Multi-Class Classification (SFinGe)

Bhanu Angam

03/06/2022

Table of Contents

## Data Pre-processing

df1 <- read.csv('data/SFinGe\_Default.csv', header = FALSE)  
df1$quality <- 'default'  
  
df2 <- read.csv('data/SFinGe\_HQNoPert.csv', header = FALSE)  
df2$quality <- 'hq'  
  
df3 <- read.csv('data/SFinGe\_VQAndPert.csv', header = FALSE)  
df3$quality <- 'vq'  
  
tot <- rbind(df1,df2,df3)  
tot[1:6, 1:6]

## V1 V2 V3 V4 V5 V6  
## 1 0.138715 0.169319 0.172534 0.258365 0.261067 0.000000  
## 2 0.102090 0.185952 0.162513 0.274962 0.274483 0.653367  
## 3 0.136504 0.181319 0.157137 0.259984 0.265055 0.000000  
## 4 0.123671 0.165290 0.165721 0.283043 0.262275 0.000000  
## 5 0.133556 0.185907 0.207980 0.220127 0.252430 0.000000  
## 6 0.125754 0.168781 0.164002 0.263086 0.278377 0.000000

### 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)

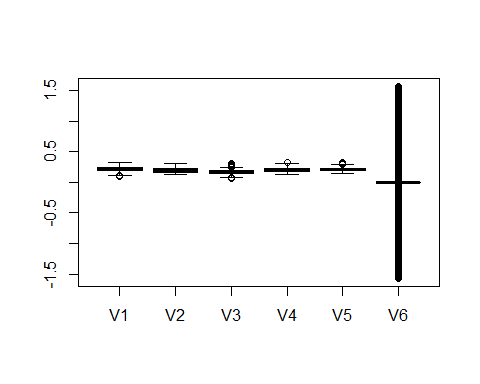
##   
## A L R T W  
## default 370 3380 3170 290 2790  
## hq 370 3380 3170 290 2790  
## vq 370 3380 3170 290 2790

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}  
  
boxplot(tot[, 1:6])



tot <- tot %>% mutate\_if(is.numeric, cap)

### Zero Varinace columns

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,4) == 0.000]

## [1] "V6" "V7" "V8" "V9" "V22" "V23" "V24" "V38" "V39"   
## [10] "V40" "V54" "V55" "V56" "V70" "V71" "V86" "V87" "V102"   
## [19] "V103" "V118" "V119" "V134" "V135" "V150" "V151" "V166" "V167"   
## [28] "V182" "V183" "V198" "V199" "V214" "V215" "V230" "V231" "V232"   
## [37] "V246" "V247" "V248" "V262" "V263" "V264" "V278" "V279" "V280"   
## [46] "V294" "V295" "V296" "V310" "V311" "V312" "V313" "V326" "V327"   
## [55] "V328" "V329" "V342" "V343" "V344" "V345" "V1179" "V1180" "V1181"  
## [64] "V1182" "V1183" "V1184" "V1185" "V1186" "V1187" "V1188" "V1189" "V1190"  
## [73] "V1191" "V1192" "V1193" "V1194" "V1195" "V1196" "V1197" "V1198" "V1199"  
## [82] "V1200" "V1203" "V1205" "V1206" "V1207" "V1208" "V1209" "V1210" "V1211"  
## [91] "V1212" "V1213" "V1214" "V1215" "V1216" "V1217" "V1218" "V1219" "V1220"  
## [100] "V1221" "V1222" "V1223" "V1224" "V1225" "V1226" "V1227" "V1231" "V1297"  
## [109] "V1299" "V1301" "V1312" "V1314" "V1323" "V1335" "V1347" "V1348" "V1349"  
## [118] "V1350" "V1359" "V1360" "V1361" "V1362" "V1371" "V1383" "V1396" "V1398"

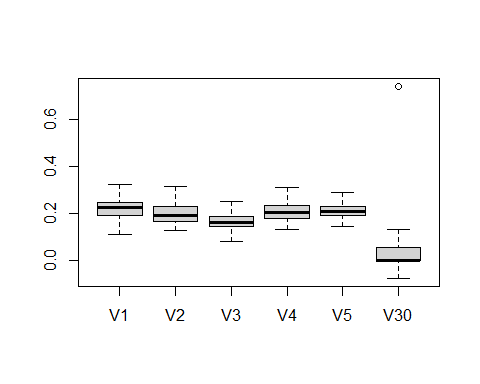
zero\_var <- vardf[order(vardf$variation),]  
str(tot$V1312)

## num [1:30000] 1 1 1 1 1 1 1 1 1 1 ...

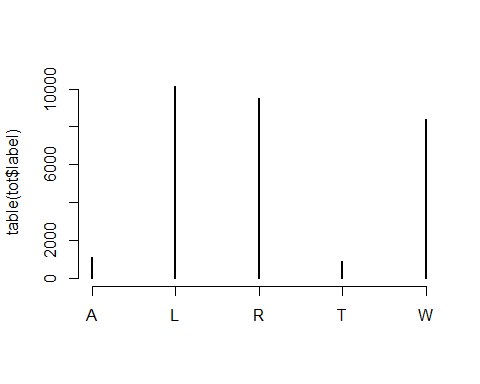
label <- tot$label  
quality <- tot$quality  
# remove columns with zero variance  
tot <- tot[,!(round(vardf$variation, 4) == 0.000)]  
tot$label <- label  
tot$quality <- quality  
  
tot <- (tot[, sapply(tot, function(x) length(unique(x)) > 3)[1:(ncol(tot)-2)]])  
dim(tot)

## [1] 30000 1294

boxplot(tot[, 1:6])

 ### Class Balancing

#-------------------------------balancing class using SMOTE for TRAINSET --------------------   
plot(table(tot$label), type="h")



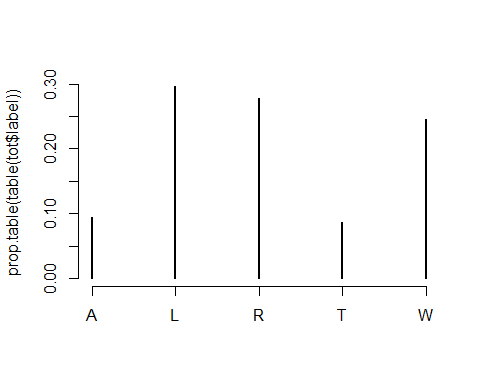
prop.table(table(tot$label))

##   
## A L R T W   
## 0.037 0.338 0.317 0.029 0.279

tot\_de <- tot[tot$quality == 'default',]  
tot\_hq <- tot[tot$quality == 'hq',]  
tot\_vq <- tot[tot$quality == 'vq',]  
dim(tot\_de)

## [1] 10000 1294

## SMOTE for class A  
smoted\_deA <- oversample\_smote((tot\_de %>% select(-quality)), "A", 'label', 700)  
smoted\_hqA <- oversample\_smote((tot\_hq %>% select(-quality)), "A", 'label', 700)  
smoted\_vqA <- oversample\_smote((tot\_vq %>% select(-quality)), "A", 'label', 700)  
  
smoted\_deA$quality <- 'default'  
smoted\_hqA$quality <- 'hq'  
smoted\_vqA$quality <- 'vq'  
  
tot <- rbind(tot, smoted\_deA, smoted\_hqA, smoted\_vqA)  
  
## SMOTE for class T  
smoted\_deT <- oversample\_smote((tot\_de %>% select(-quality)), "T", 'label', 700)  
smoted\_hqT <- oversample\_smote((tot\_hq %>% select(-quality)), "T", 'label', 700)  
smoted\_vqT <- oversample\_smote((tot\_vq %>% select(-quality)), "T", 'label', 700)  
  
smoted\_deT$quality <- 'default'  
smoted\_hqT$quality <- 'hq'  
smoted\_vqT$quality <- 'vq'  
  
tot <- rbind(tot, smoted\_deT, smoted\_hqT, smoted\_vqT)  
  
plot(prop.table(table(tot$label)))



dim(tot)

## [1] 34200 1294

sum(is.na(tot))

## [1] 0

# tot %>% glimpse()  
  
# saving the pre processed SFinGe data   
# saveRDS(tot, 'processed\_SFinGe.rds')  
# tot <- readRDS('processed\_SFinGe.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) %>% 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: 23939  
## Test cases: 10261

## KNN with full data

### Bayesian Optimization for Hyper parameter Tuning

##### Model Specifications

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/SFinGe/knn\_tuned\_full.rds')  
knn\_tuned\_full <- readRDS('models/SFinGe/knn\_tuned\_full.rds')

Training the model on full data with the optimum Hyper-parameters

knn\_best\_param <- knn\_tuned\_full %>% select\_best('roc\_auc')  
  
knn\_final\_full <- knn\_mod %>%   
 finalize\_model(knn\_best\_param) %>%  
 fit( formula = label ~ .,data = tot\_train)  
   
saveRDS(knn\_final\_full, 'models/SFinGe/knn\_final\_full.rds')  
knn\_final\_full <- readRDS('models/SFinGe/knn\_final\_full.rds')

knn\_final\_full <- readRDS('models/SFinGe/knn\_final\_full.rds')  
knn\_final\_full$spec

## K-Nearest Neighbor Model Specification (classification)  
##   
## Main Arguments:  
## neighbors = 50  
## weight\_func = gaussian  
##   
## Computational engine: kknn   
##   
## Model fit template:  
## kknn::train.kknn(formula = missing\_arg(), data = missing\_arg(),   
## ks = min\_rows(50L, data, 5), kernel = "gaussian")

### testing accuracies with full model

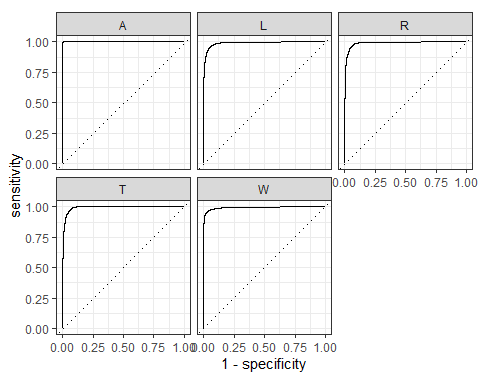
cm\_knn\_full\_SFinGe <- read\_rds('results/SFinGe/cm\_knn\_full\_SFinGe.rds')  
  
# pred\_df <- bind\_cols(ytest,  
# predict(knn\_final\_full, tot\_test),  
# predict(knn\_final\_full, tot\_test, type = "prob"))  
# saveRDS(pred\_df, 'results/SFinGe/knn\_pred\_df.rds')  
pred\_df <- readRDS('results/SFinGe/knn\_pred\_df.rds')  
  
colnames(pred\_df) <- c("obs", "pred","pred\_A","pred\_L", "pred\_R","pred\_T","pred\_W")  
  
pred\_df %>%  
 f\_meas(obs, pred)

## # A tibble: 1 × 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 f\_meas macro 0.894

cm\_knn\_full\_SFinGe <- confusionMatrix(pred\_df$pred, ytest)  
  
# saveRDS(cm\_knn\_full\_SFinGe, 'results/SFinGe/cm\_knn\_full\_SFinGe.rds')  
cm\_knn\_full\_SFinGe

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A L R T W  
## A 1002 15 22 7 7  
## L 1 2792 9 13 184  
## R 0 8 2586 9 204  
## T 0 237 204 851 74  
## W 0 2 4 0 2030  
##   
## Overall Statistics  
##   
## Accuracy : 0.9025   
## 95% CI : (0.8966, 0.9082)  
## No Information Rate : 0.2976   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8731   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: A Class: L Class: R Class: T Class: W  
## Sensitivity 0.99900 0.9142 0.9154 0.96705 0.8123  
## Specificity 0.99449 0.9713 0.9703 0.94510 0.9992  
## Pos Pred Value 0.95157 0.9310 0.9213 0.62299 0.9971  
## Neg Pred Value 0.99989 0.9639 0.9679 0.99674 0.9430  
## Prevalence 0.09775 0.2976 0.2753 0.08576 0.2435  
## Detection Rate 0.09765 0.2721 0.2520 0.08294 0.1978  
## Detection Prevalence 0.10262 0.2923 0.2736 0.13313 0.1984  
## Balanced Accuracy 0.99675 0.9427 0.9428 0.95607 0.9058

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



## Boost Tree on 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 <-   
 workflows::workflow() %>%  
 add\_model(xgboost\_model) %>%   
 add\_formula(label ~ .)  
  
# hyperparameter tuning  
library(doParallel)  
all\_cores <- parallel::detectCores(logical = FALSE)  
cls <- makePSOCKcluster(all\_cores)  
registerDoParallel(cls)  
  
set.seed(234)  
xgboost\_tuned <- tune::tune\_grid(  
 object = xgboost\_wf,  
 resamples = tot\_folds,  
 grid = xgboost\_grid,  
 metrics = yardstick::metric\_set(roc\_auc, accuracy),  
 control = tune::control\_grid(verbose = TRUE, save\_pred = TRUE))  
  
saveRDS(xgboost\_tuned, '../models/xgboost\_tuned.rds')  
xgboost\_tuned\_full <- readRDS('models/SFinGe/xgboost\_tuned\_full.rds')  
# plot(xgboost\_tuned$.notes)  
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 <- 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/SFinGe/xgboost\_final\_full.rds')  
xgboost\_final\_full <- readRDS('models/SFinGe/xgboost\_final\_full.rds')

#### Testing model on Test data

xgboost\_final\_full <- readRDS('models/SFinGe/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`

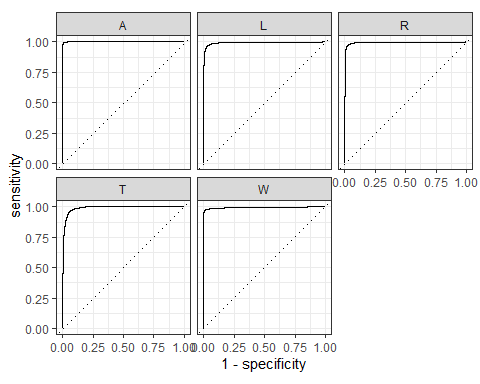
colnames(pred\_df) <- c("obs","pred","pred\_A","pred\_L", "pred\_R","pred\_T","pred\_W")  
  
saveRDS(pred\_df, 'results/SFinGe/xgboost\_final\_pred\_df.rds')  
pred\_df <- readRDS('results/SFinGe/xgboost\_final\_pred\_df.rds')  
  
pred\_df %>%  
 f\_meas(obs, pred)

## # A tibble: 1 × 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 f\_meas macro 0.943

cm\_xgboost\_full <- confusionMatrix(pred\_df$pred, testing(tot\_split)$label)  
cm\_xgboost\_full

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A L R T W  
## A 994 6 5 7 0  
## L 3 2909 17 63 34  
## R 4 22 2736 68 39  
## T 2 87 53 741 11  
## W 0 30 14 1 2415  
##   
## Overall Statistics  
##   
## Accuracy : 0.9546   
## 95% CI : (0.9504, 0.9585)  
## No Information Rate : 0.2976   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9402   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: L Class: R Class: T Class: W  
## Sensitivity 0.99103 0.9525 0.9685 0.84205 0.9664  
## Specificity 0.99806 0.9838 0.9821 0.98369 0.9942  
## Pos Pred Value 0.98221 0.9613 0.9536 0.82886 0.9817  
## Neg Pred Value 0.99903 0.9800 0.9880 0.98516 0.9892  
## Prevalence 0.09775 0.2976 0.2753 0.08576 0.2435  
## Detection Rate 0.09687 0.2835 0.2666 0.07222 0.2354  
## Detection Prevalence 0.09863 0.2949 0.2796 0.08713 0.2397  
## Balanced Accuracy 0.99454 0.9681 0.9753 0.91287 0.9803

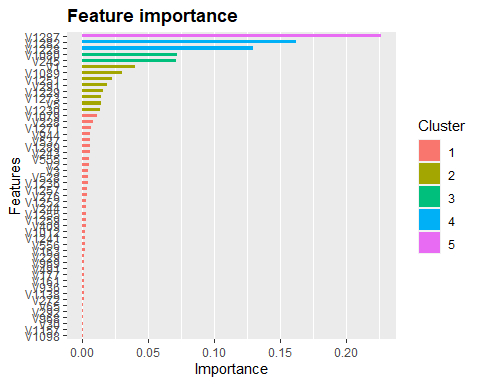
saveRDS(cm\_xgboost\_full, 'results/SFinGe/cm\_xgboost\_full.rds')  
pred\_df %>%  
 roc\_curve(obs, pred\_A:pred\_W) %>%  
 autoplot()



#### Feature Importance

xgboost\_final\_full <- readRDS('models/SFinGe/xgboost\_final\_full.rds')  
imp\_df = xgb.importance(model=xgboost\_final\_full$fit)  
head(imp\_df$Feature, 10)  
saveRDS(imp\_df, 'models/SFinGe/feature\_imp.rds')

imp\_df <- readRDS('models/SFinGe/feature\_imp.rds')  
xgb.importance(model=xgboost\_final\_full$fit) %>% xgb.ggplot.importance(top\_n=50, measure=NULL, rel\_to\_first = FALSE)



dim(imp\_df)

## [1] 433 4

## 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] 23939 301

dim(dftest)

## [1] 10261 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/SFinGe/xgboost\_tuned\_features.rds')  
xgboost\_tuned\_features <- readRDS('models/SFinGe/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/SFinGe/xgboost\_final\_features.rds')  
xgboost\_final\_features <- readRDS('models/SFinGe/xgboost\_final\_features.rds')

#### Testing model on Test data

xgboost\_final\_features <- readRDS('models/SFinGe/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`

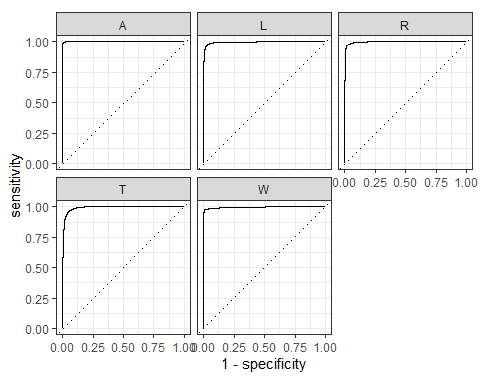
saveRDS(pred\_df, 'results/SFinGe/xgboost\_features\_pred\_df.rds')  
  
pred\_df <- readRDS('results/SFinGe/xgboost\_features\_pred\_df.rds')  
  
colnames(pred\_df) <- c("obs","pred","pred\_A","pred\_L", "pred\_R","pred\_T","pred\_W")  
  
pred\_df %>%  
 f\_meas(obs, pred)

## # A tibble: 1 × 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 f\_meas macro 0.946

cm\_xgboost\_features <- confusionMatrix(pred\_df$pred, testing(tot\_split)$label)  
# saveRDS(cm\_xgboost\_features, 'results/SFinGe/cm\_xgboost\_features.rds')  
cm\_xgboost\_features <- readRDS('results/SFinGe/cm\_xgboost\_features.rds')  
cm\_xgboost\_features

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A L R T W  
## A 992 4 7 14 1  
## L 2 2920 24 65 30  
## R 4 20 2732 50 32  
## T 5 83 46 749 5  
## W 0 27 16 2 2431  
##   
## Overall Statistics  
##   
## Accuracy : 0.9574   
## 95% CI : (0.9533, 0.9612)  
## No Information Rate : 0.2976   
## P-Value [Acc > NIR] : < 2e-16   
##   
## Kappa : 0.9439   
##   
## Mcnemar's Test P-Value : 0.09285   
##   
## Statistics by Class:  
##   
## Class: A Class: L Class: R Class: T Class: W  
## Sensitivity 0.98903 0.9561 0.9671 0.85114 0.9728  
## Specificity 0.99719 0.9832 0.9857 0.98518 0.9942  
## Pos Pred Value 0.97446 0.9602 0.9626 0.84347 0.9818  
## Neg Pred Value 0.99881 0.9814 0.9875 0.98602 0.9913  
## Prevalence 0.09775 0.2976 0.2753 0.08576 0.2435  
## Detection Rate 0.09668 0.2846 0.2663 0.07299 0.2369  
## Detection Prevalence 0.09921 0.2964 0.2766 0.08654 0.2413  
## Balanced Accuracy 0.99311 0.9697 0.9764 0.91816 0.9835

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



## KNN nearest neighbours on selected Features

### 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\_recipe(knn\_rec)  
  
knn\_param <-   
 knn\_wflow %>%   
 parameters() %>%   
 update(  
 neighbors = neighbors(c(3, 30)),  
 weight\_func = weight\_func(values = c("rectangular", "gaussian", "triangular"))  
 )  
  
# ctrl <- control\_bayes(verbose = TRUE)  
# set.seed(2022)  
# knn\_search <- tune\_bayes(knn\_wflow, resamples = df\_fold, initial = 5, iter = 20,  
# param\_info = knn\_param, control = ctrl)  
# saveRDS(knn\_search, 'models/SFinGe/knn\_tuned\_features.rds')  
knn\_tuned\_features <- readRDS('models/SFinGe/knn\_tuned\_features.rds')

knn\_best\_param <- knn\_tuned\_features %>% select\_best('roc\_auc')  
  
knn\_final\_features <- knn\_mod %>% finalize\_model(knn\_best\_param) %>%  
 # fit the model on all the training data  
 fit( formula = label ~ .,data = dftrain)  
  
knn\_final\_features$fit$fitted.values  
saveRDS(knn\_final\_features, 'models/SFinGe/knn\_final\_features.rds')  
knn\_final\_features <- readRDS('models/SFinGe/knn\_final\_features.rds')

#### Predictions on Test data

knn\_final\_features <- readRDS('models/SFinGe/knn\_final\_features.rds')  
  
pred\_df <- bind\_cols(  
 ytest,  
 predict(knn\_final\_features, as.data.frame(tot\_test)),  
 predict(knn\_final\_features, as.data.frame(tot\_test), type = "prob"))

## New names:  
## • `` -> `...1`

saveRDS(pred\_df, 'results/SFinGe/knn\_features\_pred\_df.rds')  
pred\_df <- readRDS('results/SFinGe/knn\_features\_pred\_df.rds')  
  
colnames(pred\_df) <- c("obs","pred","pred\_A","pred\_L", "pred\_R","pred\_T","pred\_W")  
  
pred\_df %>%  
 f\_meas(obs, pred)

## # A tibble: 1 × 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 f\_meas macro 0.953

cm\_knn\_features <- confusionMatrix(pred\_df$pred, testing(tot\_split)$label)  
  
saveRDS(cm\_knn\_features, 'results/SFinGe/cm\_knn\_features.rds')  
cm\_knn\_features <- readRDS('results/SFinGe/cm\_knn\_features.rds')  
cm\_knn\_features

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A L R T W  
## A 1003 10 12 0 2  
## L 0 2910 10 2 56  
## R 0 8 2713 2 81  
## T 0 122 84 876 20  
## W 0 4 6 0 2340  
##   
## Overall Statistics  
##   
## Accuracy : 0.9592   
## 95% CI : (0.9552, 0.9629)  
## No Information Rate : 0.2976   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9465   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: L Class: R Class: T Class: W  
## Sensitivity 1.00000 0.9528 0.9604 0.99545 0.9364  
## Specificity 0.99741 0.9906 0.9878 0.97591 0.9987  
## Pos Pred Value 0.97663 0.9772 0.9675 0.79492 0.9957  
## Neg Pred Value 1.00000 0.9802 0.9850 0.99956 0.9799  
## Prevalence 0.09775 0.2976 0.2753 0.08576 0.2435  
## Detection Rate 0.09775 0.2836 0.2644 0.08537 0.2280  
## Detection Prevalence 0.10009 0.2902 0.2733 0.10740 0.2290  
## Balanced Accuracy 0.99870 0.9717 0.9741 0.98568 0.9675

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

