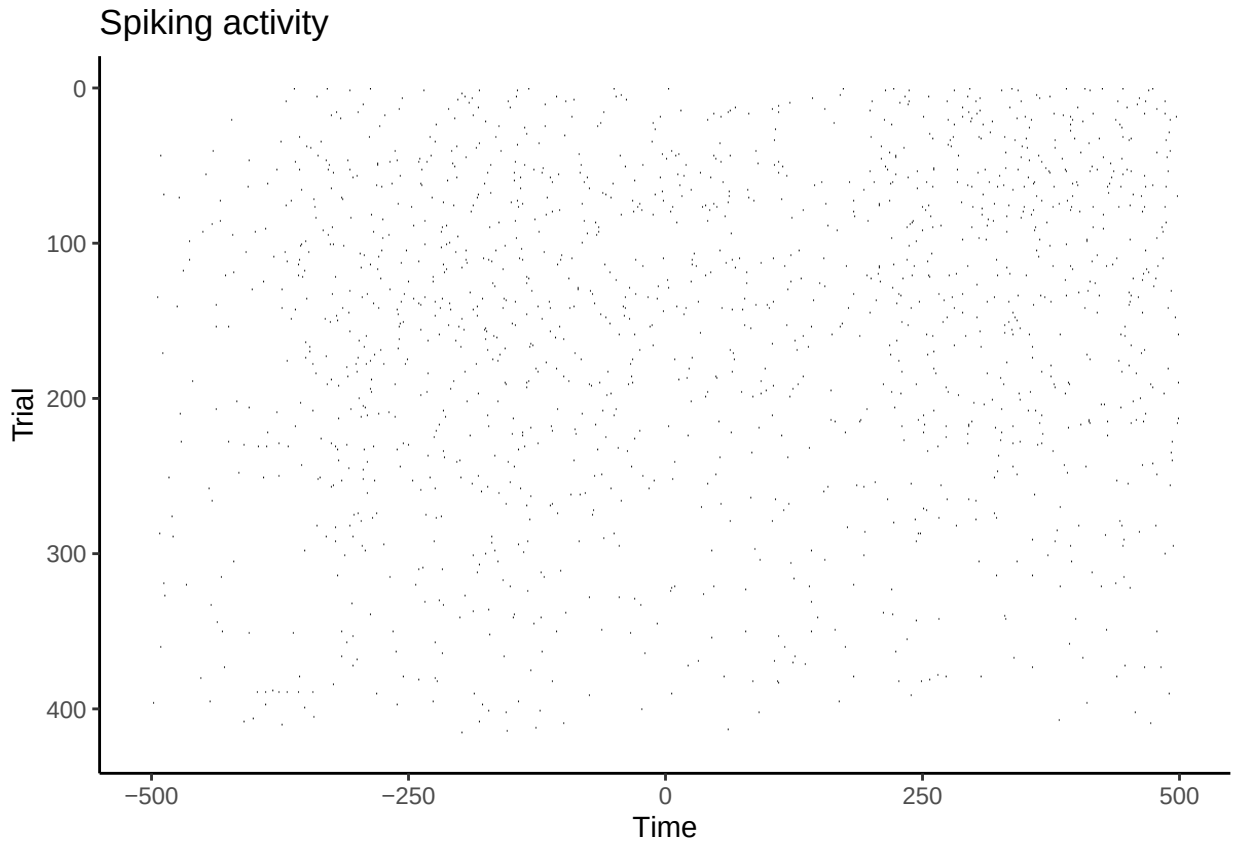
neuro_decode_practice

Brennan

2022-11-07

```
# Load in the data in the desired format
raster_dir_name <- file.path(system.file("extdata", package = "NeuroDecodeR"),
                             "Zhang_Desimone_7object_raster_data_small_rda")
file_name <- "bp1001spk_01A_raster_data.rda"
load(file.path(raster_dir_name, file_name))

test_valid_raster_format(file.path(raster_dir_name, file_name))

# Graph the data as a sanity check
plot(raster_data)
```

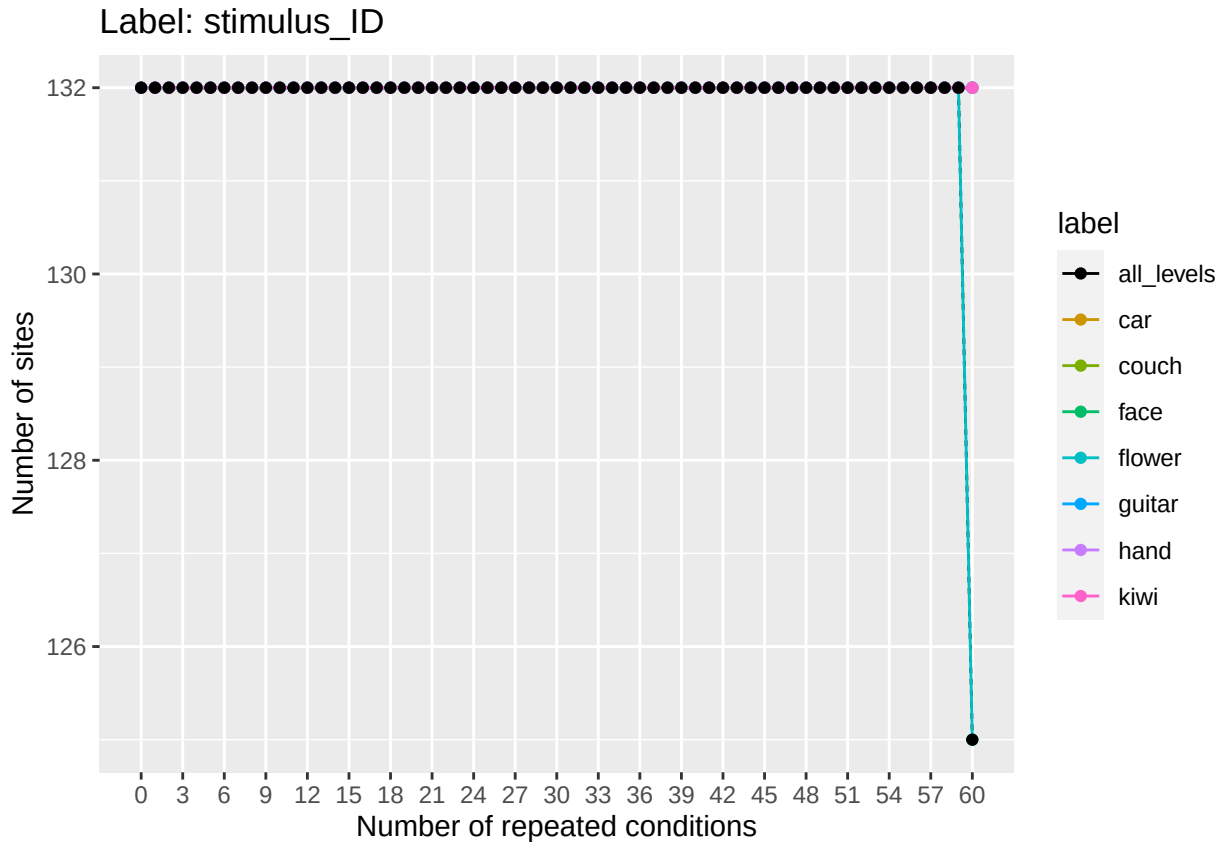



```
# Group the data into bins
save_dir_name <- file.path("Test_Binned")
```

```
binned_file_name <- create_binned_data(raster_dir_name, file.path(save_dir_name, "ZD"),
                                       150, 50, num_parallel_cores = 2)
```

```
##   |                                                                      |
```

```
# Visualize how many times each stimulus was presented
binned_file_name <- system.file(file.path("extdata", "ZD_150bins_50sampled.Rda"), package="NeuroDecodeR
label_rep_info <- get_num_label_repetitions(binned_file_name, "stimulus_ID")
plot(label_rep_info)
```



Decoding Analysis

Performing a decoding analysis involves several steps:

1. Creating a datasource (DS) object that generates training and test splits of the data.

```
# Get the data
binned_file_name <- system.file(file.path("extdata", "ZD_150bins_50sampled.Rda"),
                                package="NeuroDecodeR")

# Get the response variable
variable_to_decode <- "stimulus_ID"

# Determine the number of splits (this is lower than maximum in this case for runtime)
```

```
num_cv_splits <- 20

# Create the dataset
ds <- ds_basic(binned_file_name, variable_to_decode, num_cv_splits)
```

```
## Automatically selecting sites_IDs_to_use. Since num_cv_splits = 20 and num_label_repeats_per_cv_spli
```

2. Optionally creating feature-preprocessor (FP) objects that learn parameters from the training data, and preprocess the training and test data.

```
# Store a function to normalize z-scores of data
fps <- list(fp_zscore())
```

3. Creating a classifier (CL) object that learns the relationship between the training data and training labels, and then evaluates the strength of this relationship on the test data.

```
# Store a function to classify the training data
cl <- cl_max_correlation()
```

4. Creating result metric (RM) objects that aggregate the predictions to create result summaries.

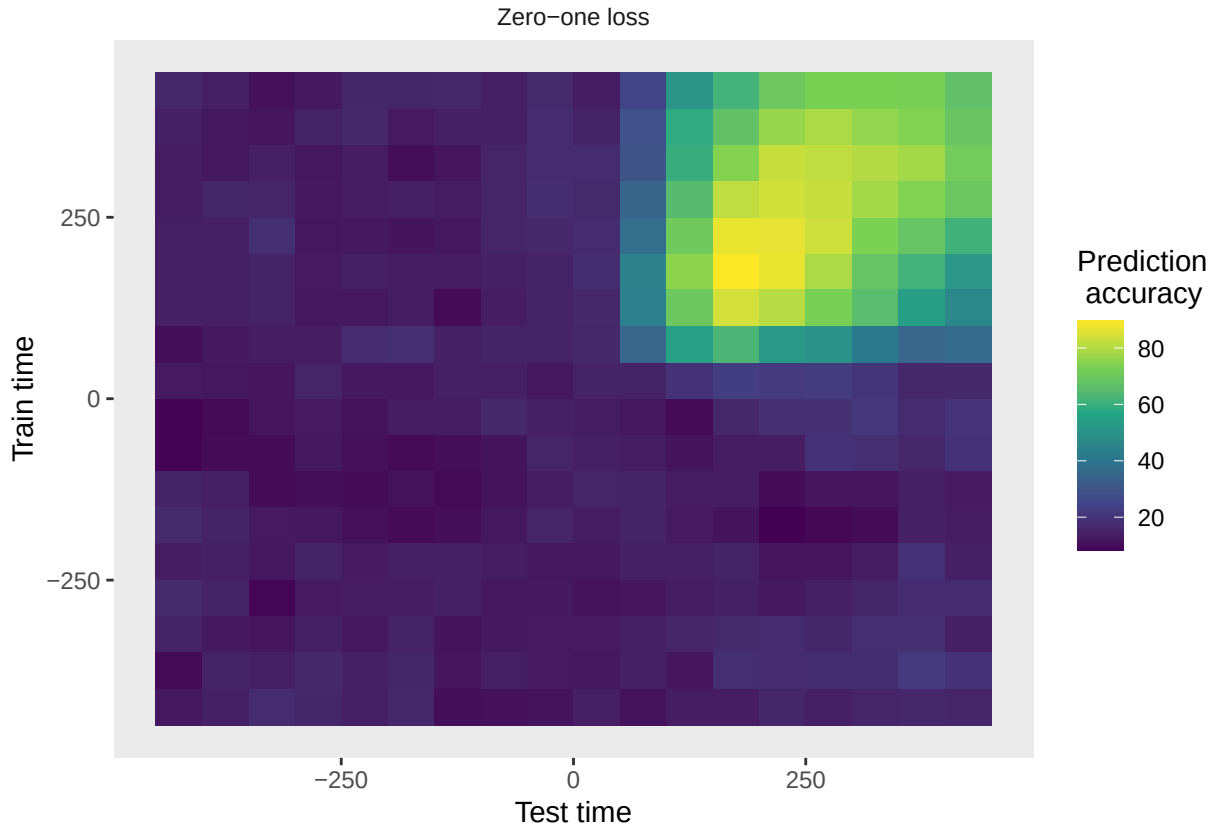
```
# Store a function for plotting relevant graphs for analysis
rms <- list(rm_main_results(), rm_confusion_matrix())
```

5. Running a cross-validator object that using the datasource (DS), the feature-preprocessor (FP) and the classifier (CL) objects to do a cross-validation procedure that estimates the decoding accuracy.

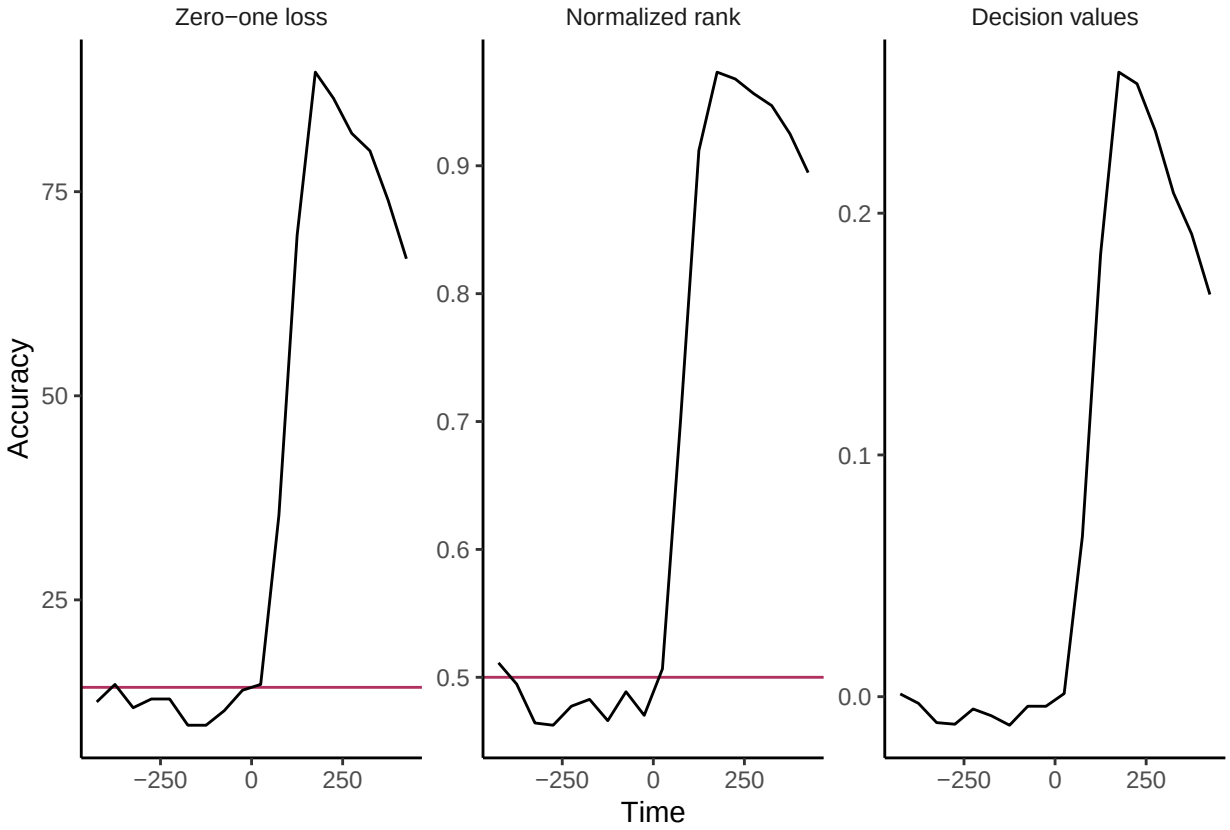
```
# This is where all of the testing happens and the previous three functions are used

# Create the cross-validate function
cv <- cv_standard(datasource = ds,
                  classifier = cl,
                  feature_preprocessors = fps,
                  result_metrics = rms,
                  num_resample_runs = 2)

# Run the function and get results
DECODING_RESULTS <- run_decoding(cv)
```

```
##   |                                                                      |
```

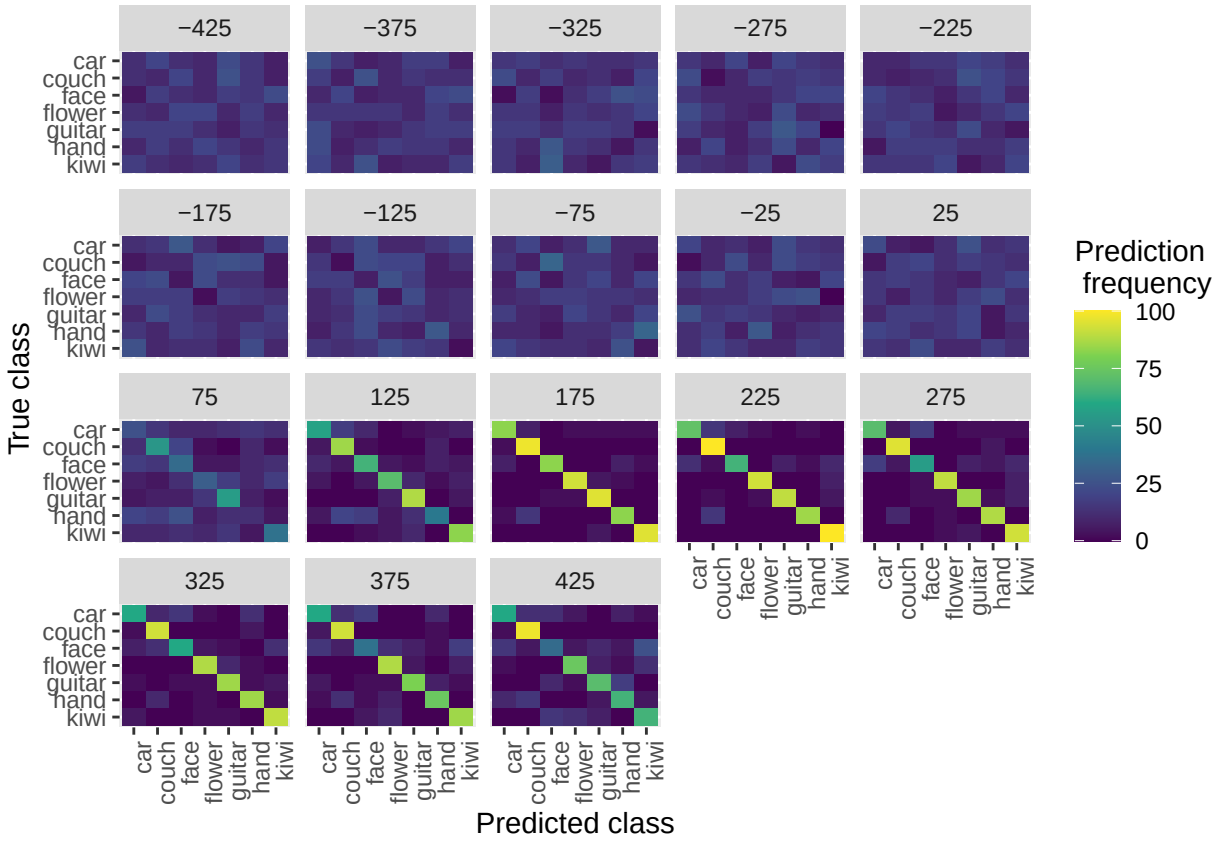
```
# Plot main results
plot(DECODING_RESULTS$rm_main_results)
```



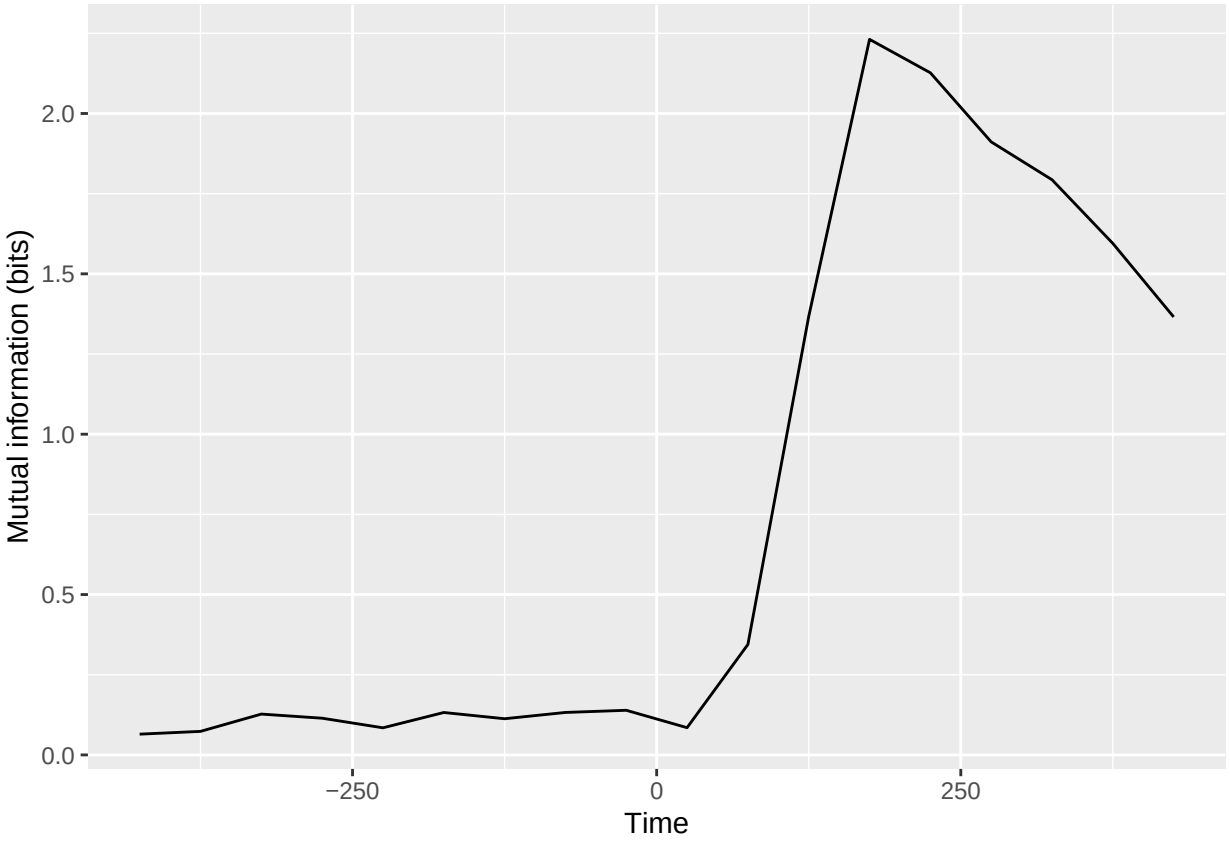

```
plot(DECODING_RESULTS$rm_main_results, results_to_show = 'all', type = 'line')
```




```
# Plot the confusion matrix
plot(DECODING_RESULTS$rm_confusion_matrix)
```




```
plot(DECODING_RESULTS$rm_confusion_matrix, results_to_show = "mutual_information")
```




```
# This code saves the data to a file if there is a valid directory listed
results_dir_name <- file.path(tempdir(), "results", "")
dir.create(results_dir_name)

log_save_results(DECODING_RESULTS, results_dir_name)
```

```
## Warning in log_save_results(DECODING_RESULTS, results_dir_name): The manifest file does not exist.
##   Assuming this is the first result that is saved and creating manifest file
```