = mean(trainData$qual != rf.pred_train)

# Building confusion matrix
test.pred.prob8 <- predict(rf.fit, newdata = testData,
                           type = "prob")
test.pred8 <- rep("good", length(test.pred.prob8$good))
test.pred8[test.pred.prob8$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred8),
                reference = testData$qual,
                positive = "good")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good poor
##       good  224   46
##       poor   32  177
##
```

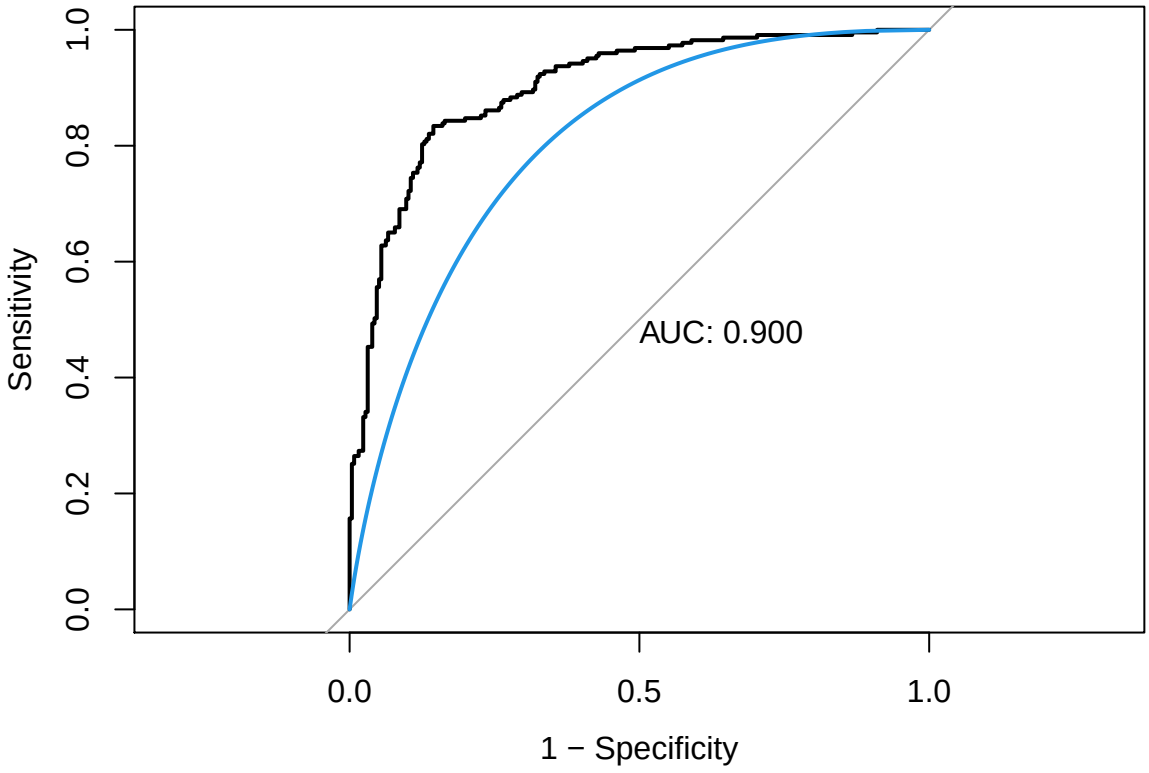
```
##                Accuracy : 0.8372
##                  95% CI : (0.801, 0.8691)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6714
##
##  Mcnemar's Test P-Value : 0.141
##
##             Sensitivity : 0.8750
##             Specificity : 0.7937
##          Pos Pred Value : 0.8296
##          Neg Pred Value : 0.8469
##              Prevalence : 0.5344
##          Detection Rate : 0.4676
##    Detection Prevalence : 0.5637
##       Balanced Accuracy : 0.8344
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.rf <- roc(testData$qual, test.pred.prob8$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```
plot(roc.rf, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.ctree), col = 4, add = TRUE)
```



SVM

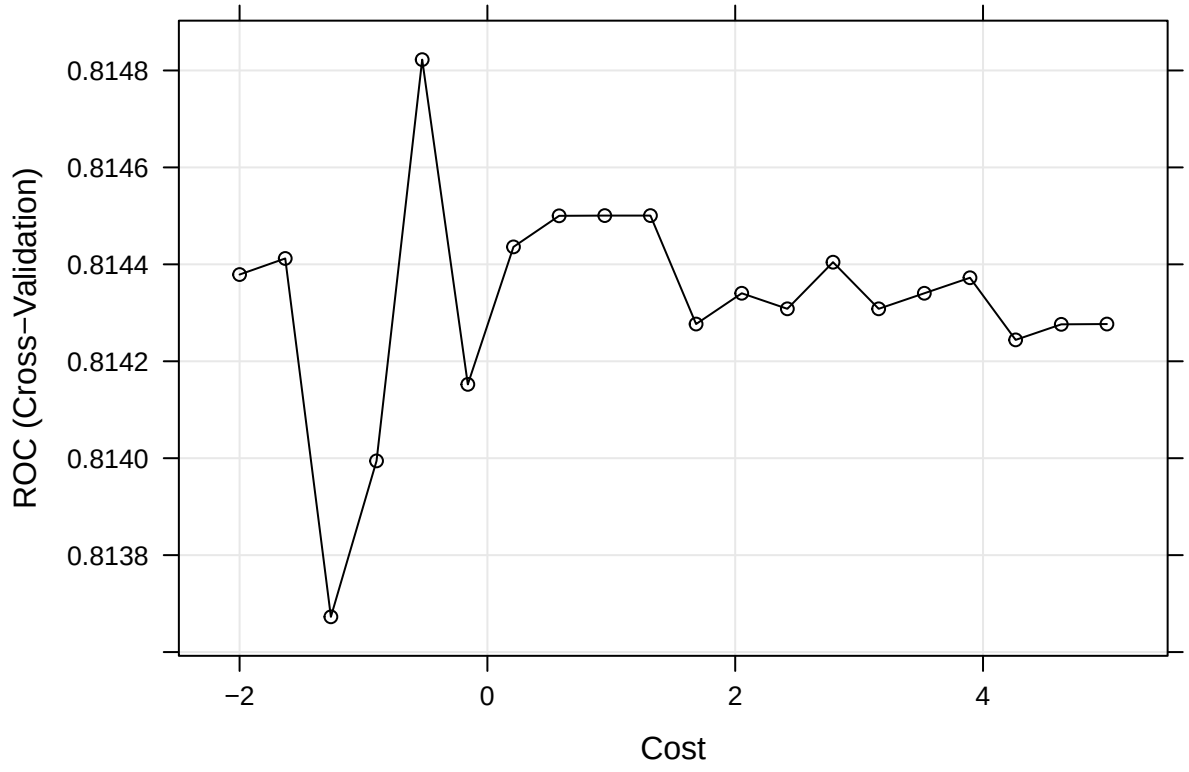
We used SVM with linear kernal and we tuned over cost. We found that the model with maximized ROC had cost = 0.59078. ROC for this model is 0.816, accuracy is 0.75 (test error is 0.25) (95%CI: 0.71-0.79). The most important variable for quality prediction is alcohol; volatile acidity and total sulfur dioxide are also relatively important variables. When performing SVM with radial kernal, we tuned over both cost and sigma, and found that the model with maximized ROC had sigma = 0.0286 and cost = 17.9733. ROC for this model is 0.821, accuracy is 0.75 (test error is 0.25) (95%CI: 0.71-0.79). Same as SVM using linear kernal, the most important variable for quality prediction is alcohol; sulphates and volatile acidity are the second and third most important variables. If the true boundary is non-linear, SVM with radial kernel performs better.

```
## SVM with linear kernel
set.seed(1)

svml.fit <- train(qual ~ . ,
                  data = trainData,
                  method = "svmLinear2",
                  tuneGrid = data.frame(cost = exp(seq(-2,5,len = 20))),
                  trControl = ctrl)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```r
plot(svml.fit, highlight = TRUE, xTrans = log)
```




```r
svml.fit$bestTune
```

```
##        cost
## 5 0.5907775
```

```r
# Building confusion matrix
test.pred.prob7 <- predict(svml.fit, newdata = testData,
                           type = "prob")
test.pred7 <- rep("good", length(test.pred.prob7$good))
test.pred7[test.pred.prob7$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred7),
                reference = testData$qual,
                positive = "good")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good poor
##       good  192   56
##       poor   64  167
##
```

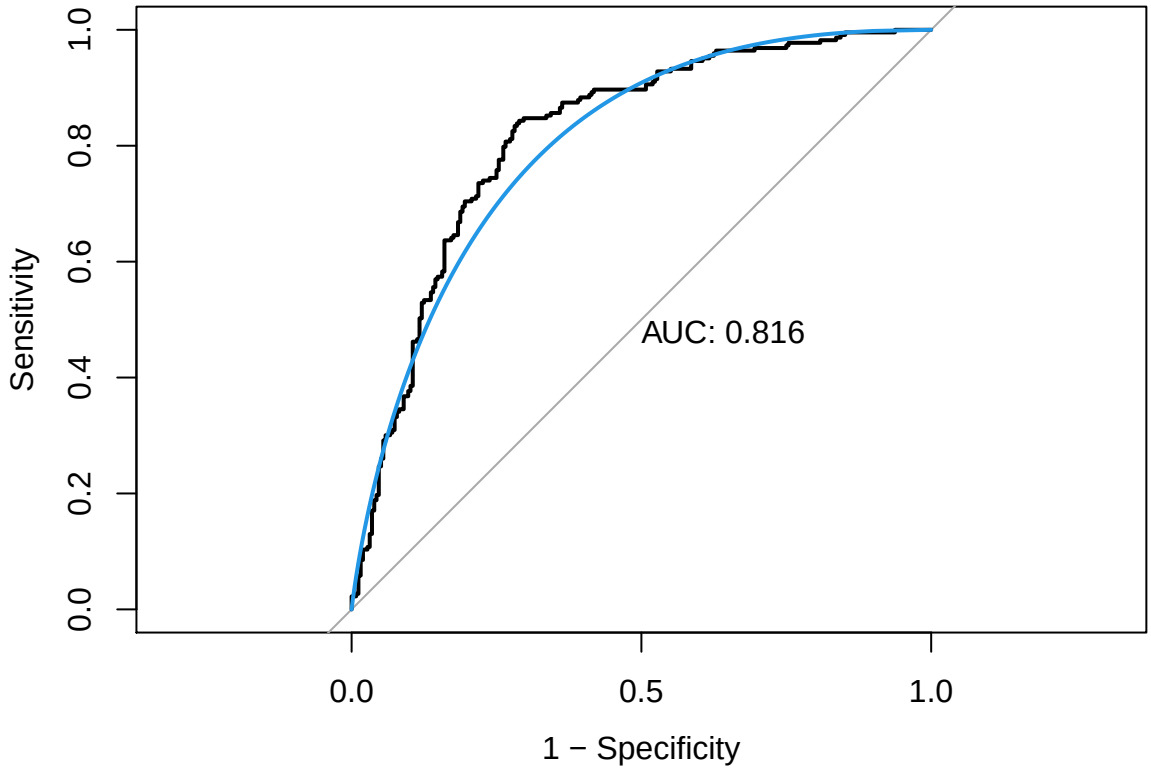
```
##                Accuracy : 0.7495
##                  95% CI : (0.7082, 0.7877)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4977
##
##  Mcnemar's Test P-Value : 0.5228
##
##             Sensitivity : 0.7500
##             Specificity : 0.7489
##          Pos Pred Value : 0.7742
##          Neg Pred Value : 0.7229
##              Prevalence : 0.5344
##          Detection Rate : 0.4008
##    Detection Prevalence : 0.5177
##       Balanced Accuracy : 0.7494
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.svml <- roc(testData$qual, test.pred.prob7$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```
plot(roc.svml, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.svml), col = 4, add = TRUE)
```

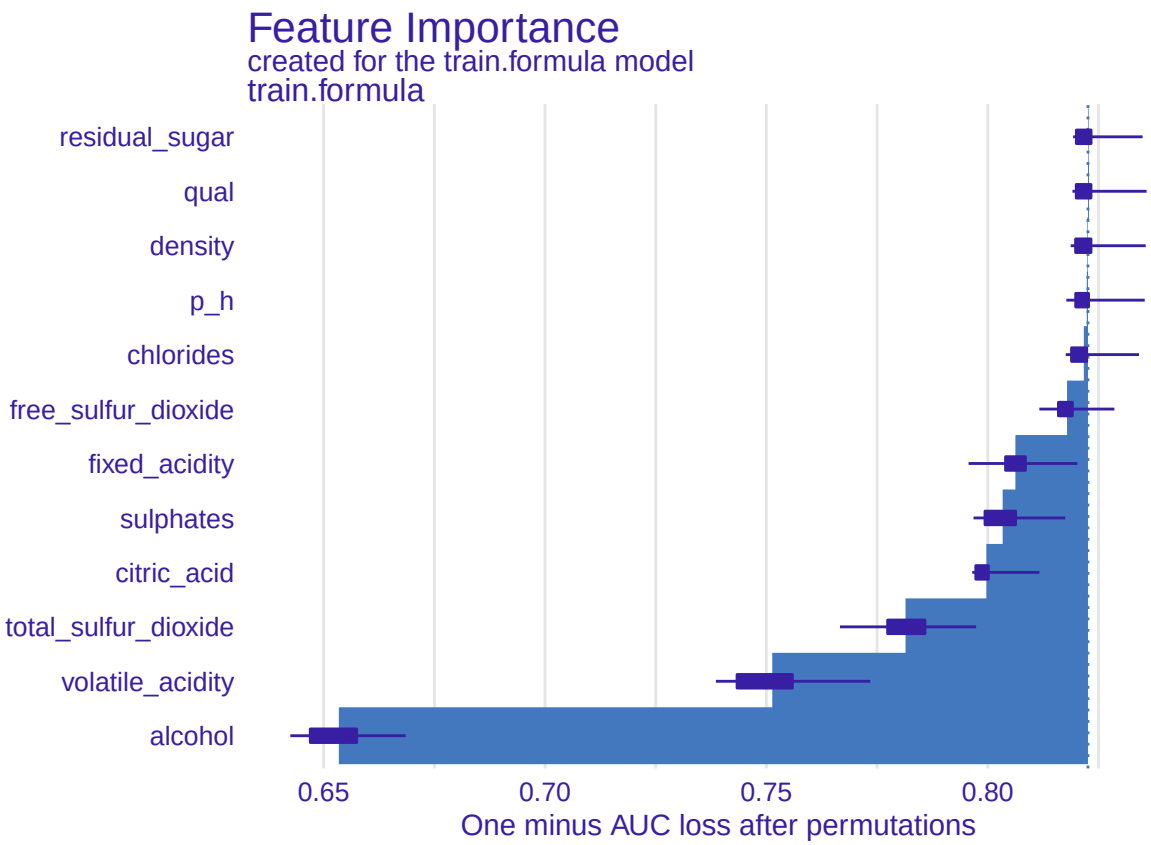



```
# Calculate the test error
error.test.svml <- mean(testData$qual != test.pred7)


# Calculate the train error rate
pred.train.svml = predict(svml.fit, newdata = trainData,type = "raw")
error.train.svml = mean(trainData$qual != pred.train.svml)

# variable importance
explainer_svm <- explain(svml.fit,
                         abel = "svml",
                         ata = trainData %>% dplyr::select(-qual),
                         y = as.numeric(trainData$qual == "good"),
                         verbose = FALSE)

vi_svm <- model_parts(explainer_svm)
plot(vi_svm)
```



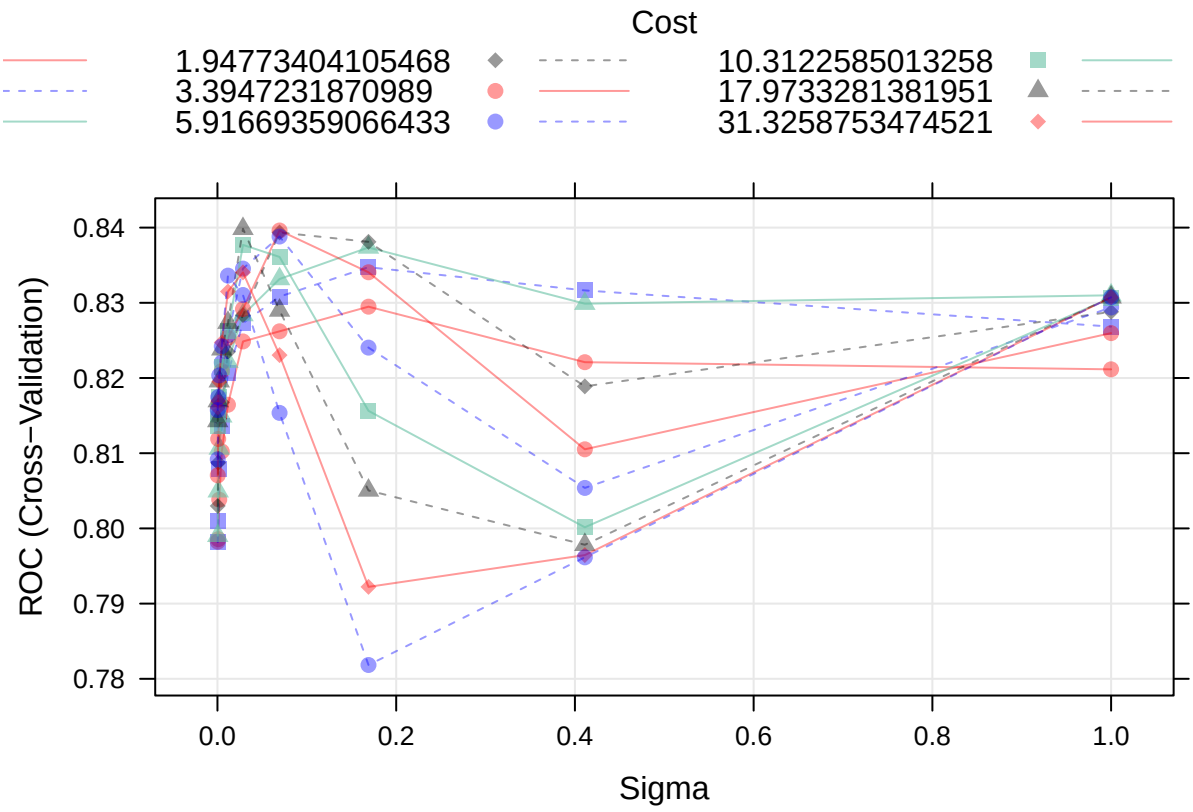

```
## SVM with radial kernel
set.seed(1)

svmr.grid <- expand.grid(C = exp(seq(-1,4,len=10)),
                         sigma = exp(seq(-8,0,len=10)))

svmr.fit <- train(qual ~ .,
                  data = trainData,
                  method = "svmRadialSigma",
                  preProcess = c("center", "scale"),
                  tuneGrid = svmr.grid,
                  trControl = ctrl)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```
plot(svmr.fit, highlight = TRUE)
```




```
svmr.fit$bestTune
```

```
##        sigma        C
## 76 0.0285655 17.97333
```

```
# Building confusion matrix
test.pred.prob8 <- predict(svmr.fit, newdata = testData,
                           type = "prob")
test.pred8 <- rep("good", length(test.pred.prob8$good))
test.pred8[test.pred.prob8$good < 0.5] <- "poor"

confusionMatrix(data = as.factor(test.pred8),
                reference = testData$qual,
                positive = "good")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good poor
##       good  197   62
##       poor   59  161
##
##                Accuracy : 0.7474
##                  95% CI : (0.706, 0.7857)
##     No Information Rate : 0.5344
```

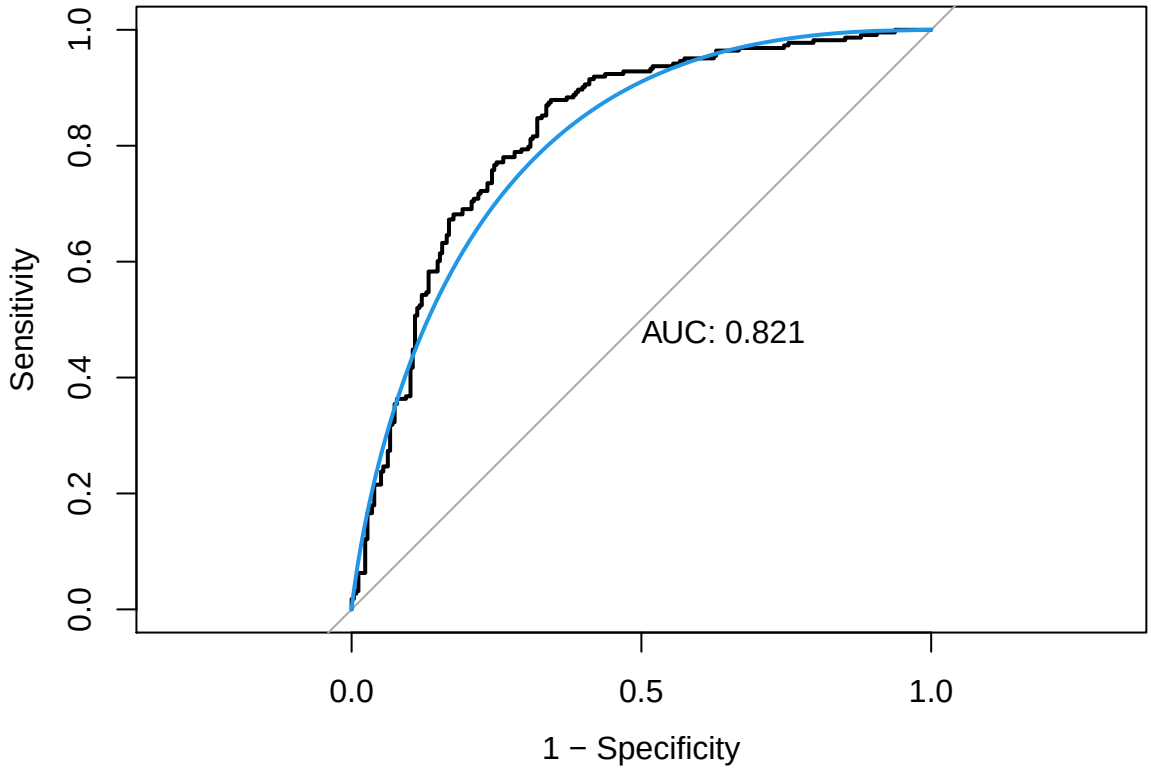
```
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4919
##
##  Mcnemar's Test P-Value : 0.8557
##
##             Sensitivity : 0.7695
##             Specificity : 0.7220
##          Pos Pred Value : 0.7606
##          Neg Pred Value : 0.7318
##              Prevalence : 0.5344
##          Detection Rate : 0.4113
##    Detection Prevalence : 0.5407
##       Balanced Accuracy : 0.7458
##
##        'Positive' Class : good
##
```

```
# Plot the test ROC
roc.svmr <- roc(testData$qual, test.pred.prob8$good)
```

```
## Setting levels: control = good, case = poor
```

```
## Setting direction: controls > cases
```

```
plot(roc.svmr, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.svmr), col = 4, add = TRUE)
```

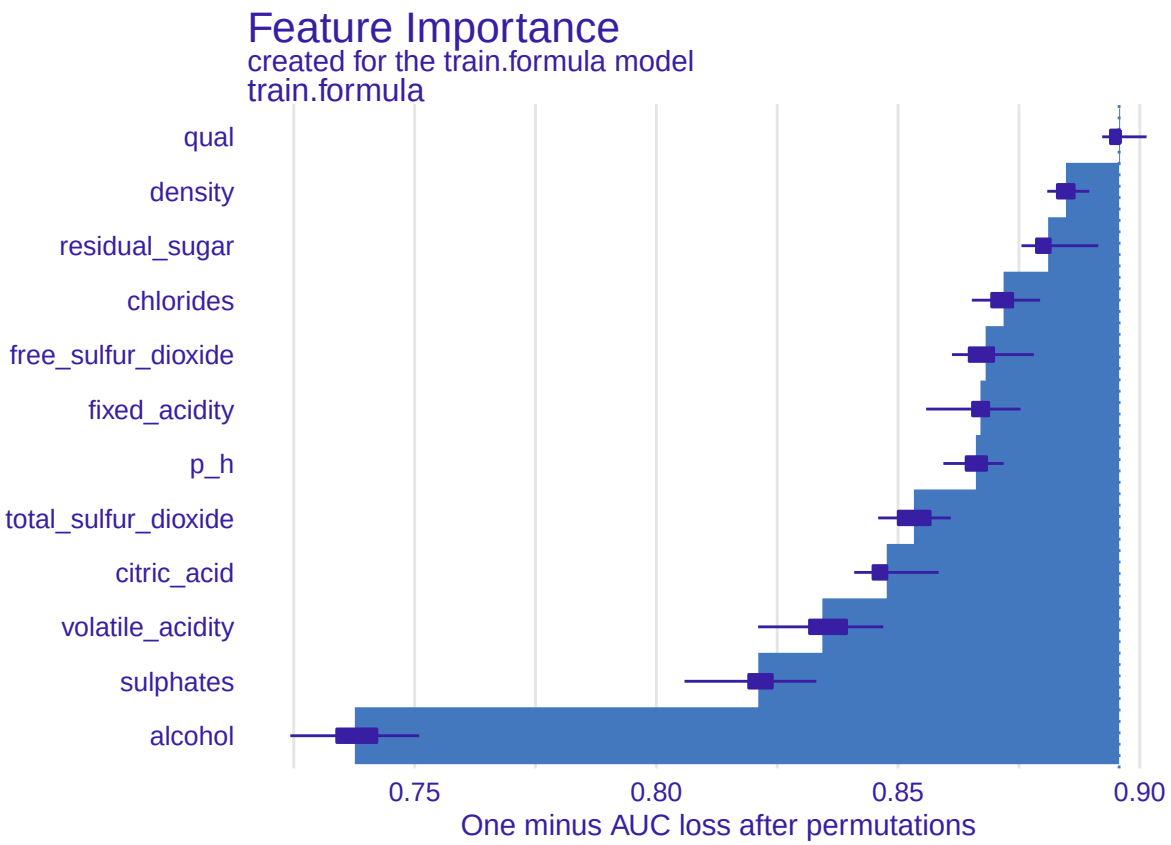



```
# Calculate the test error
error.test.svmr <- mean(testData$qual != test.pred8)


# Calculate the train error rate
pred.train.svmr = predict(svmr.fit, newdata = trainData,type = "raw")
error.train.svmr = mean(trainData$qual != pred.train.svmr)

# variable importance
explainer_svmr <- explain(svmr.fit,
                          abel = "svmr",
                          ata = trainData %>% dplyr::select(-qual),
                          y = as.numeric(trainData$qual == "good"),
                          verbose = FALSE)

vi_svmr <- model_parts(explainer_svmr)
plot(vi_svmr)
```



Model Comparison

After model building, we conducted model comparisons based on the training and test performance of all models. The following tables shows the cross-validation classification error rates and ROCs of all the models. In the results, random forest model has the largest AUC value, while KNN has the smallest. Therefore, we selected the random forest model as the best predictive classification model for our data. Based on the random forest model, alcohol, sulphates, volatile acidity, total sulfur dioxide and density are the top 5 important predictors that help us predict the classification of wine quality. Since factors such as alcohol, sulphates and volatile acidity are the ones that may determine the flavor and taste of wines, so such findings meet our expectation.

While looking at the summary of each model, we realize that KNN model has the lowest AUC value and the largest test classification error rate, 0.367. The other nine models have close AUC values that are about 82%.

```
resamp = resamples(list(logistic = model.glm,
                        penalized_logistic = model.glmn,
                        LDA = model.lda,
                        GAM = model.gam,
                        MARS = model.mars,
                        knn = fit.knn,
                        QDA = model.qda,
                        ClassTree = class.tree,
                        RandomForest = rf.fit,
                        SVML = svml.fit,
```

```
                        SVMR = svmr.fit
                        ), metric = "accuracy" )

summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: logistic, penalized_logistic, LDA, GAM, MARS, knn, QDA, ClassTree, RandomForest, SVML, SVMR
## Number of resamples: 10
##
## ROC
##                          Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## logistic           0.7512821 0.7829327 0.8112179 0.8141979 0.8472888 0.8769231
## penalized_logistic 0.7522436 0.7849359 0.8123397 0.8146170 0.8463454 0.8753205
## LDA                0.7477564 0.7834135 0.8134615 0.8141959 0.8463282 0.8772436
## GAM                0.7782051 0.7967949 0.8243590 0.8302269 0.8598512 0.8872229
## MARS               0.7855769 0.8127653 0.8300481 0.8315484 0.8432692 0.8871795
## knn                0.6456731 0.6965946 0.7371863 0.7222954 0.7475742 0.7737179
## QDA                0.6971154 0.7500000 0.8036859 0.7912686 0.8302099 0.8512821
## ClassTree          0.7546474 0.7943109 0.8202724 0.8058090 0.8245708 0.8274038
## RandomForest       0.8397436 0.8540865 0.8700895 0.8754754 0.8869391 0.9349359
## SVML               0.7490385 0.7879006 0.8099359 0.8148223 0.8523059 0.8705128
## SVMR               0.8060897 0.8112981 0.8203526 0.8398396 0.8709089 0.9041667
##                    NA's
## logistic              0
## penalized_logistic    0
## LDA                   0
## GAM                   0
## MARS                  0
## knn                   0
## QDA                   0
## ClassTree             0
## RandomForest          0
## SVML                  0
## SVMR                  0
##
## Sens
##                          Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## logistic           0.6333333 0.7083333 0.7416667 0.7498023 0.8000000 0.8813559
## penalized_logistic 0.6333333 0.7083333 0.7416667 0.7498023 0.8000000 0.8813559
## LDA                0.6333333 0.6750000 0.7416667 0.7347458 0.7791667 0.8474576
## GAM                0.6666667 0.7166667 0.7750000 0.7647458 0.7958333 0.8474576
## MARS               0.7333333 0.7666667 0.7833333 0.7981638 0.8000000 0.8983051
## knn                0.5666667 0.6500000 0.7166667 0.7046328 0.7666667 0.8000000
## QDA                0.7166667 0.7750000 0.8083333 0.8114407 0.8661017 0.8833333
## ClassTree          0.6666667 0.7083333 0.7563559 0.7446045 0.7791667 0.8000000
## RandomForest       0.7500000 0.7833333 0.8083333 0.8013842 0.8270480 0.8333333
## SVML               0.6166667 0.6708333 0.7083333 0.7130226 0.7625000 0.8135593
## SVMR               0.7333333 0.7500000 0.7666667 0.7764689 0.7833333 0.8813559
##                    NA's
## logistic              0
```

```
## penalized_logistic    0
## LDA                   0
## GAM                   0
## MARS                  0
## knn                   0
## QDA                   0
## ClassTree             0
## RandomForest          0
## SVML                  0
## SVMR                  0
##
## Spec
##                            Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## logistic           0.5769231 0.6826923 0.7500000 0.7312409 0.7800254 0.8461538
## penalized_logistic 0.5769231 0.6875000 0.7523585 0.7350871 0.7836538 0.8461538
## LDA                0.5961538 0.7307692 0.7523585 0.7466255 0.7884615 0.8461538
## GAM                0.6153846 0.6923077 0.7115385 0.7235849 0.7464623 0.8461538
## MARS               0.6346154 0.6746190 0.7019231 0.7160015 0.7548077 0.8076923
## knn                0.5384615 0.6027758 0.6346154 0.6488026 0.6875000 0.7692308
## QDA                0.4230769 0.5913462 0.6442308 0.6198476 0.6728955 0.7115385
## ClassTree          0.6346154 0.6696299 0.7307692 0.7197750 0.7500000 0.7884615
## RandomForest       0.6730769 0.7307692 0.7596154 0.7619013 0.8056060 0.8269231
## SVML               0.6538462 0.7307692 0.7523585 0.7620102 0.7836538 0.8846154
## SVMR               0.6730769 0.7211538 0.7692308 0.7580914 0.7884615 0.8269231
##                    NA's
## logistic              0
## penalized_logistic    0
## LDA                   0
## GAM                   0
## MARS                  0
## knn                   0
## QDA                   0
## ClassTree             0
## RandomForest          0
## SVML                  0
## SVMR                  0
```

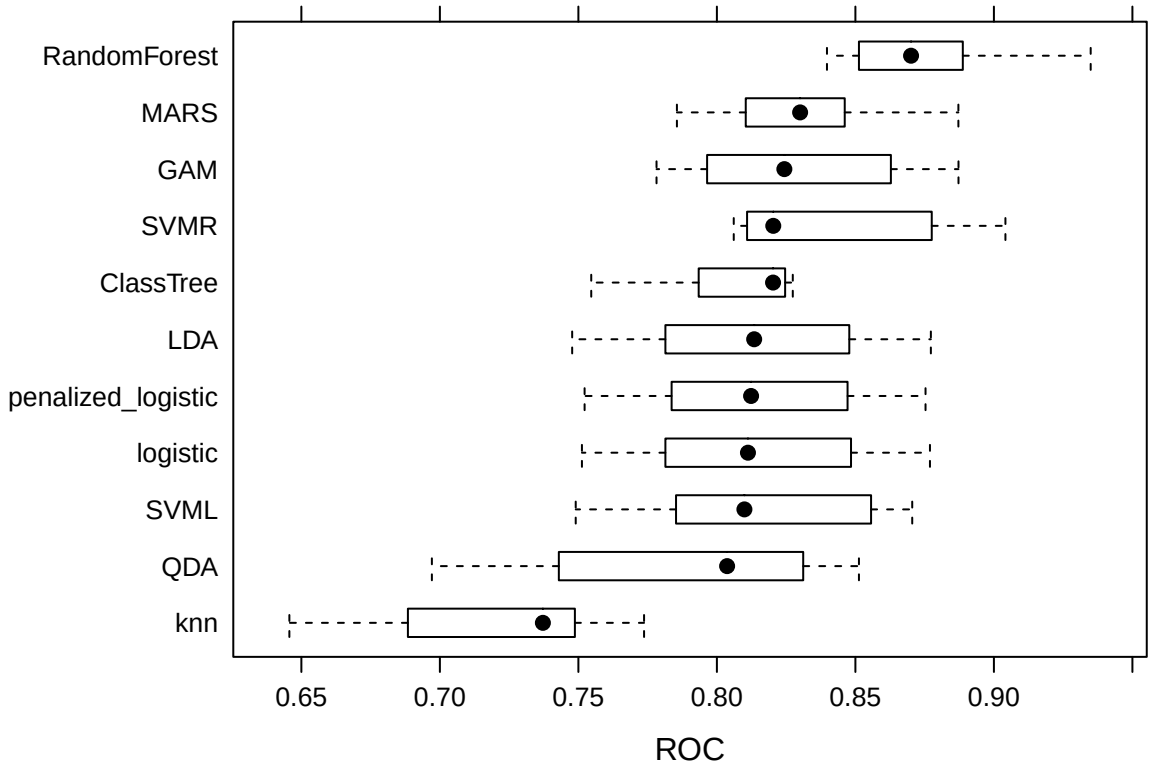
```
comparison = summary(resamp)$statistics$ROC
r_square = summary(resamp)$statistics$Rsquared

knitr::kable(comparison[,1:6])
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
logistic	0.7512821	0.7829327	0.8112179	0.8141979	0.8472888	0.8769231
penalized_logistic	0.7522436	0.7849359	0.8123397	0.8146170	0.8463454	0.8753205
LDA	0.7477564	0.7834135	0.8134615	0.8141959	0.8463282	0.8772436
GAM	0.7782051	0.7967949	0.8243590	0.8302269	0.8598512	0.8872229
MARS	0.7855769	0.8127653	0.8300481	0.8315484	0.8432692	0.8871795
knn	0.6456731	0.6965946	0.7371863	0.7222954	0.7475742	0.7737179
QDA	0.6971154	0.7500000	0.8036859	0.7912686	0.8302099	0.8512821
ClassTree	0.7546474	0.7943109	0.8202724	0.8058090	0.8245708	0.8274038
RandomForest	0.8397436	0.8540865	0.8700895	0.8754754	0.8869391	0.9349359
SVML	0.7490385	0.7879006	0.8099359	0.8148223	0.8523059	0.8705128

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
SVMR	0.8060897	0.8112981	0.8203526	0.8398396	0.8709089	0.9041667

```
bwplot(resamp, metric = "ROC")
```




```
Model_Name <- c("logistic regression", "penalized logistic regression", "LDA", "GAM", "MARS", "KNN", "Q
Train_Error <- c(error.trian.glm,error.trian.glmn,error.train.lda, error.trian.gam, error.trian.mars, e
Test_Error <- c(error.test.glm,error.test.glmn,error.test.lda, error.test.gam, error.test.mars, error.t
Test_ROC = c(0.818, 0.818, 0.819, 0.829, 0.823, 0.672, 0.784, 0.809, 0.900, 0.816, 0.821)


df <- data.frame(Model_Name, Train_Error, Test_Error, Test_ROC)

knitr::kable(df)
```

Model_Name	Train_Error	Test_Error	Test_ROC
logistic regression	0.2553571	0.2505219	0.818
penalized logistic regression	0.2553571	0.2463466	0.818
LDA	0.2571429	0.2379958	0.819
GAM	0.2151786	0.2400835	0.829
MARS	0.2321429	0.2463466	0.823
KNN	0.2928571	0.3674322	0.672
QDA	0.2526786	0.2922756	0.784

Model_Name	Train_Error	Test_Error	Test_ROC
Classification Tree	0.1508929	0.2421712	0.809
Random forest	0.0000000	0.1628392	0.900
SVM with linear kernel	0.2589286	0.2505219	0.816
SVM with radial kernel	0.1866071	0.2526096	0.821

Conlcusion

The process of model building shows that in the training dataset, alcohol, sulphates, volatile acidity, total sulfur dioxide and density are the top 5 important predictors for classification of wine quality. We selected the random forest model because of its largest AUC value and lowest classification error rate. This model also performs well in the test dataset. Therefore, this random forest model is an effective method for classification of wine quality.