My Document

Neural Decoding

Neural decoding is a data analysis method that uses neural activity to predict which experimental conditions are present in different experimental trials. Neural decoding can be used to assess how information is coded in populations of neural activity.

When running a decoding analysis, neural activity from different sites do not need to be recorded simultaneously, but instead one can create 'pseudo-populations' where responses of simultaneously recorded neural populations are approximated by combining recordings made across multiple experimental sessions.

In many situations it is not currently practical or possible to record simultaneously from many neurons (for example, it is currently difficult to implant multielectrode arrays in deep brain structures such as macaque inferior temporal cortex).

Methods

The NeuroDecorderR method works by 'training' a machine learning algorithm, called a pattern classifier, to learn the relationship between neural activity and particular experimental conditions on a subset of data called the training set. Once the classifier has 'learned' the relationship between the neural data and experimental conditions, one assesses whether this relationship is reliable by having the pattern classifier predict which experimental conditions are present in a separate test set of data.

Using the NeuroDecodeR package

The NeuroDecodeR package is designed around five abstract object types which allows researchers to easily run a range of different decoding analyses.

- Data Sources (ds)
- Feature preprocessors (fp)
- Classifiers (cl)
- Result metrics (rm)
- Cross-validator (cv)

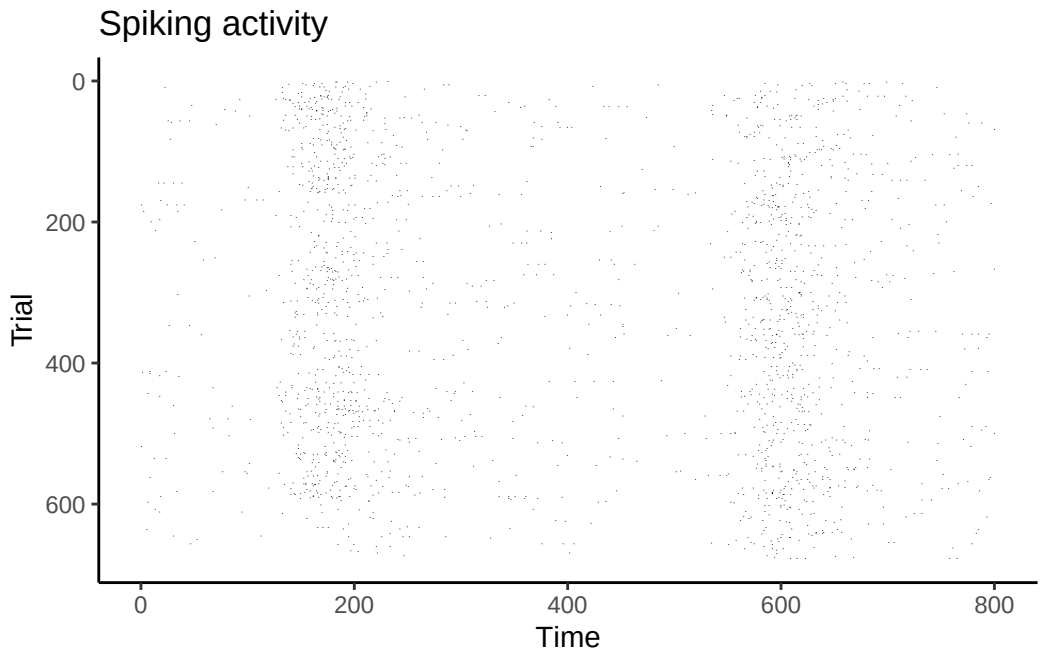
In this example, we'll use data from the "Friewald Tsao Face Views AM data set"

```
library(NeuroDecodeR)

# In order to use the NeuroDecodeR package, neural data must be put into a particular format called "raste
raster_dir_name <- "data"

raster_data <- read_raster_data(file.path(raster_dir_name,'raster_data_bert_am_site021.rda'))
```

```
plot(raster_data)
```



When looking at the plot of the raster data example neuron shown, we see that there is a large increase in spiking activity a little before 200 ms post-stimulus onset, which corresponds to the response latency of this neuron, and then another large increase in spiking activity a little before 600 ms post-stimulus onset, which corresponds to the response of the next stimulus. Seeing that the pattern of responses matches what we expect based on the design of the experiment is a good sanity check that we have correctly formatted the data.

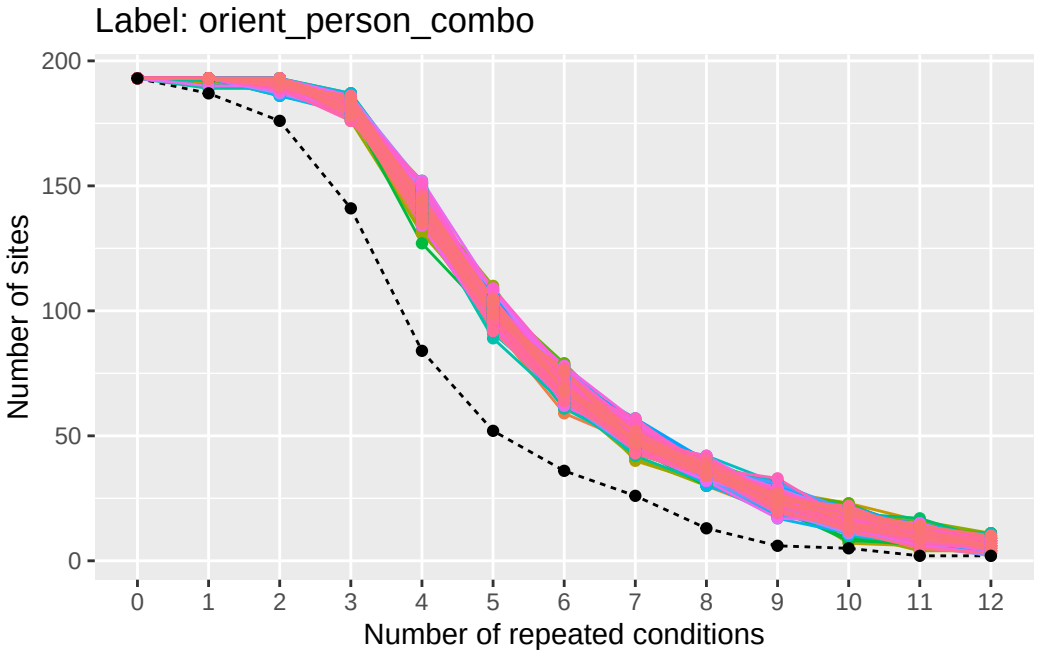
Objective

We will decode which of the 25 individuals were shown on each trial using only the left profile images.

Before starting to run a decoding analysis, it is useful to assess how many trials were collected from each recording site for each stimulus that was shown

```
label_info <- get_num_label_repetitions(binned_file_name,
                                        labels = "orient_person_combo")

plot(label_info, show_legend=FALSE)
```



The plot illustrate the trade-off between the number of cross-validation splits we would like to use (k), and the number of neurons available.

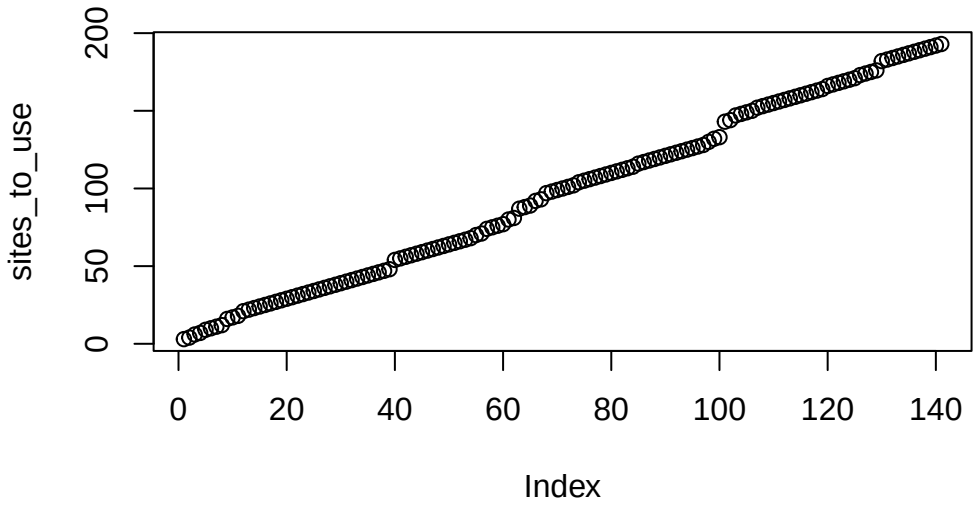
The black dashed line shows how many sites have k repetitions of all the stimuli.

From looking at this black dashed line, we see that there are a little less than 150 neurons that have 3 repetitions of each of the 25 stimuli, and there are a little more than 75 neurons that have 4 repetitions of each stimulus. Thus, if we run a 3 fold cross-validation, our pseudo-population vectors could consist of a little less than 150 neurons, and if we run a 4 fold cross-validation analysis, our pseudo-population vectors could consist of a little more than 75 neurons. In the subsequent analyses, we will run a 3 fold cross-validation, although a 4 fold cross-validation with fewer neurons would also be a reasonable choice.

Decoding analysis 1

```
sites_to_use <- get_siteIDs_with_k_label_repetitions(
  "FV_AM_30bins_10sampled.Rda",
  labels = "orient_person_combo",
  k= 3)

plot(sites_to_use)
```




```
left_profile_levels <- paste("left profile", 1:25)

# We can then run a decoding analysis by creating each of the 5 object types
```

1. ds_basic data source to create pseudo-populations of data. The arguments we pass to this constructor are:

   a) the name of the binned data file

   b) the variable we want to decode (e.g., "orient_person_combo")

   c) the number of cross-validation splits to use

   d) a vector with the levels we want to use (i.e., the levels that start with "left profile")

   e) the site IDs for all the sites that have at least 3 label repetitions.

2. fp_zscore feature preprocessor to normalize the data. This feature preprocessor ensures that neurons with higher firing rates do not dominate over neurons with lower firing rates.
3. cl_max_correlation classifier to make our predictions.
4. rm_main_results and rm_confusion_matrix result metrics to show our decoding accuracies.
5. cv_standard cross-validator to run the full decoding analysis. The arguments we pass to this constructor are:

   a) the data source

   b) the classifier

   c) the feature pre-processor

   d) the result metrics

   e) a number specifying how many resample runs to use.

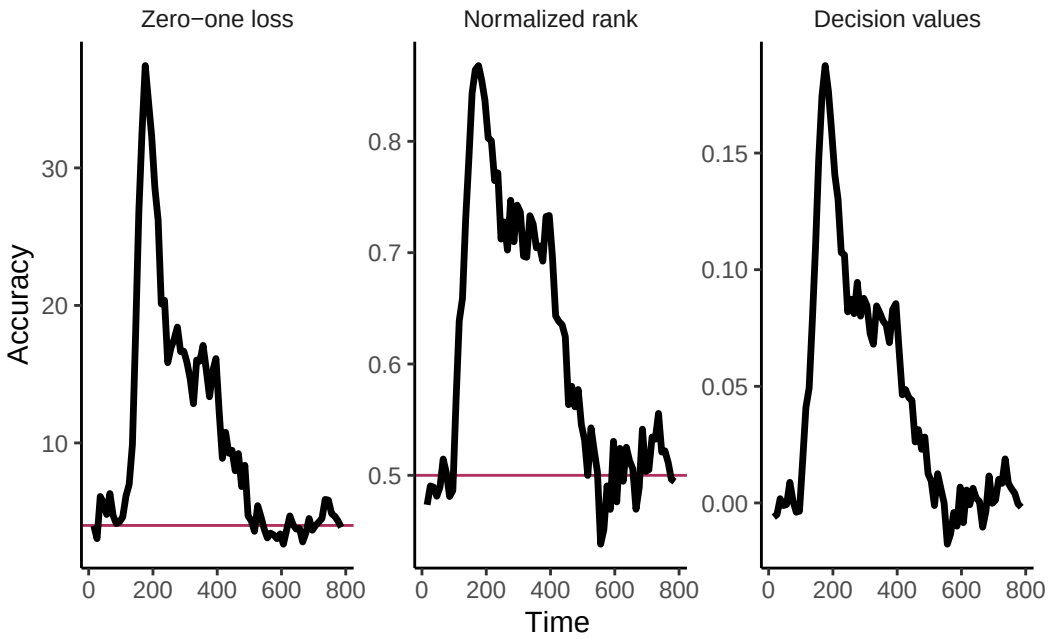
```
ds <- ds_basic(binned_data = "FV_AM_30bins_10sampled.Rda",
               labels = "orient_person_combo",
               num_cv_splits = 3,
               label_levels = left_profile_levels,
               site_IDs_to_use = sites_to_use)

fps <- list(fp_zscore())
cl <- cl_max_correlation()
rms <- list(rm_main_results(), rm_confusion_matrix())

cv <- cv_standard(datasource = ds,
               classifier = cl,
               feature_preprocessors = fps,
               result_metrics = rms)
```

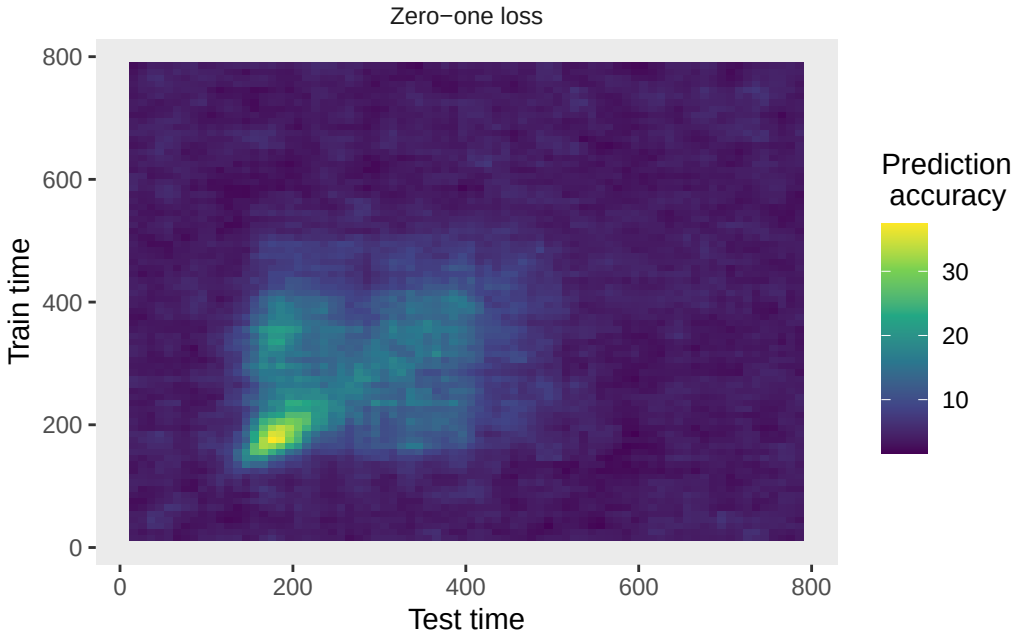
```
DECODING_RESULTS <- run_decoding(cv)
```

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



From looking at the results we see that the decoding accuracy rises above the chance level of 1/25 around 150 ms after stimulus onset, and the zero–one loss, normalized rank and raw decision value results look similar, which is often the case.

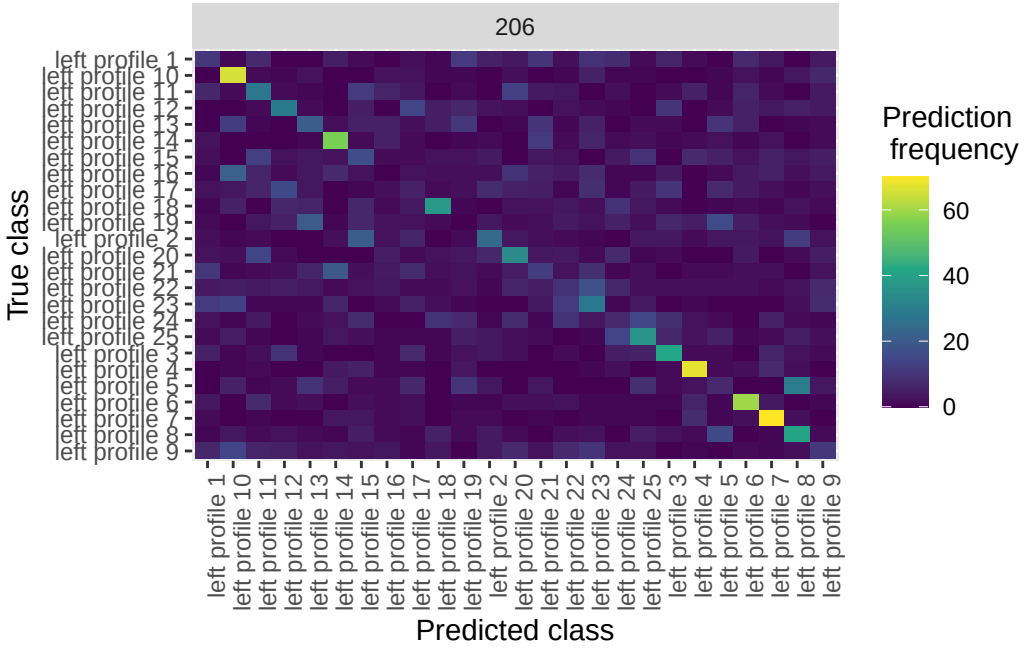
```
plot(DECODING_RESULTS$rm_main_results, type = "TCD")
```



A temporal cross-decoding (TCD) plot based on using the plot(rm_main_results, type = "TCD") function. The lack of a strong diagonal band on the plot indicates that information is not contained in a highly dynamic code. Overall, the reason the classification accuracy is relatively low is because we are only training the classifier on two examples from each class.

The rm_confusion_matrix result metric plot() function allows one to view a sequence of confusion matrices for each time period that was decoded. Because we binned the data with a relatively small sampling interval of 10 ms, plotting a sequence of confusion matrices for all decoded time bins will be rather cluttered, so instead we will just plot the confusion matrix that starts around 200 ms after stimulus onset, which we can do by setting the argument start_time_to_plot = 200.

```
plot(DECODING_RESULTS$rm_confusion_matrix,
    plot_only_one_train_time = 206)
```



We see that some classes (i.e., face identities) were predicted more accurately than others, for example, the person 10 was correctly predicted about 60% of the time.