NIST Classification

Bhanu Angam

5/24/2022

Table of Contents

## Data Pre-processing

df1 <- read.csv('data/NISTDB4-F.csv', header=FALSE)  
df1$quality <- 'F'  
  
df2 <- read.csv('data/NISTDB4-S.csv', header=FALSE)  
df2$quality <- 'S'  
  
tot <- rbind(df1,df2)  
tot[1:6,1:6]

## V1 V2 V3 V4 V5 V6  
## 1 0.176250 0.169871 0.157203 0.209183 0.287494 0.907002  
## 2 0.139261 0.162739 0.210994 0.234681 0.252325 0.962001  
## 3 0.129169 0.173096 0.147397 0.288801 0.261536 0.000000  
## 4 0.158882 0.169906 0.157303 0.223310 0.290599 0.000000  
## 5 0.112672 0.160529 0.165923 0.290734 0.270142 0.599817  
## 6 0.121959 0.160766 0.167344 0.261005 0.288926 1.077620

### Check for NA

colnames(tot)[ncol(tot)-1] <- 'label'  
tot$quality <- as.factor(tot$quality)  
tot$label <- as.factor(tot$label)  
table(tot$quality, tot$label)

##   
## A L R T W  
## F 380 378 373 123 396  
## S 380 378 373 123 396

map\_chr(tot, typeof) %>%   
 tibble() %>%   
 table()

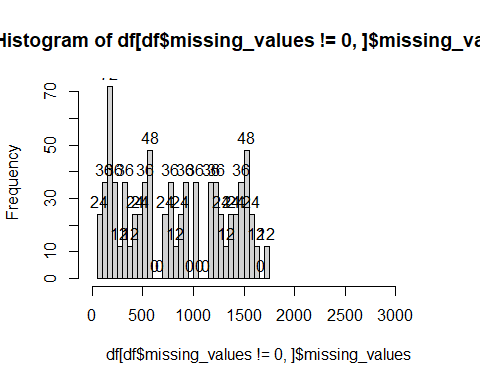
## .  
## double integer   
## 1830 186

sum(is.na(tot))

## [1] 666600

#### Missing Values Treatment

df <- modify(tot[,1:(ncol(tot)-2)], is.na) %>%   
 colSums() %>%  
 tibble(names = colnames(tot[,1:(ncol(tot)-2)]),missing\_values=.) %>%   
 arrange(-missing\_values)  
  
hist(df[df$missing\_values != 0,]$missing\_values,labels = TRUE, xlim = c(0,3300), breaks = 50)



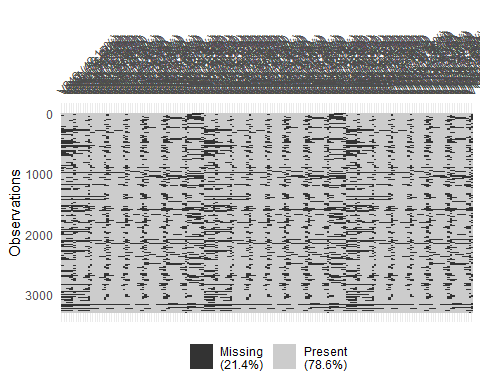
# Removing columns with 50% of total rows having missing values  
names <- modify(tot[,1:(ncol(tot)-2)], is.na) %>%   
 colSums() %>%  
 tibble(names = colnames(tot[,1:(ncol(tot)-2)]), missing\_values=.) %>%   
 filter(missing\_values < 1500) %>%   
 dplyr::select(1)  
  
tot <- tot[c(names$names, 'quality', 'label')]  
  
  
## Imputing remaining predictors with less than 1500 missing values  
library(naniar)

##   
## Attaching package: 'naniar'

## The following object is masked from 'package:skimr':  
##   
## n\_complete

vis\_miss(tot[,846:1019])

## Warning: `gather\_()` was deprecated in tidyr 1.2.0.  
## Please use `gather()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was generated.

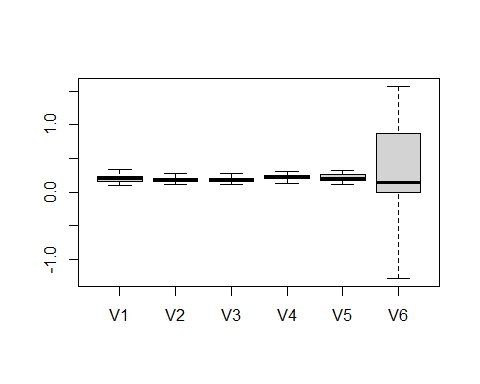


tot <- tot %>%   
 mutate\_if(is.numeric, function(x) ifelse(is.na(x), median(x, na.rm = T), x))  
sum(is.na(tot))

## [1] 0

### Remove Outliers

cap <- function(x){  
 quantiles <- quantile( x, c(.05, 0.25, 0.75, .95 ) )  
 x[ x < quantiles[2] - 1.5\*IQR(x) ] <- quantiles[1]  
 x[ x > quantiles[3] + 1.5\*IQR(x) ] <- quantiles[4]  
 x}  
  
tot <- tot %>% mutate\_if(is.numeric, cap)  
  
boxplot(tot[, 1:6])



### Zero Varinace columns

Finding if columns have Zero Variance that gives NAs while scaling

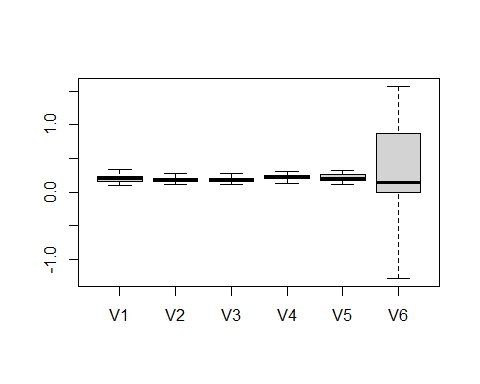
##### Finding if columns have Zero Variance that gives NAs while scaling  
x <- cbind(lapply(tot[,1:(ncol(tot)-2)], FUN = var, na.rm = T))  
  
vardf <- data.frame('col' = rownames(x), 'variation' = unlist(x))  
vardf$col[round(vardf$variation, 5) == 0.0000]

## [1] "V1667" "V1668" "V1669" "V1670" "V1671" "V1672" "V1673" "V1674" "V1675"  
## [10] "V1676" "V1677" "V1678" "V1679" "V1680" "V1681" "V1682" "V1683" "V1684"  
## [19] "V1685" "V1686" "V1687" "V1688" "V1694" "V1695" "V1696" "V1697" "V1698"  
## [28] "V1699" "V1700" "V1701" "V1702" "V1703" "V1704" "V1705" "V1706" "V1707"  
## [37] "V1708" "V1709" "V1710" "V1711" "V1712" "V1713" "V1714" "V1715" "V1859"

zero\_var <- vardf[order(vardf$variation),]  
# str(tot$V1312)  
  
# remove columns with zero variance  
quality <- tot$quality  
label <- tot$label  
tot <- tot[,!(round(vardf$variation, 5) == 0.0000)]  
tot$label <- label  
tot$quality <- quality  
rm(quality)  
rm(label)  
dim(tot)

## [1] 3300 1875

boxplot(tot[, 1:6])



### Class Balancing

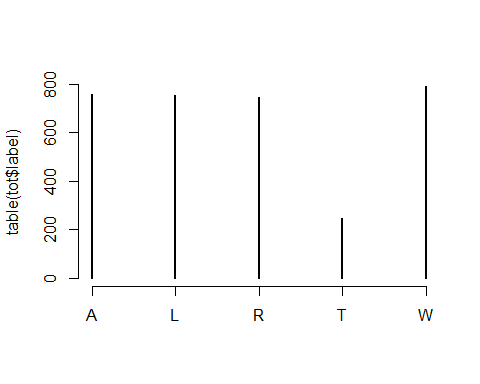
table(tot$label)

##   
## A L R T W   
## 760 756 746 246 792

prop.table(table(tot$label))

##   
## A L R T W   
## 0.23030303 0.22909091 0.22606061 0.07454545 0.24000000

plot(table(tot$label), type="h")



tot\_f <- tot[tot$quality == 'F',]  
tot\_s <- tot[tot$quality == 'S',]  
  
smoted\_fT <- oversample\_smote((tot\_f %>% dplyr::select(-'quality')), "T", "label", 150)  
smoted\_sT <- oversample\_smote((tot\_s %>% dplyr::select(-'quality')), "T", "label", 150)  
  
smoted\_fT$quality = 'F'  
smoted\_sT$quality = 'S'  
table(smoted\_fT$label)

##   
## T   
## 150

table(smoted\_sT$label)

##   
## T   
## 150

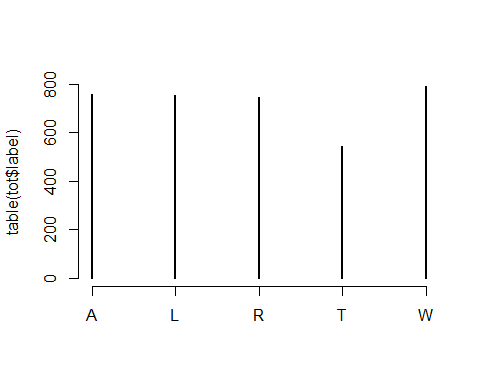
tot <- rbind(tot, smoted\_fT, smoted\_sT)  
  
table(tot$label)

##   
## A L R T W   
## 760 756 746 546 792

prop.table(table(tot$label))

##   
## A L R T W   
## 0.2111111 0.2100000 0.2072222 0.1516667 0.2200000

plot(table(tot$label), type="h")



## Glimpse first 10 columns  
head(tot[, 1:10], n = 14)

## V1 V2 V3 V4 V5 V6 V7 V8  
## 1 0.176250 0.169871 0.157203 0.209183 0.287494 0.907002 0.9343530 0.9736690  
## 2 0.139261 0.162739 0.210994 0.234681 0.252325 0.962001 0.9017870 0.8352160  
## 3 0.129169 0.173096 0.147397 0.288801 0.261536 0.000000 0.0000000 1.5429700  
## 4 0.158882 0.169906 0.157303 0.223310 0.290599 0.000000 0.0000000 0.0000000  
## 5 0.112672 0.160529 0.165923 0.290734 0.270142 0.599817 0.5549920 0.5275150  
## 6 0.121959 0.160766 0.167344 0.261005 0.288926 1.077620 0.8099300 0.7829870  
## 7 0.220777 0.183854 0.166993 0.198650 0.229726 0.202632 1.0379700 1.0908900  
## 8 0.136199 0.149354 0.177415 0.246785 0.290247 0.923321 0.7973780 0.6491520  
## 9 0.162786 0.149191 0.209091 0.218244 0.260688 1.124200 1.0705400 0.9868830  
## 10 0.142169 0.168210 0.177313 0.257223 0.255084 0.250729 0.1997890 0.1675730  
## 11 0.137683 0.171983 0.161767 0.243440 0.285127 1.568070 -0.2356836 -0.1469033  
## 12 0.165984 0.189287 0.158808 0.213524 0.272397 1.136500 1.1282600 1.1157100  
## 13 0.146466 0.156443 0.199567 0.225695 0.271829 0.867468 0.8514730 0.8227150  
## 14 0.148994 0.196293 0.186888 0.204488 0.263337 0.133917 0.1582920 0.1513280  
## V9 V10  
## 1 1.00229000 0.99892500  
## 2 0.80259900 0.80839800  
## 3 0.76367200 0.17376600  
## 4 0.00000000 0.00000000  
## 5 0.48682100 0.46142700  
## 6 0.74598600 0.71305900  
## 7 1.04437000 1.04088000  
## 8 0.61425900 0.63339700  
## 9 0.85081200 0.83263800  
## 10 0.25196400 0.51510800  
## 11 -0.07520578 -0.02597753  
## 12 1.07574000 1.05620000  
## 13 0.79631000 0.78283500  
## 14 0.09605270 0.07050630

dim(tot)

## [1] 3600 1875

sum(is.na(tot))

## [1] 0

# saving the pre processed NIST data   
# saveRDS(tot, 'processed\_NIST.rds')  
tot <- readRDS('processed\_NIST.rds')

## Train Test Splits

set.seed(2022)  
# Split data 70%-30% into training set and test set  
tot\_split <- as\_tibble(tot) %>%  
 mutate\_if(is.numeric, scale) %>%  
 initial\_split(prop = 0.70, strata = label)  
  
# Extract data in each split  
tot\_train <- training(tot\_split)  
tot\_test <- testing(tot\_split) %>% dplyr::select(-label)  
ytest <- testing(tot\_split)$label  
  
tot\_folds <- vfold\_cv(training(tot\_split), v = 5, strata = label)  
# Print the number of observations in each split  
cat("Training cases: ", nrow(tot\_train), "\n",  
 "Test cases: ", nrow(tot\_test), sep = "")

## Training cases: 2519  
## Test cases: 1081

## KNN with full data

### Bayesian Optimization for Hyper parameter Tuning

### Model Specification

knn\_rec <- recipe(label ~ ., data=tot\_train) %>%  
 step\_normalize(all\_numeric\_predictors)  
  
knn\_mod <- nearest\_neighbor(neighbors = tune(), weight\_func = tune()) %>%   
 set\_engine("kknn") %>%   
 set\_mode("classification")  
  
knn\_wflow <- workflow() %>%   
 add\_model(knn\_mod) %>%   
 add\_formula(label ~ .)  
  
knn\_param <- knn\_wflow %>%   
 parameters() %>%   
 update(  
 neighbors = neighbors(c(3, 50)),  
 weight\_func = weight\_func(values = c("rectangular", "gaussian", "triangular"))  
 )  
  
library(doParallel)  
all\_cores <- detectCores(logical = FALSE)  
cls <- makePSOCKcluster(all\_cores)  
registerDoParallel(cls)  
ctrl <- control\_bayes(verbose = TRUE)  
set.seed(2022)  
# Hyper parameter tuning by bayesian optimization  
knn\_search\_full <- tune\_bayes(knn\_wflow, resamples = tot\_folds,   
 initial = 5, iter = 20,  
 param\_info = knn\_param, control = ctrl)  
  
saveRDS(knn\_search\_full, 'models/NIST/knn\_tuned\_full.rds')  
knn\_tuned\_full <- readRDS('models/NIST/knn\_tuned\_full.rds')

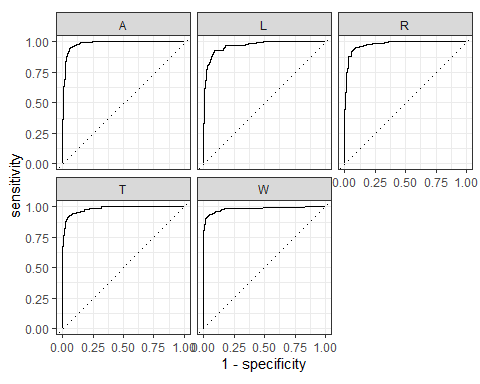
knn\_final\_full <- readRDS('models/NIST/knn\_final\_full.rds')  
  
pred\_df <- bind\_cols(ytest,  
 predict(knn\_final\_full, tot\_test),  
 predict(knn\_final\_full, tot\_test, type = "prob"))

## New names:  
## • `` -> `...1`

colnames(pred\_df) <- c("obs", "pred","pred\_A","pred\_L", "pred\_R","pred\_T","pred\_W")  
  
saveRDS(pred\_df, 'results/NIST/knn\_full\_pred\_df.rds')  
pred\_df <- readRDS('results/NIST/knn\_full\_pred\_df.rds')  
  
cm\_knn\_full\_NIST <- confusionMatrix(pred\_df$pred, ytest)  
saveRDS(cm\_knn\_full\_NIST, 'results/NIST/cm\_knn\_full\_NIST.rds')  
cm\_knn\_full\_NIST

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A L R T W  
## A 212 18 19 11 8  
## L 0 174 0 0 22  
## R 0 0 190 0 22  
## T 16 35 15 153 8  
## W 0 0 0 0 178  
##   
## Overall Statistics  
##   
## Accuracy : 0.839   
## 95% CI : (0.8158, 0.8605)  
## No Information Rate : 0.2202   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.799   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: L Class: R Class: T Class: W  
## Sensitivity 0.9298 0.7665 0.8482 0.9329 0.7479  
## Specificity 0.9343 0.9742 0.9743 0.9193 1.0000  
## Pos Pred Value 0.7910 0.8878 0.8962 0.6740 1.0000  
## Neg Pred Value 0.9803 0.9401 0.9609 0.9871 0.9336  
## Prevalence 0.2109 0.2100 0.2072 0.1517 0.2202  
## Detection Rate 0.1961 0.1610 0.1758 0.1415 0.1647  
## Detection Prevalence 0.2479 0.1813 0.1961 0.2100 0.1647  
## Balanced Accuracy 0.9321 0.8704 0.9113 0.9261 0.8739

pred\_df %>%  
 roc\_curve(obs, pred\_A:pred\_W) %>%  
 autoplot()



## Boost Tree with full Data

#### model specification

# XGBoost model specification  
xgboost\_model <-   
 parsnip::boost\_tree(  
 mode = "classification",  
 trees = 100,  
 min\_n = tune(),  
 tree\_depth = tune(),  
 learn\_rate = tune(),  
 loss\_reduction = tune()  
 ) %>%  
 set\_engine("xgboost")  
  
# grid specification  
xgboost\_params <-   
 dials::parameters(  
 min\_n(),  
 tree\_depth(),  
 learn\_rate(),  
 loss\_reduction()  
 )  
  
xgboost\_grid <-   
 dials::grid\_max\_entropy(  
 xgboost\_params,   
 size = 4  
 )  
  
head(xgboost\_grid)  
  
xgboost\_wf <- workflow() %>%  
 add\_model(xgboost\_model) %>%   
 add\_formula(label ~ .)  
  
# hyper parameter tuning  
library(doParallel)  
all\_cores <- detectCores(logical = FALSE)  
cls <- makePSOCKcluster(all\_cores)  
registerDoParallel(cls)  
set.seed(234)  
xgboost\_tuned\_full <- tune\_grid(  
 object = xgboost\_wf,  
 resamples = tot\_folds,  
 grid = xgboost\_grid,  
 metrics = metric\_set(roc\_auc, accuracy),  
 control = control\_grid(verbose = TRUE, save\_pred = TRUE)  
)  
saveRDS(xgboost\_tuned\_full, 'models/NIST/xgboost\_tuned\_full.rds')  
xgboost\_tuned\_full <- readRDS('models/NIST/xgboost\_tuned\_full.rds')  
  
xgboost\_tuned\_full %>%  
 collect\_metrics(metric='accuracy') %>%  
 knitr::kable()  
xgboost\_tuned\_full %>%  
 select\_best('accuracy')  
  
xgboost\_best\_param <- xgboost\_tuned\_full %>%  
 select\_best('roc\_auc')  
## fit the model on all the training data  
xgboost\_final\_full <- xgboost\_model %>%  
 finalize\_model(xgboost\_best\_param) %>%  
 # fit the model on all the training data  
 fit( formula = label ~ .,data = tot\_train)  
saveRDS(xgboost\_final\_full, 'models/NIST/xgboost\_final\_full.rds')  
xgboost\_final\_full <- readRDS('models/NIST/xgboost\_final\_full.rds')

#### Testing model on Test data

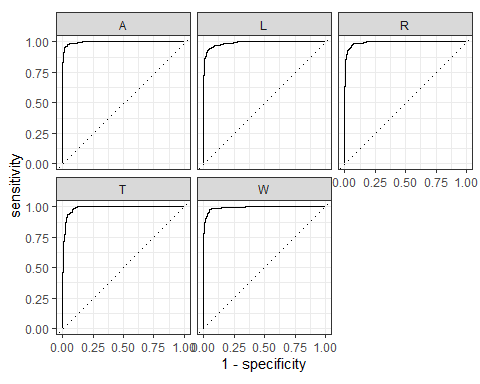
xgboost\_final\_full <- readRDS('models/NIST/xgboost\_final\_full.rds')  
  
pred\_df <- bind\_cols(  
 testing(tot\_split)$label,  
 predict(xgboost\_final\_full, tot\_test),  
 predict(xgboost\_final\_full, tot\_test, type = "prob"))

## New names:  
## • `` -> `...1`

colnames(pred\_df) <- c("obs","pred","pred\_A","pred\_L", "pred\_R","pred\_T","pred\_W")  
saveRDS(pred\_df, 'results/NIST/xgboost\_full\_pred\_df')  
pred\_df <- readRDS('results/NIST/xgboost\_full\_pred\_df')  
  
cm\_xgboost\_full\_NIST <- confusionMatrix(pred\_df$pred, testing(tot\_split)$label)  
cm\_xgboost\_full\_NIST

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A L R T W  
## A 216 2 2 8 1  
## L 3 204 0 7 7  
## R 1 2 207 3 14  
## T 7 13 10 146 2  
## W 1 6 5 0 214  
##   
## Overall Statistics  
##   
## Accuracy : 0.913   
## 95% CI : (0.8946, 0.9292)  
## No Information Rate : 0.2202   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.891   
##   
## Mcnemar's Test P-Value : 0.151   
##   
## Statistics by Class:  
##   
## Class: A Class: L Class: R Class: T Class: W  
## Sensitivity 0.9474 0.8987 0.9241 0.8902 0.8992  
## Specificity 0.9848 0.9801 0.9767 0.9651 0.9858  
## Pos Pred Value 0.9432 0.9231 0.9119 0.8202 0.9469  
## Neg Pred Value 0.9859 0.9733 0.9801 0.9801 0.9719  
## Prevalence 0.2109 0.2100 0.2072 0.1517 0.2202  
## Detection Rate 0.1998 0.1887 0.1915 0.1351 0.1980  
## Detection Prevalence 0.2118 0.2044 0.2100 0.1647 0.2091  
## Balanced Accuracy 0.9661 0.9394 0.9504 0.9277 0.9425

saveRDS(cm\_xgboost\_full\_NIST, 'results/NIST/cm\_xgboost\_full\_NIST.rds')  
pred\_df %>%  
 roc\_curve(obs, pred\_A:pred\_W) %>%  
 autoplot()

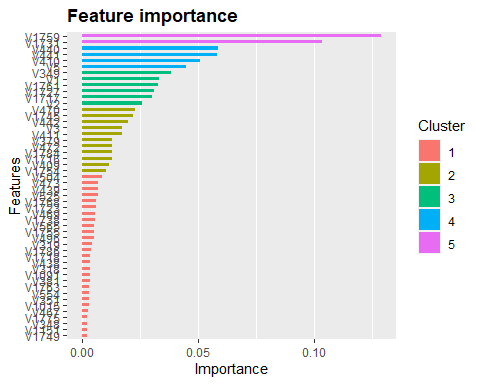


#### Feature Importance

imp\_df = xgb.importance(model=xgboost\_final\_full$fit)  
head(imp\_df$Feature, 10)

## [1] "V1759" "V1731" "V440" "V441" "V410" "V5" "V349" "V1" "V1761"  
## [10] "V1727"

xgb.importance(model=xgboost\_final\_full$fit) %>% xgb.ggplot.importance(top\_n=50, measure=NULL, rel\_to\_first = FALSE)



dim(imp\_df)

## [1] 538 4

saveRDS(imp\_df, 'models/NIST/feature\_imp.rds')  
imp\_df <- readRDS('models/NIST/feature\_imp.rds')

## Selecting Features from Boost Trees

nf <- 300  
dftrain <- training(tot\_split) %>% dplyr::select(imp\_df$Feature[1:nf], label)  
dftest <- testing(tot\_split) %>% dplyr::select(imp\_df$Feature[1:nf])  
ytest <- testing(tot\_split)$label  
df\_folds <- vfold\_cv(training(tot\_split) %>% dplyr::select(imp\_df$Feature[1:nf], label), v = 5, strata = label)  
  
dim(dftrain)

## [1] 2519 301

dim(dftest)

## [1] 1081 300

## Boost Tree with Feature Selection

##### model specification

# XGBoost model specification  
xgboost\_model <- parsnip::boost\_tree(mode = "classification",  
 trees = 100,  
 min\_n = tune(),  
 tree\_depth = tune(),  
 learn\_rate = tune(),  
 loss\_reduction = tune()) %>%  
 set\_engine("xgboost")  
  
# grid specification  
xgboost\_params <- dials::parameters(min\_n(),  
 tree\_depth(),  
 learn\_rate(),  
 loss\_reduction())  
  
xgboost\_grid <- dials::grid\_max\_entropy(  
 xgboost\_params,   
 size = 4)  
  
head(xgboost\_grid)  
  
xgboost\_wf <- workflow() %>%  
 add\_model(xgboost\_model) %>%   
 add\_formula(label ~ .)  
  
# hyper parameter tuning  
library(doParallel)  
all\_cores <- detectCores(logical = FALSE)  
cls <- makePSOCKcluster(all\_cores)  
registerDoParallel(cls)  
set.seed(234)  
## Grid Tune  
xgboost\_tuned\_features <- tune\_grid(  
 object = xgboost\_wf,  
 resamples = df\_folds,  
 grid = xgboost\_grid,  
 metrics = metric\_set(roc\_auc, accuracy),  
 control = control\_grid(verbose = TRUE, save\_pred = TRUE))  
  
saveRDS(xgboost\_tuned\_features, 'models/NIST/xgboost\_tuned\_features.rds')  
xgboost\_tuned\_features <- readRDS('models/NIST/xgboost\_tuned\_features.rds')  
  
xgboost\_tuned\_features %>%  
 collect\_metrics(metric='accuracy') %>%  
 knitr::kable()  
xgboost\_tuned\_features %>%  
 select\_best('accuracy')  
  
xgboost\_best\_param <- xgboost\_tuned\_features %>%  
 select\_best('roc\_auc')  
## fit the model on all the training data  
xgboost\_final\_features <- xgboost\_model %>%  
 finalize\_model(xgboost\_best\_param) %>%  
 # fit the model on all the training data  
 fit( formula = label ~ .,data = dftrain)  
  
saveRDS(xgboost\_final\_features, 'models/NIST/xgboost\_final\_features.rds')  
xgboost\_final\_features <- readRDS('models/NIST/xgboost\_final\_features.rds')

#### Testing model on Test data

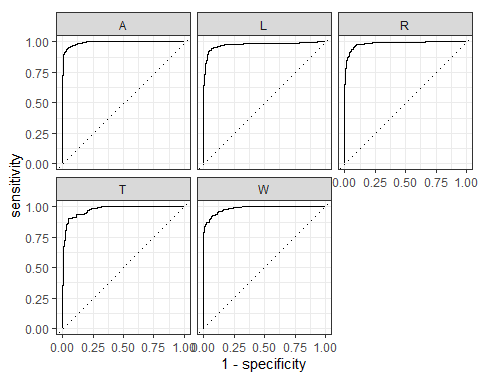
xgboost\_final\_features <- readRDS('models/NIST/xgboost\_final\_features.rds')  
  
pred\_df <- bind\_cols(  
 testing(tot\_split)$label,  
 predict(xgboost\_final\_features, dftest),  
 predict(xgboost\_final\_features, dftest, type = "prob"))

## New names:  
## • `` -> `...1`

colnames(pred\_df) <- c("obs","pred","pred\_A","pred\_L", "pred\_R","pred\_T","pred\_W")  
  
saveRDS(pred\_df, 'results/NIST/xgboost\_features\_pred\_df.rds')  
pred\_df <- readRDS('results/NIST/xgboost\_features\_pred\_df.rds')  
  
cm\_xgboost\_features <- confusionMatrix(pred\_df$pred, testing(tot\_split)$label)  
cm\_xgboost\_features

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A L R T W  
## A 214 5 5 12 1  
## L 5 204 0 12 12  
## R 1 2 202 2 18  
## T 7 10 12 138 3  
## W 1 6 5 0 204  
##   
## Overall Statistics  
##   
## Accuracy : 0.8899   
## 95% CI : (0.8697, 0.908)  
## No Information Rate : 0.2202   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.862   
##   
## Mcnemar's Test P-Value : 0.004233   
##   
## Statistics by Class:  
##   
## Class: A Class: L Class: R Class: T Class: W  
## Sensitivity 0.9386 0.8987 0.9018 0.8415 0.8571  
## Specificity 0.9730 0.9660 0.9732 0.9651 0.9858  
## Pos Pred Value 0.9030 0.8755 0.8978 0.8118 0.9444  
## Neg Pred Value 0.9834 0.9729 0.9743 0.9715 0.9607  
## Prevalence 0.2109 0.2100 0.2072 0.1517 0.2202  
## Detection Rate 0.1980 0.1887 0.1869 0.1277 0.1887  
## Detection Prevalence 0.2192 0.2155 0.2081 0.1573 0.1998  
## Balanced Accuracy 0.9558 0.9324 0.9375 0.9033 0.9215

saveRDS(cm\_xgboost\_features, 'results/NIST/cm\_xgboost\_features.rds')  
pred\_df %>%  
 roc\_curve(obs, pred\_A:pred\_W) %>%  
 autoplot()



## KNN nearest neighbours with Feature Selection

### Bayesian Optimization for Hyper parameter Tuning

##### Model Specification

knn\_rec <- recipe(label ~ ., data=tot\_train) %>%  
 step\_scale(all\_numeric\_predictors)  
  
knn\_mod <- nearest\_neighbor(neighbors = tune(), weight\_func = tune()) %>%   
 set\_engine("kknn") %>%   
 set\_mode("classification")  
  
knn\_wflow <-   
 workflow() %>%   
 add\_model(knn\_mod) %>%   
 add\_formula(label ~ .)  
  
knn\_param <-   
 knn\_wflow %>%   
 parameters() %>%   
 update(  
 neighbors = neighbors(c(3, 50)),  
 weight\_func = weight\_func(values = c("rectangular", "gaussian", "triangular"))  
 )  
  
  
library(doParallel)  
all\_cores <- detectCores(logical = FALSE)  
cls <- makePSOCKcluster(all\_cores)  
registerDoParallel(cls)  
ctrl <- control\_bayes(verbose = TRUE)  
set.seed(2022)  
# Hyper parameter tuning by bayesian optimization  
knn\_search\_features <- tune\_bayes(knn\_wflow, resamples = df\_folds, initial = 5, iter = 20,  
 param\_info = knn\_param, control = ctrl)  
saveRDS(knn\_search\_features, 'models/NIST/knn\_tuned\_features.rds')  
knn\_tuned\_features <- readRDS('models/NIST/knn\_tuned\_features.rds')

knn\_best\_param <- knn\_tuned\_features %>% select\_best('roc\_auc')  
## Fit Full train data with best parameters from tuning  
knn\_final\_features <- knn\_mod %>%  
 finalize\_model(knn\_best\_param) %>%  
 fit( formula = label ~ .,data = dftrain)  
  
saveRDS(knn\_final\_features, 'models/NIST/knn\_final\_features.rds')  
knn\_final\_features <- readRDS('models/NIST/knn\_final\_features.rds')

knn\_final\_features <- readRDS('models/NIST/knn\_final\_features.rds')  
  
pred\_df <- bind\_cols(  
 ytest,  
 predict(knn\_final\_features, dftest),  
 predict(knn\_final\_features, dftest, type = "prob"))

## New names:  
## • `` -> `...1`

colnames(pred\_df) <- c("obs", "pred","pred\_A","pred\_L", "pred\_R","pred\_T","pred\_W")  
saveRDS(pred\_df, 'results/NIST/knn\_features\_pred\_df.rds')  
pred\_df <- readRDS('results/NIST/knn\_features\_pred\_df.rds')  
  
cm\_knn\_features\_NIST <- confusionMatrix(pred\_df$pred, ytest)  
cm\_knn\_features\_NIST

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A L R T W  
## A 223 14 9 10 6  
## L 0 189 0 0 13  
## R 0 0 198 1 22  
## T 5 22 16 153 5  
## W 0 2 1 0 192  
##   
## Overall Statistics  
##   
## Accuracy : 0.8834   
## 95% CI : (0.8628, 0.902)  
## No Information Rate : 0.2202   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8542   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: L Class: R Class: T Class: W  
## Sensitivity 0.9781 0.8326 0.8839 0.9329 0.8067  
## Specificity 0.9543 0.9848 0.9732 0.9477 0.9964  
## Pos Pred Value 0.8511 0.9356 0.8959 0.7612 0.9846  
## Neg Pred Value 0.9939 0.9568 0.9698 0.9875 0.9481  
## Prevalence 0.2109 0.2100 0.2072 0.1517 0.2202  
## Detection Rate 0.2063 0.1748 0.1832 0.1415 0.1776  
## Detection Prevalence 0.2424 0.1869 0.2044 0.1859 0.1804  
## Balanced Accuracy 0.9662 0.9087 0.9285 0.9403 0.9016

saveRDS(cm\_knn\_features\_NIST, 'results/NIST/cm\_knn\_NIST\_features.rds')  
  
pred\_df %>%  
 roc\_curve(obs, pred\_A:pred\_W) %>%  
 autoplot()

