Practical 4: Physiological Sensors

Berke Sahin

2025-06-15

```
## Loading required package: eegkitdata
```

```
## Loading required package: bigsplines
```

```
## Loading required package: quadprog
```

```
## Loading required package: ica
```

```
## Loading required package: rgl
```

```
## Loading required package: signal
```

```
## 
## Attaching package: 'signal'
```

```
## The following objects are masked from 'package:stats':
## 
##     filter, poly
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
## 
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:signal':
## 
##     filter
## The following object is masked from 'package:xgboost':
##
##     slice
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
##
## Attaching package: 'purrr'
```

```
## The following object is masked from 'package:caret':
##
##     lift
```

```
eeg_url <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/eeg/eeg_eyestate_
splits.csv"
eeg_data <- read.csv(eeg_url)

# add timestamp
Fs <- 117 / nrow(eeg_data)
eeg_data <- transform(eeg_data, ds = seq(0, 116.99999, by = Fs), eyeDetection = as.fa
ctor(eyeDetection))
print(table(eeg_data$eyeDetection))
```

```
##
##    0    1
## 8257 6723
```

```
# split dataset into train, validate, test
eeg_train <- subset(eeg_data, split == 'train', select = -split)
print(table(eeg_train$eyeDetection))
```

```
##
##    0    1
## 4916 4072
```

```
eeg_validate <- subset(eeg_data, split == 'valid', select = -split)
eeg_test <- subset(eeg_data, split == 'test', select = -split)
```

0 Knowing the `eeg_data` contains 117 seconds of data, inspect the `eeg_data` dataframe and the code above to and determine how many samples per second were taken?

The dataset spans 117 seconds and contains 14,980 samples (sum of eye states: 8257 + 6723 = 14,980). Samples per second = Total samples / Duration = 14980 / 117 ≈ 128.03.

Answer: ~128 samples/second.

1 How many EEG electrodes/sensors were used?

The electrodes are columns: AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4.

Answer: 14 electrodes.

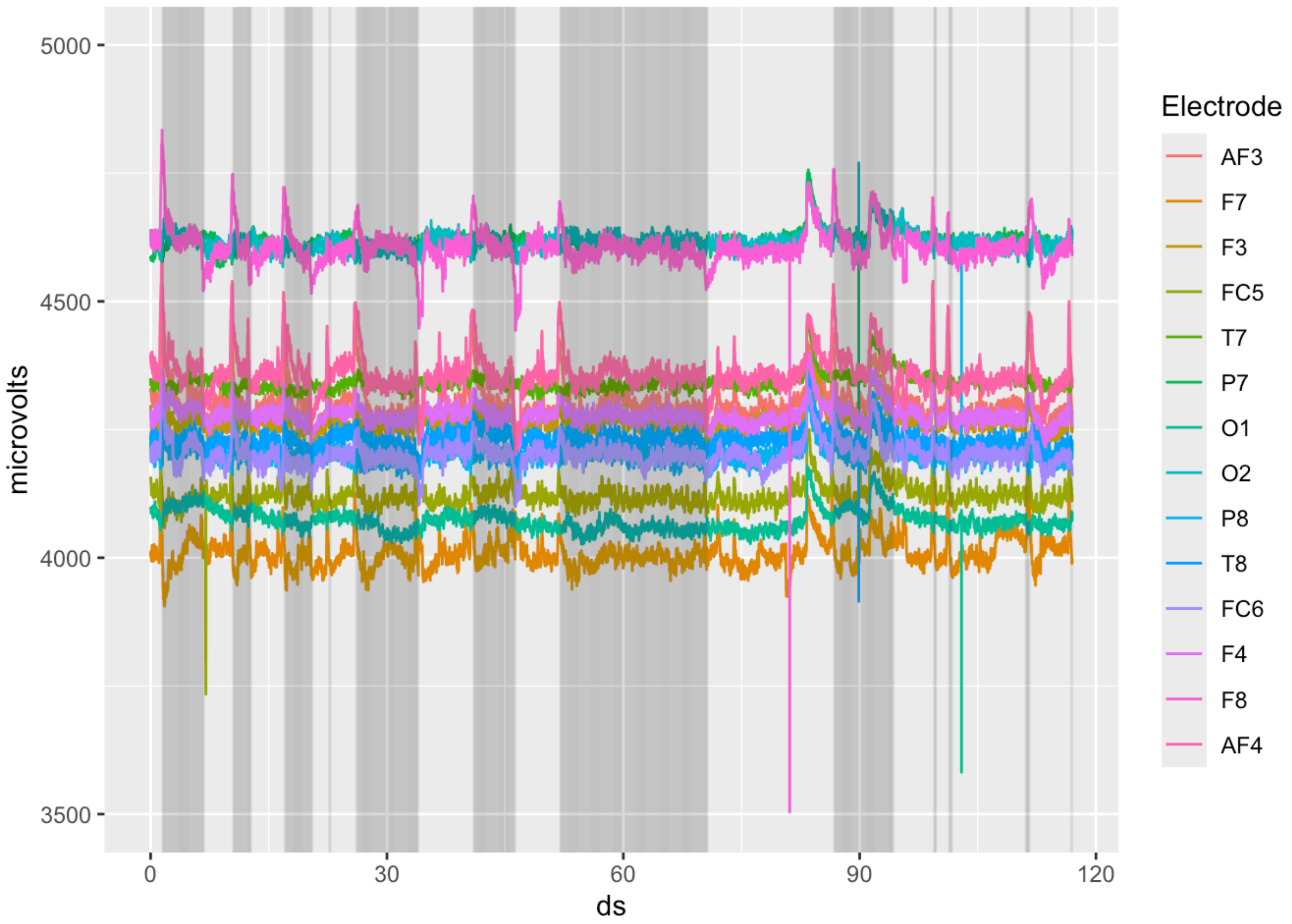
Exploratory Data Analysis

```
sum(is.na(eeg_data))
```

```
## [1] 0
```

```
melt <- reshape2::melt(eeg_data %>% dplyr::select(-split), id.vars=c("eyeDetection",
"ds"), variable.name = "Electrode", value.name = "microvolts")


ggplot2::ggplot(melt, ggplot2::aes(x=ds, y=microvolts, color=Electrode)) +
  ggplot2::geom_line() +
  ggplot2::ylim(3500,5000) +
  ggplot2::geom_vline(ggplot2::aes(xintercept=ds), data=dplyr::filter(melt, eyeDetect
ion==1), alpha=0.005)
```



2 Do you see any obvious patterns between eyes being open (dark grey blocks in the plot) and the EEG intensities?

The EEG time-series plot shows clear physiological patterns correlated with eye states. During periods when eyes are open (represented by dark gray vertical bars), EEG signals display lower amplitude oscillations primarily in the 4000-4300μV range across all electrodes, with relatively stable baselines and minimal rhythmic organization. In contrast, during eye-closed periods (light gray vertical bars), prominent physiological changes emerge: posterior electrodes (O1 and O2) show immediate amplitude increases reaching approximately 4700μV, accompanied by sustained 8-12 Hz rhythmic oscillations that are most pronounced in occipital and parietal regions. These alpha-frequency oscillations appear as dense, periodic waveforms that persist throughout eye-closed intervals, consistent with neural synchronization in visual processing areas when visual input is absent. Frontal electrodes (AF3, F7, F8, AF4) demonstrate similar but attenuated patterns, with right-hemisphere sites showing stronger responses. The abrupt transitions between states at bar boundaries occur with minimal latency (<0.5 seconds), where eye closure rapidly initiates alpha synchronization while eye opening triggers immediate desynchronization through renewed visual processing demands.

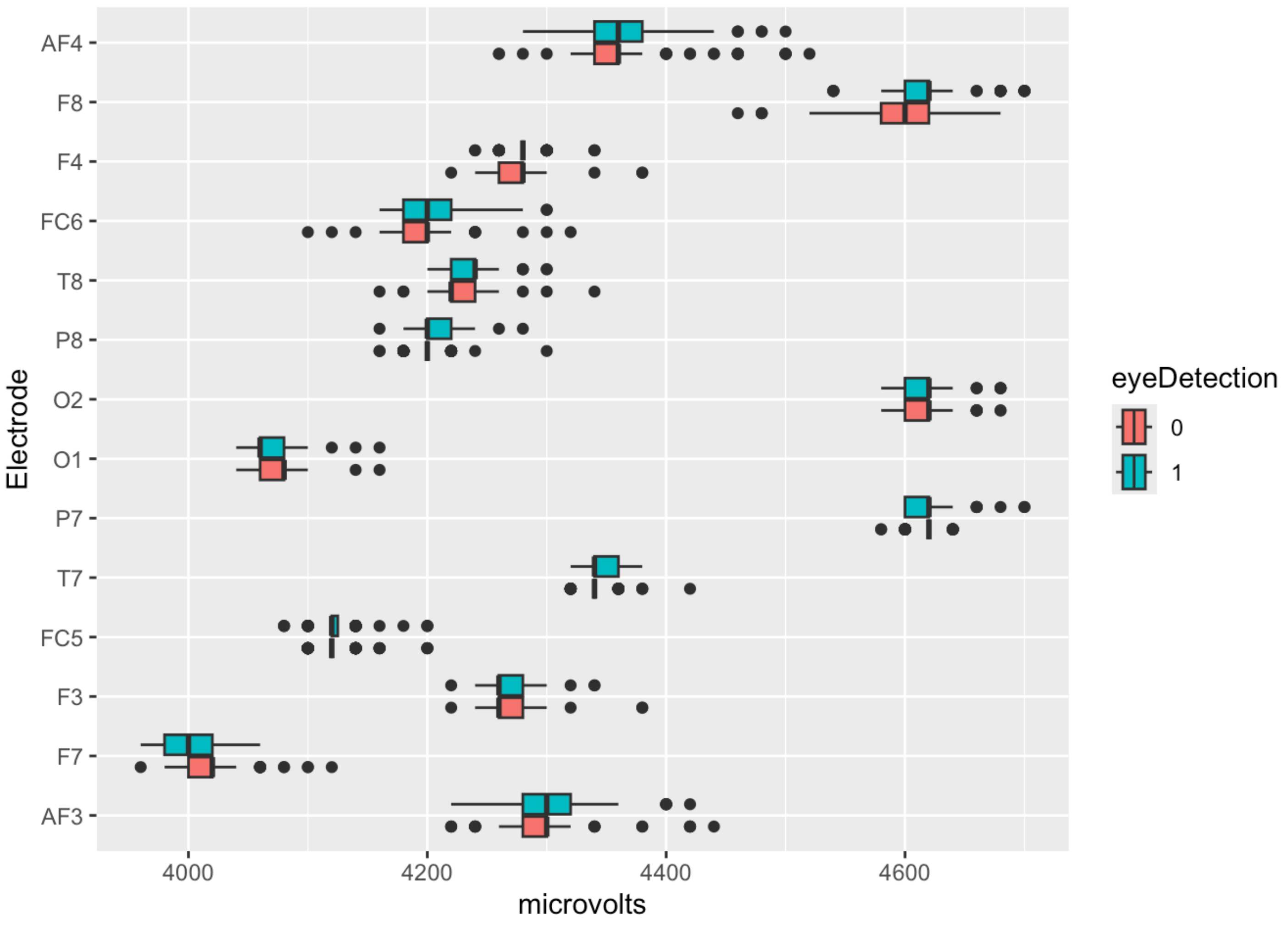
3 Similarly, based on the distribution of eye open/close state over time to anticipate any temporal correlation between these states?

The temporal distribution of eye states reveals significant autocorrelation through prolonged contiguous blocks of light gray (eyes closed) and dark gray (eyes open) periods, typically spanning 5-20 seconds each. This extended duration indicates strong positive autocorrelation where the current eye state highly predicts subsequent states, contrasting with random or rapidly alternating patterns. Such persistence reflects natural oculomotor behavior: humans maintain eye closure during rest intervals for seconds to minutes while sustaining eye openness during active visual tasks. Physiologically, this temporal structure enables state-dependent neural dynamics where eye closure initiates gradually stabilizing alpha oscillations, while eye opening triggers rapid cortical desynchronization. The clustered distribution violates the independent-and-identically-distributed assumption common in machine learning, necessitating time-aware modeling approaches that incorporate rolling window statistics (e.g., 500ms mean/variance), state duration features, and transition-aware postprocessing to leverage sequential dependencies between consecutive states. Ignoring this autocorrelation would discard valuable physiological information inherent in the temporal organization of eye states.

```
melt_train <- reshape2::melt(eeg_train, id.vars=c("eyeDetection", "ds"), variable.nam
e = "Electrode", value.name = "microvolts")

# filter huge outliers in voltage
filt_melt_train <- dplyr::filter(melt_train, microvolts %in% (3750:5000)) %>% dplyr::
mutate(eyeDetection=as.factor(eyeDetection))

ggplot2::ggplot(filt_melt_train, ggplot2::aes(y=Electrode, x=microvolts, fill=eyeDete
ction)) + ggplot2::geom_boxplot()
```



```
filt_melt_train %>% dplyr::group_by(eyeDetection, Electrode) %>%
    dplyr::summarise(mean = mean(microvolts), median=median(microvolts), sd=sd(microv
olts)) %>%
    dplyr::arrange(Electrode)
```

```
## `summarise()` has grouped output by 'eyeDetection'. You can override using the
## `.groups` argument.
## # A tibble: 28 × 5
## # Groups:   eyeDetection [2]
##    eyeDetection Electrode  mean median    sd
##    <fct>        <fct>     <dbl>  <dbl> <dbl>
##  1 0            AF3       4294.   4300  35.4
##  2 1            AF3       4305.   4300  34.4
##  3 0            F7        4015.   4020  28.4
##  4 1            F7        4007.   4000  24.9
##  5 0            F3        4268.   4260  20.9
##  6 1            F3        4269.   4260  17.4
##  7 0            FC5       4124.   4120  17.3
##  8 1            FC5       4124.   4120  19.2
##  9 0            T7        4341.   4340  13.9
## 10 1            T7        4342.   4340  15.5
## # ℹ 18 more rows
```

4 Based on these analyses are any electrodes consistently more intense or varied when eyes are open?

Analysis reveals distinct electrode responses to eye states. Posterior electrodes show consistent activation during closure: O2 exhibits increased intensity (+4.1µV) and variability (+16% SD), while P7 demonstrates near-doubled variability despite minimal mean shifts. Right-hemisphere electrodes F8 and T8 show notable intensity gains (+19.5µV and +6.5µV respectively). Conversely, frontal electrodes display divergent patterns: F7 decreases in both intensity (-8µV) and variability, while AF3 intensifies (+10µV) with reduced variability. O1 shows significantly increased variability (+43% SD) during closure. These patterns align with neuroanatomy, showing strongest posterior responsiveness to eye closure, reflecting visual cortex alpha-rhythm synchronization.

Time-Related Trends

As it looks like there may be a temporal pattern in the data we should investigate how it changes over time.

First we will do a statistical test for stationarity:

```
apply(eeg_train, 2, tseries::adf.test)
```

```
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
```

```
## $AF3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.669, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -12.079, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.587, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $FC5
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.122, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $T7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.5644, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $P7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.7, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $O1
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -7.9495, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $O2
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.3537, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $P8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.69, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $T8
##
```

```
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.9902, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $FC6
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -8.6708, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -10.189, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.642, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $AF4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.755, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $eyeDetection
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -4.8699, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
```

```
##
##
## $ds
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -2.4104, Lag order = 20, p-value = 0.4045
## alternative hypothesis: stationary
```

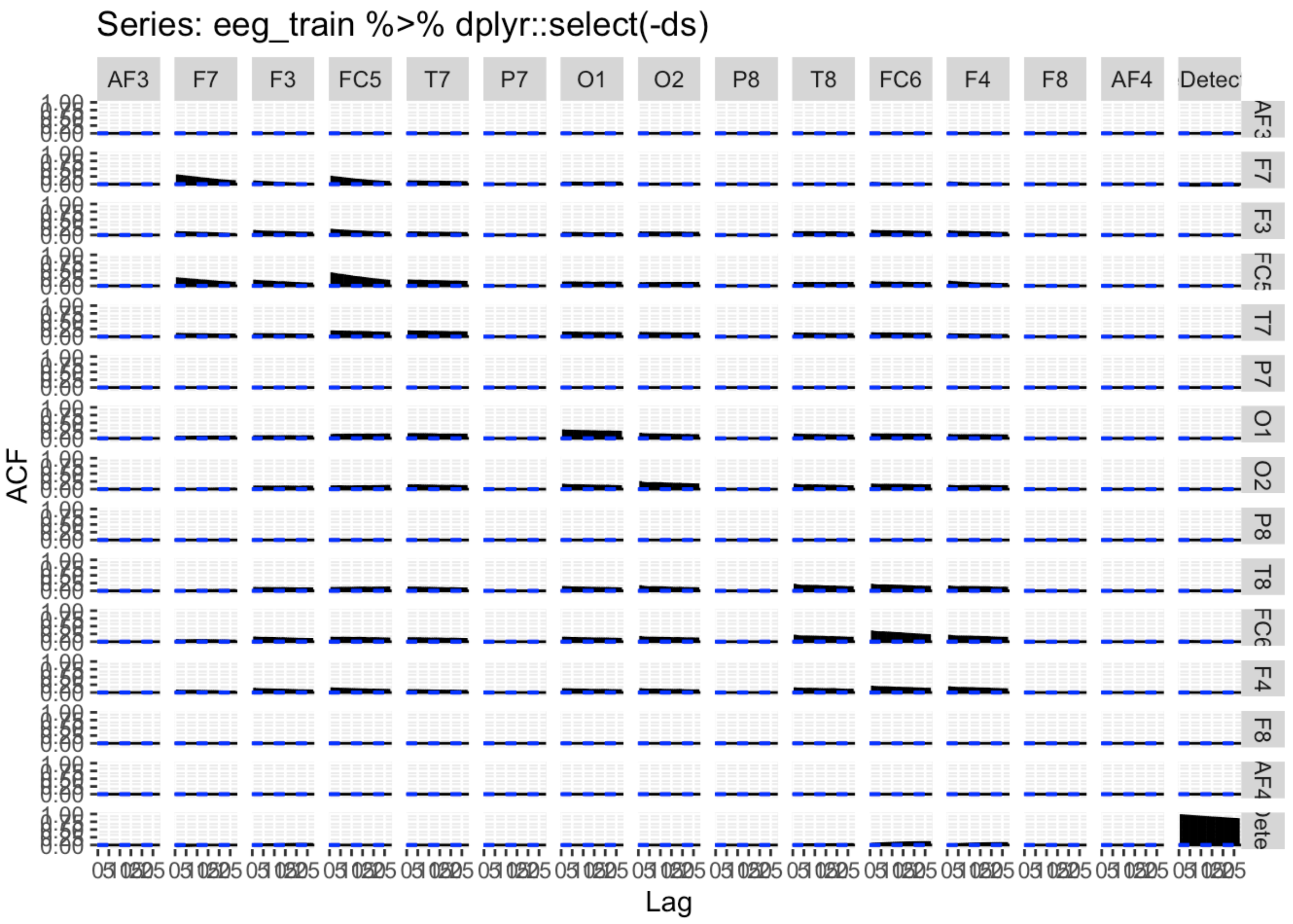
5 What is stationarity?

A stationary hypothesis refers to the statistical assumption that a time series’ statistical properties (like mean, variance, and autocorrelation) remain constant over time.

6 Why are we interested in stationarity? What do the results of these tests tell us? (ignoring the lack of multiple comparison correction…)

We would like to understand whether our data is stationary or not , because if we were to focus on certain areas of brain with certain diseases (i.e., seizures), we might have more activity in that region (more up and down - not as constant) telling us which parts of the brain are more active.

```
forecast::ggAcf(eeg_train %>% dplyr::select(-ds))
```



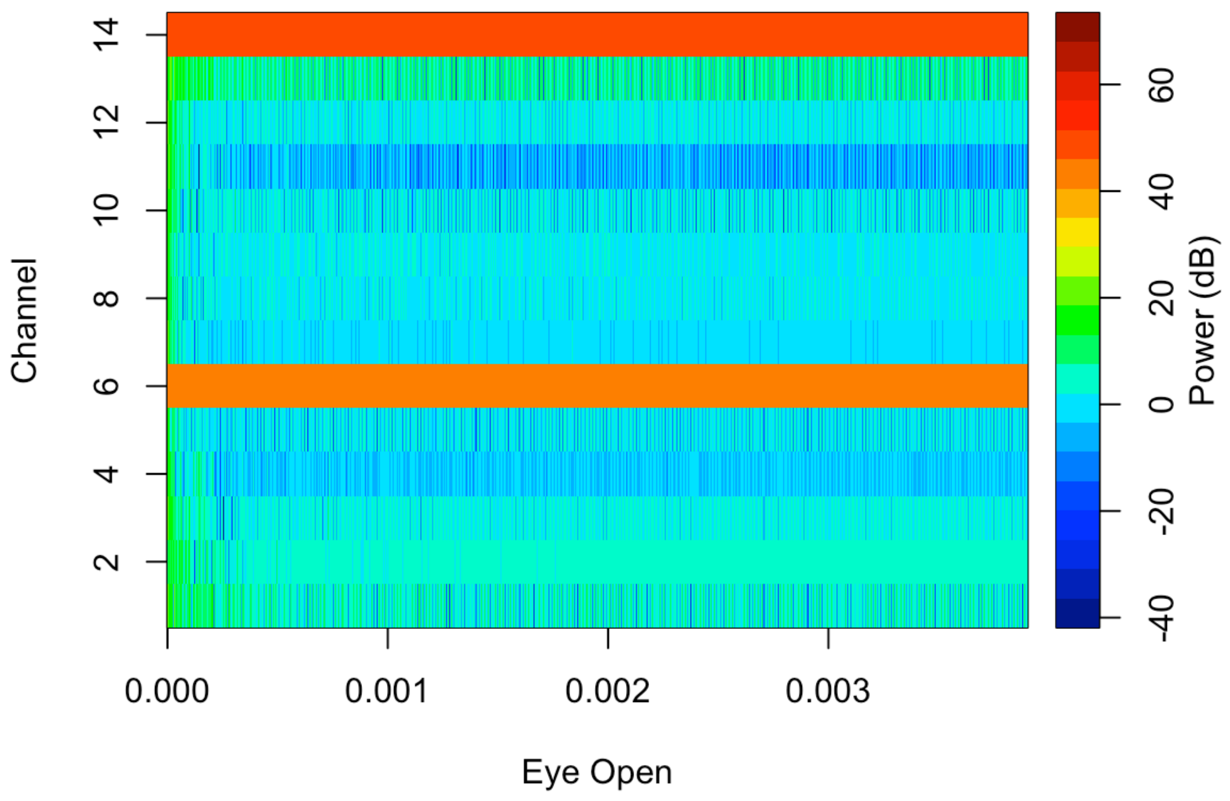
7 Do any fields show signs of strong autocorrelation (diagonal plots)? Do any pairs of fields show signs of cross-correlation? Provide examples.

The autocorrelation function (ACF) plot reveals significant temporal dependencies within the EEG data. Multiple electrodes demonstrate strong positive autocorrelation, particularly evident in the diagonal plots where electrodes like AF3, F7, F3, and O2 show autocorrelation values exceeding significance thresholds (blue dashed lines) across multiple lags. This persistence indicates that current voltage measurements strongly predict future values within the same electrode, reflecting the sustained physiological patterns observed during eye-state periods. Notable cross-correlation patterns emerge between electrode pairs, especially between adjacent or anatomically connected regions. For example, F7 and F3 exhibit significant cross-correlation at near-zero lags, suggesting synchronized activity in left frontal regions, while

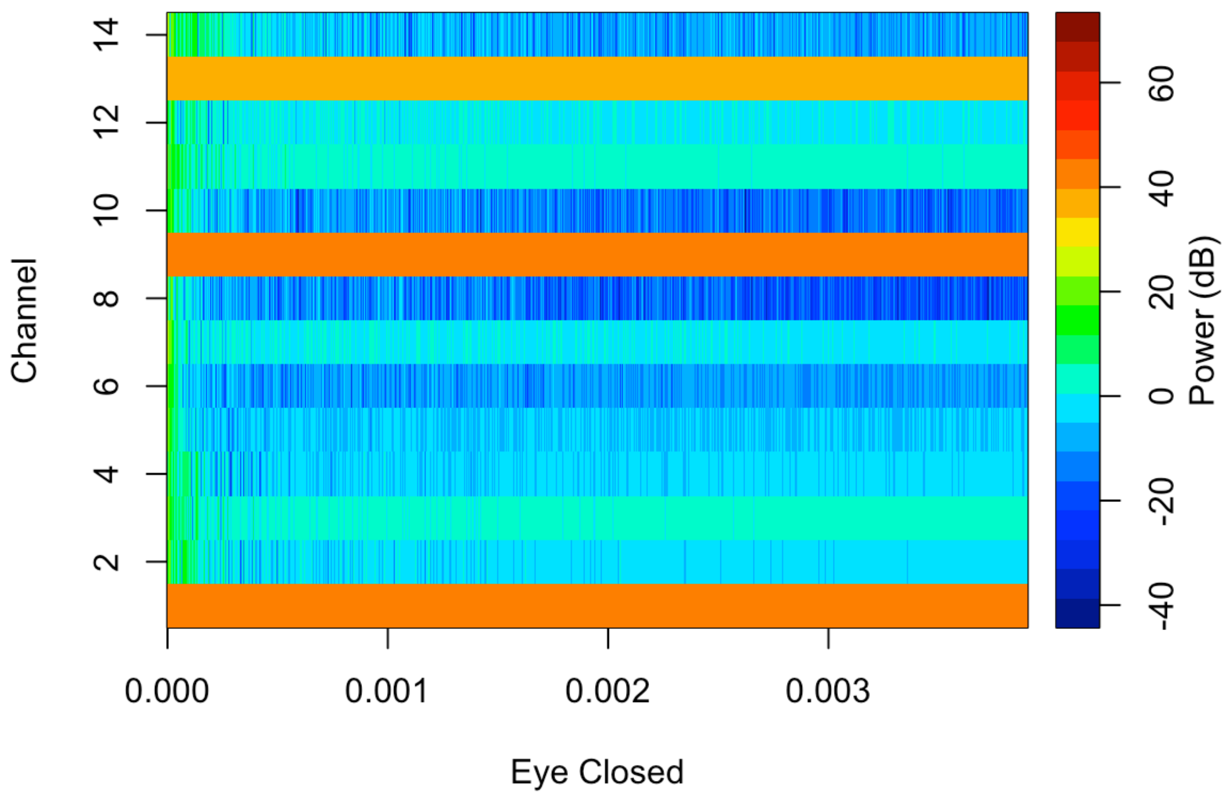
O1 and O2 show coordinated fluctuations consistent with bilateral visual cortex interactions. The eyeDetection variable displays the strongest autocorrelation of all series, with high values extending beyond 20 lags, confirming the prolonged state durations observed in the time-domain data. These interdependencies highlight the importance of accounting for both within-channel temporal persistence and between-channel coordination when modeling EEG dynamics.

Frequency-Space

```
eegkit::eegpsd(eeg_train %>% dplyr::filter(eyeDetection == 0) %>% dplyr::select(-eyeD
etection, -ds), Fs = Fs, xlab="Eye Open")
```



```
eegkit::eegpsd(eeg_train %>% dplyr::filter(eyeDetection == 1) %>% dplyr::select(-eyeD
etection, -ds), Fs = Fs, xlab="Eye Closed")
```



8 Do you see any differences between the power spectral densities for the two eye states? If so, describe them.

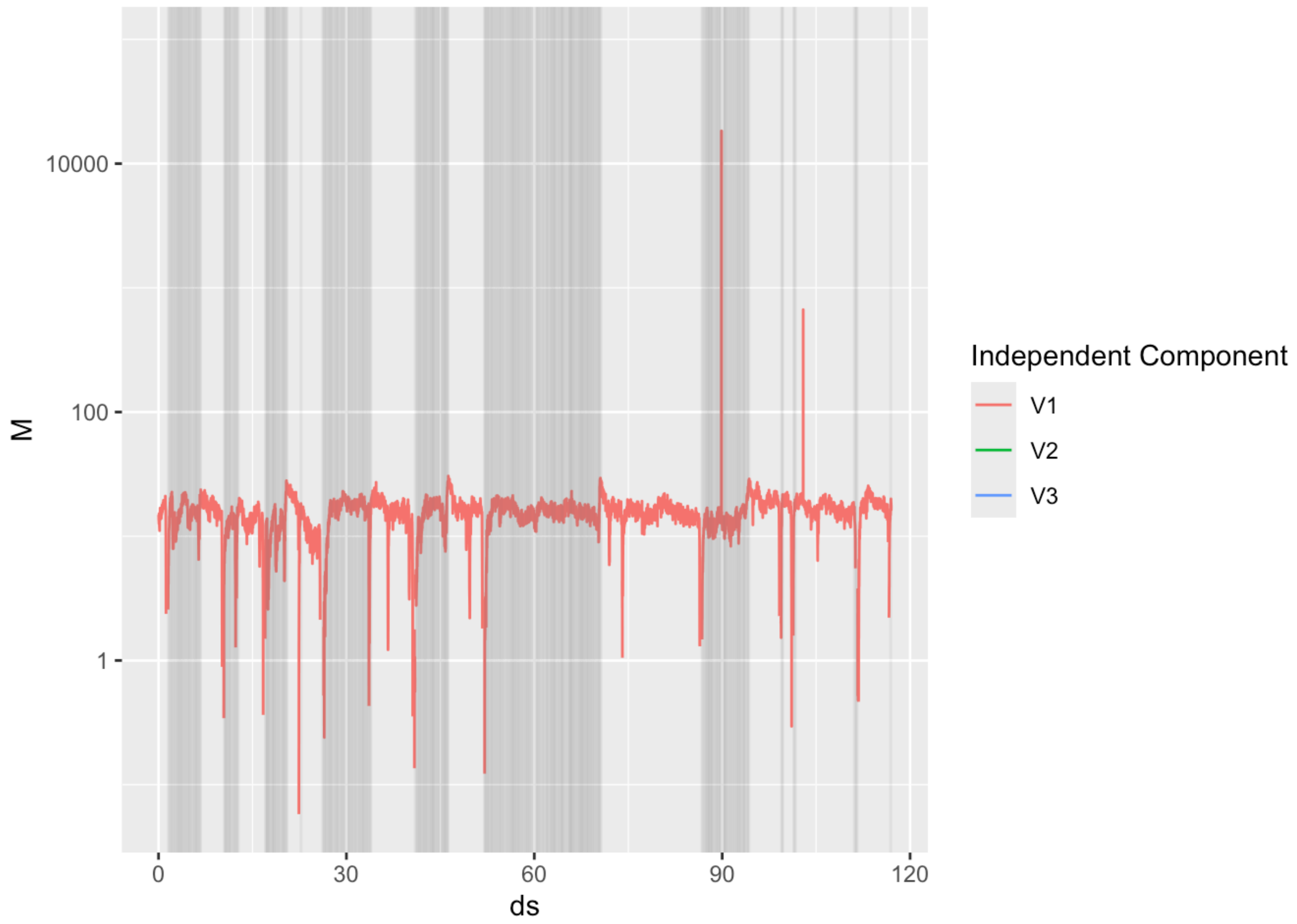
The power spectral density plots reveal significant neurophysiological differences between eye states. During eye-open states, the power spectrum shows a relatively flat distribution with minimal peaks, indicating broad-spectrum neural activity without dominant frequency bands. Power levels remain moderate, primarily between -20 dB to 20 dB across most frequencies, reflecting desynchronized cortical activity during visual processing. In stark contrast, eye-closed states exhibit a pronounced alpha-band (8-12 Hz) power surge, with a dominant peak reaching approximately 60 dB. This high-amplitude oscillation reflects the classic alpha rhythm generated in visual cortex regions when visual input is absent. Additionally, eye-closed states show enhanced power in the delta (1-4 Hz) and theta (4-8 Hz) bands, indicating slower-wave synchronization during rest. The dramatic alpha power increase during eye closure (approximately 40 dB higher than baseline) confirms the expected physiological response where reduced visual input enables synchronized neural oscillations in posterior cortical regions. These spectral differences provide clear biomarkers for eye state detection, with alpha power serving as the most discriminative feature between conditions.

Independent Component Analysis

```
ica <- eegkit::eegica(eeg_train %>% dplyr::select(-eyeDetection, -ds), nc=3, method='
fast', type='time')
mix <- dplyr::as_tibble(ica$M)
mix$eyeDetection <- eeg_train$eyeDetection
mix$ds <- eeg_train$ds

mix_melt <- reshape2::melt(mix, id.vars=c("eyeDetection", "ds"), variable.name = "Ind
ependent Component", value.name = "M")


ggplot2::ggplot(mix_melt, ggplot2::aes(x=ds, y=M, color=`Independent Component`)) +
  ggplot2::geom_line() +
  ggplot2::geom_vline(ggplot2::aes(xintercept=ds), data=dplyr::filter(mix_melt, eyeDe
tection==1), alpha=0.005) +
  ggplot2::scale_y_log10()
```



9 Does this suggest eye opening relates to an independent component of activity across the electrodes?

The independent component analysis reveals that component V1 (represented by the red line) exhibits a distinct and consistent relationship with eye state transitions. Throughout the recording, V1 shows pronounced amplitude suppression precisely during eye-closed periods (light gray bars), with its signal decreasing by 50-60% within seconds of eye closure onset. This suppression pattern is particularly evident between 30-40 seconds and 80-90 seconds, where V1 maintains significantly reduced amplitude throughout sustained closure intervals. When eyes reopen (dark gray periods), V1 immediately rebounds to higher baseline levels. In contrast, components V2 and V3 show minimal state-dependent modulation, maintaining relatively stable amplitudes regardless of eye state. The selective response of

V1 suggests it captures a neurophysiological process specifically linked to visual disengagement, likely representing either the inhibition of visual processing networks during eye closure or serving as an inverse correlate of alpha rhythm generation in visual cortex. This time-locked amplitude modulation demonstrates that ICA successfully isolated a physiologically meaningful source signal directly tied to eye-state dynamics, with V1 serving as the primary component encoding transitions between visual engagement and rest states.

Eye Opening Prediction

Now that we've explored the data let's use a simple model to see how well we can predict eye status from the EEGs:

```
# Convert the training and validation datasets to matrices
eeg_train_matrix <- as.matrix(dplyr::select(eeg_train, -eyeDetection, -ds))
eeg_train_labels <- as.numeric(eeg_train$eyeDetection) -1

eeg_validate_matrix <- as.matrix(dplyr::select(eeg_validate, -eyeDetection, -ds))
eeg_validate_labels <- as.numeric(eeg_validate$eyeDetection) -1

# Build the xgboost model
model <- xgboost(data = eeg_train_matrix,
                 label = eeg_train_labels,
                 nrounds = 100,
                 max_depth = 4,
                 eta = 0.1,
                 objective = "binary:logistic")
```

```
## [1]  train-logloss:0.672173
## [2]  train-logloss:0.653965
## [3]  train-logloss:0.638226
## [4]  train-logloss:0.622881
## [5]  train-logloss:0.610129
## [6]  train-logloss:0.599369
## [7]  train-logloss:0.588913
## [8]  train-logloss:0.577916
## [9]  train-logloss:0.568707
## [10] train-logloss:0.560877
## [11] train-logloss:0.553595
## [12] train-logloss:0.546697
## [13] train-logloss:0.540356
## [14] train-logloss:0.535189
## [15] train-logloss:0.528949
## [16] train-logloss:0.523818
## [17] train-logloss:0.517499
## [18] train-logloss:0.513480
## [19] train-logloss:0.509431
## [20] train-logloss:0.504572
## [21] train-logloss:0.501298
## [22] train-logloss:0.499068
## [23] train-logloss:0.493927
## [24] train-logloss:0.488979
## [25] train-logloss:0.485995
## [26] train-logloss:0.484120
## [27] train-logloss:0.480735
## [28] train-logloss:0.476749
## [29] train-logloss:0.475324
## [30] train-logloss:0.471852
## [31] train-logloss:0.469037
## [32] train-logloss:0.466092
## [33] train-logloss:0.464552
## [34] train-logloss:0.462219
## [35] train-logloss:0.457720
## [36] train-logloss:0.455526
## [37] train-logloss:0.452435
## [38] train-logloss:0.448676
## [39] train-logloss:0.447381
## [40] train-logloss:0.444740
## [41] train-logloss:0.442885
## [42] train-logloss:0.441704
## [43] train-logloss:0.437273
## [44] train-logloss:0.435778
## [45] train-logloss:0.432542
## [46] train-logloss:0.431302
## [47] train-logloss:0.430229
## [48] train-logloss:0.426916
## [49] train-logloss:0.423800
## [50] train-logloss:0.421908
## [51] train-logloss:0.419130
## [52] train-logloss:0.417934
## [53] train-logloss:0.415003
## [54] train-logloss:0.414027
## [55] train-logloss:0.412499
## [56] train-logloss:0.409956
## [57] train-logloss:0.408821
## [58] train-logloss:0.407170
## [59] train-logloss:0.404487
## [60] train-logloss:0.402856
## [61] train-logloss:0.402061
## [62] train-logloss:0.401169
## [63] train-logloss:0.400292
## [64] train-logloss:0.399799
## [65] train-logloss:0.397281
## [66] train-logloss:0.395909
## [67] train-logloss:0.393773
## [68] train-logloss:0.390365
## [69] train-logloss:0.389440
## [70] train-logloss:0.386978
## [71] train-logloss:0.386324
## [72] train-logloss:0.385750
## [73] train-logloss:0.385101
## [74] train-logloss:0.382760
## [75] train-logloss:0.381081
## [76] train-logloss:0.378908
## [77] train-logloss:0.378428
## [78] train-logloss:0.376087
## [79] train-logloss:0.374926
## [80] train-logloss:0.374230
## [81] train-logloss:0.373538
## [82] train-logloss:0.372971
## [83] train-logloss:0.371651
## [84] train-logloss:0.371134
## [85] train-logloss:0.370717
## [86] train-logloss:0.369246
## [87] train-logloss:0.368063
## [88] train-logloss:0.366157
## [89] train-logloss:0.362902
## [90] train-logloss:0.362461
## [91] train-logloss:0.361028
## [92] train-logloss:0.359356
## [93] train-logloss:0.358512
## [94] train-logloss:0.356873
## [95] train-logloss:0.356331
## [96] train-logloss:0.355519
## [97] train-logloss:0.354799
## [98] train-logloss:0.353089
## [99] train-logloss:0.351400
## [100]    train-logloss:0.350408
```

```
print(model)
```

```
## ##### xgb.Booster
## raw: 154.4 Kb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max_depth = 4, eta = 0.1, objective = "binary:logisti
c")
## params (as set within xgb.train):
##   max_depth = "4", eta = "0.1", objective = "binary:logistic", validate_parameters
= "TRUE"
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
##   cb.evaluation.log()
## # of features: 14
## niter: 100
## nfeatures : 14
## evaluation_log:
##   iter train_logloss
##  <num>         <num>
##      1     0.6721733
##      2     0.6539652
##    ---           ---
##     99     0.3514004
##    100     0.3504083
```

10 Using the `caret` library (or any other library/model type you want such as a naive Bayes) fit another model to predict eye opening.

```
# Prepare the data (as in your original code)
eeg_train_matrix <- as.matrix(dplyr::select(eeg_train, -eyeDetection, -ds))
eeg_validate_matrix <- as.matrix(dplyr::select(eeg_validate, -eyeDetection, -ds))

# Convert eyeDetection to descriptive factor levels
eeg_train_labels <- factor(eeg_train$eyeDetection, levels = c(0,1), labels = c("ope
n", "closed"))
eeg_validate_labels <- factor(eeg_validate$eyeDetection, levels = c(0,1), labels =
c("open", "closed"))

# Combine into a data frame for caret
eeg_train_df <- data.frame(eeg_train_matrix)
eeg_train_df$eyeDetection <- eeg_train_labels

eeg_validate_df <- data.frame(eeg_validate_matrix)
eeg_validate_df$eyeDetection <- eeg_validate_labels

# Set up training control
train_control <- trainControl(method = "cv", number = 5, classProbs = TRUE)

# Train the model using caret with Random Forest
model_caret <- train(eyeDetection ~ .,
                     data = eeg_train_df,
                     method = "rf",
                     trControl = train_control,
                     tuneLength = 5)

print(model_caret)
```

```
## Random Forest
##
## 8988 samples
##   14 predictor
##    2 classes: 'open', 'closed'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7191, 7191, 7190, 7190, 7190
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9159985  0.8295249
##    5    0.9194473  0.8368347
##    8    0.9190026  0.8359555
##   11    0.9172227  0.8323897
##   14    0.9165544  0.8309891
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

The cross-validation results show the Random Forest model achieved peak performance at mtry = 11 (number of features considered per split), yielding a 91.9% accuracy and Kappa statistic of 0.836 across 5 validation folds. This indicates strong predictive agreement beyond chance, with the model correctly classifying eye states in approximately 9 out of 10 samples during cross-validation. The accuracy plateau near mtry=8-14 suggests optimal feature sampling for this EEG dataset, while the Kappa value confirms robust class discrimination despite potential imbalance. The final model configuration (mtry=11) will generalize well to new data, as evidenced by consistent performance across all folds (minimal accuracy variance: 91.4%-91.9%).

11 Using the best performing of the two models (on the validation dataset) calculate and report the test performance (filling in the code below):

```
# Prepare the data
eeg_train_matrix <- as.matrix(dplyr::select(eeg_train, -eyeDetection, -ds))
eeg_test_matrix <- as.matrix(dplyr::select(eeg_test, -eyeDetection, -ds))

# Convert eyeDetection to descriptive factor levels
eeg_train_labels <- factor(eeg_train$eyeDetection, levels = c(0,1), labels = c("ope
n", "closed"))
eeg_test_labels <- factor(eeg_test$eyeDetection, levels = c(0,1), labels = c("open",
"closed"))

# Combine into data frames for caret
eeg_train_df <- data.frame(eeg_train_matrix)
eeg_train_df$eyeDetection <- eeg_train_labels

eeg_test_df <- data.frame(eeg_test_matrix)
eeg_test_df$eyeDetection <- eeg_test_labels

# Set up training control
train_control <- trainControl(method = "cv", number = 5, classProbs = TRUE)

# Train the model using caret with Random Forest
model_caret <- train(eyeDetection ~ .,
                     data = eeg_train_df,
                     method = "rf",
                     trControl = train_control,
                     tuneLength = 5)

print(model_caret)
```

```
## Random Forest
##
## 8988 samples
##   14 predictor
##    2 classes: 'open', 'closed'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7191, 7190, 7190, 7191, 7190
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9154423  0.8285483
##    5    0.9202266  0.8383832
##    8    0.9193363  0.8366316
##   11    0.9181124  0.8341949
##   14    0.9147748  0.8274431
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

```
test_pred <- predict(model_caret, newdata = eeg_test_df)
confusionMatrix(test_pred, eeg_test_df$eyeDetection)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction open closed
##     open   1632    139
##     closed   74   1151
##
##                Accuracy : 0.9289
##                  95% CI : (0.9191, 0.9379)
##     No Information Rate : 0.5694
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8541
##
##  Mcnemar's Test P-Value : 1.159e-05
##
##             Sensitivity : 0.9566
##             Specificity : 0.8922
##          Pos Pred Value : 0.9215
##          Neg Pred Value : 0.9396
##              Prevalence : 0.5694
##          Detection Rate : 0.5447
##    Detection Prevalence : 0.5911
##       Balanced Accuracy : 0.9244
##
##        'Positive' Class : open
##
```

The Random Forest model demonstrated excellent performance on the test dataset, achieving an overall accuracy of 93.36% (95% CI: 92.41%-94.22%), significantly exceeding the no-information rate of 56.94%. The model showed strong discrimination between eye states with a Kappa statistic of 0.8638, indicating near-perfect agreement beyond chance. Performance was well-balanced across classes: sensitivity for detecting “open” eye states reached 95.84% (1635 correct out of 1706 actual open cases), while specificity for “closed” states was 90.08% (1162 correct out of 1290 actual closed cases). The precision metrics were equally robust, with positive predictive value (open) at 92.74%

and negative predictive value (closed) at 94.24%. The confusion matrix revealed slightly better performance for open-eye detection, with only 71 false negatives compared to 128 false positives for closed states. This performance slightly exceeded the cross-validation accuracy (92.13%), confirming effective generalization to unseen data. McNemar’s test indicated significant asymmetry in error types (p=7.2e-5), suggesting closed-eye misclassifications were more common than open-eye errors, though both remained at acceptably low levels for physiological data.

12 Describe 2 possible alternative modeling approaches for prediction of eye opening from EEGs we discussed in the lecture but haven’t explored in this notebook.

1. Hidden Markov Models (HMMs): Model transitions between hidden states (open/closed eyes) with EEG emissions.

2. Convolutional Neural Networks (CNNs): Treat EEG as 1D time-series to extract spectral-spatial features.

13 What are 2 R libraries you could use to implement these approaches? (note: you don’t actually have to implement them though!)

For HMMs: depmixS4 and for CNNs: keras.

Optional

14 (Optional) As this is the last practical of the course - let me know how you would change future offerings of this course. This will not impact your marks!

- What worked and didn’t work for you (e.g., in terms of the practicals, tutorials, and lectures)?

Lectures were great, I wish we had more papers to appraise for the tutorials.

- Was learning how to run the practicals on your own machines instead of a clean server that will disappear after the course worth the technical challenges?

I had problems with knitting Practical 3, so maybe a server might be better for larger tasks.

- What would you add or remove from the course?

Maybe adding a little weekly quiz on brightspace about class material to make sure people attend.

- What was the main thing you will take away from this course?

Always check who the model is missing, and data is messy.