Project2

Bet Baradaji, Jens Lagemann

2023-01-13

Separating noisy and clean ecgs

Loading the data

Loading the ECG data from the two noisy folders and the training2017 as clean data. Iterating through every file, calculating the spectral power density with pwelch and the autocorrelation using acf. Since noise in ECG data is mostly low frequency baseline wandering, power line artifacts and white noise, we particularly selected the amplitudes from 0-5 Hz, 50 Hz, and an average across all frequencies. For the correlation we limited it to a maximum lag of 30 to limit the dimensions and data. Finally we compiled a single data frame, labeling as "noisy" and "clean". To avoid having to repeat the data processing every time, we saved that dataframe to a csv.

```r
# load noisy data from two folders
f_noisy1 <- list.files("./EcgNOISY1")
f_noisy2 <- list.files("./ECGNOISY2")
f_noisy <- append(f_noisy1, f_noisy2)

# load clean data
f_clean <- list.files("./training2017", pattern="*.mat")

# iterate through data, making pwelch 0-5 and 50 and acf lag.max=30
noisy_data_p <- data.frame()
noisy_data_a <- data.frame()
for (f in f_noisy1){
  path <- paste("./EcgNOISY1/", f, sep = "")
  ecg <- readMat(path)$newval[1,]
  p <- pwelch(ecg, fs = 300)
  a <- acf(ecg, lag.max = 30, plot = FALSE)
  spec <- append(p$spec[1:6], p$spec[51])
  spec <- append(spec, mean(p$spec))
  noisy_data_p <- rbind(noisy_data_p, spec)
  noisy_data_a <- rbind(noisy_data_a, a$acf[2:31])
}

for (f in f_noisy2){
  path <- paste("./ECGNOISY2/", f, sep = "")
  ecg <- readMat(path)$newval[1,]
  p <- pwelch(ecg, fs = 300)
  a <- acf(ecg, lag.max = 30, plot = FALSE)
  spec <- append(p$spec[1:6], p$spec[51])
  spec <- append(spec, mean(p$spec))
  noisy_data_p <- rbind(noisy_data_p, spec)
  noisy_data_a <- rbind(noisy_data_a, a$acf[2:31])
}
```

```r
noisy_data_p <- na.omit(noisy_data_p)
colnames(noisy_data_p) <- c("Hz0", "Hz1", "Hz2", "Hz3", "Hz4", "Hz5", "Hz50", "WhiteNoise")
colnames(noisy_data_a) <- c(1:30)
noisy_data_p$class <- "noisy"
noisy_data_a$class <- "noisy"

# same for clean data
clean_data_p <- data.frame()
clean_data_a <- data.frame()
for (f in f_clean){
  path <- paste("./training2017/", f, sep = "")
  ecg <- readMat(path)$val[1,]
  p <- pwelch(ecg, fs = 300)
  a <- acf(ecg, lag.max = 30, plot = FALSE)
  spec <- append(p$spec[1:6], p$spec[51])
  spec <- append(spec, mean(p$spec))
  clean_data_p <- rbind(clean_data_p, spec)
  clean_data_a <- rbind(clean_data_a, a$acf[2:31])
}
colnames(clean_data_p) <- c("Hz0", "Hz1", "Hz2", "Hz3", "Hz4", "Hz5", "Hz50", "WhiteNoise")
colnames(clean_data_a) <- c(1:30)
clean_data_p$class <- "clean"
clean_data_a$class <- "clean"

# concat data
p_data <- rbind(na.omit(clean_data_p), na.omit(noisy_data_p))
p_data$class <- as.factor(p_data$class)
a_data <- rbind(na.omit(clean_data_a), na.omit(noisy_data_a))
a_data$class <- as.factor(a_data$class)
write.csv(p_data, "./p_data.csv", row.names = FALSE)
write.csv(a_data, "./a_data.csv", row.names = FALSE)
```

Training classifier

To train the classifier we first split the data randomly into a 70/30 split. We train two distinct linear svm, one on the spectral density data, one on the autocorrelation data. After training the performance is evaluated on the test data and a confusion matrix is calculated.

```r
if (!(exists("p_data"))){
  p_data <- read.csv("./p_data.csv")
}

if (!(exists("a_data"))){
  a_data <- read.csv("./a_data.csv")
}

# split train/test
split <- sample( c(TRUE, FALSE), nrow(p_data),
                 replace = TRUE, prob = c(0.7, 0.3))
p_training <- p_data[split, ]
p_test <- p_data[!split, ]

split <- sample( c(TRUE, FALSE), nrow(a_data),
                 replace = TRUE, prob = c(0.7, 0.3))
```

```r
a_training <- a_data[split, ]
a_test <- a_data[!split, ]
# train predictor
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
p_knn <- train(class ~ .,
             data = p_training,
             method = "svmLinear",
             trControl = trctrl,
             tuneLength = 10)

a_knn <- train(class ~ .,
               data = a_training,
               method = "svmLinear",
               trControl = trctrl,
               tuneLength = 10)

# test
p_knn_test <- predict(p_knn, p_test)
confusionMatrix(table(p_knn_test, p_test$class))
```

```
## Confusion Matrix and Statistics
##
##
## p_knn_test clean noisy
##      clean  2403   719
##      noisy    20  1659
##
##                Accuracy : 0.8461
##                  95% CI : (0.8356, 0.8562)
##     No Information Rate : 0.5047
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6913
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9917
##             Specificity : 0.6976
##          Pos Pred Value : 0.7697
##          Neg Pred Value : 0.9881
##              Prevalence : 0.5047
##          Detection Rate : 0.5005
##    Detection Prevalence : 0.6503
##       Balanced Accuracy : 0.8447
##
##        'Positive' Class : clean
##
```

```r
a_knn_test <- predict(a_knn, a_test)
confusionMatrix(table(a_knn_test, a_test$class))
```

```
## Confusion Matrix and Statistics
##
##
```

```
## a_knn_test clean noisy
##      clean  2570   280
##      noisy    94  2222
##
##                Accuracy : 0.9276
##                  95% CI : (0.9202, 0.9345)
##     No Information Rate : 0.5157
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8547
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9647
##             Specificity : 0.8881
##          Pos Pred Value : 0.9018
##          Neg Pred Value : 0.9594
##              Prevalence : 0.5157
##          Detection Rate : 0.4975
##    Detection Prevalence : 0.5517
##       Balanced Accuracy : 0.9264
##
##        'Positive' Class : clean
##
```

Classifying ECG data based on heart rythm

Preprocessing

Firstly we define functions of the preprocessing the ECG data. To filter the data and be able to extract features of the QRS complex, we apply a simplified model of the pan-tompkins algorithm. This consists of first applying a passband filter, trying to remove noise. As the R peak is one of the clearest mark of the heartbeat in ECG data, we focus on exaggerating those marks and extract them. This is done by applying a derivative filter, squaring the values to achieve only positive values and the integrating using a moving window average to restore the dynamics of the original signal. Once we have identified the R-peaks, we extract the sequence and calculate the features from it.

```r
pan_tompkins <- function(t_series, sampling_freq = 300){
  # passband filter
  fil_N <- 8
  fs <- 200 #sampling rate
  nf <- fs/2
  fc <- c(5, 32) # remove frequencies from 5Hz to 32Hz
  fc_norm <- fc/nf

  re_series <- resample(t_series, fs, sampling_freq)

  fir_filter <- fir1(fil_N, fc_norm, type="pass")
  f_x <- filtfilt(fir_filter, re_series)

  # derivative filter
  d_x <- c()
  for (i in 3:(length(f_x)-2)){
    d_x[i] <- (1/(8/nf)) * (-f_x[i-2] - 2*f_x[i-1] + 2*f_x[i+1] + f_x[i+1])
  }
  # integration filter by moving window average
  m_x <- c()
  for (i in 1:length(d_x)){
    m_x[i] <- mean(d_x[i:i+8])
  }

  # determine peaks
  peaks <- findpeaks(m_x, minpeakheight = mean(m_x, na.rm = TRUE),
                     minpeakdistance = 50)
  peaks_d <- data.frame(peaks)
  colnames(peaks_d) <- c("height", "loc", "start", "end")
  no_peaks <- m_x
  for (i in 1:length(peaks_d$loc)){
    p_start <- peaks_d$start[i]
    p_end <- peaks_d$end[i]
    no_peaks[p_start: p_end] <- c(NA * (p_end - p_start))
  }
  npk <- sd(no_peaks, na.rm = TRUE)
  spk <- mean(peaks_d$height)
  threshold1 <- npk + 0.25 * (spk - npk)
  threshold2 <- 0.5 * threshold1
  peaks_d <- peaks_d[peaks_d$height > threshold1,]
  peaks_d$loc <- peaks_d$loc
  return(sort(peaks_d$loc) * 5)
}

nn_features <- function(peak_seq){
  nn <- peak_seq[2:length(peak_seq)] - peak_seq[1:length(peak_seq) - 1]
  succ_dif <- nn[2:length(nn)] - nn[1:length(nn)-1]
  sdnn <- sd(nn)
  sdsd <- sd(succ_dif)
  pnn50 = length(succ_dif[abs(succ_dif) > 50])/length(succ_dif)
  pnn20 = length(succ_dif[abs(succ_dif) > 20])/length(succ_dif)
  features <- data.frame(sdnn, sdsd, pnn50, pnn20)
  return(features)
}
```

Loading the data

We load the data from the folder and prepare a data frame with the features. Again we save the data frame to a csv to be able to avoid this time intensive step when not necessary.

```
help("list.files")
dir <- "./training2017/"
files <- list.files(dir, pattern = "*.mat")
labels <- read.csv("./REFERENCE.csv", header = FALSE)
colnames(labels) <- c("file", "class")
ecg_set <- data.frame()
columns <- c("file", "class", "sdnn", "sdsd", "pnn50", "pnn20")
ecg_dataset = data.frame(matrix(nrow = 0, ncol = length(columns)))
for (i in 1:length(files)){
  # get file name
  recording <- labels$file[i]
  path <- paste(dir, recording, ".mat", sep = "")
  series <- readMat(path)$val[1,]
  r_peaks <- pan_tompkins(series)
  features <- nn_features(r_peaks)
  labels$sdnn[i] <- features$sdnn
  labels$sdsd[i] <- features$sdsd
  labels$pnn50[i] <- features$pnn50
  labels$pnn20[i] <- features$pnn20
}
write.csv(labels, "./feature_data.csv")
```

Training and testing

After again splitting into training/test at 70/30, we train a k-nearest neighbor algorithm on the data and evaluate on the test set using a confusion matrix. Since in this case we have lower dimensionality than in the autocorrelation case before, we decided on a knn approach rather than a linear-svm, since lower dimensional data is often harder to separate linearly, where in turn higher dimensional data is often sparse.

```
data <- read.csv("./feature_data.csv")
data$class <- as.factor(data$class)
data <- na.omit(data)
split <- sample( c(TRUE, FALSE), nrow(data), replace = TRUE, prob = c(0.7, 0.3))
training_data <- data[split, ]
test_data <- data[!split, ]
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
knn <- train(class ~ sdnn + sdsd + pnn50 + pnn20,
             data = training_data,
             method = "knn",
             trControl = trctrl,
             tuneLength = 10)
test_pred <- predict(knn, test_data[,4:7])
confusionMatrix(table(test_pred, test_data$class))
```

```
## Confusion Matrix and Statistics
##
##
## test_pred    ~    A    N    O
##         ~    2    0    0    0
##         A    6   38    7   53
##         N   42   77 1353  419
##         O   24  102  156  281
##
## Overall Statistics
##
##                Accuracy : 0.6539
##                  95% CI : (0.6351, 0.6723)
##     No Information Rate : 0.5922
##     P-Value [Acc > NIR] : 8.176e-11
##
##                   Kappa : 0.3
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
```

```
##                      Class: ~ Class: A Class: N Class: O
## Sensitivity          0.0270270  0.17512   0.8925   0.3732
## Specificity          1.0000000  0.97183   0.4847   0.8439
## Pos Pred Value       1.0000000  0.36538   0.7155   0.4991
## Neg Pred Value       0.9718530  0.92712   0.7564   0.7636
## Prevalence           0.0289063  0.08477   0.5922   0.2941
## Detection Rate       0.0007813  0.01484   0.5285   0.1098
## Detection Prevalence 0.0007813  0.04063   0.7387   0.2199
## Balanced Accuracy    0.5135135  0.57347   0.6886   0.6086
```