Classifying Wines by Data Mining from Physicochemical Properties

Std Nr: 160223367

2/05/2019

Analysis

Intrdocution

We attempt here to build a classifier, using methods of machine learning (ML), which based on 12 physical and chemical measurements of a wine predict if is good or not as good (poor) determined by experts. The dataset to our disposal consists of 5000 observation. The explanatory variables include measurements of, e.g. alcohol, chloride, citric acid concentrations, or of the wine color (white, red). There is also a binary variable quality, determined by an expert assessment, which we will be trying to predict for new observations using the ML model.

In the original publication (Cortez et al. 2009) the white and red wine datasets were modeled separately since, "red and white tastes are quite different." In the original dataset, the quality was an ordinal variable, with ten levels. Therefore, to model, the response regression models were used, and the authors reported the mean absolute deviation as a measure of model performance. For our study, the quality was collapsed into two categories $quality \leq 5$ and $quality > 5$ which we renamed here in "poor" and "good." Hence, we represented the quality as a categorical variable (factor) and use binary classification. To asses the performance of the binary classifiers, we will use a measure suitable for binary classification, namely the ROC curves, as well as well as Cohen's Kappa and Sensitivity and Specificity measures. We also do not split the dataset into white and red wines but build on model for both types of wines and use the wine color as an additional explanatory variable.

Most of the analysis is performed using the `caret` package (Kuhn 2008), which provides a unified interface to many ML algorithms. It also facilitates data preprocessing, and model comparison. A further feature of the `caret` package we utilized is that it facilitates tuning of model hyperparameters.

The Data

Table @ref(tab:datanr) shows that the dataset includes three times more white than red wines. In the original study, the dataset was split into the white and red wines, and separate models were trained for these two types of wine. We run a few preliminary tests, and observed that models build on the subset of data do not perform significantly better than on the entire dataset (results not shown).

We also see that approximately two-thirds of the ratings were positive. That means we can obtain a prediction accuracy of approximately 60% by just always predicting that the wine is good. Because of this, we decided to use the Cohens Kappa measure, which normalizes for the imbalance in the classes, to asses the performance of the ML models.

Table 1: Summary of the dataset

Wine	rating_up_to_5	rating_above_5	nr	percentage
all	36	64	5000	100
white	33	67	3771	75
red	45	55	1229	25

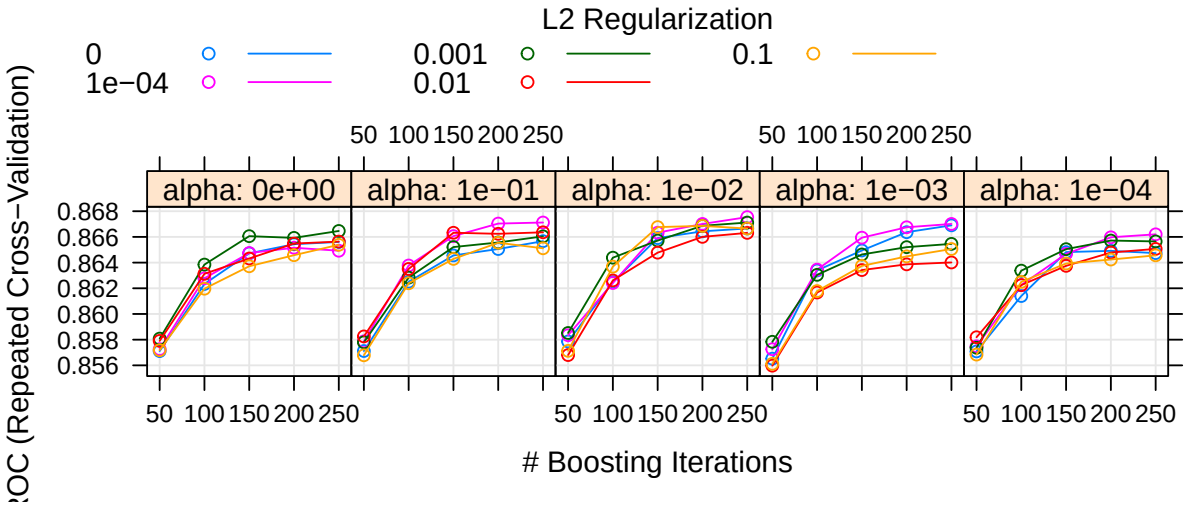

Figure 1: Hyperparamter scan for the xgbLinear model. Y axis AUC - ROC.

Data preprocessing

Transforming and scaling the variables towards normality can improve the performance of ML models. To this task, we used the `preProcess` method of the `caret` package and transformed the data using the BoxCox transformation. The supplementary Table @ref(tab:datasummary) shows the five-number summaries of all variables in the original dataset.

Furthermore, we split the data into a training (80%) and test (20%) dataset. The training dataset will be used to train the models as well as to tune the hyperparameters. We used the test dataset to compare the tuned ML models among each other and select the best ML model.

Hyperparameter tuning

Many ML algorithms have hyperparameters, and their performance will depend on them. Caret uses heuristics to propose a reasonable design to perform the parameter scan. We only needed to specify the number of points on the design grid. To measure how good a design point performance and to compare it with others a validation step is needed. To this task, we use three steps of ten-fold cross-validation. Caret trains the model at each grid points on three folds (90% of the training data) and computes performance summaries on the withhold data. It then computes the average of the scores obtained for the three folds to compare the model performance and select the best. Since we have here a two-class problem, we have used the receiver operator curve (ROC) to compare the different hyperparameters.

The number of hyperparameters for the Machine learning algorithms tested ranges between 0 (e.g. LDA, glm) to up to 7 (mlpKerasDropoutCost) or 8 (xgbDART). Therefore, for some of the models, the training had to be performed at several hundred design points which can be time-consuming. Figure @ref(fig:xgbLinearHyper) shows the results of evaluating the model `xgbLinear` at 125 design points sampling the parameters λ, α, at 5 locations.

The number of hyperparameters for the Machine learning algorithms tested ranges between 0 (e.g. lda, glm) to up to 7 (mlpKerasDropoutCost) or 8 (xgbDART) (see Table @ref(tab:modelSummary)). Therefore, for some of models the training had to be performed at several hundred design points which obviously takes a lot of time.

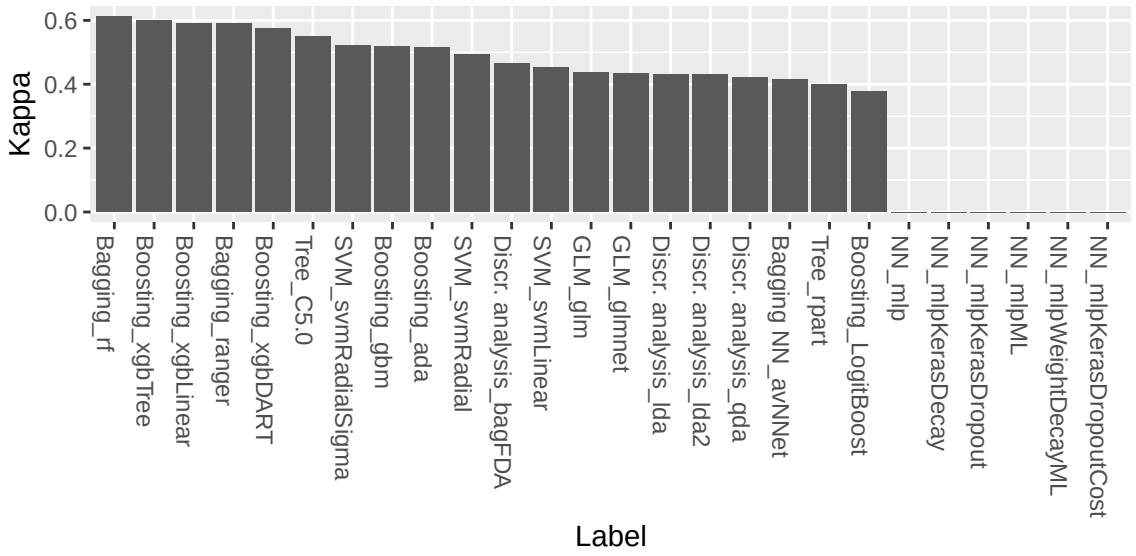

Figure 2: Cohens Kappa values for the algorithms tested using the caret API

Comparing models

After the parameter tuning, using the ten-fold cross-validation, we will be comparing the models using our test dataset (20 of the data). Using the test set, we made predictions from all trained ML models, determined the confusion matrix, and computed various performance measures using the function `confusionMatrix,` which we will use to compare the models.

To compare the models we used measures Cohens Kappa, Sensitivity and Specificity as well as prediction speed. "Cohen's kappa coefficient is a statistic which measures inter-rater agreement for qualitative (categorical) items" (Wikipedia). According to (**???**) a kappa over 0.75 is excellent, 0.40 to 0.75 is fair or reasonable, while below 0.40 it is poor. Figure @ref(fig:figureKapp) shows most of the other algorithms are in the range $0.4 - 0.75$ meaning they are fair or reasonable. Table @ref(tab:modelSummary) shows the results for predicting from the test dataset for all ML algorithms tested.

The neural network algorithms, trained using the `caret` application programming interface (API) have a kappa below 0.4, in fact, it is close to zero. All of those NN algorithms failed to converge. That might be a problem of the `caret` package. However, our direct implementation of a sequential neural network with dropout using the R `Keras` package (see supplement) resulted in aKappa of 0.37, which still is lower than for all the other algorithms. For some network architectures we observed also lack of convergence. We remove all algorithms with a $Kappa < 0.4$ from further comparisons.

Figure @ref(fig:sensspec) shows the specificity and sensitivity of all the algorithms with a Kappa above 0.4. We see that the best performers are those implemented in the package `xgboost`. Also the random forest implementations `rf` and `ranger` show similar performance. The Support Vector Machines, parameterized with various kernels, which performed best in the original publication for regression problems do not not hold their ground here. Linear discriminant analysis (`lda`, `lda2`), or generalized linear regression (`glm` and `glmnet`), show slightly worse performances.

We also compare the prediction speed of the generated models. In practice, ML models are deployed in settings where compute power might be limited while prediction speed is critical (Cars, Security). How much time it takes to learn the model, given today's computer capabilities, is secondary. Figure @ref(fig:predictiontime) shows the prediction time on the x-axis and kappa on the y-axis. We observed that simple models, e.g. `lda`, `glm` predict fast, while random forests e.g. `ranger`, `rf`, are rather slow. Luckily some of the best models xgbTree, xgbLinear, and xgbDART are well implemented and make very fast predictions fast. Hence, for this

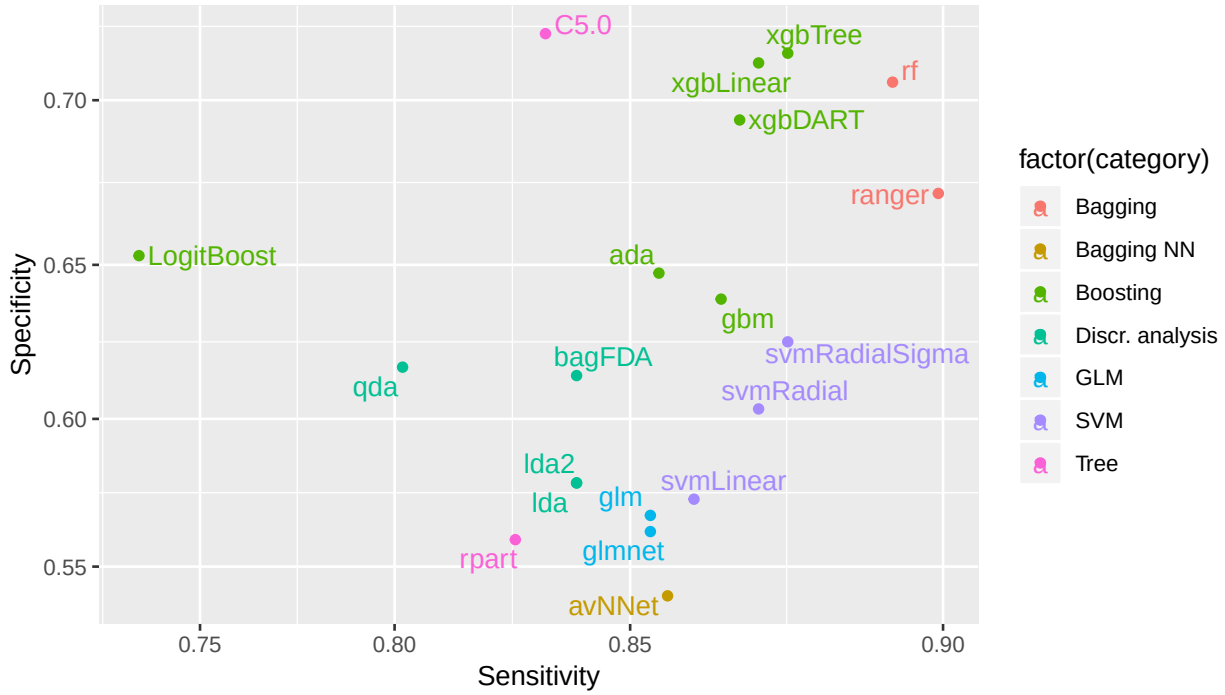

Figure 3: Sensitivity vs Specificity of algorithms with Kappa > 0.4.

reason we prefer them over the random forrest implementations.

Discussion and Conclusion

Caret interfaces dozen of various ML algorithms, and most of those support classification problems. We selected the algorithms based on the lecture notes, i.e., Trees, Trees with bagging (i.e., Random Forrest), Boosting algorithms (i.e., gbm and xgboost), Supported Vector Machines and Neural Networks. We also included also Linear Discriminant Analysis and generalized linear models as benchmarks. Most of the models performed similarly reasonable (Kappa 0.4-0.75) with one exceptions: Neural Networks. Neural Networks performed worse, because we either were not able to determine a suitable network structure (standard implementation) or because the network structure proposed by the `caret` package was not converging. So we conclude, that although NN might perform very well for some type of problems, this type of models are very difficult to tune. It might also be that the training dataset we had is to small to train an NN. We also observe that, the performance of algorithms although in principle similar might differ. For instance, see results obtained with two different implementation of gradient boosting, i.e. `gbm` (Kappa = 0.52) and `xgboost` (Kappa = 0.6). The `caret` package greatly simplified tuning of various ML algorithms. Because of this, the used ML algorithms can be used as a black box. Also, the parameter hyperparameter tuning heuristic might not work optimally, as it is case for the NN.

Our choice of model to predict wine quality is the `xgbTree` algorithm. It not only showed the highest sensitivity and specificity among the tested algorithms, but it also was computationally efficient when predicting.

Deployment

The model consists of the data transformation applied to the inputs as well as the ML model. We created the function `should_I_drink_these_wines` which contains the transformation and ML model all this data

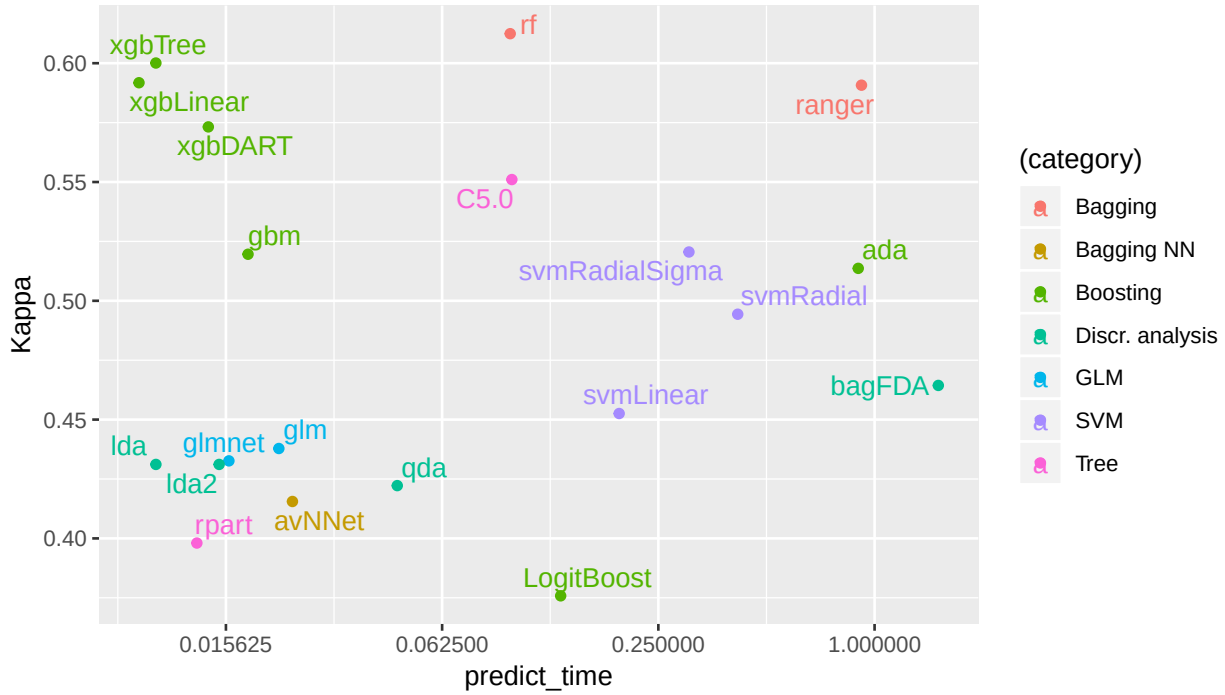

Figure 4: Kappa vs prediction time for 1000 tests cases in seconds

and stored in the file `should_I_drink_these_wines.rds`. In order to make predictions with new data what is needed is to deserialize it and pass a dataframe with the explanatory variables (see code snippet below).

```
library(tidyverse)
library(caret)
library(xgboost)
rm(list=ls())
should_I_drink_these_wines <- readRDS(file = "should_I_drink_these_wines.rds")

data_RW <- read_csv("winetrain.csv") %>% dplyr::select(-X1)

test_sample_X <- sample_n(data_RW,10)
test_sample <- dplyr::select(test_sample_X, -quality)

confusionMatrix(should_I_drink_these_wines(test_sample), as.factor(test_sample_X$quality))
```

Recreating the analysis

Run the following R files from the `R` folder in Sequence:

- `dataSetup.R`
- `testingAllModelsAndSaving.R` (very long running script)
- `analysingResults.R`
- `deployment.R`

The file `modelFunctions.R` contains the `caras` code.

Supplement

Table 2: Data summary.

type	variable	complete	ordered	mean	sd	p0	p25	p50	p75	p100
factor	colour	5000	FALSE	NA	NA	NA	NA	NA	NA	NA
factor	quality	5000	FALSE	NA	NA	NA	NA	NA	NA	NA
numeric	alcohol	5000	NA	10.5	1.2	8	9.5	10.3	11.3	14.9
numeric	chlorides	5000	NA	0.056	0.035	0.009	0.038	0.047	0.065	0.61
numeric	citric.acid	5000	NA	0.32	0.15	0	0.25	0.31	0.39	1.66
numeric	density	5000	NA	0.99	0.003	0.99	0.99	0.99	1	1.04
numeric	fixed.acidity	5000	NA	7.21	1.3	3.8	6.4	7	7.7	15.9
numeric	free.sulfur.dioxide	5000	NA	30.42	17.79	1	17	29	41	289
numeric	pH	5000	NA	3.22	0.16	2.72	3.11	3.21	3.32	4.01
numeric	residual.sugar	5000	NA	5.37	4.72	0.6	1.8	3	8	65.8
numeric	sulphates	5000	NA	0.53	0.15	0.22	0.43	0.51	0.6	2
numeric	total.sulfur.dioxide	5000	NA	115.69	56.64	6	77	118	156	440
numeric	volatile.acidity	5000	NA	0.34	0.17	0.08	0.23	0.29	0.4	1.58

Table 3: ML algorithm summaries. pred time - prediction time in s for 1000 samples

name	description	category	hyperparms	pred_time	Kappa
ada	Boosted Classification Trees	Boosting	2	0.90	0.51
avNNet	Model Averaged Neural Network	Bagging NN	2	0.02	0.42
gbm	Stochastic Gradient Boosting	Boosting	3	0.02	0.52
glm	Generalized Linear Model	GLM	0	0.02	0.44
glmnet	Generalized Linear Model - Lasso	GLM	1	0.02	0.43
lda	Linear Discr. Analysis	Discr. analysis	0	0.01	0.43
lda2	Linear Discr. Analysis	Discr. analysis	0	0.01	0.43
LogitBoost	Boosted Logistic Regression	Boosting	0	0.13	0.38
mlp	Multi-Layer Perceptron	NN	0	0.05	0.00
mlpKerasDecay	Multil. Perc. Net. with Weight Decay	NN	6	1.83	0.00
mlpKerasDropout	Multil. Perc. Network with Dropout	NN	6	1.80	0.00
mlpKerasDropoutCost	Multil. Perc. Network with Dropout	NN	7	4.75	0.00
mlpML	Multil. Perc. Network	NN	2	0.03	0.00
mlpWeightDecayML	Multil. Perc. Network	NN	3	0.04	0.00
qda	Quadratic Discr. Analysis	Discr. analysis	0	0.05	0.42
rf	Random Forest	Bagging	0	0.10	0.61
ranger	Random Forest	Bagging	2	0.92	0.59
rpart	CART	Tree	0	0.01	0.40
svmLinear	SVM	SVM	0	0.19	0.45
svmRadial	SVM	SVM	1	0.42	0.49
svmRadialSigma	SVM	SVM	1	0.30	0.52
xgbDART	Boosted Trees	Boosting	8	0.01	0.57
xgbLinear	Boosted Trees	Boosting	3	0.01	0.59
xgbTree	Boosted Trees	Boosting	6	0.01	0.60
bagFDA	Bagged Flexible Discr. Analysis	Discr. analysis	1	1.50	0.46
C5.0	Tree	Tree	2	0.10	0.55

Data setup code

```
rm(list=ls())
library(tidyverse)
library(readr)
library(factoextra)
library(GGally)
library(caret)

data_RW <- read_csv("winetrain.csv") %>% dplyr::select(-X1)

data_RW$quality <- as.factor(ifelse(data_RW$quality==0,"poor","good"))
data_RW$colour <- as.factor(ifelse(data_RW$colour==0,"white","red"))

xx <- data_RW %>% select_if(is.numeric)
bb <- as.matrix(xx)
prProc <- preProcess( bb , method="BoxCox" )
saveRDS(prProc, file="preProcess_BoxCoxModel.rds")

gg <- predict( prProc , bb)
gg <- data.frame(gg)
data_RW_t <- data.frame(colour = data_RW$colour, quality = data_RW$quality, gg)
data_RW <- data_RW_t
data_RW <- as_tibble(data_RW)

data_W <- data_RW %>% filter(colour=="white") %>% select(-colour)
data_R <- data_RW %>% filter(colour=="red") %>% select(-colour)

trainIndex_RW <- createDataPartition(data_RW$quality , p = .8,
                                     list = FALSE,
                                     times = 1)
trainIndex_W <- createDataPartition(data_W$quality , p = .8,
                                    list = FALSE,
                                    times = 1)
trainIndex_R <- createDataPartition(data_R$quality , p = .8,
                                    list = FALSE,
                                    times = 1)

data_RW_train <- data_RW[trainIndex_RW,]
data_RW_test <- data_RW[-trainIndex_RW,]
data_W_train <- data_W[trainIndex_W,]
data_W_test <- data_W[-trainIndex_W,]
data_R_train <- data_R[trainIndex_R,]
data_R_test <- data_R[-trainIndex_R,]

reslist <- list(data_RW_train = data_RW_train, data_RW_test = data_RW_test,
                data_W_train = data_W_train,
                data_W_test = data_W_test,
                data_R_train = data_R_train,
                data_R_test = data_R_test)
#saveRDS(reslist, file=paste0("AllData_",gsub("[ :]", "_",date()) , ".rds"))
saveRDS(reslist, file="AllData_Mon_Feb_04_12_50_06_2019.rds")
```

Training the models

```
rm(list=ls())
library(caret)
source("modelFunctions.R")

traindata <- readRDS("AllData_Mon_Feb_04_12_50_06_2019.rds")

path <- "results_models_Mon_Feb_04_12_50_06_2019"
dir.create(path)


data_RW_train <- traindata$data_RW_train
data_W_train <- traindata$data_W_train
data_R_train <- traindata$data_R_train


ctrl <- trainControl(method = "repeatedcv",
                     number=10,
                     repeats = 3,
                     allowParallel = TRUE,
                     classProbs = TRUE,
                     savePredictions = TRUE,
                     summaryFunction = twoClassSummary
)

runModelsClassification(data_RW_train,
                        data_W_train,
                        data_R_train,
                        ctrl,
                        path,
                        tuneLength = 5)
```

The function `runModelsClassification` is implemented as follows:

```
run_train <- function(data, trControl, dataLabel = "RW", method = "xgbDART",
                      tuneLength = tuneLength, metric = "ROC", path="."){
  modelFile <- file.path(path,paste0("mod_fit_",method,"_",dataLabel,".rds"))
  if(!file.exists(modelFile)){
    print(paste0("create:", modelFile))
    set.seed(1234)
    mod_fit <- caret::train(quality ~ .,  data=data, method=method,
                            tuneLength = tuneLength, trControl = trControl , metric = metric)
    saveRDS(mod_fit, file=modelFile)
    return(mod_fit)
  }
  return(NULL)
}


runModelsClassification <- function(data_RW_train,data_W_train,data_R_train,ctrl,path, tuneLength = 5) {

  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "glm", tuneLength = tuneLength, path=path)
```

```
mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "glmnet", tuneLength = tuneLength, path=path)

mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "lda", tuneLength = tuneLength, path=path)

mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "lda2", tuneLength = tuneLength, path=path)

if(Sys.info()["nodename"] != "DESKTOP-45T6438"){
  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "bagFDA", tuneLength = tuneLength, path=path)
}
if(FALSE){
  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "randomGLM", tuneLength = tuneLength, path=path)
}

mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "qda", tuneLength = tuneLength, path=path)

mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "rpart", tuneLength = tuneLength, path=path)

mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "ranger", tuneLength = tuneLength, path=path)

mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "C5.0", tuneLength = tuneLength, path=path)


mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "rf", tuneLength = tuneLength, path=path)


mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "gbm", tuneLength = tuneLength, path=path)

mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "xgbLinear", tuneLength = tuneLength, path=path)

# xgbDART ----
mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "xgbDART", tuneLength = tuneLength, path=path)

# xgbTree ----
mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "xgbTree", tuneLength = tuneLength, path=path)

# LogitBoost ----
mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                     method = "LogitBoost", tuneLength = tuneLength, path=path)
```

```
  # ADA -----
  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "ada", tuneLength = tuneLength, path=path)


  # SVM ----
  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "svmLinear", tuneLength = tuneLength, path=path)



  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "svmRadial", tuneLength = tuneLength, path=path)
  # SVMRadialSigma ----
  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "svmRadialSigma", tuneLength = tuneLength, path=path)


  # Model Averaged Neural Network ----

  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "avNNet", tuneLength = tuneLength, path=path)

  # Multi-Layer Perceptron ----

  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "mlp", tuneLength = tuneLength, path=path)

  # Multi-Layer Perceptron, multiple layers ----

  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "mlpWeightDecayML", tuneLength = tuneLength, path=path)

  # Multi-Layer Perceptron, with multiple layers -----
  mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                       method = "mlpML", tuneLength = tuneLength, path=path)

  if(FALSE){
    mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                         method = "mlpKerasDropout", tuneLength = tuneLength, path=path)
    mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
                         method = "mlpKerasDropoutCost", tuneLength = tuneLength, path=path)

    mod_fig <- run_train(data_RW_train, ctrl, dataLabel = "RW",
  }
}
```

R - keras NN implementation

```
rm(list=ls())
library(caret)
library(keras)
library(dplyr)

traindata <- readRDS("AllData_Mon_Feb_04_12_50_06_2019.rds")

data_RW_train <- traindata$data_RW_train
data_RW_train <-(data_RW_train %>% mutate(colour = ifelse(colour == "white", 1, 0)))
data_RW_train <-(data_RW_train %>% mutate(quality = ifelse(quality == "good", 1, 0)))

training <- data_RW_train

data_RW_test <- traindata$data_RW_test
data_RW_test <-(data_RW_test %>% mutate(colour = ifelse(colour == "white", 1, 0)))
data_RW_test <-(data_RW_test %>% mutate(quality = ifelse(quality == "good", 1, 0)))
testing <-data_RW_test

x_train <- training %>% dplyr::select( -quality)%>% as.matrix()
y_train <- to_categorical(training$quality, 2)

x_test <- testing %>% dplyr::select( -quality) %>% as.matrix()
y_test <- to_categorical(testing$quality, 2)


model <- keras_model_sequential()

model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = ncol(x_train)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 2, activation = 'sigmoid')

model %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)

model %>% fit(
  x_train, y_train,
  epochs = 30,
  batch_size = 100,
  validation_split = 0.2
)


keraspred <- model %>% predict_classes(x_test)
confusionMatrix(as.factor(keraspred), as.factor(testing$quality))
```

Comparing the models

```
library(tidyverse)
library(caret)
library(mlbench)


traindata <- readRDS("AllData_Mon_Feb_04_12_50_06_2019.rds")
traindata$data_RW_test

path <- "results_models_Mon_Feb_04_12_50_06_2019/"

allModels <- dir(path)
rwModels <- grep("_RW.rds",allModels,value = TRUE )

models <- list()
confMatrix <- list()
byClass <- list()
overall <- list()
evalTime <- list()
predictions <- list()


for(file in rwModels){
  print(file)
  rfFit <- readRDS(file.path(path, file))
  start_time <- Sys.time()
  predtest <- predict(rfFit, traindata$data_RW_test)
  end_time <- Sys.time()
  evaltime <- end_time - start_time
  dd <- confusionMatrix(predtest, traindata$data_RW_test$quality)
  models[[file]]<-rfFit
  predictions[[file]] <- predtest
  evalTime[[file]] <- evaltime
  confMatrix[[file]] <- dd
  byClass[[file]] <- dd$byClass
  overall[[file]] <- dd$overall
}



designpoints <- sapply(models, function(x){nrow(x$results)})
hyperparamters <- sapply(models, function(x){ncol(x$results)-7})

hyperparamters <- data.frame(ML = names(hyperparamters),
                             nr_hyperparameters = hyperparamters)
designpoints <- data.frame(ML = names(designpoints),nr_designpoints=designpoints)

bclass <- lapply( byClass, function(x){as.data.frame(t(x))} )


bclass <-data.frame( ML=names(bclass) , bind_rows(bclass) )
```

```r
overall <- lapply(overall, function(x){as.data.frame(t(x))})
overall <-data.frame(ML=names(overall),bind_rows(overall))

evalTime <- data.frame(ML = names(evalTime), eval_time = unlist(evalTime))
modelEval <- inner_join(inner_join(evalTime, overall), bclass)

modelEval <- inner_join(modelEval, hyperparamters)

bb<-c("mod_fit_ada_RW.rds",  "Boosted Classification Trees" ,"Boosting",
      "mod_fit_avNNet_RW.rds", "Model Averaged Neural Network", "Bagging NN",
      "mod_fit_gbm_RW.rds", "Stochastic Gradient Boosting", "Boosting",
      "mod_fit_glm_RW.rds", "Generalized Linear Model", "GLM",
      "mod_fit_glmnet_RW.rds", "Generalized Linear Model - Lasso", "GLM",
      "mod_fit_lda_RW.rds", "Linear Discr. Analysis", "Discr. analysis",
      "mod_fit_lda2_RW.rds", "Linear Discr. Analysis", "Discr. analysis",
      "mod_fit_LogitBoost_RW.rds", "Boosted Logistic Regression", "Boosting",
      "mod_fit_mlp_RW.rds", "Multi-Layer Perceptron", "NN",
      "mod_fit_mlpKerasDecay_RW.rds", "Multilayer Perceptron Network with Weight Decay", "NN",
      "mod_fit_mlpKerasDropout_RW.rds", "Multilayer Perceptron Network with Dropout", "NN",
      "mod_fit_mlpKerasDropoutCost_RW.rds", "Multilayer Perceptron Network with Dropout", "NN",
      "mod_fit_mlpML_RW.rds", "Multi-Layer Perceptron, with multiple layers", "NN",
      "mod_fit_mlpWeightDecayML_RW.rds", "Multi-Layer Perceptron, multiple layers", "NN",
      "mod_fit_qda_RW.rds", "Quadratic Discr. Analysis", "Discr. analysis",
      "mod_fit_rf_RW.rds","Random Forest","Bagging",
      "mod_fit_ranger_RW.rds","Random Forest","Bagging",
      "mod_fit_rpart_RW.rds", "CART","Tree",
      "mod_fit_svmLinear_RW.rds", "SVM","SVM",
      "mod_fit_svmRadial_RW.rds", "SVM","SVM",
      "mod_fit_svmRadialSigma_RW.rds", "SVM","SVM",
      "mod_fit_xgbDART_RW.rds", "Boosted Trees","Boosting",
      "mod_fit_xgbLinear_RW.rds","Boosted Trees","Boosting",
      "mod_fit_xgbTree_RW.rds", "Boosted Trees","Boosting",
      "mod_fit_bagFDA_RW.rds","Bagged Flexible Discr. Analysis","Discr. analysis",
      "mod_fit_C5.0_RW.rds","Tree","Tree"
)

bb <- data.frame(matrix(bb, ncol=3, byrow = T))
colnames(bb) <- c("ML", "name", "category")

modelEval <- full_join(bb, modelEval)
modelEval$ML2 <- gsub("mod_fit_","",gsub("_RW.rds","",modelEval$ML))

modelSummary <- modelEval %>%
  dplyr::select(ML2, name, category,
                nr_hyp = nr_hyperparameters,
                predict_time = eval_time,
                Kappa, Sensitivity,
                Specificity, Balanced.Accuracy)

write_csv(modelSummary,path = "modelSummary.txt")
```

Creating deployment

```
make_should_I_drink_these_wines <- function(){
  prProc <- readRDS(prProc, file="preProcess_BoxCoxModel.rds")
  bestModel <- readRDS("results_models_Mon_Feb_04_12_50_06_2019/mod_fit_xgbTree_RW.rds")

  preprocessdata <- function(data_RW){
    data_RW$colour <- as.factor(ifelse(data_RW$colour==0,"white","red"))
    xx <- data_RW %>% select_if(is.numeric)
    bb <- as.matrix(xx)
    gg <- predict( prProc , bb)
    gg <- data.frame(gg)

    data_RW_t <- data.frame(colour = data_RW$colour,  gg)
    return( as_tibble(data_RW_t) )
  }

  predict_mm <- function(data_RW){
    xx <- preprocessdata(data_RW)
    bb <- predict(bestModel,xx)
    bb <- as.factor(ifelse(bb == "good" ,1, 0))
    return(bb)
  }
  return(predict_mm)
}

should_I_drink_these_wines<-make_should_I_drink_these_wines()
saveRDS(should_I_drink_these_wines, file = "should_I_drink_these_wines.rds")
```

References

Cortez, Paulo, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. "Modeling Wine Preferences by Data Mining from Physicochemical Properties." *Decision Support Systems* 47 (4): 547–53.

Kuhn, Max. 2008. "Building Predictive Models in R Using the Caret Package." *Journal of Statistical Software, Articles* 28 (5): 1–26. https://doi.org/10.18637/jss.v028.i05.