Capstone Part 2: Choose your own

Jeremy

25/08/2020

Preface

The following machine learning (ML) project is part of the Harvardx Data Science: Capstone course, where learners are given an opportunity to showcase ML within R-programming of one's choosing.

Introduction

As computer and technological systems improve, many different sectors use machine learning to do what cannot be physically done by a human being within short time spans. Within the field of abnormal psychology, there are also many opportunities to apply machine learning. With the increase in detecting many facets of the human brain, machine learning can process the data captured to identify different psychopathologic disorders. Thus, ML can help in prioritising patients in need of further testing, and if so, lead to timely interventions.

In this analysis, I will be using a UCI Parkinsons data set as the basis. This data set consists of 23 speech attributes from 195 voice recordings. From this set, I will be dividing the data set into two, and once I trained the algorithm on the training set, I will apply it to the test set to get the best accuracy. As the result is a categorical result, the two main methods used is K-nearest neighbours (KNN) and random forest. The results will show the method used, the accuracy obtained from the algorithm, and the F1 score.

Methods and analysis

Here, the packages used are loaded into the script. The dataset is downloaded directly from the database.

https://archive.ics.uci.edu/ml/datasets/Parkinsons

1. Packages and data Load in

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(caretEnsemble)) install.packages("caretEnsemble", repos = "http://cran.us.r-project.org")
if(!require(kableExtra)) install.packages("kableExtra", repos = "http://cran.us.r-project.org")

dat <- read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/parkinsons.data")
```

2. Dividing into train and test sets

Here, the dataset is split into 2 partitions, by a 50/50 split. As the data set is relatively small, the 50:50 split ensures that there is equal amount for the algorithm to work on, before it can be tested on the test set. Also, a further analysis of the data set shows that there is no even balance between healthy and parkison's patients. Thus, the prevalence narrows how the data set should be split. Later in the analysis, the randomforest method will elaborate on k-fold methods.

```
dat <- as.data.frame(dat)
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
index <- createDataPartition(dat$status, times = 1, p = 0.5, list = F)

test_dat <- dat[index,]
train_dat <- dat[-index,]
```

3. Preparing the training dataset

These terms were given by the UCI source on measures used for detecting Parkinson's.

name - ASCII subject name and recording number MDVP:Fo(Hz) - Average vocal fundamental frequency MDVP:Fhi(Hz) - Maximum vocal fundamental frequency MDVP:Flo(Hz) - Minimum vocal fundamental frequency MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several measures of variation in fundamental frequency MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shim - Several measures of variation in amplitude NHR,HNR - Two measures of ratio of noise to tonal components in the voice status - Health status of the subject (one) - Parkinson's, (zero) - healthy RPDE,D2 - Two nonlinear dynamical complexity measures DFA - Signal fractal scaling exponent spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variation

However, the column names will be renamed for better processing and understanding.

Subject - ASCII subject name and recording number Avg.vocal.freq - Average vocal fundamental frequency Max - Maximum vocal fundamental frequency Min.vocal.freq - Minimum vocal fundamental frequency Jitter.perc,Jitter.Abs,RAP,PPQ,DDP - Several measures of variation in fundamental frequency Shimmer,Shimmer.dB,Shimmer.APQ3,Shimmer.APQ5,APQ,DDA - Several measures of variation in amplitude NHR,HNR - Two measures of ratio of noise to tonal components in the voice status - Health status of the subject (one) - Parkinson's, (zero) - healthy RPDE,D2 - Two nonlinear dynamical complexity measures DFA - Signal fractal scaling exponent spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variation

```
#renaming columns
names(train_dat) <- c("Subject",
                      "Avg.vocal.freq",
                      "Max.vocal.freq",
                      "Min.vocal.freq",
                      "Jitter.perc",
                      "Jitter.Abs",
                      "RAP",
                      "PPQ",
                      "DDP",
                      "Shimmer",
                      "Shimmer.dB",
                      "Shimmer.APQ3",
                      "Shimmer.APQ5",
                      "APQ",
                      "Shimmer.DDA",
                      "NHR",
                      "HNR",
                      "status",
                      "RPDE",
                      "D2",
                      "DFA",
                      "Spread1",
                      "Spread2",
                      "PPE")

train_dat <- as.data.frame(train_dat)
train_dat_numeric <- train_dat %>% select(-"Subject", -"status")

colnames <- colnames(train_dat_numeric)

#preparing the test set:
names(test_dat) <- c("Subject",
                      "Avg.vocal.freq",
                      "Max.vocal.freq",
                      "Min.vocal.freq",
                      "Jitter.perc",
                      "Jitter.Abs",
                      "RAP",
                      "PPQ",
                      "DDP",
                      "Shimmer",
                      "Shimmer.dB",
                      "Shimmer.APQ3",
                      "Shimmer.APQ5",
                      "APQ",
                      "Shimmer.DDA",
                      "NHR",
                      "HNR",
                      "status",
                      "RPDE",
                      "D2",
                      "DFA",
                      "Spread1",
                      "Spread2",
                      "PPE")

test_dat <- as.data.frame(test_dat)
```

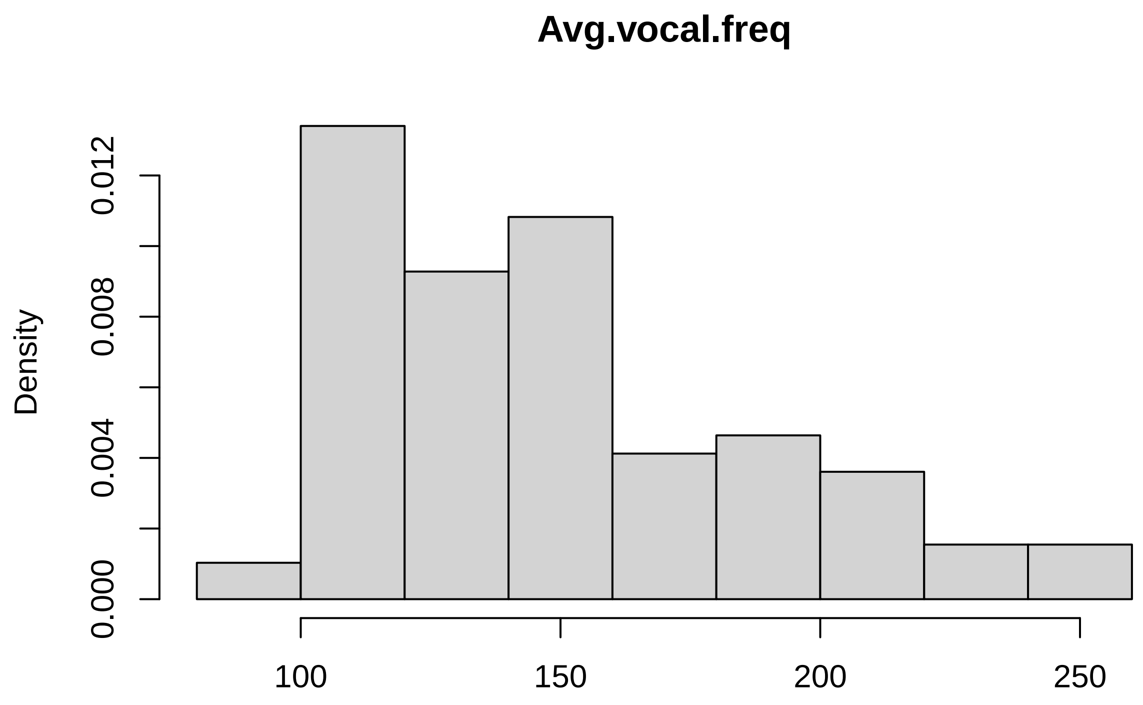
3.1 Exploring the training dataset

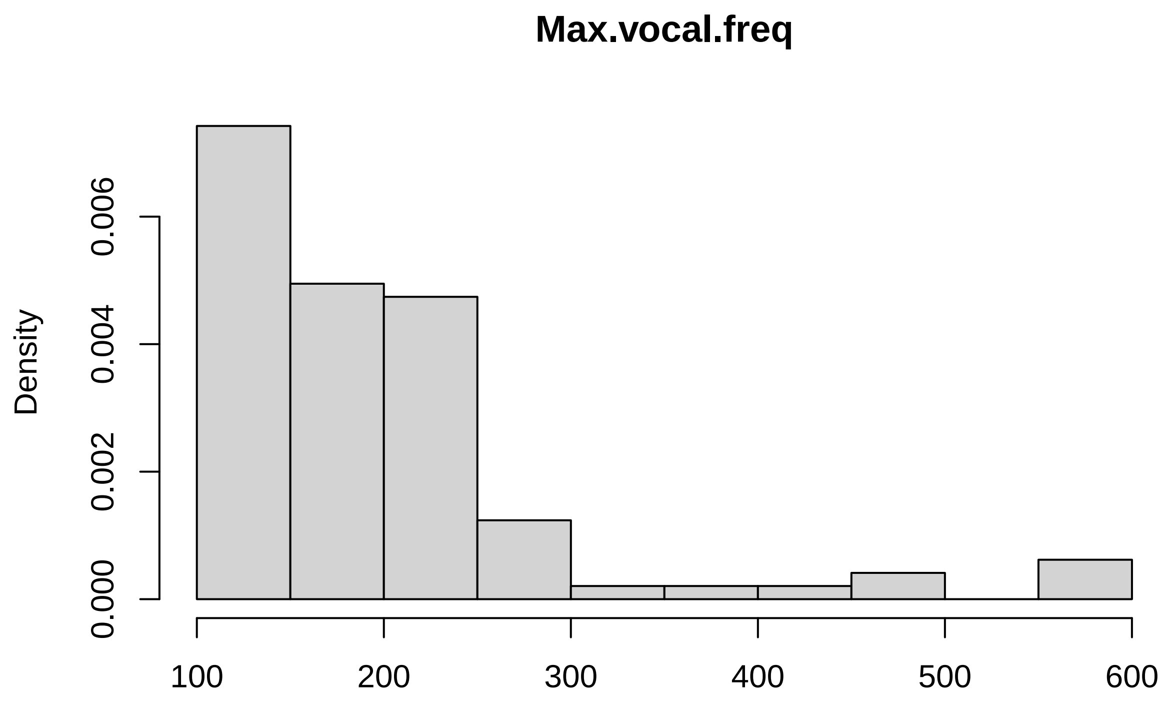
In this section we will see how the scores are distributed. Initially, the histogram does not give much information other than the spread of the predictors. However, when visualised by comparing the status (healthy vs parkinson's), clear distinctions in the scores form.

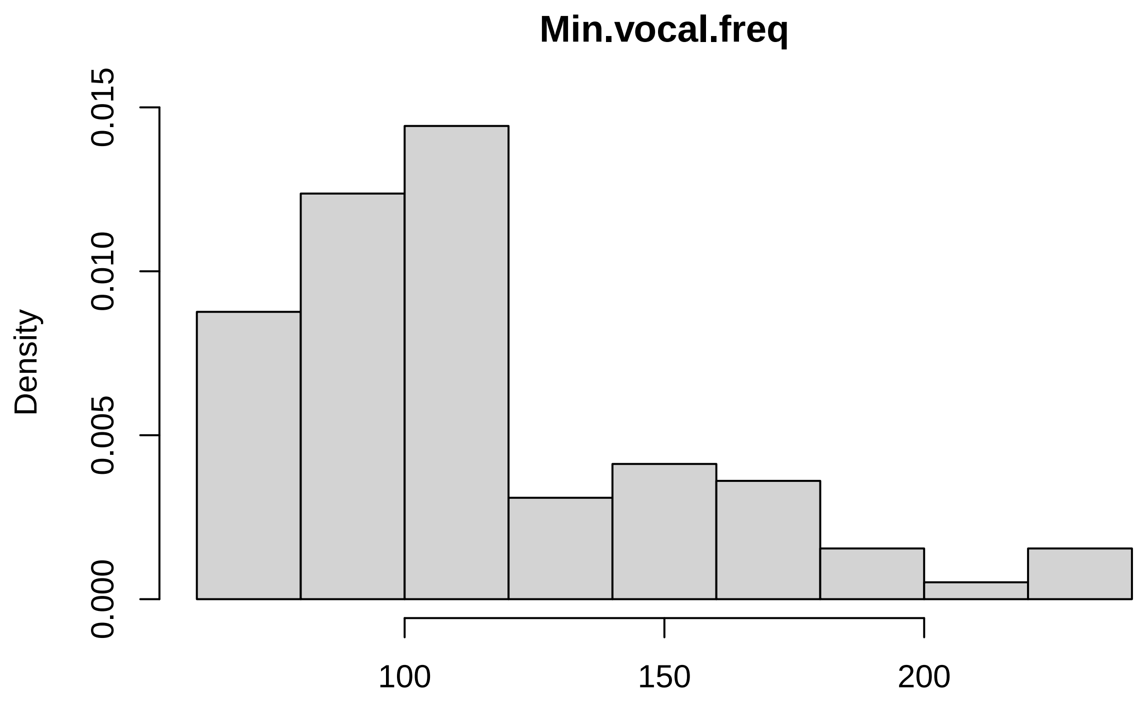
```
#data set
head(train_dat)
```

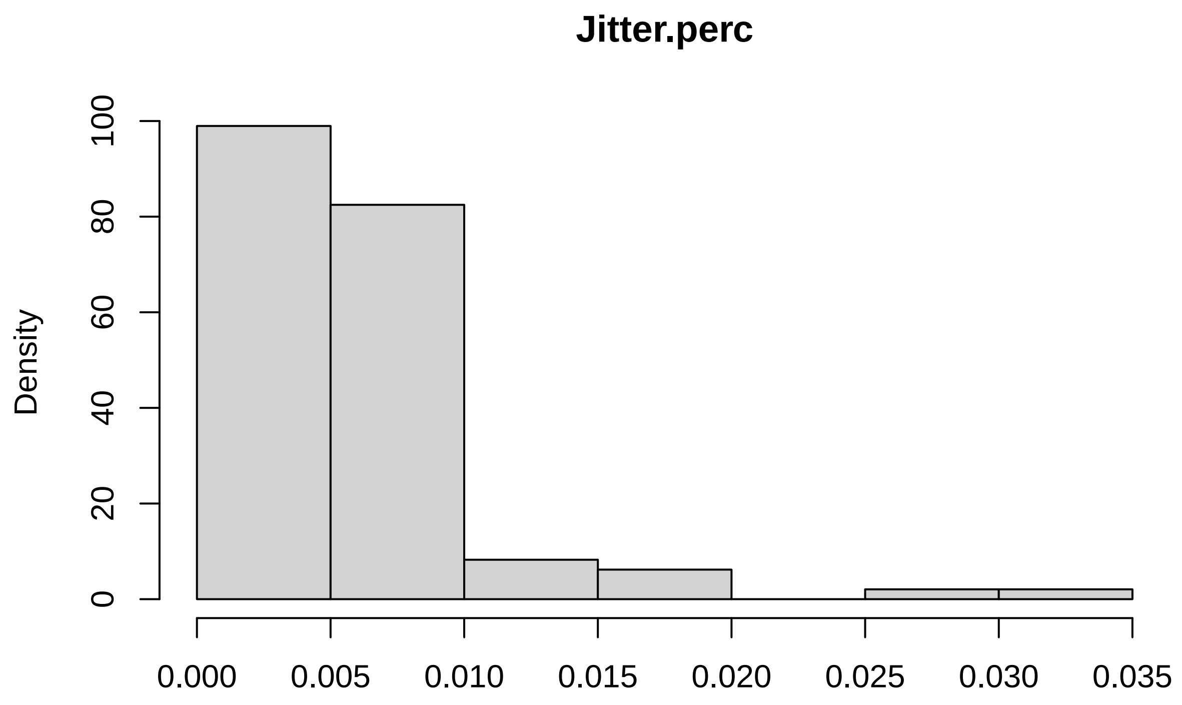
```
##          Subject Avg.vocal.freq Max.vocal.freq Min.vocal.freq Jitter.perc
## 1 phon_R01_S01_1        119.992        157.302         74.997     0.00784
## 2 phon_R01_S01_2        122.400        148.650        113.819     0.00968
## 5 phon_R01_S01_5        116.014        141.781        110.655     0.01284
## 6 phon_R01_S01_6        120.552        131.162        113.787     0.00968
## 8 phon_R01_S02_2        107.332        113.840        104.315     0.00290
## 9 phon_R01_S02_3         95.730        132.068         91.754     0.00551
##   Jitter.Abs     RAP     PPQ     DDP Shimmer Shimmer.dB Shimmer.APQ3
## 1    0.00007 0.00370 0.00554 0.01109 0.04374      0.426      0.02182
## 2    0.00008 0.00465 0.00696 0.01394 0.06134      0.626      0.03134
## 5    0.00011 0.00655 0.00908 0.01966 0.06425      0.584      0.03490
## 6    0.00008 0.00463 0.00750 0.01388 0.04701      0.456      0.02328
## 8    0.00003 0.00144 0.00182 0.00431 0.01567      0.134      0.00829
## 9    0.00006 0.00293 0.00332 0.00880 0.02093      0.191      0.01073
##   Shimmer.APQ5     APQ Shimmer.DDA     NHR    HNR status     RPDE       D2
## 1      0.03130 0.02971     0.06545 0.02211 21.033      1 0.414783 0.815285
## 2      0.04518 0.04368     0.09403 0.01929 19.085      1 0.458359 0.819521
## 5      0.04825 0.04465     0.10470 0.01767 19.649      1 0.417356 0.823484
## 6      0.03526 0.03243     0.06985 0.01222 21.378      1 0.415564 0.825069
## 8      0.00946 0.01256     0.02487 0.00344 26.892      1 0.637420 0.763262
## 9      0.01277 0.01717     0.03218 0.01070 21.812      1 0.615551 0.773587
##         DFA  Spread1  Spread2      PPE
## 1 -4.813031 0.266482 2.301442 0.284654
## 2 -4.075192 0.335590 2.486855 0.368674
## 5 -3.747787 0.234513 2.332180 0.410335
## 6 -4.242867 0.299111 2.187560 0.357775
## 8 -6.167603 0.183721 2.064693 0.163755
## 9 -5.498678 0.327769 2.322511 0.231571
```

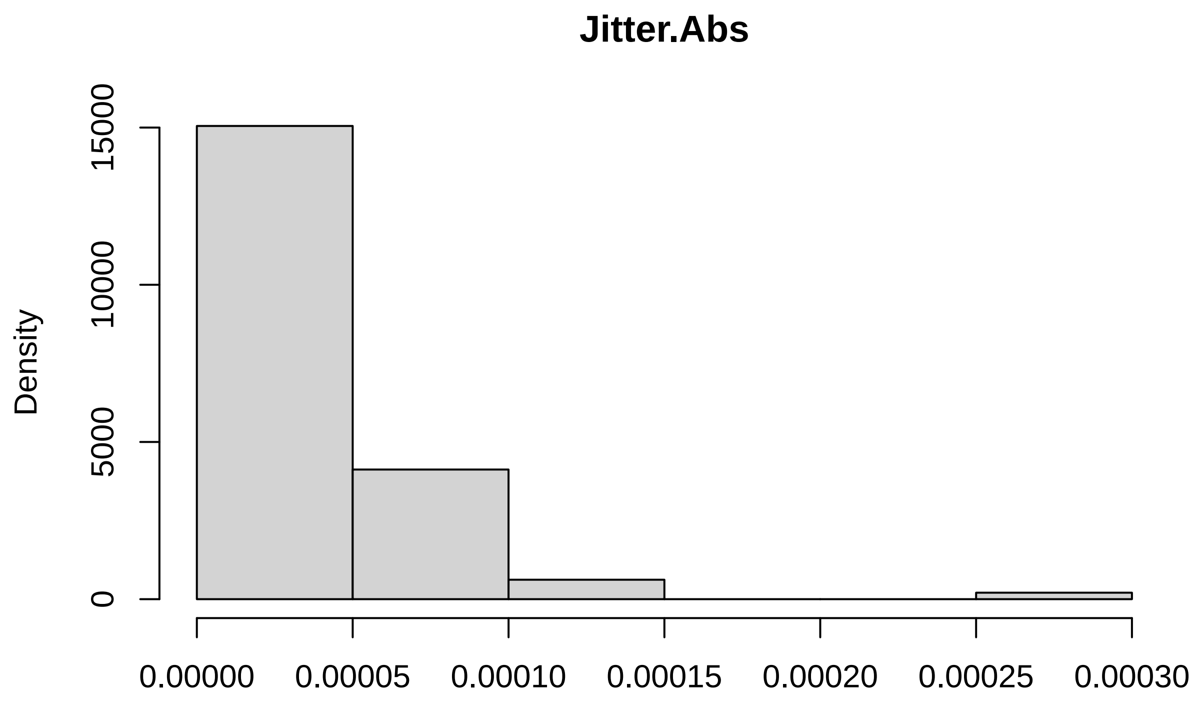
```
#visualising the distribution of scores
for (i in 1:22) {
    hist(train_dat_numeric[,i], probability=TRUE,
         main = colnames[i],
         xlab = NULL)
}
```

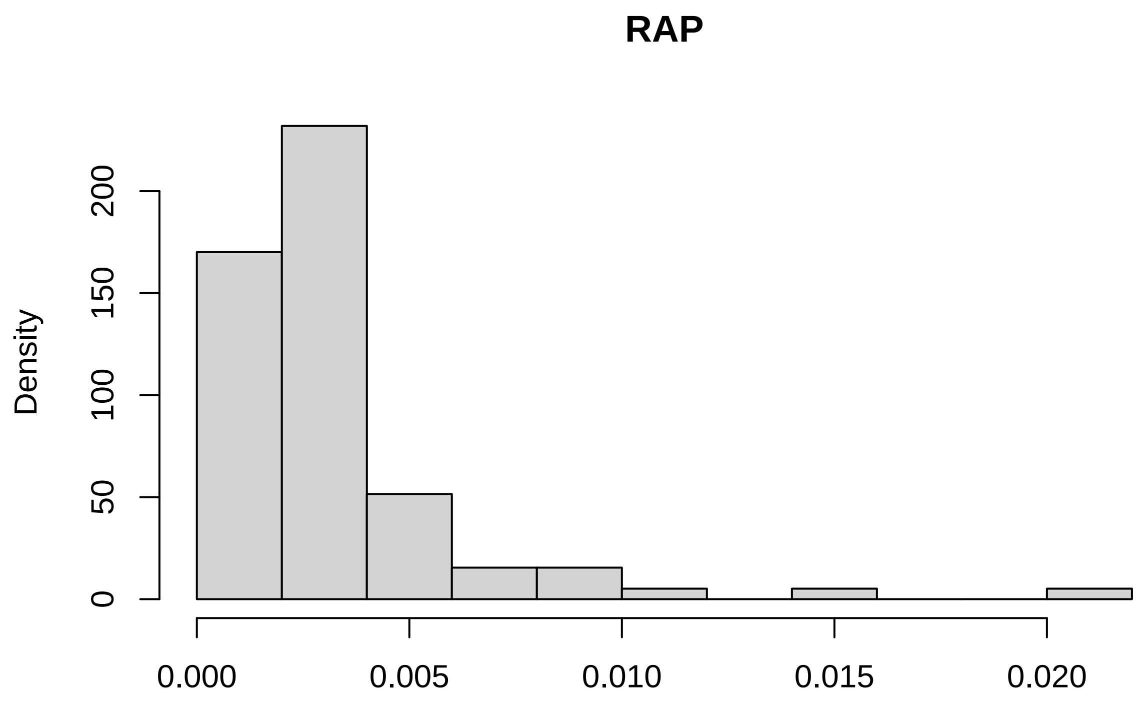

Avg.vocal.freq
Density
0.000
0.004
0.008
0.012
100
150
200
250

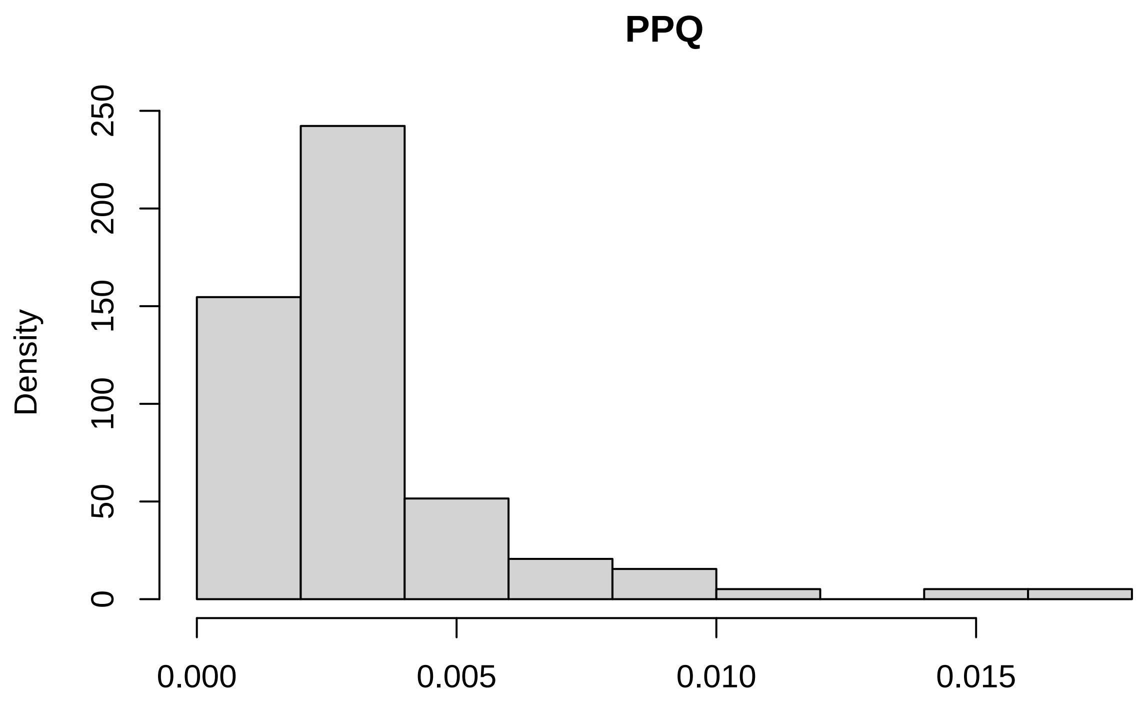

Max.vocal.freq
Density
0.000
0.002
0.004
0.006
100
200
300
400
500
600

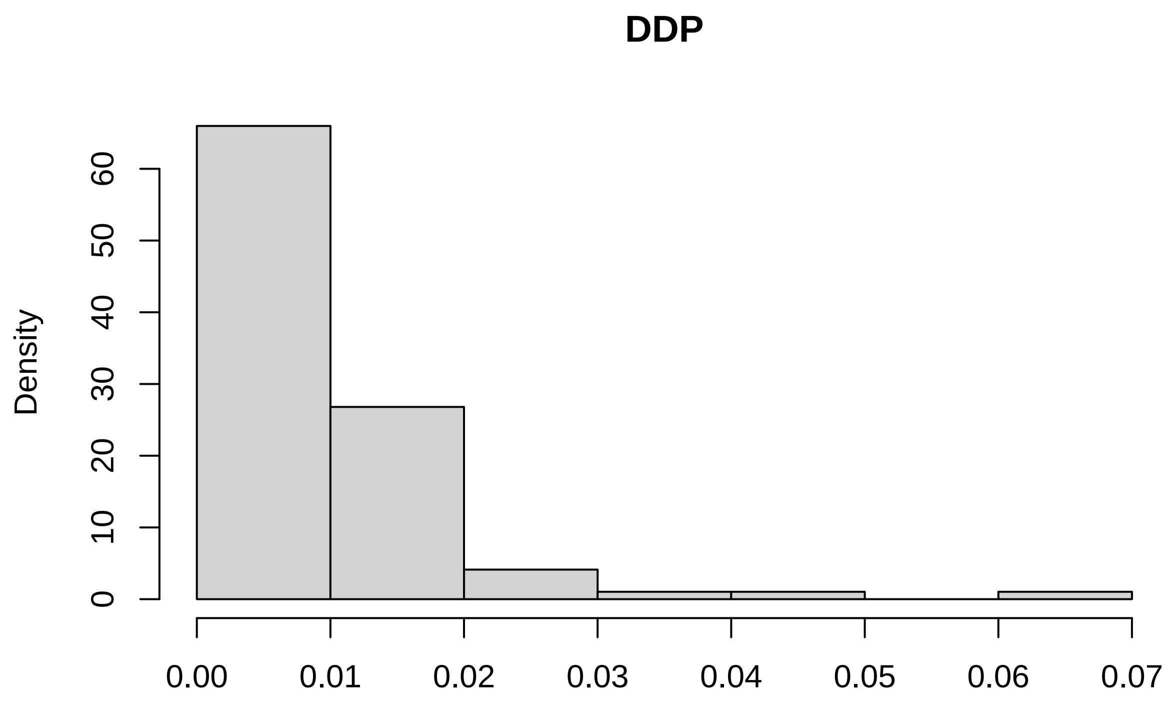

Min.vocal.freq
Density
0.000
0.005
0.010
0.015
100
150
200

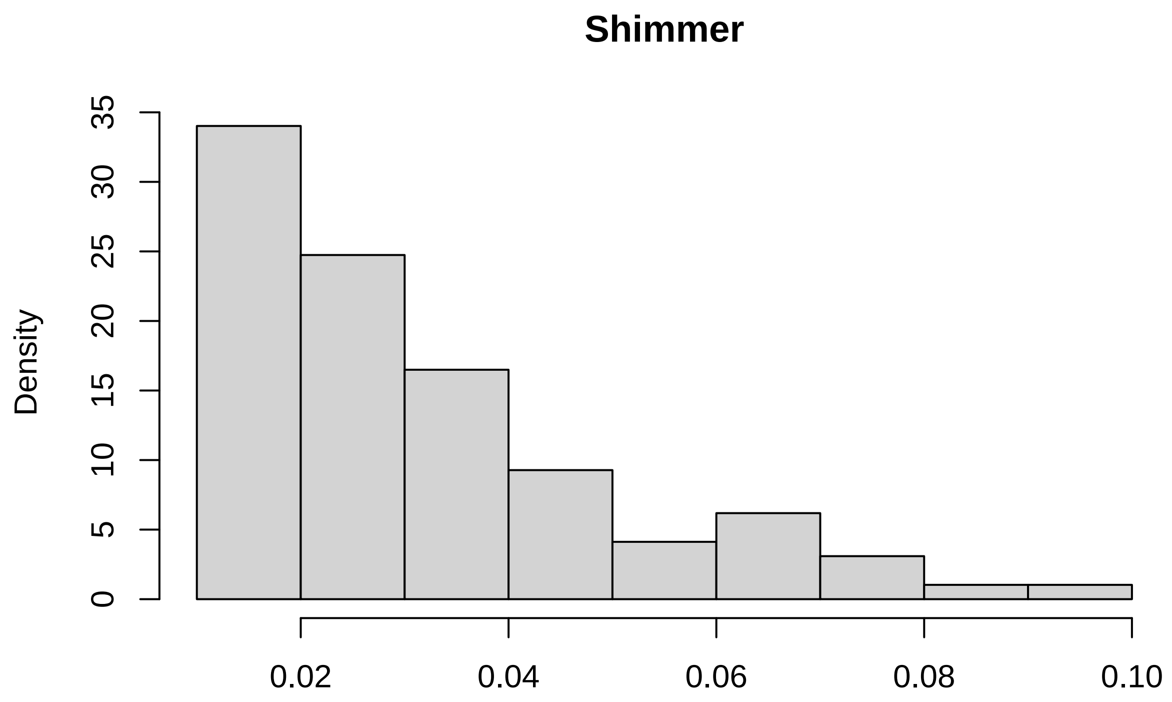

Jitter.perc
Density
0
20
40
60
80
100
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035


Jitter.Abs
Density
0
5000
10000
15000
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025
0.00030

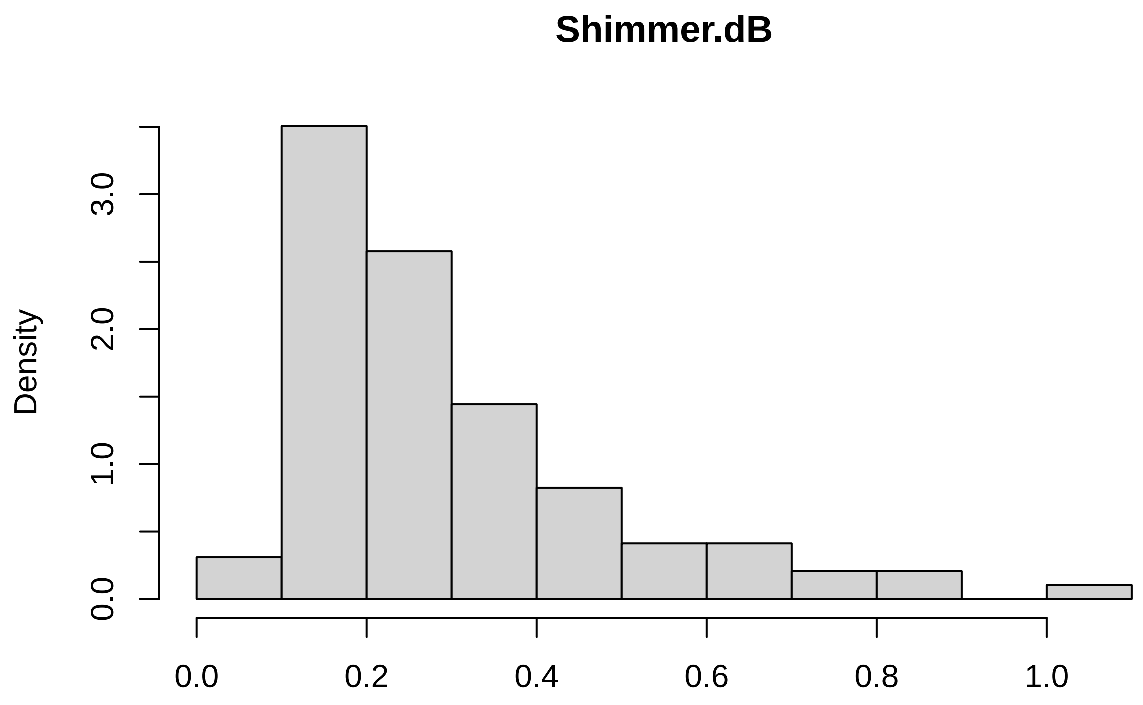

RAP
Density
0
50
100
150
200
0.000
0.005
0.010
0.015
0.020

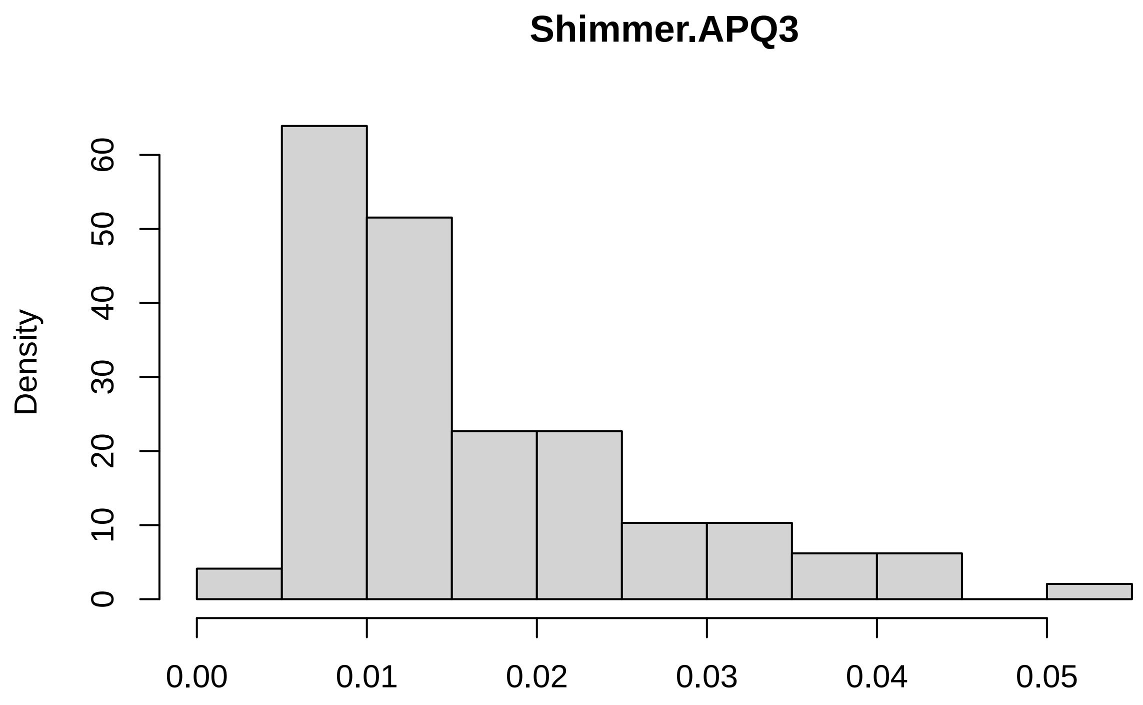

PPQ
Density
0
50
100
150
200
250
0.000
0.005
0.010
0.015

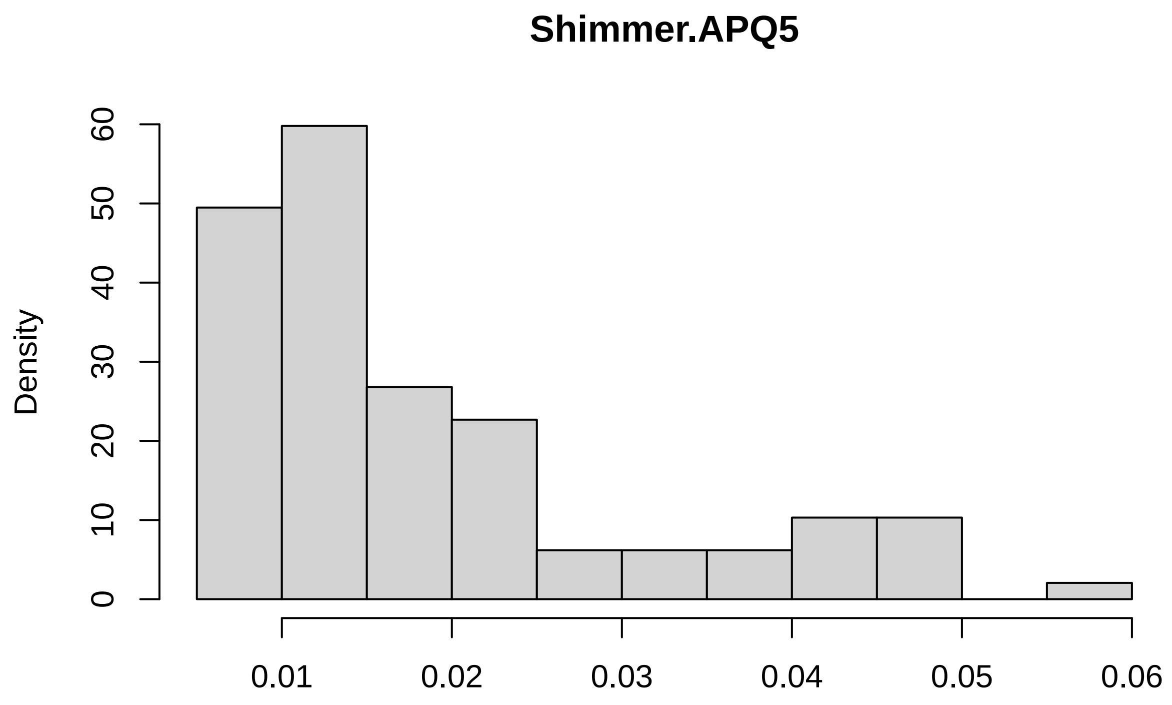

DDP
Density
0
10
20
30
40
50
60
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07

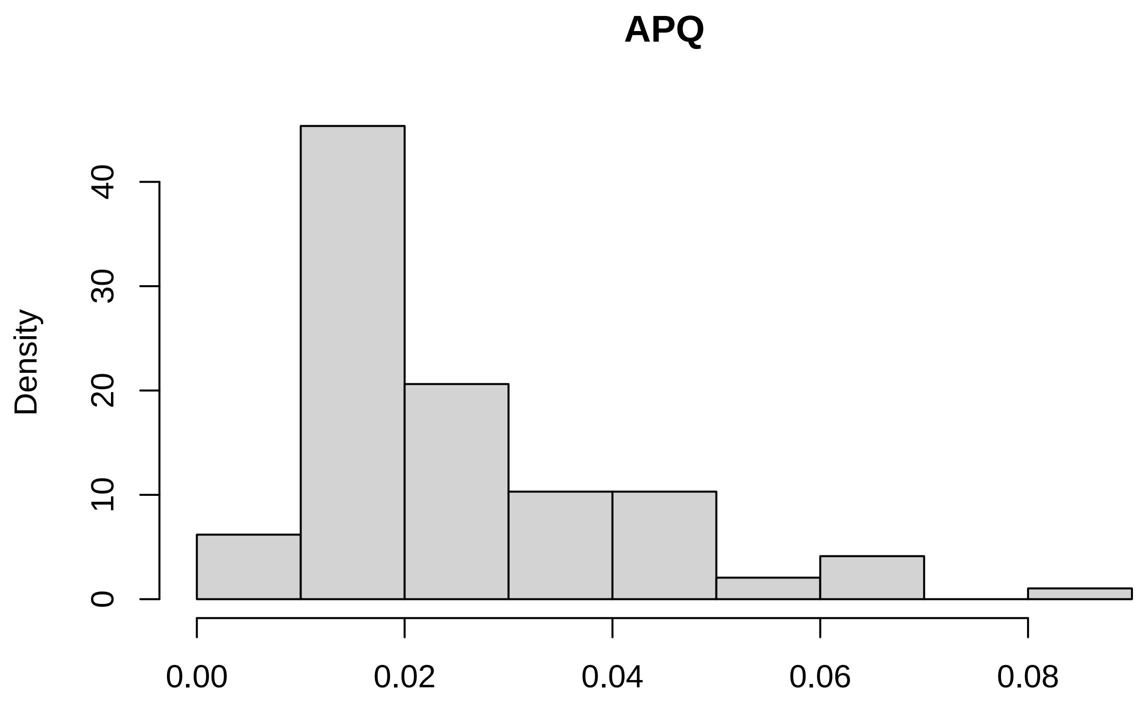


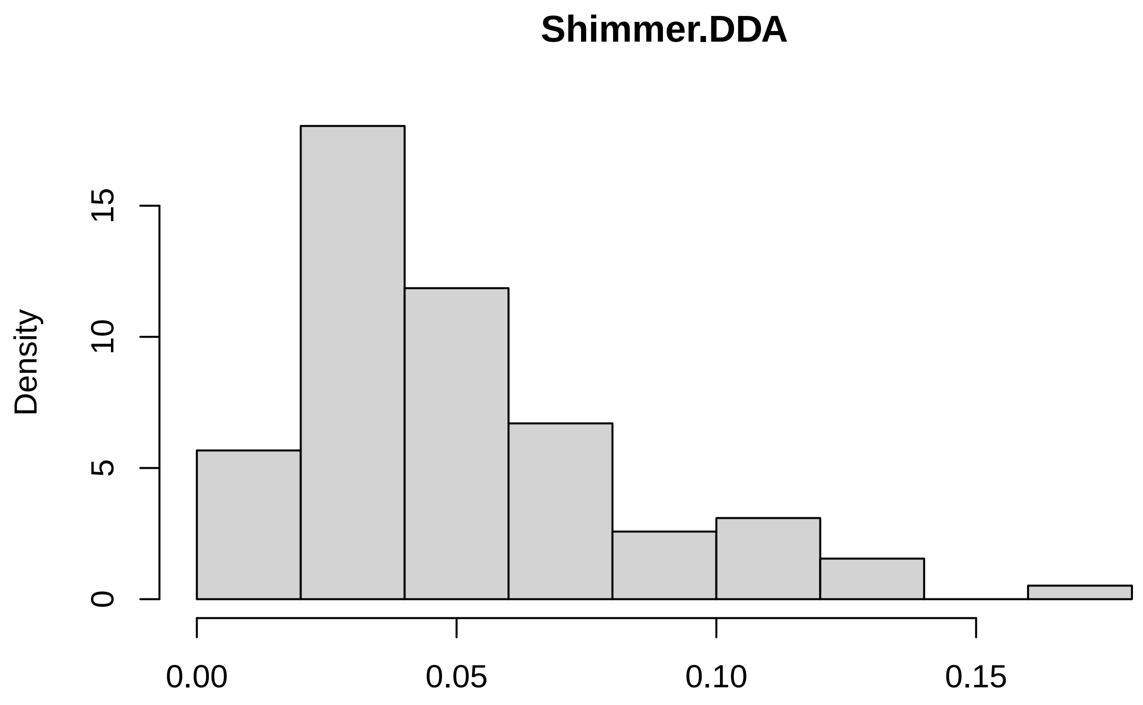
Shimmer
Density
0
5
10
15
20
25
30
35
0.02
0.04
0.06
0.08
0.10

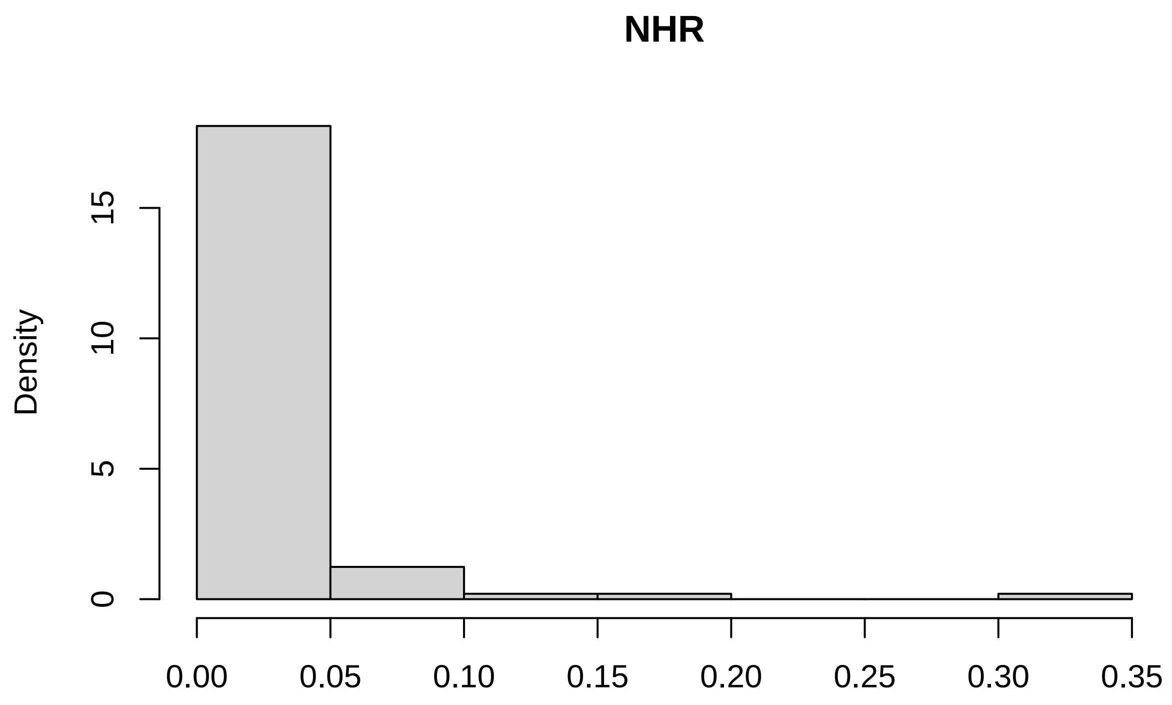

Shimmer.dB
Density
0.0
1.0
2.0
3.0
0.0
0.2
0.4
0.6
0.8
1.0

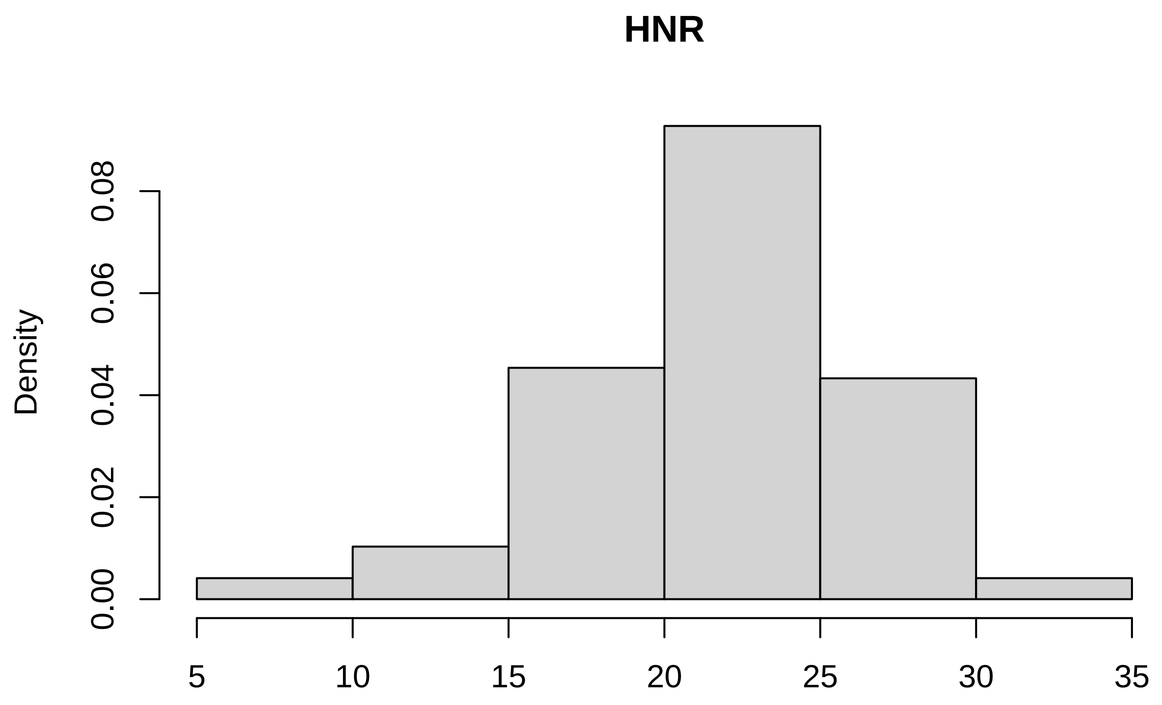

Shimmer.APQ3
Density
0
10
20
30
40
50
60
0.00
0.01
0.02
0.03
0.04
0.05

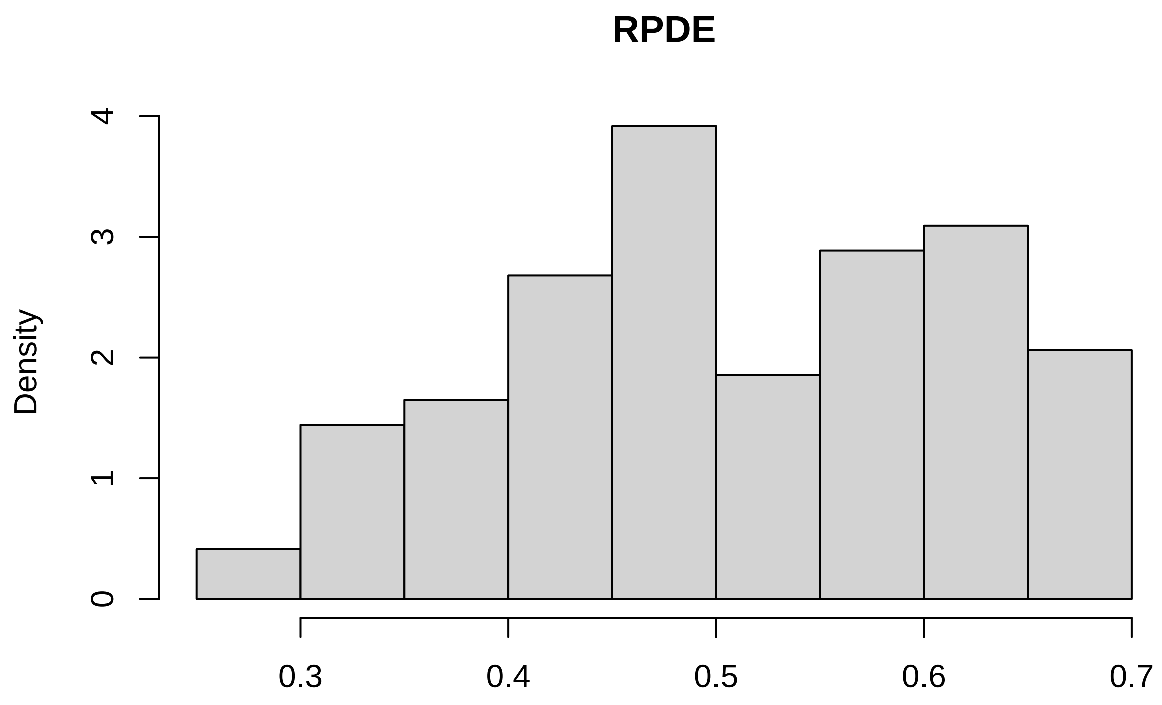

Shimmer.APQ5
Density
0
10
20
30
40
50
60
0.01
0.02
0.03
0.04
0.05
0.06

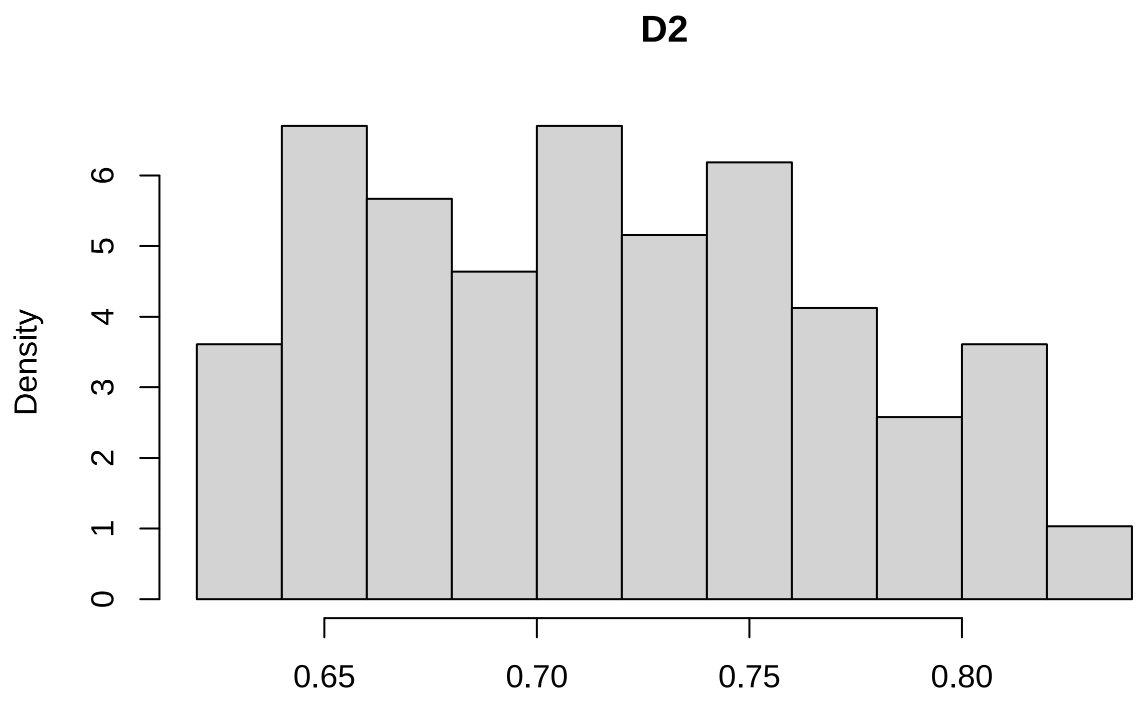

APQ
Density
0
10
20
30
40
0.00
0.02
0.04
0.06
0.08

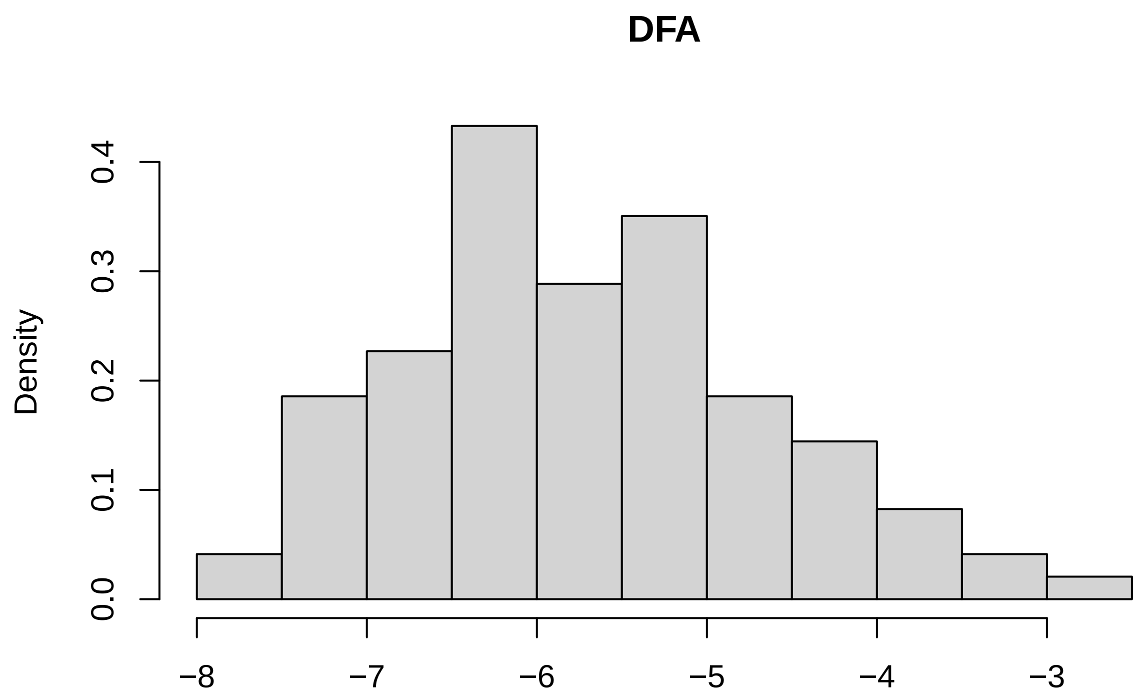

Shimmer.DDA
Density
0
5
10
15
0.00
0.05
0.10
0.15

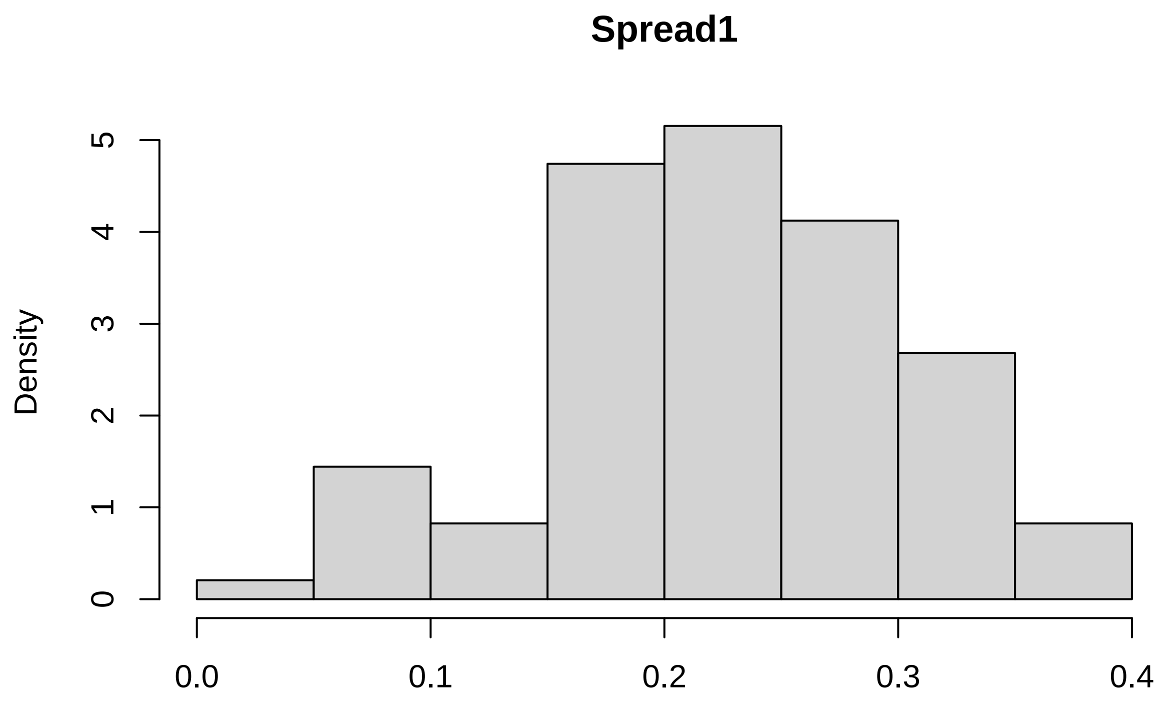

NHR
Density
0
5
10
15
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35

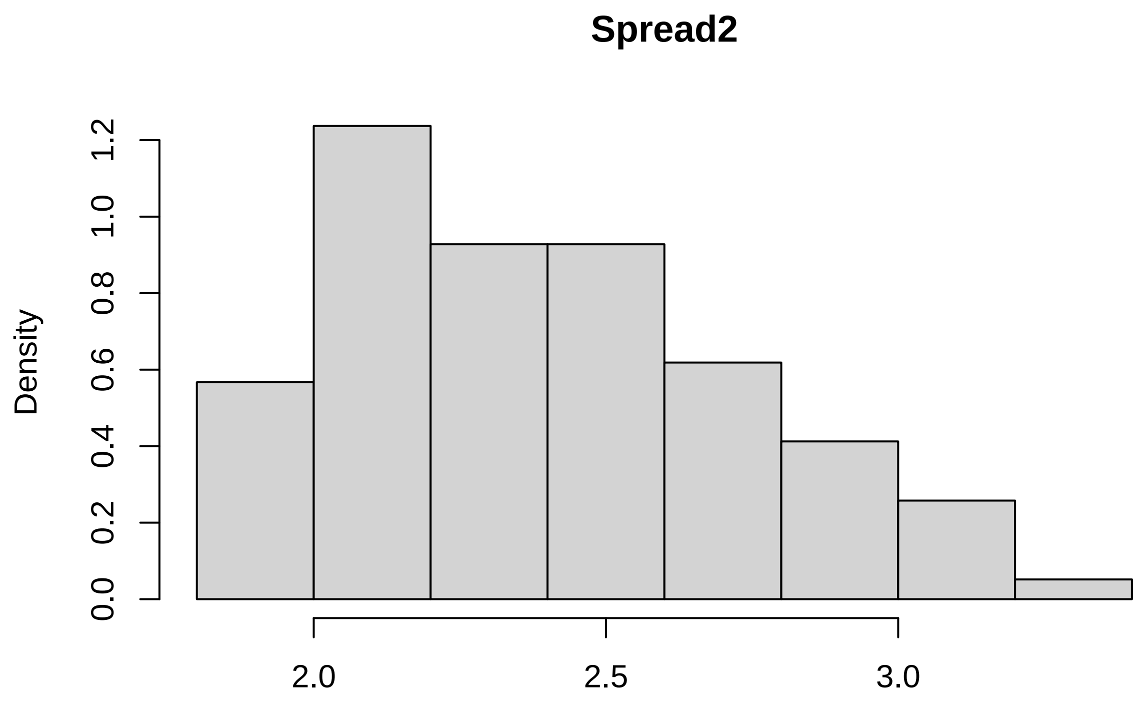

HNR
Density
0.00
0.02
0.04
0.06
0.08
5
10
15
20
25
30
35

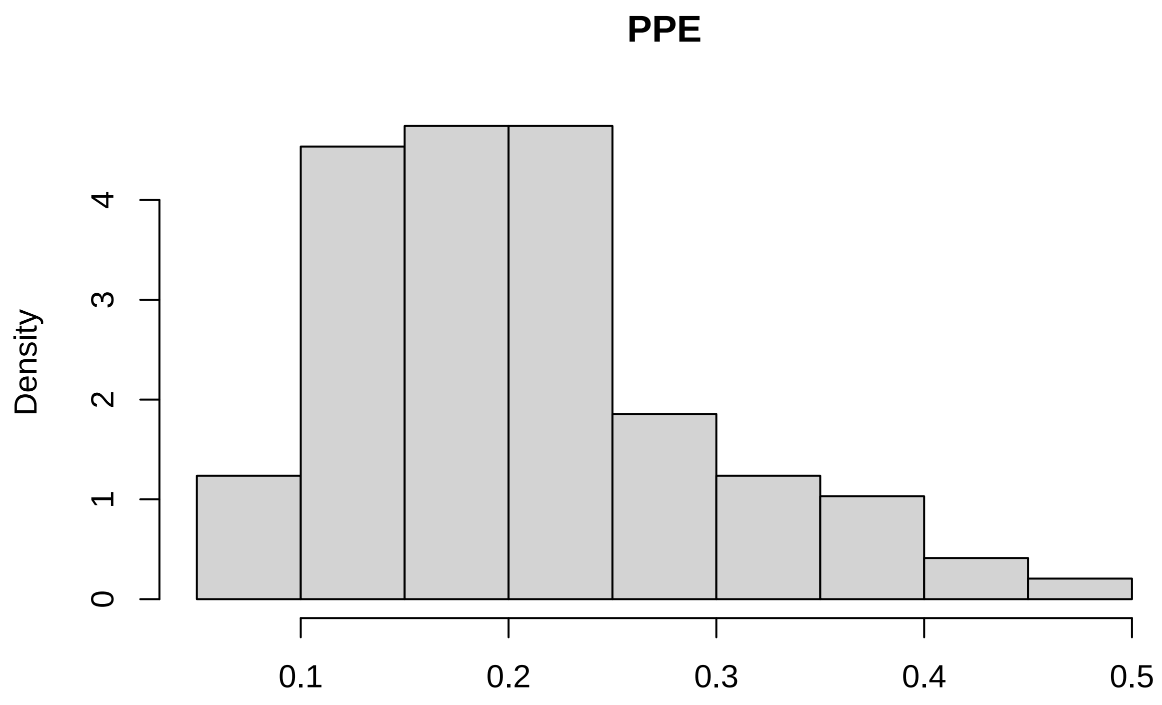

RPDE
Density
0
1
2
3
4
0.3
0.4
0.5
0.6
0.7


D2
Density
0
1
2
3
4
5
6
0.65
0.70
0.75
0.80


DFA
Density
0.0
0.1
0.2
0.3
0.4
−8
−7
−6
−5
−4
−3


Spread1
Density
0
1
2
3
4
5
0.0
0.1
0.2
0.3
0.4

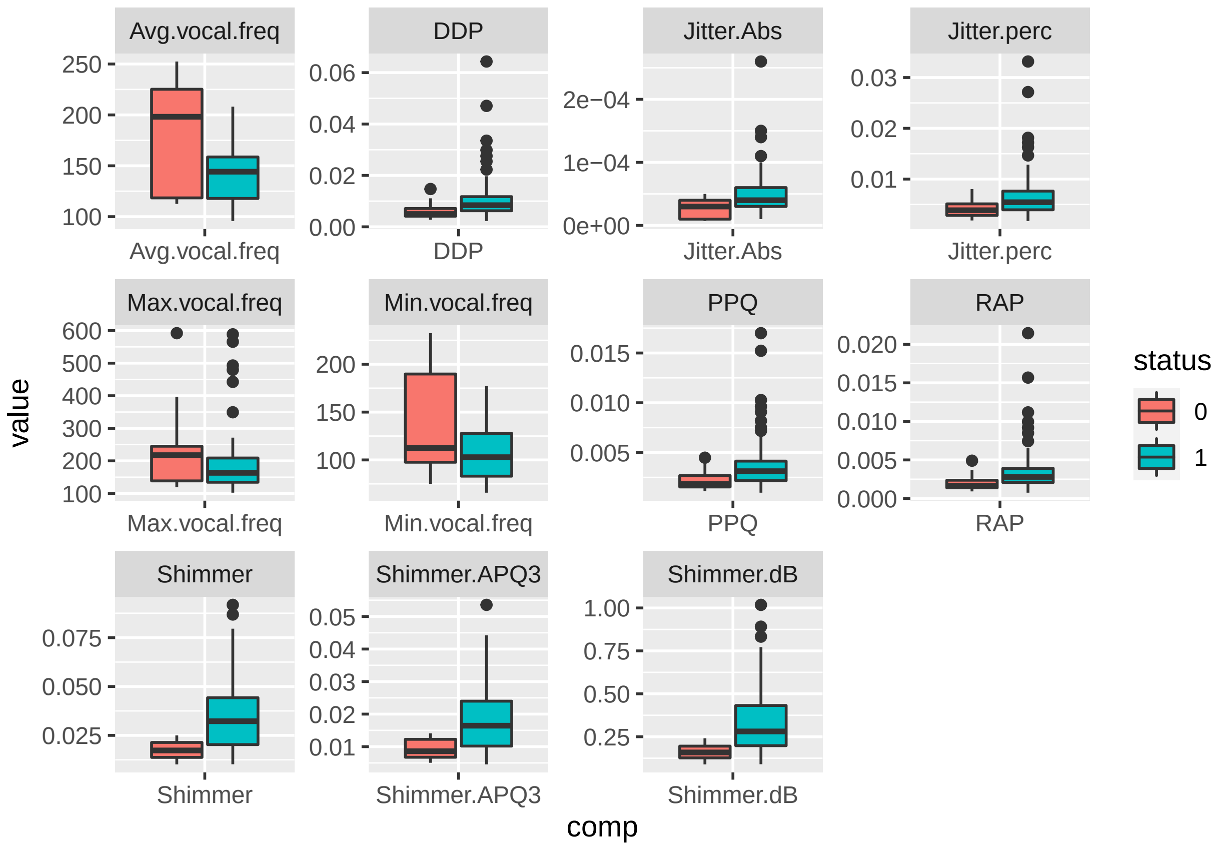

Spread2
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
2.0
2.5
3.0



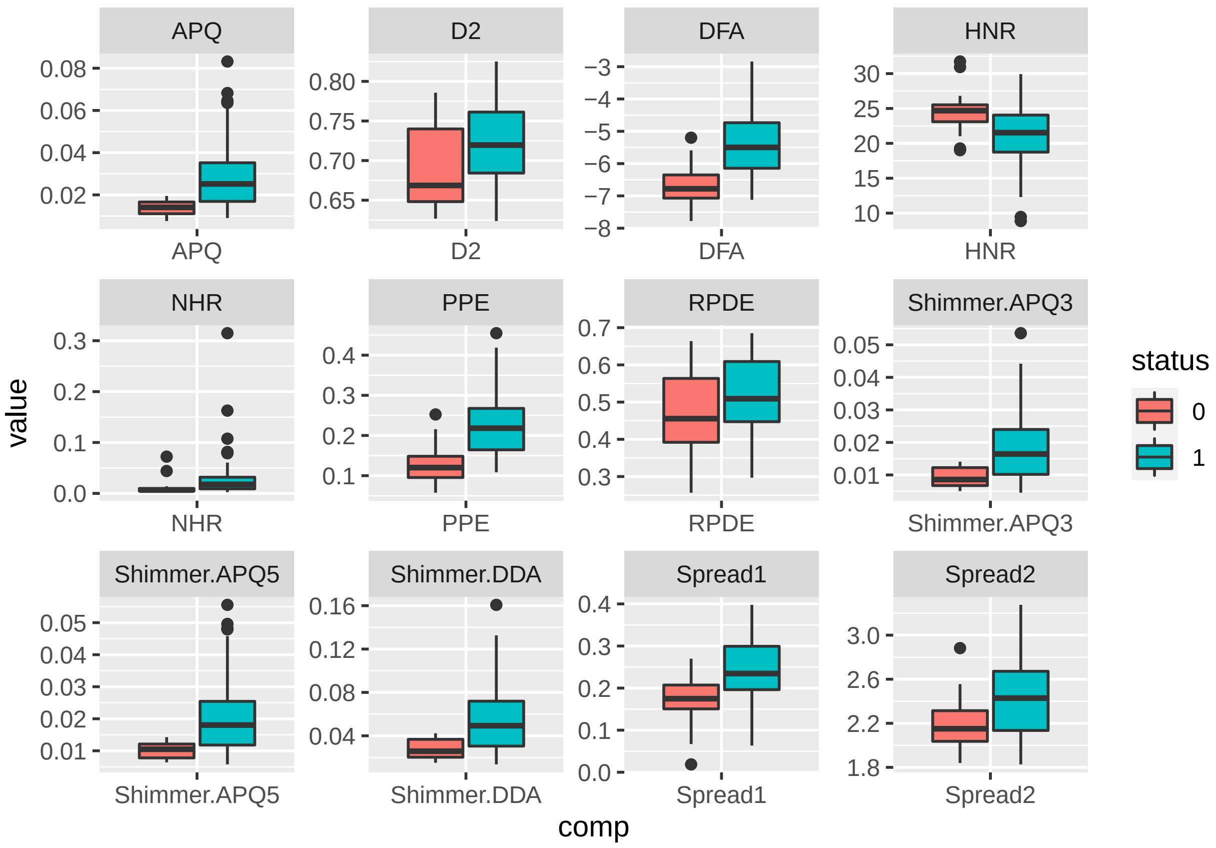
```r
#visualising if values change according to status
#part 1:
a <- train_dat %>%
  select(Avg.vocal.freq:Shimmer.APQ3, status) %>%
  mutate(status = as.factor(status)) %>%
  gather(comp, value, -status) %>%
  ggplot(aes(comp,value, fill = status))+
  geom_boxplot()+
  facet_wrap(~comp, scales = "free")

#part 2:
b <- train_dat %>%
  select(Shimmer.APQ3:PPE) %>%
  mutate(status = as.factor(status)) %>%
  gather(comp, value, -status) %>%
  ggplot(aes(comp,value, fill = status))+
  geom_boxplot()+
  facet_wrap(~comp, scales = "free")

a;b
```



Avg.vocal.freq
DDP
Jitter.Abs
Jitter.perc
Max.vocal.freq
Min.vocal.freq
PPQ
RAP
Shimmer
Shimmer.APQ3
Shimmer.dB
value
comp
status
0
1



From here, the following components have visually distinct differences between healthy and Parkinson's recordings: - Jitter.Abs, Jitter.perc, PPQ, Shimmer, Shimmer.dB, APQ, DFA, Shimmer.APQ3, Shimmer.APQ5, Shimmer.DDA, Spread1

Analysis: Fitting the algorithms

4. Fitting models: KNN

K-nearest neighbours, or KNN, estimates the average of the observations used (the neighbourhood). The flexibility in the method comes from the k parameter, which we will see fine tuned in this section. Here, *all* predictors are used for the analysis.

```
#prep work
train_dat1 <- train_dat %>%
  select(-Subject) %>%
  mutate(status = as.factor(status))
test_dat1 <- test_dat %>% select(-Subject) %>%
  mutate(status = as.factor(status))

ks <- seq(1, 96, 2)

accuracy <- map_df(ks, function(k){
  fit <- knn3(status ~ ., data = train_dat1, k = k)
  y_hat <- predict(fit, train_dat1, type="class")
  cm_train <- confusionMatrix(y_hat, train_dat1$status)
  train_error <- cm_train$overall["Accuracy"]

  y_hat <- predict(fit, test_dat1,type="class")
  cm_test <- confusionMatrix(y_hat, test_dat1$status)
  test_error <- cm_test$overall["Accuracy"]

  tibble(train = train_error, test = test_error)
})

accuracy
```

```
## # A tibble: 48 x 2
##    train  test
##    <dbl> <dbl>
##  1 1     0.857
##  2 0.887 0.847
##  3 0.876 0.827
##  4 0.866 0.837
##  5 0.825 0.816
##  6 0.825 0.816
##  7 0.825 0.806
##  8 0.784 0.745
##  9 0.784 0.765
## 10 0.773 0.735
## # ... with 38 more rows
```

```
which.max(accuracy$test) #when k = 5, accuracy highest
```

```
## Accuracy
##        1
```

```
#using k=5
fit.knn <- knn3(status ~ ., data = train_dat1, k = 5)
y_hat_knn <- predict(fit.knn, test_dat1, type="class")
cm_test <- confusionMatrix(y_hat_knn, test_dat1$status)
cm_test$overall["Accuracy"] #0.82653
```

```
##  Accuracy
## 0.8265306
```

```
#checking F1 scores
F_meas(y_hat_knn,test_dat1$status) #0.58537
```

```
## [1] 0.5853659
```

```
#prevalence in test_dat
mean(1-test_dat$status)
```

```
## [1] 0.2653061
```

```r
acc_results <- data_frame(Method = "Tuned KNN (k=5)", #column 1: method
                          Accuracy = cm_test$overall["Accuracy"],
                         F1.score  = F_meas(y_hat_knn,test_dat1$status))
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
acc_results
```

```
## # A tibble: 1 x 3
##   Method          Accuracy F1.score
##   <chr>              <dbl>    <dbl>
## 1 Tuned KNN (k=5)    0.827    0.585
```

```r
#prevalence
mean(train_dat$status == 1)
```

```
## [1] 0.7731959
```

4.1 Tuning the F1 score

As in the last section, we see that the F1 score is quite low. The F1 score refers to the harmonic average between both specificity and sensitivity. Calculating the F1 score here is important as the prevalence of the status is not an exact 50/50 split (77.3% of the set has detected parkinsons). A low F1 score could mean that the algorithm have poor precision and poor recall.

Here, we find that the F1 score can improve.

```r
ks <- seq(1, 50, 2)
Fvalue <- map_df(ks, function(k){
  fit.knn <- knn3(status ~ ., data = train_dat1, k = k)
  y_hat_knn <- predict(fit.knn, test_dat1, type="class")
  cm_train <- confusionMatrix(y_hat_knn, test_dat1$status)
  Fvalue <- F_meas(y_hat_knn,test_dat1$status)

  tibble(Fvalue)
})
Fvalue
```

```
## # A tibble: 25 x 1
##     Fvalue
##      <dbl>
##  1  0.731
##  2  0.694
##  3  0.585
##  4  0.619
##  5  0.471
##  6  0.471
##  7  0.457
##  8  0.0741
##  9  0.207
## 10 NA
## # ... with 15 more rows
```

```
ks[which.max(Fvalue$Fvalue)] #when k = 1, F1 score is 0.73077
```

```
## [1] 1
```

4.2 Using components with clear distinctions

Here, KNN is used with tuning of the k value for *selected* predictors.

As seen in section 3: The following components have visually distinct differences between healthy and parkinson-recordings: - Jitter.Abs, Jitter.perc, PPQ, Shimmer, Shimmer.dB, APQ, DFA, Shimmer.APQ3, Shimmer.APQ5, Shimmer.DDA, Spread1

```
#selecting only components with clear distinctions
train_dat_special <- train_dat1 %>% select(Jitter.Abs, Jitter.perc, PPQ, Shimmer, Shimmer.dB, Shimmer.A
test_dat_special <- test_dat1 %>% select(Jitter.Abs, Jitter.perc, PPQ, Shimmer, Shimmer.dB, Shimmer.APQ

#tuning knn
ks <- seq(1, 96,2)
accuracy <- map_df(ks, function(k){
  fit <- knn3(status ~ ., data = train_dat_special, k = k)
  y_hat <- predict(fit, test_dat1, type="class")
  cm_train <- confusionMatrix(y_hat, test_dat_special$status)
  test_error <- cm_train$overall["Accuracy"]
  tibble(test = test_error)
})
accuracy
```

```
## # A tibble: 48 x 1
##     test
##    <dbl>
##  1 0.806
##  2 0.755
##  3 0.816
##  4 0.827
##  5 0.878
##  6 0.857
##  7 0.857
##  8 0.847
##  9 0.867
## 10 0.878
## # ... with 38 more rows
```

```
ks[which.max(accuracy$test)] #accuracy is highest at k = 9
```

```
## [1] 9
```

```
#using k = 9
fit.knn.s <- knn3(status ~ ., data = train_dat_special, k = 9)
y_hat_knn_s <- predict(fit.knn.s, test_dat1, type="class")
cm_test_knn_s <- confusionMatrix(y_hat_knn_s, test_dat_special$status)
cm_test_knn_s$overall["Accuracy"]
```

```
## Accuracy
## 0.877551
```

```
#accuracy at 0.8776

#tuning F1 score
F.value_s <- map_df(ks, function(k){
  fit <- knn3(status ~ ., data = train_dat_special, k = k)
  y_hat <- predict(fit, test_dat, type="class")
  cm_train <- confusionMatrix(y_hat, test_dat_special$status)
  Fvalue.s <- F_meas(y_hat,test_dat1$status)

  tibble(Fvalue.s)
})
F.value_s[5,]
```

```
## # A tibble: 1 x 1
##   Fvalue.s
##      <dbl>
## 1    0.739
```

```
ks[which.max(F.value_s$Fvalue.s)] #when k = 9, F1 score is 0.73913 (highest)
```

```
## [1] 9
```

```
#inputting the results
acc_results1 <- bind_rows(acc_results,
                          data.frame (Method = "Selected predictors, Tuned KNN (k=9)",
                           Accuracy = cm_test_knn_s$overall["Accuracy"],
                          F1.score  = F_meas(y_hat_knn_s,test_dat1$status)))


acc_results1
```

```
## # A tibble: 2 x 3
##   Method                                  Accuracy F1.score
## * <chr>                                      <dbl>    <dbl>
## 1 Tuned KNN (k=5)                            0.827    0.585
## 2 Selected predictors, Tuned KNN (k=9)    0.878    0.739
```

Therefore, using components that are visually distinct can offer better accuracy and F1 scores to the results.

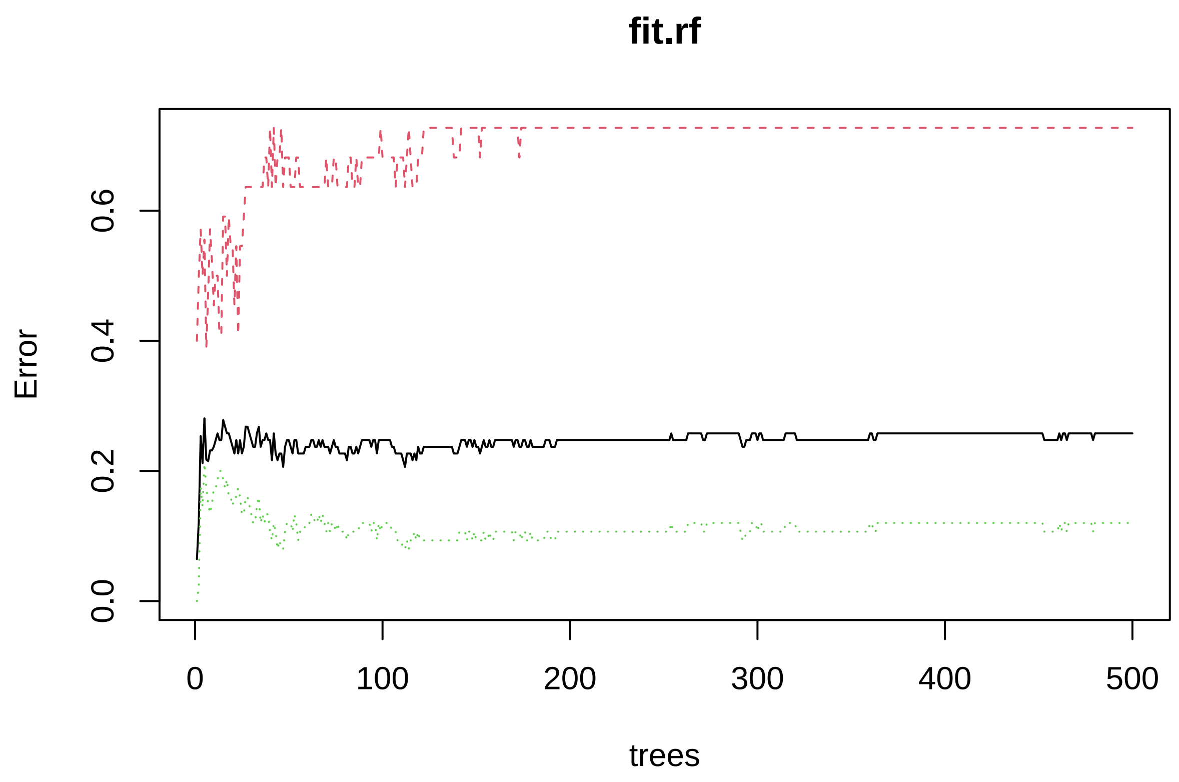
5. Fitting models: Random forest

Random forest (RF) is a classification method that is a step up from classification trees, which generates a single decision tree from the data. To address the shortcomings of decision trees, the rf method averages multiple decision trees. The flexibility in the method comes from the mtry parameter, which is the number of variables randomly sampled as candidates at each split.

Here, random forest is used for *selected* predictors.

```
#random forest without tuning
fit.rf <- randomForest(status~., data = train_dat_special)
fit.rf;plot(fit.rf)
```

```
##
## Call:
##  randomForest(formula = status ~ ., data = train_dat_special)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 25.77%
## Confusion matrix:
##   0  1 class.error
## 0 6 16   0.7272727
## 1 9 66   0.1200000
```



```
y_hat_rf_s <- predict(fit.rf, test_dat1, type = "class")
cm_basic_rf <- confusionMatrix(y_hat_rf_s, test_dat1$status)
```

Plotting: The red curve is the error rate for the Setosa class, the green and blue curves above are for Versicolor and Virginica while the black curve is the Out-of-Bag error rate

5.1 Tuning randomForest

Here, the algorithm is tuned for best accuracy by considering various levels of mtry.

```
#tuning mtry parameter
m <- c(1:11)
fit.rf1 <- sapply(m, function(m){
  fit <- randomForest(status~., data = train_dat_special, mtry = m)
  y_hat <- predict(fit, test_dat1, type="class")
  cm_train <- confusionMatrix(y_hat, test_dat_special$status)
  test_error <- cm_train$overall["Accuracy"]
  tibble(test_error)
  })

as.data.frame(fit.rf1)
```

```
##          test_error test_error.1 test_error.2 test_error.3 test_error.4
## Accuracy  0.8367347    0.8673469     0.877551    0.8673469    0.8673469
##          test_error.5 test_error.6 test_error.7 test_error.8 test_error.9
## Accuracy    0.8673469     0.877551     0.877551     0.877551    0.8673469
##          test_error.10
## Accuracy      0.877551
```

```
m[which.max(as.data.frame(fit.rf1))] #accuracy highest when mtry = 7
```

```
## [1] 3
```

```
#checking results
fit.rf.test <- randomForest(status~., data = train_dat_special, mtry = 7)
y_hat.rf <- predict(fit.rf.test, test_dat1, type="class")
cm_test_rf_s <- confusionMatrix(y_hat.rf, test_dat_special$status)
cm_test_rf_s
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 19  5
##          1  7 67
##
##                Accuracy : 0.8776
##                  95% CI : (0.7959, 0.9351)
##     No Information Rate : 0.7347
##     P-Value [Acc > NIR] : 0.000474
```

```
##
##                   Kappa : 0.678
##
##  Mcnemar's Test P-Value : 0.772830
##
##             Sensitivity : 0.7308
##             Specificity : 0.9306
##          Pos Pred Value : 0.7917
##          Neg Pred Value : 0.9054
##              Prevalence : 0.2653
##          Detection Rate : 0.1939
##    Detection Prevalence : 0.2449
##       Balanced Accuracy : 0.8307
##
##        'Positive' Class : 0
##
```

```
cm_test_rf_s$overall["Accuracy"]
```

```
## Accuracy
## 0.877551
```

```
F_meas(y_hat.rf, test_dat_special$status)
```

```
## [1] 0.76
```

```
#inputting the results
acc_results2 <- bind_rows(acc_results1,
                          data.frame (Method = c("Selected predictors, RF",
                                      "Selected predictors, Tuned RF (mtry=7)"),
                           Accuracy = c(cm_basic_rf$overall["Accuracy"],
                           cm_test_rf_s$overall["Accuracy"]),
                          F1.score  = c(F_meas(y_hat_rf_s,test_dat1$status),
                          F_meas(y_hat.rf,test_dat1$status))))

acc_results2
```

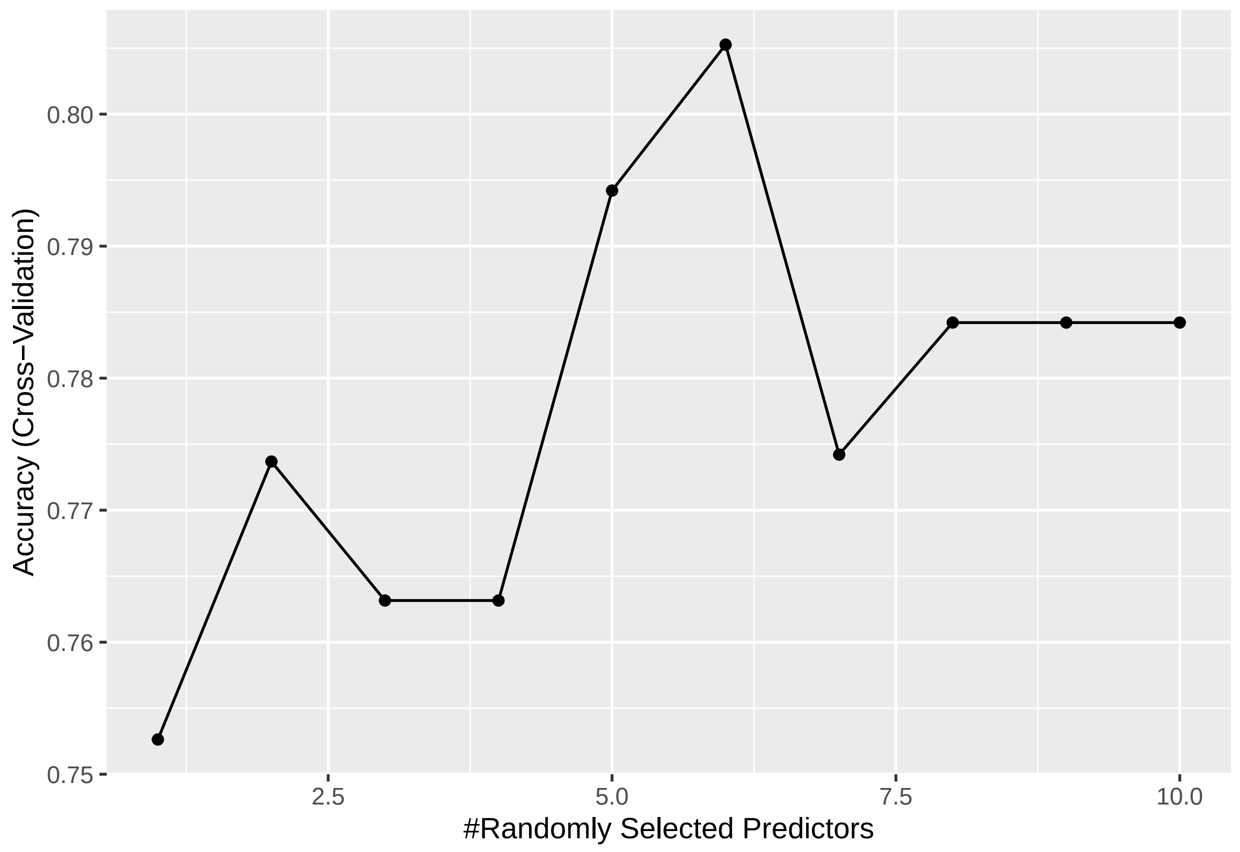
```
## # A tibble: 4 x 3
##   Method                                Accuracy F1.score
## * <chr>                                    <dbl>    <dbl>
## 1 Tuned KNN (k=5)                          0.827    0.585
## 2 Selected predictors, Tuned KNN (k=9)     0.878    0.739
## 3 Selected predictors, RF                  0.847    0.694
## 4 Selected predictors, Tuned RF (mtry=7)   0.878    0.76
```

When compared to without mtry tuning, there is much better accuracy and F1 score.

5.2 Further tuning for random forests

Here, various fine tuning for random forest is performed.

```r
#train control - 5 fold
control <- trainControl(method="cv", number = 5)
grid <- data.frame(mtry = c(1:10))
train_rf <-  train(status~., data = train_dat_special,
                   method = "rf",
                   ntree = 150,
                   trControl = control,
                   tuneGrid = grid,
                   nSamp = 5000)
ggplot(train_rf)
```



```r
train_rf
```

```
## Random Forest
##
## 97 samples
## 11 predictors
##  2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 78, 77, 78, 78, 77
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.7526316  0.2377882
##    2    0.7736842  0.2913093
##    3    0.7631579  0.3199903
##    4    0.7631579  0.3114806
##    5    0.7942105  0.3820569
##    6    0.8052632  0.3786895
##    7    0.7742105  0.3235806
##    8    0.7842105  0.3402986
##    9    0.7842105  0.3694082
##   10    0.7842105  0.3649140
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

```
fit.rf2 <- randomForest(status~., data = train_dat_special,
                       minNode = train_rf$bestTune$mtry)
y_hat_rf_s2 <- predict(fit.rf2, test_dat_special)
cm_test_rf2 <- confusionMatrix(y_hat_rf_s2, test_dat_special$status) #acc at 0.8673
cm_test_rf2$overall["Accuracy"]
```

```
## Accuracy
## 0.877551
```

```
F_meas(y_hat_rf_s2,test_dat1$status)
```

```
## [1] 0.75
```

```
#inputting the results
acc_results3 <- bind_rows(acc_results2,
                          data.frame (Method = "Selected predictors, Tuned RF (5 fold with varying mtry
                           Accuracy = cm_test_rf2$overall["Accuracy"],
                          F1.score  = F_meas(y_hat_rf_s2,test_dat1$status)))

acc_results3
```

```
## # A tibble: 5 x 3
##   Method                                                  Accuracy F1.score
## * <chr>                                                      <dbl>    <dbl>
## 1 Tuned KNN (k=5)                                            0.827    0.585
## 2 Selected predictors, Tuned KNN (k=9)                       0.878    0.739
## 3 Selected predictors, RF                                    0.847    0.694
## 4 Selected predictors, Tuned RF (mtry=7)                     0.878    0.76
## 5 Selected predictors, Tuned RF (5 fold with varying mtry)   0.878    0.75
```

6. Ensembles

Ensembles are a way to combine different algorithms to produce a better accuracy than a single algorithm. Here, the analysis will combine the top two methods to produce an ensemble. However, this process does not always ensure a better accuracy, as we shall see.

```
#preparing predictions for ensemble comparison
pred <- cbind(y_hat_knn_s, y_hat.rf)
pred <- pred - 1 #not sure why table produces 1 and 2s

#Ensemble Calculation
park <- rowMeans(pred == "1")
y_hat <- ifelse(park > 0.5, "1", "0")
mean(as.factor(y_hat) == test_dat1$status)
```

```
## [1] 0.8673469
```

```
#Accuracy at 0.878, lower than tuned randomForest (more methods needed)
#inputting the results
acc_results4 <- bind_rows(acc_results3,
                          data.frame (Method = "Ensemble of method 2 & 3",
                           Accuracy = mean(as.factor(y_hat) == test_dat1$status),
                          F1.score  = NA))

acc_results4
```

```
## # A tibble: 6 x 3
##   Method                                                  Accuracy F1.score
## * <chr>                                                      <dbl>    <dbl>
## 1 Tuned KNN (k=5)                                            0.827    0.585
## 2 Selected predictors, Tuned KNN (k=9)                       0.878    0.739
## 3 Selected predictors, RF                                    0.847    0.694
## 4 Selected predictors, Tuned RF (mtry=7)                     0.878    0.76
## 5 Selected predictors, Tuned RF (5 fold with varying mtry)   0.878    0.75
## 6 Ensemble of method 2 & 3                                   0.867   NA
```

Above, the ensemble method has a lower accuracy that random forests with tuning.

Conclusion

In conclusion, after comparing two methods in classification, KNN and RF, we see that the RF method with tuning has the highest Accuracy 0.877551 and F1 score of 0.76 for predicting healthy versus Parkinson's patients. However, this accuracy is the same for the tuned KNN and Tuned RF with varying mtry. Yet, this algorithm yields the highest F1 score.

Limitations

There are two limitations involved in this analysis.

First, while the dataset has many predictors, the number of outcomes is limited ($n = 195$). Furthermore, the prevalence of status = 1 (parkinsons) is 77% of the overall figure. This could lead to lower F1 scores of sensitivity and specificity, which could affect the efficacy of the model.

Second, due to the nature of the course, only two main algorithms were used. The caret package offers a wide range of different training options that can be applied. Using these options, together with an ensemble, could yield a higher accuracy for the overall result.

Future work

Future work could address the above limitations. By working on a larger dataset, reducing the prevalence of one outcome over the other, and including more algorithms, the accuracy could increase and therefore lead to better detection of parkinson's disease in patients.

Developing this algorithm with these recommendations would be crucial in diagnosing parkinson's disease, and even work for other related pathologies. A powerful model could lead to early intervention for patients afflicted with these diseases and empower the psychological and medical community better.

References

Max A. Little, Patrick E. McSharry, Eric J. Hunter, Lorraine O. Ramig (2008), 'Suitability of dysphonia measurements for telemonitoring of Parkinson's disease', IEEE Transactions on Biomedical Engineering (to appear).

Irizarry, R. (2020) Introduction to Data Science. Retrieved from https://rafalab.github.io/dsbook/