Decoding spikes

In this project the aim is to classify channels in IBL data as right or left selective based on their firing rate after visual stimulus.

Decoding Framework

the decoding procedures implemented in the `DecodingFramework_OnCluster` class, which is used to classify firing rate based on logistic regression. This class is built to facilitate the decoding of right vs. left visual stimuli from firing rates using data from both passive and active conditions. ; active means the data is recorded during the behavioral protocol while passive represent the data recorded during passive stimulation. see

Key Components

1. **Data Initialization** The framework accepts both passive and active firing rates for a single channel.
   - `data_passive` & `data_active`: Arrays of shape `(n_trials, n_clusters, n_time_bins)` representing firing rates for each channel.
   - `labels_passive` & `labels_active`: Labels for each trial, indicating the condition. right = 1 left = −1 and no_stim= 0
2. **Feature Selection** Feature selection can be applied using one of the following methods:
   - **PCA**: Principal Component Analysis to reduce dimensionality by combining cluster and time bin information. `reduced_data = (n_trials, n_components_of(n_clusters * n_time_bins))`
   - **Average Clusters**: Averaging the neural activity over clusters to reduce complexity.
3. **Decoding and Cross-Validation** The decoding step uses **Logistic Regression** with an `L1` penalty, balanced class weights, and maximum iterations set to 1000.
   - **Cross-Validation**: A Stratified K-Fold cross-validation (`StratifiedKFold`) is used for test strategy `'passive'` to validate the model.
   - Training and testing datasets are chosen based on the `test_strategy`:
     - `'passive'`: Train and test using passive data with cross-validation.
     - `'active'`: Train on passive data and test on active data.
     - `'both'`: Train on part of the passive data, and test on the rest along with the active data.
4. **Null Distribution and Statistical Validation** To validate the decoding accuracy, a **null distribution** is generated using label permutation with `n_permutations` iterations.
   - **Null Distribution**: Accuracy scores computed after random shuffling of labels, repeated across specified permutations.

- **p-Value Calculation**: p-values are computed to assess statistical significance by comparing true accuracy to the null distribution.

Summary of the Workflow

1. **Data Preparation**: Mask trials to separate right vs. no stimulus and left vs. no stimulus for passive and active conditions.
2. **Feature Selection**: Apply PCA or average clusters to reduce the dimensionality of the dataset.
3. **Model Training and Testing**:
   - Train a logistic regression model on the prepared dataset.
   - Apply cross-validation for test strategy `'passive'`, or train/test on different data splits for other strategies.
4. **Validation**: Generate null distributions and calculate p-values to verify the statistical significance of the model's performance.

Key Parameters

- `n_folds`: Number of folds for cross-validation.
- `n_components`: Number of components to retain for PCA.
- `n_permutations`: Number of permutations used for null distribution generation.
- `test_strategy`: Determines how passive and active datasets are used for training and testing.

Results

The final output of the `decode` method includes:

- **True Accuracies** (`true_accuracy_right`, `true_accuracy_left`)
- **Null Distributions** (`null_distribution_right`, `null_distribution_left`)
- **p-Values** (`p_value_right`, `p_value_left`)

These results provide insights into the model's performance for both right and left visual stimuli, and the significance of the observed accuracies compared to random label shuffling.

Usage

see `decoding.ipynb` Jupyter notebook for one session example with different decoding procedures.

Preprocessing Data

The functions implemented here are designed to extract relevant features, filter the data, and compile information that is essential for applying decoding procedures on data.

Overview

The preprocessing workflow involves two main functions:

- **`pre_processed_active_data()`**: Processes the active dataset.
- **`pre_processed_passive_data()`**: Processes the passive dataset.

Both functions aim to extract spiking data, filter it based on specific conditions, and output firing rates, trial metadata, and channel metadata.

Preprocessing Steps

1. Extract Trial Information

- **Active Data**: Behavioral data is loaded using `get_behavior()`. The trial information includes contrasts, trial onset times, and other metadata. The trials are filtered based on contrast levels (`contrast_filter`) and probability left values (`probabilityLeft_filter`). Each trial is labeled based on whether it contains right stimulus (`1`), left stimulus (`-1`), or no stimulus (`0`).
- **Passive Data**: The passive dataset is loaded using the `passiveGabor` object from the `ONE` API. Trials are filtered based on contrast levels, and labels are assigned based on the visual field location of the stimulus (right or left).

2. Load and Filter Spiking Data

- Spiking data is loaded using `get_spikes()`, and channel information is obtained using `get_channels()`. Each cluster is assigned to a specific channel.
- **Filtering**:
  - Clusters can be filtered to only include "good" clusters based on quality metrics (`only_good_clusters` parameter).
  - Regions can be filtered using the `filter_regions` parameter, which ensures that only specific brain regions are considered for the analysis.

3. Calculate Firing Rates

- Firing rates are computed for each cluster using the `firingRate_OnClusters()` function. This function bins spike times around each trial onset, and then normalizes firing rates by calculating z-scores relative to the baseline activity before stimulus onset.
- The firing rates (`z_score_firing_rate`) are computed separately for each trial and cluster, resulting in a tensor of shape `(n_trials, n_clusters, n_time_bins)`. The time bins are filtered to include only those after a specified minimum time (`min_time`).

4. Extract Metadata

- **Channel Metadata**: Information such as channel depth, atlas ID, coordinates (x, y, z), and acronyms are extracted for each selected channel and stored in a DataFrame (`channel_info`).
- **Trial Metadata**: Metadata such as trial indices, labels, contrasts, and distances to the latest block change are compiled into a DataFrame (`trial_info`).

5. Output

The final preprocessed data for both active and passive datasets is returned as a dictionary with the following keys: - **`firing_rates`**: Firing rate data for each channel, stored as a dictionary where each key is a channel, and values are z-scored firing rates for that channel. - **`trial_info`**: DataFrame containing trial-level metadata such as labels, contrasts, and assigned side. - **`channel_info`**: DataFrame containing channel-level metadata such as coordinates and region acronyms. - **`time_bins`**: Array representing the time bins used for calculating the firing rates.

Key Parameters

- **`min_contrast`**: Minimum contrast value used to filter the trials.
- **`t_bin`**: Size of the time bin (in seconds) used for calculating firing rates.
- **`pre_stim`** and **`post_stim`**: Time (in seconds) before and after stimulus onset to consider when calculating firing rates.
- **`filter_regions`**: List of brain regions to include in the analysis.
- **`only_good_clusters`**: Boolean flag indicating whether to filter clusters to include only those classified as "good".
- **`contrast_filter`** and **`probabilityLeft_filter`**: Lists specifying valid contrast and probability left values for trial filtering.

Usage

Preprocessing Data

```
pre_processed_data = pre_processed_active_data(eid, pid, min_contrast=0.25,
t_bin=0.02, pre_stim=0.5, post_stim=1.0)

pre_processed_data = pre_processed_passive_data(eid, pid, min_contrast=0.25,
t_bin=0.02, pre_stim=0.5, post_stim=1.0)
```
