# ML Final

2025-04-28

```r
library(formatR)
knitr::opts_chunk$set(echo = TRUE,
                      error = TRUE,
                      tidy.opts=list(width.cutoff=70),
                      tidy  = TRUE,
                      eval = TRUE, collapse = TRUE)
```

Data Preprocessing

```r
library(readr)
library(xgboost)
## Warning: package 'xgboost' was built under R version 4.3.3
library(nnet)
library(MASS)
library(randomForest)
## Warning: package 'randomForest' was built under R version 4.3.3
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
library(caret)
## Warning: package 'caret' was built under R version 4.3.3
## Loading required package: ggplot2
##
## Attaching package: 'ggplot2'
## The following object is masked from 'package:randomForest':
##
##     margin
## Loading required package: lattice
library(class)
library(gbm)
## Warning: package 'gbm' was built under R version 4.3.3
## Loaded gbm 2.2.2
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
library(dplyr)
##
## Attaching package: 'dplyr'
## The following object is masked from 'package:randomForest':
##
##     combine
## The following object is masked from 'package:MASS':
##
##     select
## The following object is masked from 'package:xgboost':
##
##     slice
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
wine <- read_csv("wine-quality-white-and-red.csv")
## Rows: 6497 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): type
## dbl (12): fixed acidity, volatile acidity, citric acid, residual sugar, chlo...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
p <- ncol(wine) - 1
n <- nrow(wine)
unique(wine$quality)  # This shows that quality takes value from 3 to 9
## [1] 6 5 7 8 4 3 9


wine$type <- trimws(wine$type)  # Remove all the spaces in the column, so I could let R read string and

wine$type <- ifelse(wine$type == "white", TRUE, FALSE)  # Convert the wine type into binary categorical


num_cols <- setdiff(names(wine), c("quality", "type"))
wine[num_cols] <- scale(wine[num_cols])  # These two lines scales the predictors except 'type' into val

names(wine) <- make.names(names(wine))  # This is prepared for random forest
```

Spliting dataset into training and test set.

```
set.seed(1)
train_index <- sample(1:n, size = 0.7 * n)  # randomly pick indexes for training set
wine_train <- wine[train_index, ]
wine_test <- wine[-train_index, ]

sum(wine$type)/n  # 0.7538864, proportion of white in original dataset
## [1] 0.7538864
sum(wine_train$type)/(0.7 * n)  #  0.7500165, proportion of white in train set
## [1] 0.7500165
sum(wine_test$type)/(0.3 * n)  #  0.7629162, proportion of white in test set
## [1] 0.7629162
# These three lines are to make sure that the split is roughly equal
# for white and red, so that there would be a set with white way more
# than red. It turns out the porportion is very close to the orignal
# proportion, which is good!
```

Fit a Logistic Regression Model

```
logit_model <- multinom(quality ~ . - quality, data = wine_train)
## # weights:  98 (78 variable)
## initial  value 8848.053448
## iter  10 value 5679.452802
## iter  20 value 5326.744291
## iter  30 value 4990.782552
## iter  40 value 4875.776360
## iter  50 value 4825.772259
## iter  60 value 4806.322174
## iter  70 value 4799.896646
## iter  80 value 4799.166038
## iter  90 value 4798.450969
## iter 100 value 4797.613877
## final  value 4797.613877
## stopped after 100 iterations
# Cross Validation
set.seed(1)
k <- 5
folds <- sample(1:k, nrow(wine_train), replace = TRUE)
cv_accuracy <- rep(NA, k)
for (j in 1:k) {
    train_fold <- wine_train[folds != j, ]
    valid_fold <- wine_train[folds == j, ]

    model_fold <- multinom(quality ~ . - quality, data = train_fold)

    preds <- predict(model_fold, newdata = valid_fold)

    cv_accuracy[j] <- mean(preds == valid_fold$quality)
}
## # weights:  98 (78 variable)
## initial  value 7092.842493
## iter  10 value 4301.160457
## iter  20 value 4147.158183
## iter  30 value 3948.240719
## iter  40 value 3887.175328
## iter  50 value 3864.162906
## iter  60 value 3859.187694
## iter  70 value 3854.148862
## iter  80 value 3851.480330
## iter  90 value 3849.819962
## iter 100 value 3848.856741
## final  value 3848.856741
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 7063.653841
## iter  10 value 4279.265993
## iter  20 value 4157.956272
## iter  30 value 3931.482015
## iter  40 value 3872.863711
## iter  50 value 3851.439173
## iter  60 value 3845.907668
## iter  70 value 3841.023659
## iter  80 value 3838.409230
## iter  90 value 3837.082502
## iter 100 value 3836.774668
## final  value 3836.774668
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 7137.598427
## iter  10 value 4365.492616
## iter  20 value 4186.619393
## iter  30 value 3955.392119
## iter  40 value 3896.139914
## iter  50 value 3869.131387
## iter  60 value 3861.930036
## iter  70 value 3855.080808
## iter  80 value 3851.843240
## iter  90 value 3850.235797
## iter 100 value 3849.358440
## final  value 3849.358440
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 7071.437482
## iter  10 value 4399.552551
## iter  20 value 4246.226603
## iter  30 value 3928.797466
## iter  40 value 3856.852303
## iter  50 value 3825.234321
## iter  60 value 3819.701778
## iter  70 value 3814.560382
## iter  80 value 3812.627378
## iter  90 value 3811.264652
## iter 100 value 3810.607567
## final  value 3810.607567
## stopped after 100 iterations
## # weights:  84 (65 variable)
## initial  value 6470.043443
## iter  10 value 4228.154512
## iter  20 value 4053.630812
## iter  30 value 3894.645267
## iter  40 value 3834.480505
## iter  50 value 3810.891889
## iter  60 value 3806.019230
## iter  70 value 3804.043296
## final  value 3804.014306
## converged
mean(cv_accuracy)  # 0.5432214
## [1] 0.5432214
```

Fit a Random Forest Model

```
set.seed(1)
k <- 5
folds <- sample(1:k, nrow(wine_train), replace = TRUE)
```

```
cv_accuracy_rf <- rep(NA, k)

for (j in 1:k) {
    train_fold <- wine_train[folds != j, ]
    valid_fold <- wine_train[folds == j, ]

    rf_model <- randomForest(as.factor(quality) ~ . - quality, data = train_fold,
        ntree = 100)

    preds <- predict(rf_model, newdata = valid_fold)

    cv_accuracy_rf[j] <- mean(preds == valid_fold$quality)
}
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!

mean(cv_accuracy_rf)  # 0.6604858
## [1] 0.6604858
```

Fit a Boosting Model

```
set.seed(1)
cv_accuracy_boost <- rep(NA, k)

for (j in 1:k) {
    train_fold <- wine_train[folds != j, ]
    valid_fold <- wine_train[folds == j, ]

    train_fold$type_factor <- as.factor(train_fold$type)
    valid_fold$type_factor <- as.factor(valid_fold$type)

    train_fold$quality_gbm <- as.numeric(as.factor(train_fold$quality)) -
        1
    valid_fold$quality_gbm <- as.numeric(as.factor(valid_fold$quality)) -
        1

    gbm_model <- gbm(quality_gbm ~ . - quality - quality_gbm - type + type_factor,
        data = train_fold, distribution = "multinomial", n.trees = 100,
        interaction.depth = 3, shrinkage = 0.1, n.minobsinnode = 10, verbose = FALSE)

    preds_prob <- predict(gbm_model, newdata = valid_fold, n.trees = 100,
        type = "response")
    preds <- apply(preds_prob, 1, which.max) - 1

    cv_accuracy_boost[j] <- mean(preds == valid_fold$quality_gbm)
}
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.

mean(cv_accuracy_boost)  # 0.5679767
## [1] 0.5679767
```

Fit a LDA Model

```


set.seed(1)
k <- 5
folds <- sample(1:k, nrow(wine_train), replace = TRUE)

cv_accuracy_lda <- rep(NA, k)

for (j in 1:k) {
    train_fold <- wine_train[folds != j, ]
    valid_fold <- wine_train[folds == j, ]

    lda_model <- lda(as.factor(quality) ~ . - quality, data = train_fold)

    preds <- predict(lda_model, newdata = valid_fold)$class

    cv_accuracy_lda[j] <- mean(preds == valid_fold$quality)
}
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
mean(cv_accuracy_lda)  # 0.5298893
## [1] 0.5298893
```

Fit a QDA model

```r
set.seed(1)
k <- 5
folds <- sample(1:k, nrow(wine_train), replace = TRUE)

cv_accuracy_qda <- rep(NA, k)

for (j in 1:k) {
    train_fold <- wine_train[folds != j, ]
    valid_fold <- wine_train[folds == j, ]

    qda_model <- qda(as.factor(quality) ~ . - quality, data = train_fold)

    preds <- predict(qda_model, newdata = valid_fold)$class

    cv_accuracy_qda[j] <- mean(preds == valid_fold$quality)
}
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!
## Error in qda.default(x, grouping, ...): some group is too small for 'qda'

mean(cv_accuracy_qda)  # There is this error indicating some groups is too small for qda.
## [1] NA
```

QDA calculates covariance matrix for every level. There are a lot of levels of qualities, so probably one level appeared very few in a fold. QDA is too sensitive for this project. QDA is forfeit.

Do KNN

```r
set.seed(1)
k <- 5
folds <- sample(1:k, nrow(wine_train), replace = TRUE)

cv_accuracy_knn <- rep(NA, k)

for (j in 1:k) {
    train_fold <- wine_train[folds != j, ]
    valid_fold <- wine_train[folds == j, ]

    train_X <- train_fold[, !(names(train_fold) %in% c("quality", "type"))]
    valid_X <- valid_fold[, !(names(valid_fold) %in% c("quality", "type"))]
    # Type is removed because KNN is based on numeric predictors
    train_y <- as.factor(train_fold$quality)

    preds <- knn(train_X, valid_X, train_y, k = 5)

    cv_accuracy_knn[j] <- mean(preds == valid_fold$quality)
}

mean(cv_accuracy_knn)  # 0.5407512
## [1] 0.5407512
```

For all of the models, Random Forest is apparently the best. Its cv accuracy is significantly higher than the rest. Therefore, RF is the final model.

Fit the final random forest model with the entire training set

```
set.seed(1)
final_rf_model <- randomForest(as.factor(quality) ~ . - quality, data = wine_train,
    ntree = 200)
## Warning in terms.formula(formula, data = data): 'varlist' has changed (from
## nvar=13) to new 14 after EncodeVars() -- should no longer happen!

rf_preds_test <- predict(final_rf_model, newdata = wine_test)
RF_test_accuracy <- mean(rf_preds_test == wine_test$quality)
RF_test_accuracy  # The final RF test accuracy rate is 0.6779487
## [1] 0.6738462
```