Classifying wine quality from physicochemical properties

Flavia Jiang, Chunyu Wu

2023-12-01

Cleaning

Created red_df for red wine and white_df for white wine.

```
df <- read.csv("wine-quality-white-and-red.csv")
red_df <- subset(filter(df, type == "red"), select = -type)
white_df <- subset(filter(df, type == "white"), select = -type)
```

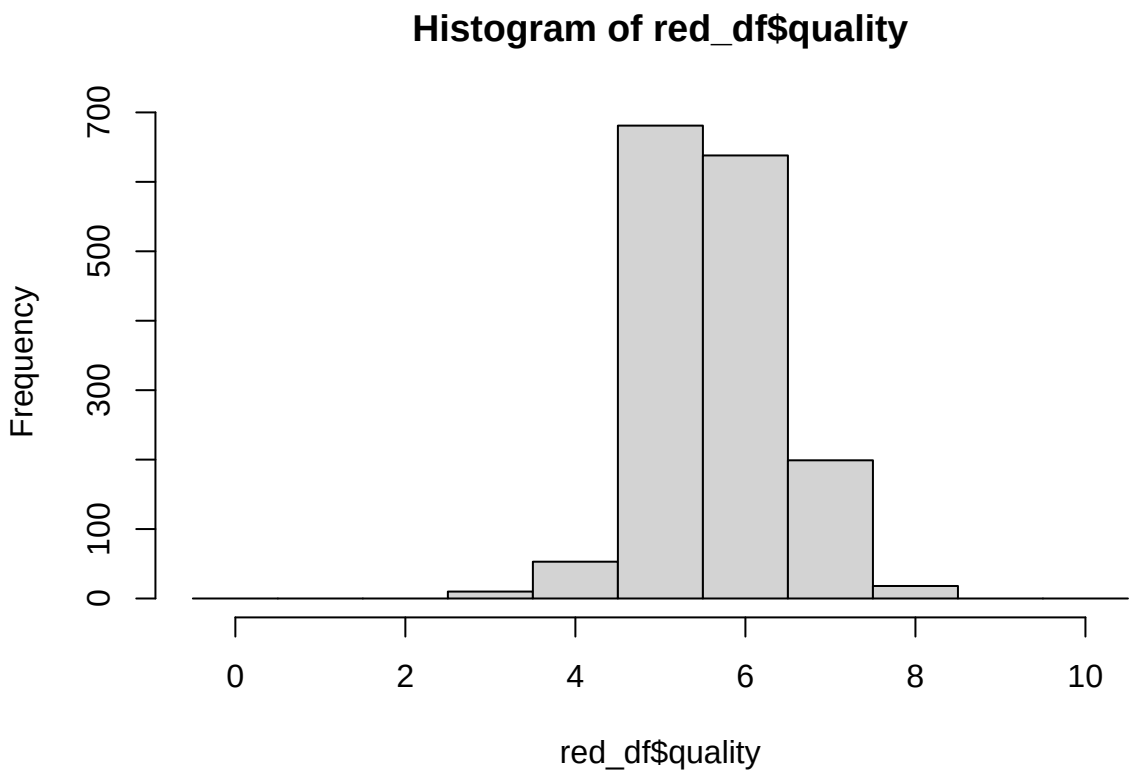
Predict red wine quality

Descriptive Statistics

```
summary(red_df)
```

```
##  fixed.acidity   volatile.acidity  citric.acid    residual.sugar
##  Min.   : 4.60   Min.   :0.120    Min.   :0.000   Min.   : 0.90
##  1st Qu.: 7.10   1st Qu.:0.390    1st Qu.:0.090   1st Qu.: 1.90
##  Median : 7.90   Median :0.520    Median :0.260   Median : 2.20
##  Mean   : 8.32   Mean   :0.528    Mean   :0.271   Mean   : 2.54
##  3rd Qu.: 9.20   3rd Qu.:0.640    3rd Qu.:0.420   3rd Qu.: 2.60
##  Max.   :15.90   Max.   :1.580    Max.   :1.000   Max.   :15.50
##    chlorides      free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.0120   Min.   : 1.0        Min.   :  6.0        Min.   :0.990
##  1st Qu.:0.0700   1st Qu.: 7.0        1st Qu.: 22.0        1st Qu.:0.996
##  Median :0.0790   Median :14.0        Median : 38.0        Median :0.997
##  Mean   :0.0875   Mean   :15.9        Mean   : 46.5        Mean   :0.997
##  3rd Qu.:0.0900   3rd Qu.:21.0        3rd Qu.: 62.0        3rd Qu.:0.998
##  Max.   :0.6110   Max.   :72.0        Max.   :289.0        Max.   :1.004
##        pH         sulphates        alcohol        quality
##  Min.   :2.74   Min.   :0.330   Min.   : 8.4   Min.   :3.00
##  1st Qu.:3.21   1st Qu.:0.550   1st Qu.: 9.5   1st Qu.:5.00
##  Median :3.31   Median :0.620   Median :10.2   Median :6.00
##  Mean   :3.31   Mean   :0.658   Mean   :10.4   Mean   :5.64
##  3rd Qu.:3.40   3rd Qu.:0.730   3rd Qu.:11.1   3rd Qu.:6.00
##  Max.   :4.01   Max.   :2.000   Max.   :14.9   Max.   :8.00
```

```
hist(red_df$quality, breaks = seq(-0.5, 10.5))
```




```
table(red_df$quality)
```

```
##
##   3   4   5   6   7   8
##  10  53 681 638 199  18
```

The distribution of response classes is unbalanced.

```
cor <- round(cor(red_df),3)
(cor != 1 & (cor > 0.7 | cor < -0.7))
```

```
##                       fixed.acidity volatile.acidity citric.acid residual.sugar
## fixed.acidity                 FALSE            FALSE       FALSE          FALSE
## volatile.acidity              FALSE            FALSE       FALSE          FALSE
## citric.acid                   FALSE            FALSE       FALSE          FALSE
## residual.sugar                FALSE            FALSE       FALSE          FALSE
## chlorides                     FALSE            FALSE       FALSE          FALSE
## free.sulfur.dioxide           FALSE            FALSE       FALSE          FALSE
## total.sulfur.dioxide          FALSE            FALSE       FALSE          FALSE
## density                       FALSE            FALSE       FALSE          FALSE
## pH                            FALSE            FALSE       FALSE          FALSE
## sulphates                     FALSE            FALSE       FALSE          FALSE
```

```
## alcohol                       FALSE           FALSE       FALSE          FALSE
## quality                       FALSE           FALSE       FALSE          FALSE
##                      chlorides free.sulfur.dioxide total.sulfur.dioxide density
## fixed.acidity            FALSE               FALSE                FALSE   FALSE
## volatile.acidity         FALSE               FALSE                FALSE   FALSE
## citric.acid              FALSE               FALSE                FALSE   FALSE
## residual.sugar           FALSE               FALSE                FALSE   FALSE
## chlorides                FALSE               FALSE                FALSE   FALSE
## free.sulfur.dioxide      FALSE               FALSE                FALSE   FALSE
## total.sulfur.dioxide     FALSE               FALSE                FALSE   FALSE
## density                  FALSE               FALSE                FALSE   FALSE
## pH                       FALSE               FALSE                FALSE   FALSE
## sulphates                FALSE               FALSE                FALSE   FALSE
## alcohol                  FALSE               FALSE                FALSE   FALSE
## quality                  FALSE               FALSE                FALSE   FALSE
##                         pH sulphates alcohol quality
## fixed.acidity        FALSE     FALSE   FALSE   FALSE
## volatile.acidity     FALSE     FALSE   FALSE   FALSE
## citric.acid          FALSE     FALSE   FALSE   FALSE
## residual.sugar       FALSE     FALSE   FALSE   FALSE
## chlorides            FALSE     FALSE   FALSE   FALSE
## free.sulfur.dioxide  FALSE     FALSE   FALSE   FALSE
## total.sulfur.dioxide FALSE     FALSE   FALSE   FALSE
## density              FALSE     FALSE   FALSE   FALSE
## pH                   FALSE     FALSE   FALSE   FALSE
## sulphates            FALSE     FALSE   FALSE   FALSE
## alcohol              FALSE     FALSE   FALSE   FALSE
## quality              FALSE     FALSE   FALSE   FALSE
```

Pair of variables highly correlated (magnitude > 0.7): none

Modeling

It is possible to do repeated k-fold CV with the trainControl function from the caret package. However, it only outputs RMSE, R-sq, and MAE for regression methods. We thought it would be more helpful to know the accuracy and AUC, so we did this manually for every method.

MLR

```
mlr_cv <- function(df){
  accuracy_mlr <- c()
  mae_mlr <- c()
  auc_mlr <- c()
  # for each seed, do a 5-fold CV
  for (seed in c(1, 10, 100, 1000, 10000)){
    set.seed(seed)
    folds <- createFolds(df$quality,k=5)
    # for each fold:
    for (i in 1:5){
      train <- df[-folds[[i]],]
```

```
      test <- df[folds[[i]],]
      model <- lm(quality~., data = train)
      pred <- predict(model, test)
      mae_mlr <- c(mae_mlr, mean(abs(test$quality-pred)))
      # round pred to the nearest integer to calculate accuracy and AUC
      accuracy_mlr <- c(accuracy_mlr, mean(round(pred) == test$quality))
      auc_mlr <- c(auc_mlr, multiclass.roc(test$quality, round(pred))$auc)
    }
  }
  return(c(mean(accuracy_mlr), mean(auc_mlr), mean(mae_mlr)))
}

mlr_cv(red_df)
```

```
## [1] 0.5923 0.7728 0.5038
```

CV accuracy = 59.23% CV AUC = 77.28% CV MAE = 0.504

Best subset selection

For computational efficiency, we would first let the regsubsets function choose the best k-variable model (k ranges from 0 to 11). Then for each k-variable model, we did a repeated 5-fold CV.

```
best_cv <- function(df){
  p <- ncol(df) - 1
  best_subset <- regsubsets(quality~., data=df, nvmax = p)
  which <- summary(best_subset)$which
  for (i in 1:p){
    columns <- c(which[i, -1], T)
    temp <- df[,columns]
    print(c(i,mlr_cv(temp)))
  }
}
best_cv(red_df)
```

```
## [1] 1.0000 0.5576 0.6911 0.5627
## [1] 2.0000 0.5690 0.7591 0.5277
## [1] 3.0000 0.5810 0.7634 0.5166
## [1] 4.0000 0.5877 0.7683 0.5094
## [1] 5.0000 0.5911 0.7666 0.5071
## [1] 6.0000 0.5921 0.7739 0.5052
## [1] 7.0000 0.5929 0.7743 0.5038
## [1] 8.0000 0.5925 0.7730 0.5041
## [1] 9.0000 0.5918 0.7743 0.5037
## [1] 10.0000  0.5909  0.7734  0.5039
## [1] 11.0000  0.5923  0.7728  0.5038
```

The 7-predictor model resulted in the highest accuracy - 59.29%.

```
summary(regsubsets(quality~., data=red_df, nvmax = 11))$which[7,]
```

```
##          (Intercept)        fixed.acidity     volatile.acidity
##                 TRUE                FALSE                 TRUE
##          citric.acid       residual.sugar            chlorides
##                FALSE                FALSE                 TRUE
##  free.sulfur.dioxide total.sulfur.dioxide              density
##                 TRUE                 TRUE                FALSE
##                   pH            sulphates              alcohol
##                 TRUE                 TRUE                 TRUE
```

LDA

```
lda_cv <- function(df){
  accuracy_lda <- c()
  mae_lda <- c()
  auc_lda <- c()
  for (seed in c(1, 10, 100, 1000, 10000)){
    set.seed(seed)
    folds <- createFolds(df$quality,k=5)
    for (i in 1:5){
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      model <- lda(quality ~ ., data = train)
      pred <- as.numeric(as.character(predict(model, test)$class))
      accuracy_lda <- c(accuracy_lda, mean(pred == test$quality))
      mae_lda <- c(mae_lda, mean(abs(test$quality - pred)))
      auc_lda <- c(auc_lda, multiclass.roc(test$quality, pred)$auc)
  }
}
  print(c(mean(accuracy_lda),mean(auc_lda), mean(mae_lda)))
}
lda_cv(red_df)
```

```
## [1] 0.5915 0.7769 0.4548
```

Naive Bayes

Assumes predictors are independent of each other.

```
nb_cv <- function(df){
  accuracy_nb <- c()
  mae_nb <- c()
  auc_nb <- c()
  for (seed in c(1, 10, 100, 1000, 10000)){
    set.seed(seed)
    folds <- createFolds(df$quality,k=5)
    for (i in 1:5){
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      model <- naiveBayes(quality~., data = train)
      pred <- as.numeric(as.character(predict(model, test)))
      accuracy_nb <- c(accuracy_nb, mean(pred == test$quality))
```

```
      mae_nb <- c(mae_nb, mean(abs(test$quality - pred)))
      auc_nb <- c(auc_nb, multiclass.roc(test$quality, pred)$auc)
  }
}
  print(c(mean(accuracy_nb), mean(auc_nb), mean(mae_nb)))
}
nb_cv(red_df)
```

```
## [1] 0.5481 0.8065 0.5216
```

Lasso regression

```
lasso_ridge_cv <- function(df, type){
  grid <- 10^seq(10, -4, length = 1000)
  accuracy <- c()
  mae <- c()
  auc <- c()
  lambdas <- c()
  for (seed in c(1, 10, 100, 1000, 10000)){
    set.seed(seed)
    folds <- createFolds(df$quality,k=5)
    for (i in 1:5){
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      train_X <- data.matrix(subset(train, select = -c(quality)))
      train_Y <- data.matrix(train$quality)
      test_X <- data.matrix(subset(test, select = -c(quality)))
      test_Y <- data.matrix(test$quality)
      cv <- cv.glmnet(train_X, train_Y, alpha = type, lambda = grid, type.measure="mae")
      bestlam <- cv$lambda.min
      lambdas <- c(lambdas, bestlam)
      model <- glmnet(train_X, train_Y, alpha = type, lambda = bestlam)
      pred <- predict(model, s=bestlam, newx = test_X)
      pred <- as.vector(pred)
      accuracy <- c(accuracy, mean(round(pred) == test$quality))
      mae <- c(mae, mean(abs(test$quality-pred)))
      auc <- c(auc, multiclass.roc(test$quality, round(pred))$auc)
  }
}
  print(c(mean(accuracy), mean(auc), mean(mae)))
  print(lambdas)
}
lasso_ridge_cv(red_df, 1)
```

```
## [1] 0.5925 0.7728 0.5038
##  [1] 0.0001000 0.0001000 0.0001623 0.0001000 0.0001000 0.0002169 0.0001000
##  [8] 0.0001000 0.0001000 0.0001000 0.0001000 0.0001000 0.0001000 0.0007394
## [15] 0.0001000 0.0001000 0.0022875 0.0001000 0.0001000 0.0001676 0.0001000
## [22] 0.0001000 0.0001000 0.0001253 0.0019467
```

It is basically a MLR because the lambdas chosen by CV were very small.

Ridge regression

```
lasso_ridge_cv(red_df, 0)
```

```
## [1] 0.5926 0.7728 0.5038
##  [1] 0.000100 0.000100 0.000100 0.000100 0.000100 0.000100 0.000100 0.000100
##  [9] 0.000100 0.000100 0.000100 0.000100 0.000100 0.006852 0.000100 0.000100
## [17] 0.013065 0.000100 0.000100 0.002520 0.000100 0.000100 0.000100 0.000100
## [25] 0.000100
```

PCR

```
pcr_cv <- function(df){
  accuracy_pcr <- c()
  mae_pcr <- c()
  auc_pcr <- c()
  n_comp <- c()
  for (seed in c(1, 10, 100, 1000, 10000)){
    set.seed(seed)
    folds <- createFolds(df$quality,k=5)
    for (i in 1:5){
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      model <- pcr(quality~., data = train, scale = T, validation = "CV")
      cv_mse <- MSEP(model) # select number of components based on MSEP
      best_ncomp <- which.min(cv_mse$val[1, , ]) -1
      n_comp <- c(n_comp, best_ncomp)
      # validationplot(model, val.type = "MSEP")
      pred <- predict(model, subset(test, select = -quality), ncomp = best_ncomp)
      accuracy_pcr <- c(accuracy_pcr, mean(round(pred) == test$quality))
      mae_pcr <- c(mae_pcr, mean(abs(test$quality-pred)))
      auc_pcr <- c(auc_pcr, multiclass.roc(test$quality, round(pred))$auc)
  }
}
  print(c(mean(accuracy_pcr), mean(auc_pcr), mean(mae_pcr)))
  print(n_comp)
}
pcr_cv(red_df)
```

```
## [1] 0.5913 0.7722 0.5037
## 10 comps 10 comps 10 comps 10 comps  9 comps 10 comps  9 comps 10 comps
##       10       10       10       10        9       10        9       10
## 10 comps 10 comps 10 comps 10 comps 10 comps 10 comps 10 comps 10 comps
##       10       10       10       10       10       10       10       10
## 10 comps  9 comps 10 comps 10 comps 10 comps 10 comps 10 comps 10 comps
##       10        9       10       10       10       10       10       10
## 11 comps
##       11
```

Classification tree

```
trees_cv <- function(df) {
  accuracy_trees <- c()
  mae_trees <- c()
  auc_trees <- c()
  for (seed in c(1, 10, 100, 1000, 10000)) {
    set.seed(seed)
    folds <- createFolds(df$quality, k=5)
    for (i in 1:5) {
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      train$quality <- factor(train$quality)
      model <- tree(quality~., train)
      pred <- as.numeric(as.character(predict(model, test, type="class")))
      accuracy_trees <- c(accuracy_trees, mean(pred == test$quality))
      mae_trees <- c(mae_trees, mean(abs(test$quality - pred)))
      auc_trees <- c(auc_trees, multiclass.roc(test$quality, pred)$auc)
    }
  }
  return(c(mean(accuracy_trees), mean(auc_trees), mean(mae_trees)))
}
trees_cv(red_df)
```

```
## [1] 0.5661 0.7580 0.4764
```

Random Forest

```
rf_cv <- function(df) {
  accuracy_rf <- c()
  mae_rf <- c()
  auc_rf <- c()
  for (seed in c(1, 10, 100, 1000, 10000)) {
    set.seed(seed)
    folds <- createFolds(df$quality, k=5)
    for (i in 1:5) {
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      train$quality <- factor(train$quality)
      model <- randomForest(quality~., train, mtry = 3, importance=TRUE)
      pred <- as.numeric(as.character(predict(model, test)))
      accuracy_rf <- c(accuracy_rf, mean(pred == test$quality))
      mae_rf <- c(mae_rf, mean(abs(test$quality - pred)))
      auc_rf <- c(auc_rf, multiclass.roc(test$quality, pred)$auc)
    }
  }
  return(c(mean(accuracy_rf), mean(auc_rf), mean(mae_rf)))
}
rf_cv(red_df)
```

```
## [1] 0.6953 0.7933 0.3376
```

```
rf_cv <- function(df) {
  accuracy_rf <- c()
  mae_rf <- c()
  auc_rf <- c()
  for (seed in c(1, 10, 100, 1000, 10000)) {
    set.seed(seed)
    folds <- createFolds(df$quality, k=5)
    for (i in 1:5) {
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      train$quality <- factor(train$quality)
      model <- randomForest(quality~., train, mtry = 4, importance=TRUE)
      pred <- as.numeric(as.character(predict(model, test)))
      accuracy_rf <- c(accuracy_rf, mean(pred == test$quality))
      mae_rf <- c(mae_rf, mean(abs(test$quality - pred)))
      auc_rf <- c(auc_rf, multiclass.roc(test$quality, pred)$auc)
    }
  }
  return(c(mean(accuracy_rf), mean(auc_rf), mean(mae_rf)))
}
rf_cv(red_df)
```

```
## [1] 0.6947 0.7924 0.3386
```

Bagging

```
bag_cv <- function(df) {
  accuracy_bag <- c()
  mae_bag <- c()
  auc_bag <- c()
  for (seed in c(1, 10, 100, 1000, 10000)) {
    set.seed(seed)
    folds <- createFolds(df$quality, k=5)
    for (i in 1:5) {
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      train$quality <- factor(train$quality)
      model <- randomForest(quality~., train, mtry=11, importance=T)
      pred <- as.numeric(as.character(predict(model, test)))
      accuracy_bag <- c(accuracy_bag, mean(pred == test$quality))
      mae_bag <- c(mae_bag, mean(abs(test$quality - pred)))
      auc_bag <- c(auc_bag, multiclass.roc(test$quality, pred)$auc)
    }
  }

  return(c(mean(accuracy_bag), mean(auc_bag), mean(mae_bag)))
}
bag_cv(red_df)
```

```
## [1] 0.6890 0.7897 0.3448
```

Boosting (regression)

```
boost_cv <- function(df) {
  accuracy_boost <- c()
  mae_boost <- c()
  auc_boost <- c()
  for (seed in c(1, 10, 100, 1000, 10000)) {
    set.seed(seed)
    folds <- createFolds(df$quality, k=5)
    for (i in 1:5) {
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      model <- gbm(quality~., train,distribution = "gaussian", n.trees = 5000,
                   interaction.depth = 4, shrinkage = 0.05)
      pred <- predict(model, test)
      accuracy_boost <- c(accuracy_boost, mean(round(pred) == test$quality))
      mae_boost <- c(mae_boost, mean(abs(test$quality - pred)))
      auc_boost <- c(auc_boost, multiclass.roc(test$quality, round(pred))$auc)
    }
  }
  return(c(mean(accuracy_boost), mean(auc_boost), mean(mae_boost)))
}
boost_cv(red_df)
```

```
## [1] 0.6549 0.8019 0.4489
```

knn

```
knn_cv <- function(df) {
  accuracy_knn <- c()
  mae_knn <- c()
  auc_knn <- c()
  for (seed in c(1, 10, 100, 1000, 10000)) {
    set.seed(seed)
    folds <- createFolds(df$quality, k=5)
    for (i in 1:5) {
      train <- df[-folds[[i]],]
      test <- df[folds[[i]],]
      pred <- knn(train, test, train$quality, k = 1)
      pred <- as.numeric(as.character(pred))
      accuracy_knn <- c(accuracy_knn, mean(round(pred) == test$quality))
      mae_knn <- c(mae_knn, mean(abs(as.numeric(test$quality) - pred)))
      auc_knn <- c(auc_knn, multiclass.roc(test$quality, round(pred))$auc)
    }
  }
  return(c(mean(accuracy_knn), mean(auc_knn), mean(mae_knn)))
}
knn_cv(red_df)
```

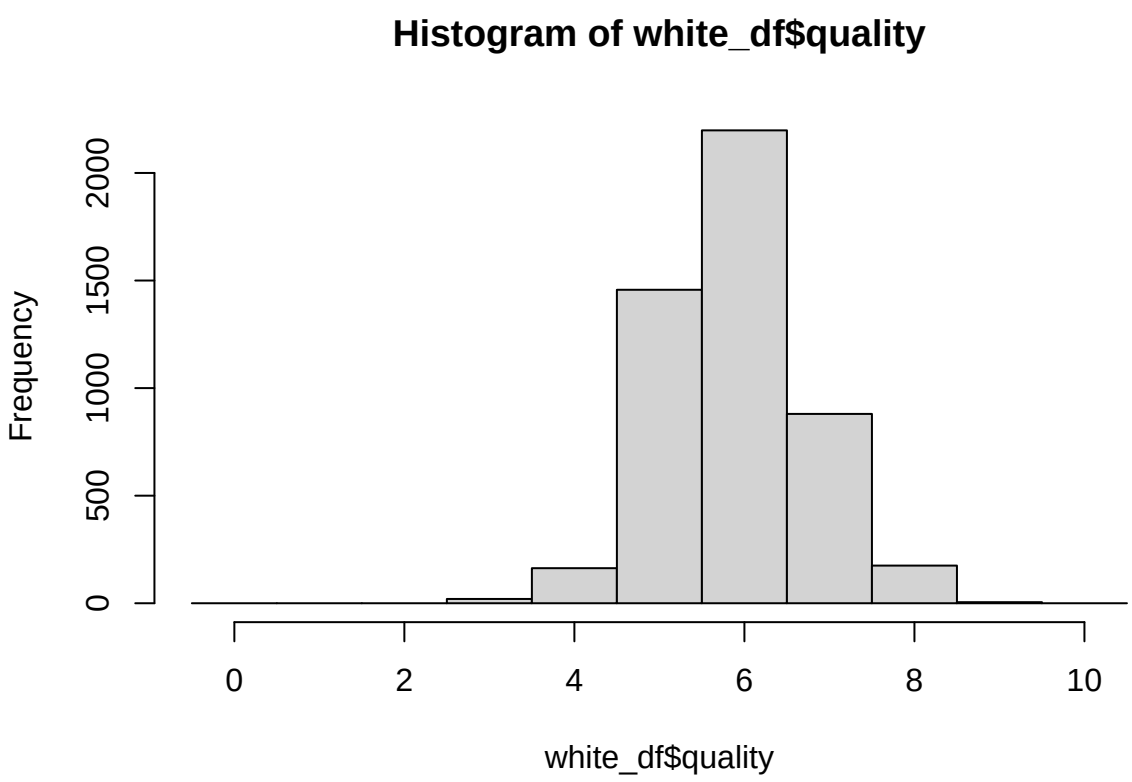
```
## [1] 0.6907 0.8421 0.3345
```

Predict white wine quality

```
summary(white_df)
```

```
##  fixed.acidity   volatile.acidity  citric.acid    residual.sugar
##  Min.   : 3.80   Min.   :0.080    Min.   :0.000   Min.   : 0.60
##  1st Qu.: 6.30   1st Qu.:0.210    1st Qu.:0.270   1st Qu.: 1.70
##  Median : 6.80   Median :0.260    Median :0.320   Median : 5.20
##  Mean   : 6.86   Mean   :0.278    Mean   :0.334   Mean   : 6.39
##  3rd Qu.: 7.30   3rd Qu.:0.320    3rd Qu.:0.390   3rd Qu.: 9.90
##  Max.   :14.20   Max.   :1.100    Max.   :1.660   Max.   :65.80
##    chlorides      free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.0090   Min.   :  2.0       Min.   :  9          Min.   :0.987
##  1st Qu.:0.0360   1st Qu.: 23.0       1st Qu.:108          1st Qu.:0.992
##  Median :0.0430   Median : 34.0       Median :134          Median :0.994
##  Mean   :0.0458   Mean   : 35.3       Mean   :138          Mean   :0.994
##  3rd Qu.:0.0500   3rd Qu.: 46.0       3rd Qu.:167          3rd Qu.:0.996
##  Max.   :0.3460   Max.   :289.0       Max.   :440          Max.   :1.039
##        pH          sulphates       alcohol        quality
##  Min.   :2.72   Min.   :0.22   Min.   : 8.0   Min.   :3.00
##  1st Qu.:3.09   1st Qu.:0.41   1st Qu.: 9.5   1st Qu.:5.00
##  Median :3.18   Median :0.47   Median :10.4   Median :6.00
##  Mean   :3.19   Mean   :0.49   Mean   :10.5   Mean   :5.88
##  3rd Qu.:3.28   3rd Qu.:0.55   3rd Qu.:11.4   3rd Qu.:6.00
##  Max.   :3.82   Max.   :1.08   Max.   :14.2   Max.   :9.00
```

```
hist(white_df$quality, breaks = seq(-0.5, 10.5))
```




```
table(white_df$quality)
```

```
##
##    3    4    5    6    7    8    9
##   20  163 1457 2198  880  175    5
```

MLR

```
mlr_cv(white_df)
```

```
## [1] 0.5189 0.7395 0.5851
```

Best subset

```
best_cv(white_df)
```

```
## [1] 1.0000 0.4835 0.6238 0.6277
## [1] 2.0000 0.5067 0.6499 0.6030
## [1] 3.0000 0.5184 0.6767 0.5958
## [1] 4.0000 0.5155 0.6972 0.5932
```

```
## [1] 5.0000 0.5169 0.7107 0.5907
## [1] 6.0000 0.5168 0.7225 0.5880
## [1] 7.0000 0.5169 0.7197 0.5858
## [1] 8.0000 0.5191 0.7391 0.5851
## [1] 9.0000 0.5200 0.7403 0.5850
## [1] 10.0000  0.5194  0.7401  0.5850
## [1] 11.0000  0.5189  0.7395  0.5851
```

```
summary(regsubsets(quality~., data=white_df, nvmax = 11))$which[9,]
```

```
##          (Intercept)        fixed.acidity     volatile.acidity
##                 TRUE                 TRUE                 TRUE
##          citric.acid       residual.sugar            chlorides
##                FALSE                 TRUE                FALSE
##  free.sulfur.dioxide total.sulfur.dioxide              density
##                 TRUE                 TRUE                 TRUE
##                   pH            sulphates              alcohol
##                 TRUE                 TRUE                 TRUE
```

LDA

```
lda_cv(white_df)
```

```
## [1] 0.5305 0.7505 0.5306
```

QDA

Error

Naive Bayes

```
nb_cv(white_df)
```

```
## [1] 0.4430 0.7354 0.6716
```

Lasso

```
lasso_ridge_cv(white_df, 1)
```

```
## [1] 0.5181 0.7395 0.5851
##  [1] 0.0001000 0.0001000 0.0001907 0.0001000 0.0001000 0.0001000 0.0001000
##  [8] 0.0005185 0.0001000 0.0004136 0.0001000 0.0001969 0.0001000 0.0001000
## [15] 0.0002390 0.0001000 0.0009268 0.0001000 0.0001000 0.0001473 0.0001426
## [22] 0.0001000 0.0001000 0.0001000 0.0001000
```

Ridge

```
lasso_ridge_cv(white_df, 0)
```

```
## [1] 0.5174 0.7391 0.5852
##  [1] 0.0025200 0.0021445 0.0022149 0.0040889 0.0016566 0.0039591 0.0003520
##  [8] 0.0033692 0.0001000 0.0020764 0.0048049 0.0005355 0.0021445 0.0004412
## [15] 0.0032622 0.0016040 0.0043615 0.0016566 0.0001426 0.0068523 0.0056461
## [22] 0.0048049 0.0040889 0.0001000 0.0022149
```

PCR

```
pcr_cv(white_df)
```

```
## [1] 0.5184 0.7349 0.5854
## 11 comps 11 comps 11 comps 11 comps 11 comps 11 comps 11 comps 11 comps 11 comps
##       11       11       11       11       11       11       11       11
## 11 comps 11 comps 10 comps 11 comps 11 comps 10 comps 11 comps 11 comps 11 comps
##       11       11       10       11       11       10       11       11
## 11 comps 11 comps 11 comps 11 comps 11 comps 11 comps 11 comps 11 comps 11 comps
##       11       11       11       11       11       11       11       11
## 11 comps
##       11
```

Classification tree

```
trees_cv(white_df)
```

```
## [1] 0.5070 0.6524 0.5629
```

Random Forest

```
rf_cv(white_df)
```

```
## [1] 0.6887 0.7997 0.3480
```

Bagging

```
bag_cv(white_df)
```

```
## [1] 0.6840 0.8035 0.3522
```

Boosting

```
boost_cv(white_df)
```

```
## [1] 0.6365 0.8031 0.4773
```

knn

```
knn_cv(white_df)
```

```
## [1] 0.6448 0.8235 0.4098
```

Reference

James, Gareth, et al.*An Introduction to Statistical Learning with Applications in R*. 2nd Edition, 2023.