Grade A Wine Analysis

Raam Pravin

2024-12-28

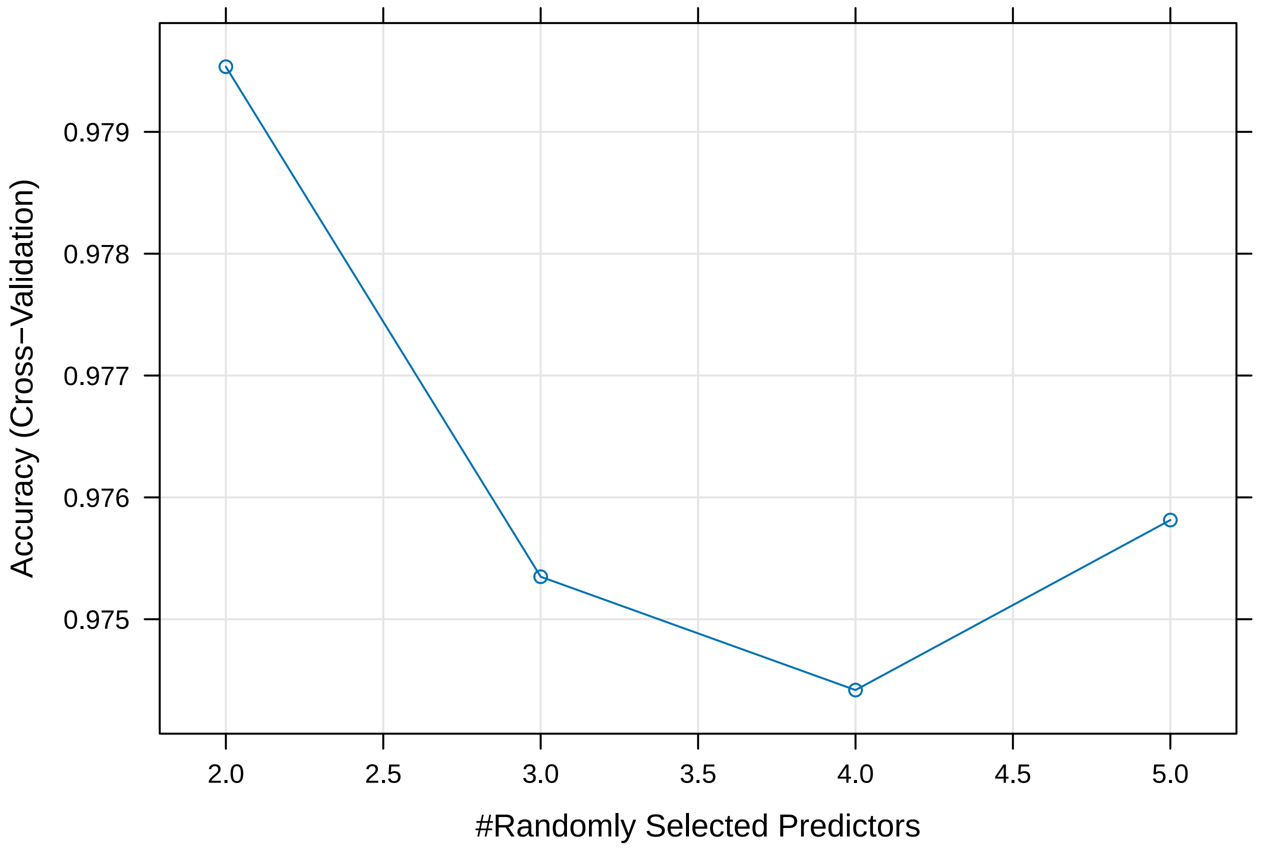
```
unregister_dopar <- function() {
  env <- foreach:::.foreachGlobals
  rm(list=ls(name=env), pos=env)
}
unregister_dopar()

#Random Forest Model for predicting Grade A Red Wine

cat("Calling extra sample storing in data frame and using cross validation and
    grid search to find optimal parameters")
```

```
## Calling extra sample storing in data frame and using cross validation and
##     grid search to find optimal parameters
```

```
red_wine_rf_extra <- oversampled_red_wine_train_list[[31]]

# Define the control for grid search with 10-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define the grid of hyper-parameters to tune
tune_grid <- expand.grid(
  mtry = c(2,3,4,5))

cat("Training the Random Forest model using grid search and
    10-fold cross-validation for Red Wine")
```

```
## Training the Random Forest model using grid search and
##     10-fold cross-validation for Red Wine
```

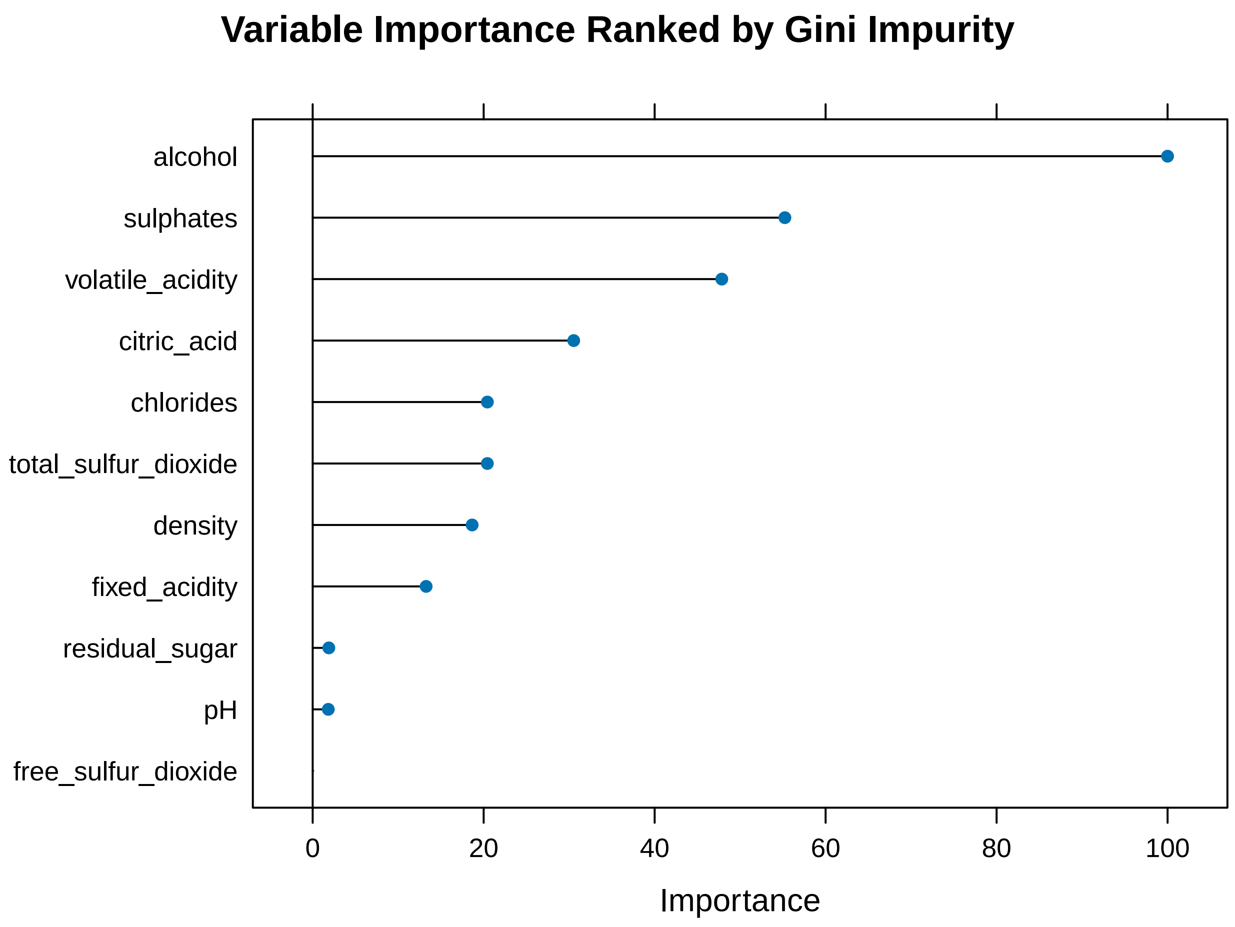
```
rf_gridsearch_red <- caret::train(quality ~ .,
                        red_wine_rf_extra,
                        method = "rf",
                        trControl = train_control,
                        tuneGrid = tune_grid,
                        importance = TRUE)
plot(rf_gridsearch_red)
```



```
cat("This plot shows that optimal number of variables
    to try at every node split is 2")
```

```
## This plot shows that optimal number of variables
##     to try at every node split is 2
```

```
#Variable Importance Plot of model predicting Grade A red wine
rf_gridsearch_red_importance <- varImp(rf_gridsearch_red, type = 2)
plot(rf_gridsearch_red_importance,
     main = "Variable Importance Ranked by Gini Impurity")
```



```
cat("This plot shows that the elbow of the importance plot is at the fifth most
    important variable so the remaining variables are dropped from future model,
    these variables are: total_sulfur_dioxide, density, fixed_acidity,
    residual_sugar, pH, free_sulfur_dioxide")
```

```
## This plot shows that the elbow of the importance plot is at the fifth most
##     important variable so the remaining variables are dropped from future model,
##     these variables are: total_sulfur_dioxide, density, fixed_acidity,
##     residual_sugar, pH, free_sulfur_dioxide
```

```
drop_columns <- c("total_sulfur_dioxide","density","fixed_acidity",
                  "residual_sugar","pH","free_sulfur_dioxide")

oversampled_red_wine_train_list <- lapply(oversampled_red_wine_train_list, function(df) {
  df %>% dplyr::select(-all_of(drop_columns))
})


cat("Examining the grid-search's plot it shows the optimal number of variables
    to randomly sample from at every node split is 2, now applying Random Forest
    Model with optimal parameter 30 times, since this is very time consuming
    using parallel processing")
```

```
## Examining the grid-search's plot it shows the optimal number of variables
##     to randomly sample from at every node split is 2, now applying Random Forest
##     Model with optimal parameter 30 times, since this is very time consuming
##     using parallel processing
```

```r
#Creating empty lists
accuracy_vector_red <- numeric(length(1:30))
conf_mat_list_red <- vector("list",length(1:30))
variable_importance_list_red <- vector("list",length(1:30))

tune_grid2 <- expand.grid(mtry = 2)

#initializing parallel processing
num_cores <- detectCores() - 2
cl <- makePSOCKcluster(num_cores)
registerDoParallel(cl)


results <- foreach (i = 1:length(oversampled_red_wine_train_list),
                    .packages = c("caret", "dplyr")) %dopar% {
# Training the Random Forest model with 30 times
  rf_model_red <- caret::train(
    quality ~ .,
    data = oversampled_red_wine_train_list[[i]],
    method = "rf",
    tuneGrid = tune_grid2,
    importance = TRUE
  )

#Confusion Matrix of final model predicting Grade A red wine
predictions_red <- predict(rf_model_red, newdata = red_wine_test_list[[i]])
confusion_mat <- confusionMatrix(predictions_red, red_wine_test_list[[i]]$quality)
#conf_mat_list_red[[i]] <- confusion_mat

accuracy_vector_red[i] <- confusion_mat$overall['Accuracy']

var_importance <- varImp(rf_model_red, type = 2)
variable_importance_list_red[[i]] <- var_importance

list(
confusion_matrix = confusion_mat,
accuracy = confusion_mat$overall['Accuracy'],
variable_importance = var_importance
)
}
stopCluster(cl)

for (i in 1:length(results)) {
  conf_mat_list_red[[i]] <- results[[i]]$confusion_matrix
  accuracy_vector_red[i] <- results[[i]]$accuracy
  variable_importance_list_red[[i]] <- results[[i]]$variable_importance
}


cat("Creating 95% Confidence Interval for Accuracy of Model
    predicting Grade A red wine")
```

```
## Creating 95% Confidence Interval for Accuracy of Model
##     predicting Grade A red wine
```

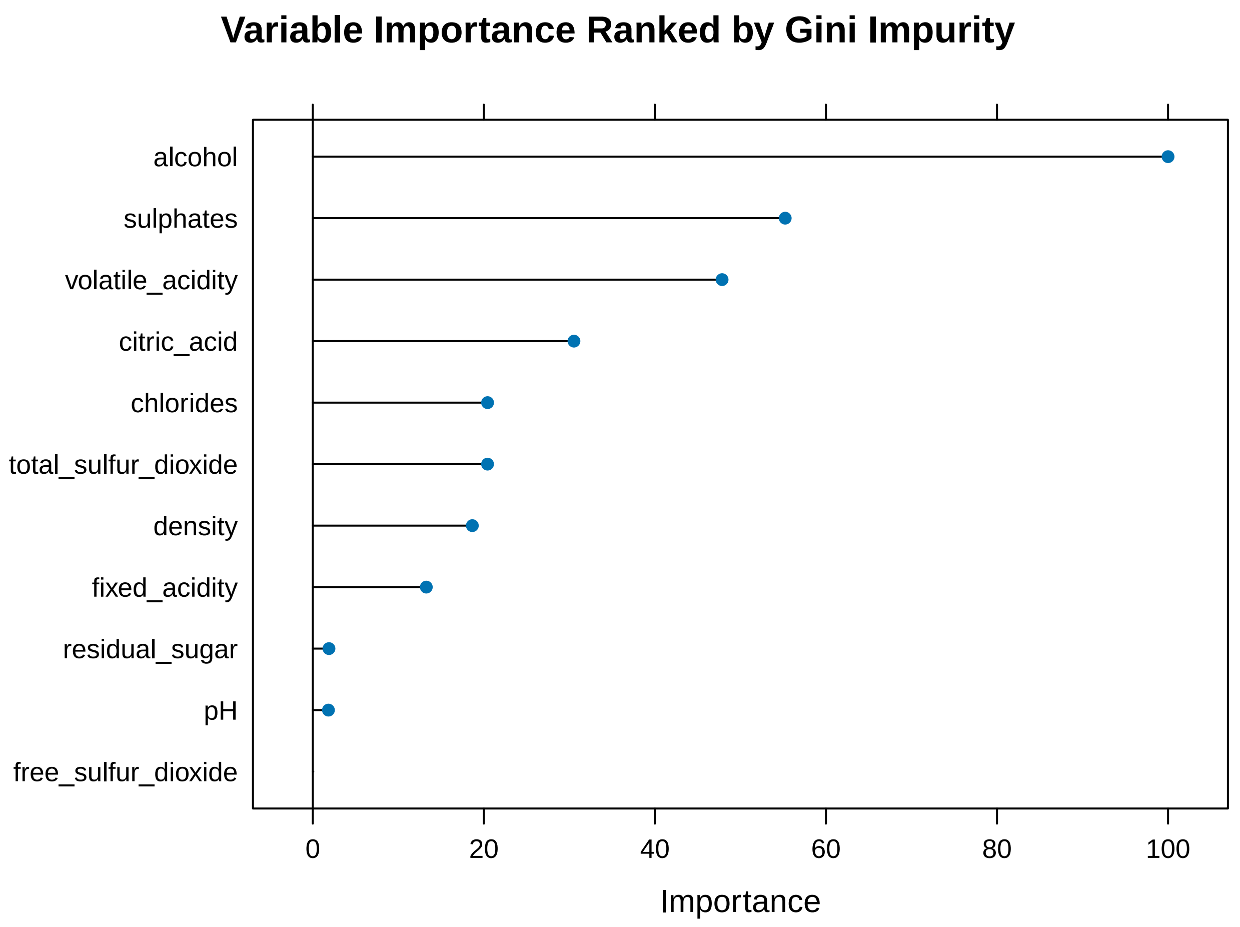
```
mean_red2_vec  <- mean(accuracy_vector_red)

#standard error
std_error_red <- sd(accuracy_vector_red) / sqrt(length(accuracy_vector_red))

#critical t value for 95% CI
critical_value_red <- qt(0.975, df = length(accuracy_vector_red) - 1)

#confidence interval
lower_ci_red <- mean_red2_vec - (critical_value_red * std_error_red)
upper_ci_red <- mean_red2_vec + (critical_value_red * std_error_red)

# 95% CI
cat("95% Confidence Interval Predicting Grade A Red Wine: [", lower_ci_red, ", ", upper_ci_red, "]\n")
```

```
## 95% Confidence Interval Predicting Grade A Red Wine: [ 0.8893452 ,  0.9029991 ]
```

```
#Finding Index of accuracy value closest to mean
closest_index_red <- which.min(abs(accuracy_vector_red - mean_red2_vec))


#Confusion Matrix of Model closest to mean accuracy
print(conf_mat_list_red[closest_index_red])
```

```
## [[1]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 275  14
##          1  21  23
##
##                Accuracy : 0.8949
##                  95% CI : (0.8569, 0.9257)
##     No Information Rate : 0.8889
##     P-Value [Acc > NIR] : 0.4051
##
##                   Kappa : 0.5086
##
##  Mcnemar's Test P-Value : 0.3105
##
##             Sensitivity : 0.9291
##             Specificity : 0.6216
##          Pos Pred Value : 0.9516
##          Neg Pred Value : 0.5227
##              Prevalence : 0.8889
##          Detection Rate : 0.8258
##    Detection Prevalence : 0.8679
##       Balanced Accuracy : 0.7753
##
##        'Positive' Class : 0
##
```

```r
#Variable Importance Plot of model
plot(rf_gridsearch_red_importance,
     main = "Variable Importance Ranked by Gini Impurity")
```



```r
unregister_dopar <- function() {
  env <- foreach:::.foreachGlobals
  rm(list=ls(name=env), pos=env)
}
unregister_dopar()

#Random Forest Model for predicting Grade A White Wine

cat("Calling extra sample storing in data frame and using cross validation and
    grid search to find optimal parameters")
```

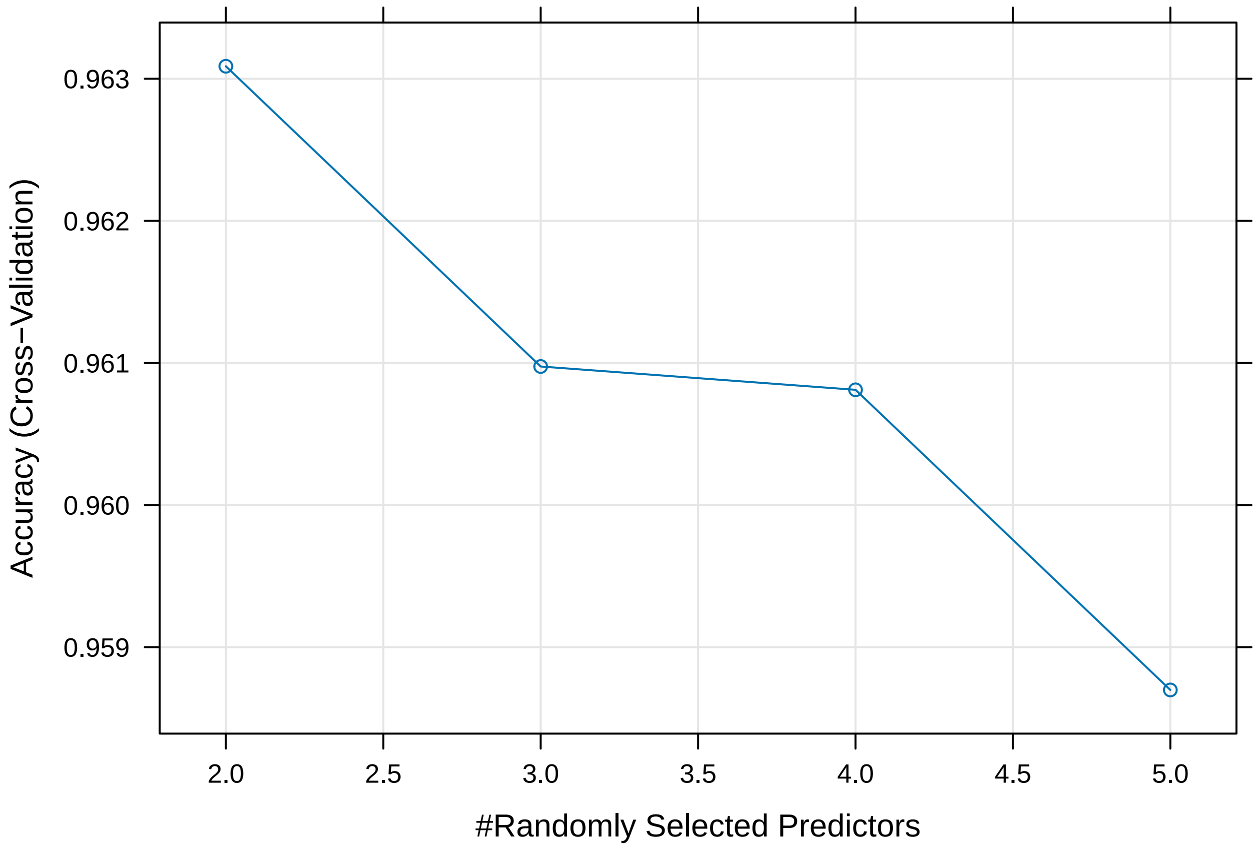
```
## Calling extra sample storing in data frame and using cross validation and
##     grid search to find optimal parameters
```

```r
white_wine_rf_extra <- oversampled_white_wine_train_list[[31]]

# Define the control for grid search with 10-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define the grid of hyper-parameters to tune
tune_grid <- expand.grid(mtry = c(2,3,4,5))
```

```r
cat("Training the Random Forest model using grid search and
    10-fold cross-validation for White Wine")
```

```
## Training the Random Forest model using grid search and
##     10-fold cross-validation for White Wine
```

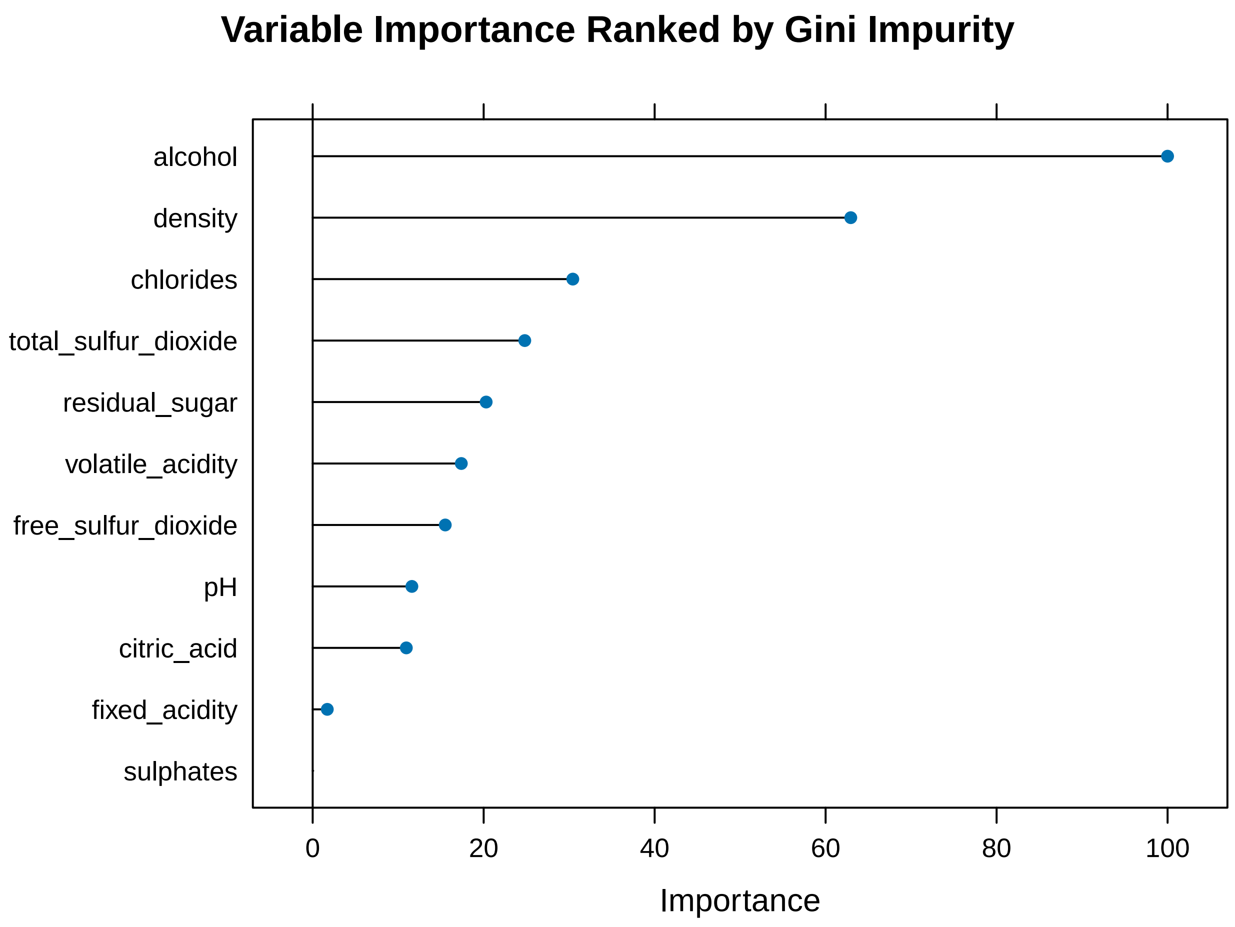
```r
rf_gridsearch_white <- caret::train(quality ~ .,
                         white_wine_rf_extra,
                         method = "rf",
                         trControl = train_control,
                         tuneGrid = tune_grid,
                         importance = TRUE)
plot(rf_gridsearch_white)
```



```r
cat("This plot shows that optimal number of variables to
    try at every node split is 2")
```

```
## This plot shows that optimal number of variables to
##     try at every node split is 2
```

```r
#Variable Importance Plot of model predicting Grade A white wine
rf_gridsearch_white_importance <- varImp(rf_gridsearch_white, type = 2)
plot(rf_gridsearch_white_importance,
     main = "Variable Importance Ranked by Gini Impurity")
```



```
cat("This plot shows that the elbow of the importance plot is at the fifth most
    important variable so the remaining variables are dropped from future model,
    these variables are: volatile_acidity, free_sulfur_dioxide, pH, citric_acid,
    fixed_acidity, sulphates")
```

```
## This plot shows that the elbow of the importance plot is at the fifth most
##     important variable so the remaining variables are dropped from future model,
##     these variables are: volatile_acidity, free_sulfur_dioxide, pH, citric_acid,
##     fixed_acidity, sulphates
```

```
drop_columns <- c("volatile_acidity","free_sulfur_dioxide", "pH", "citric_acid",
                  "fixed_acidity", "sulphates")

oversampled_white_wine_train_list <- lapply(oversampled_white_wine_train_list,
                                            function(df) {
  df %>% dplyr::select(-all_of(drop_columns))
})


cat("Examining the grid-search's plot it shows the optimal number of variables
    to randomly sample from at every node split is 2, now applying Random Forest
    Model with optimal parameter 30 times, since this is very time consuming
    using parallel processing")
```

```
## Examining the grid-search's plot it shows the optimal number of variables
##     to randomly sample from at every node split is 2, now applying Random Forest
##     Model with optimal parameter 30 times, since this is very time consuming
##     using parallel processing
```

```
#Creating empty lists
accuracy_vector_white <- numeric(length(1:30))
conf_mat_list_white <- vector("list",length(1:30))
variable_importance_list_white <- vector("list",length(1:30))

tune_grid2 <- expand.grid(mtry = 2)

#initializing parallel processing
num_cores <- detectCores() - 2
cl <- makePSOCKcluster(num_cores)
registerDoParallel(cl)


results <- foreach (i = 1:length(oversampled_white_wine_train_list),
                    .packages = c("caret", "dplyr")) %dopar% {
# Training the Random Forest model with 30 times
  rf_model_white <- caret::train(
    quality ~ .,
    data = oversampled_white_wine_train_list[[i]],
    method = "rf",
    tuneGrid = tune_grid2,
    importance = TRUE
  )

#Confusion Matrix of final model pwhiteicting Grade A white wine
predictions_white <- predict(rf_model_white, newdata = white_wine_test_list[[i]])
confusion_mat <- confusionMatrix(predictions_white,
                                 white_wine_test_list[[i]]$quality)
#conf_mat_list_white[[i]] <- confusion_mat

accuracy_vector_white[i] <- confusion_mat$overall['Accuracy']

var_importance <- varImp(rf_model_white, type = 2)
variable_importance_list_white[[i]] <- var_importance

list(
confusion_matrix = confusion_mat,
accuracy = confusion_mat$overall['Accuracy'],
variable_importance = var_importance
)
}
stopCluster(cl)

for (i in 1:length(results)) {
  conf_mat_list_white[[i]] <- results[[i]]$confusion_matrix
  accuracy_vector_white[i] <- results[[i]]$accuracy
  variable_importance_list_white[[i]] <- results[[i]]$variable_importance
}


cat("Creating 95% Confidence Interval for Accuracy of Model predicting
    Grade A white wine")
```

```
## Creating 95% Confidence Interval for Accuracy of Model predicting
##     Grade A white wine
```

```
mean_white2_vec  <- mean(accuracy_vector_white)

#standard error
std_error_white <- sd(accuracy_vector_white) / sqrt(length(accuracy_vector_white))

#critical t value for 95% CI
critical_value_white <- qt(0.975, df = length(accuracy_vector_white) - 1)

#confidence interval
lower_ci_white <- mean_white2_vec - (critical_value_white * std_error_white)
upper_ci_white <- mean_white2_vec + (critical_value_white * std_error_white)

# 95% CI
cat("95% Confidence Interval Predicting Grade A white Wine: [", lower_ci_white, ", ", upper_ci_white, "
```

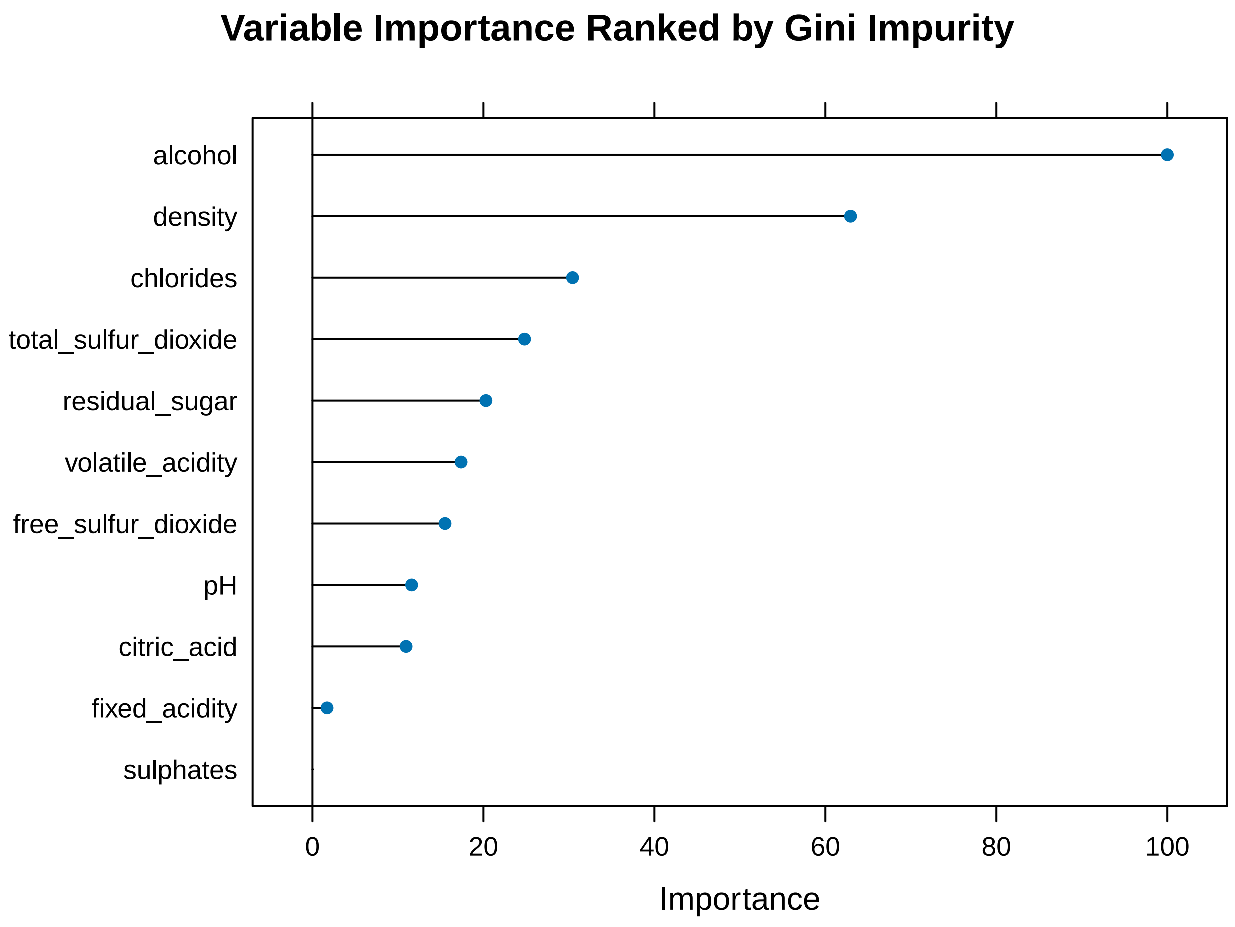
```
## 95% Confidence Interval Predicting Grade A white Wine: [ 0.8581418 ,  0.864994 ]
```

```
#Finding Index of accuracy value closest to mean
closest_index_white <- which.min(abs(accuracy_vector_white - mean_white2_vec))


#Confusion Matrix of Model closest to mean accuracy
print(conf_mat_list_white[closest_index_white])
```

```
## [[1]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 687  71
##          1  63 145
##
##                Accuracy : 0.8613
##                  95% CI : (0.8379, 0.8825)
##     No Information Rate : 0.7764
##     P-Value [Acc > NIR] : 1.63e-11
##
##                   Kappa : 0.5951
##
##  Mcnemar's Test P-Value : 0.5454
##
##             Sensitivity : 0.9160
##             Specificity : 0.6713
##          Pos Pred Value : 0.9063
##          Neg Pred Value : 0.6971
##              Prevalence : 0.7764
##          Detection Rate : 0.7112
##    Detection Prevalence : 0.7847
##       Balanced Accuracy : 0.7936
##
```

```
##        'Positive' Class : 0
##
```

```
#Variable Importance Plot of model
plot(rf_gridsearch_white_importance, main = "Variable Importance Ranked by Gini Impurity")
```


Variable Importance Ranked by Gini Impurity
alcohol
density
chlorides
total_sulfur_dioxide
residual_sugar
volatile_acidity
free_sulfur_dioxide
pH
citric_acid
fixed_acidity
sulphates
Importance