Package: NeuroDecodeR (via r-universe)

September 11, 2024

Title Decode Information from Neural Activity

Version 0.2.0

Description Neural decoding is method of analyzing neural data that uses a pattern classifiers to predict experimental conditions based on neural activity. 'NeuroDecodeR' is a system of objects that makes it easy to run neural decoding analyses. For more information on neural decoding see Meyers & Kreiman (2011) <doi:10.7551/mitpress/8404.003.0024>.

URL https://emeyers.github.io/NeuroDecodeR/, https://github.com/emeyers/NeuroDecodeR

BugReports https://github.com/emeyers/NeuroDecodeR/issues

Depends R (>= 4.1.0)

License GPL-3

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports dplyr, doSNOW, e1071, forcats, foreach, ggplot2, gridExtra, magrittr, methods, purrr, R.matlab, scales, stats, stringr, tibble, tictoc, tidyr, utils

Suggests knitr, rmarkdown, testthat

Encoding UTF-8

VignetteBuilder knitr

Repository https://emeyers.r-universe.dev

RemoteUrl https://github.com/emeyers/neurodecoder

RemoteRef HEAD

RemoteSha 826e2119a69d7879c3f58c75854272324dead2a7

Contents

`cl_max_correlation` *A maximum correlation coefficient classifier (CL)*

Description

An implementation of a maximum correlation coefficient classifier.

Usage

```
cl_max_correlation(
  ndr_container_or_object = NULL,
  return_decision_values = TRUE
)
```

Arguments

`ndr_container_or_object`
: The purpose of this argument is to make the constructor of the cl_maximum_correlation classifier work with the pipe (|>) operator. This argument should almost never be directly set by the user to anything other than NULL. If this is set to the default value of NULL, then the constructor will return a cl_max_correlation object. If this is set to an NDT container, then a cl_max_correlation object will be added to the container and the container will be returned. If this argument is set to another NDT object, then both that NDR object as well as a new cl_maximum_correlation object will be added to a new container and the container will be returned.

`return_decision_values`
: A Boolean specifying whether the prediction function should return columns that have the decision values. Setting this to FALSE will save memory so can be useful when analyzing very large high temporal resolution data sets. However if this is set to FALSE< metrics won't be able to compute decoding accuracy measures that are based on the decision values; e.g., the rm_main_results object won't be able to calculate normalized rank decision values.

Details

This CL object learns a mean population vector (template) for each class from the training set (by averaging together the all training points within each class). The classifier is tested by calculated Pearson's correlation coefficient between a test point and the templates learned from the training set, and the class with the highest correlation value is returned as the predicted label. The decision values returned by the classifier are the correlation coefficients between all test points and all templates.

Like all classifiers (CL) objects, this classifier has a private get_predictions() method which learns a model based on training data and then makes predictions on the test data.

Value

This constructor creates an NDR classifier object with the class `cl_max_correlation`. Like all NDR classifier objects, this classifier will be used by a cross-validator to learn the relationship between neural activity and experimental conditions on a training set of data, and then it will be used to make predictions on a test set of data.

See Also

Other classifier: `cl_poisson_naive_bayes()`, `cl_svm()`

Examples

```
# running a basic decoding analysis using the cl_max_correlation
data_file <- system.file(file.path("extdata", "ZD_150bins_50sampled.Rda"),
                         package = "NeuroDecodeR")
ds <- ds_basic(data_file, "stimulus_ID", 18)
fps <- list(fp_zscore())

cl <- cl_max_correlation()
```

```
cv <- cv_standard(datasource = ds,
                  classifier = cl,
                  feature_preprocessors = fps,
                num_resample_runs = 2)  # better to use more resample runs (default is 50)

DECODING_RESULTS <- run_decoding(cv)
```

`cl_poisson_naive_bayes` *A Poisson Naive Bayes classifier (CL)*

Description

An implementation of a Poisson Naive Bayes classifier.

Usage

```
cl_poisson_naive_bayes(
  ndr_container_or_object = NULL,
  return_decision_values = TRUE
)
```

Arguments

`ndr_container_or_object`
: The purpose of this argument is to make the constructor of the cl_poisson_naive_bayes classifier work with magrittr pipe (|>) operator. This argument should almost never be directly set by the user to anything other than NULL. If this is set to the default value of NULL, then the constructor will return a cl_poisson_naive_bayes object. If this is set to an ndr container, then a cl_poisson_naive_bayes object will be added to the container and the container will be returned. If this argument is set to another ndr object, then both that ndr object as well as a new cl_poisson_naive_bayes object will be added to a new container and the container will be returned.

`return_decision_values`
: A Boolean specifying whether the prediction function should return columns that have the decision values. Setting this to FALSE will save memory so can be useful when analyzing very large high temporal resolution data sets. However if this is set to FALSE< metrics won't be able to compute decoding accuracy measures that are based on the decision values; e.g., the rm_main_results object won't be able to calculate normalized rank decision values.

Details

This classifier object implements a Poisson Naive Bayes classifier. The classifier works by learning the expected number of occurrences (denoted lambda) for each feature and each class by taking the average of the training data over all trials (separately for each feature and each class). To evaluate whether a given test point belongs to class i, the log of the likelihood function is calculated using the lambda values as parameters of Poisson distributions (i.e., there is a separate Poisson distribution for each feature, that is based on the lambda value for that feature). The overall likelihood value is calculated by multiplying the probabilities for each neuron together (i.e,. Naive Bayes classifiers assume that each feature is independent), or equivalently, adding the log of the probabilities for each feature together. The class with the highest likelihood value is chosen as the predicted label, and the decision values are the log likelihood values.

Note: this classifier uses spike counts, so the binned data must be converted to use this classifier, for example, if you are using the basic_DS data source, then use_count_data = TRUE should be set in the constructor. Also, preprocessors that convert the data into values that are not integers should not be used, for example, the fp_zscore should not be used with this classifier.

Like all classifiers, this classifier learning a model based on training data and then makes predictions on new test data.

Value

This constructor creates an NDR classifier object with the class `cl_poisson_naive_bayes`. Like all NDR classifier objects, this classifier will be used by a cross-validator to learn the relationship between neural activity and experimental conditions on a training set of data, and then it will be used to make predictions on a test set of data.

See Also

Other classifier: `cl_max_correlation()`, `cl_svm()`

Examples

```
# running a basic decoding analysis using the cl_max_correlation

data_file <- system.file(file.path("extdata", "ZD_150bins_50sampled.Rda"),
                         package = "NeuroDecodeR")
ds <- ds_basic(data_file, "stimulus_ID", 18, use_count_data = TRUE)
fps <- list()

cl <- cl_poisson_naive_bayes()
cv <- cv_standard(datasource = ds,
                  classifier = cl,
                  feature_preprocessors = fps,
                num_resample_runs = 2)  # better to use more resample runs (default is 50)

DECODING_RESULTS <- run_decoding(cv)
```

`cl_svm` *A support vector machine classifier (CL)*

Description

This classifier uses the e1071 package to implement a support vector machine.

Usage

```
cl_svm(ndr_container_or_object = NULL, return_decision_values = TRUE, ...)
```

Arguments

`ndr_container_or_object`
: The purpose of this argument is to make the constructor of the cl_svm classifier works with the magrittr pipe (|>) operator. This argument should almost never be directly set by the user to anything other than NULL. If this is set to the default value of NULL, then the constructor will return a cl_svm object. If this is set to an ndr container, then a cl_svm object will be added to the container and the container will be returned. If this argument is set to another ndr object, then both that ndr object as well as a new cl_svm object will be added to a new container and the container will be returned.

`return_decision_values`
: A Boolean specifying whether the prediction function should return columns that have the decision values. Setting this to FALSE will save memory so can be useful when analyzing very large high temporal resolution data sets. However if this is set to FALSE< metrics won't be able to compute decoding accuracy measures that are based on the decision values; e.g., the rm_main_results object won't be able to calculate normalized rank decision values.

`...`
: All parameters that are available in the e1071 package svm() object should work with this CL object.

Details

A support vector machine (SVM) is a classifier that learns a function f that minimizes the hinge loss between predictions made on the training data, while also applying a penalty for more complex f (the penalty is based on the norm of f in a reproducing kernel Hilbert space). The SVM has a parameter C that controls the trade off between the empirical loss (i.e., a smaller prediction error on the training set), and the complexity of the f. SVMs can use different kernels to create nonlinear decision boundaries.

SVMs are work on binary classification problems, so to do multi-class classification, an *all-pairs* classification scheme (which is the default for the e1071 package). In the all-pairs scheme,training separate classifiers for all pairs of labels (i.e., if there are 100 different classes then nchoosek(100, 2) = 4950 different classifiers are trained). Testing the classifier in all-pairs involves having all classifiers classify the test point, and then the class label is given to the class the was chosen most often by the binary classifiers (in the case of a tie in the number of classes that won a contest the class label is randomly chosen). The decision values for all-pairs are the number of contests won by each class (for each test point).

Value

This constructor creates an NDR classifier object with the class `cl_svm`. Like all NDR classifier objects, this classifier will be used by a cross-validator to learn the relationship between neural activity and experimental conditions on a training set of data, and then it will be used to make predictions on a test set of data.

See Also

e1071

Other classifier: `cl_max_correlation()`, `cl_poisson_naive_bayes()`

Examples

```
# using the default e1071 parameters
cl <- cl_svm()

# using a linear kernel
cl <- cl_svm(kernel = "linear")
```

---

convert_matlab_raster_data

Convert raster data in MATLAB to R

Description

If one already has raster data created in MATLAB (.mat files), this function can be used to convert it to an R format (.rda files) that can be used with the NDR.

Usage

```
convert_matlab_raster_data(
  matlab_raster_dir_name,
  r_raster_dir_name = NULL,
  save_file_type = "rda",
  sampling_interval_width = 1,
  zero_time_bin = NULL,
  files_contain = "",
  add_sequential_trial_numbers = FALSE
)
```

Arguments

`matlab_raster_dir_name`
: A character string specifying the path to a directory that contains raster data in MATLAB .mat files.

`r_raster_dir_name`
: A character string specifying the path to a directory where the converted raster data in R files will be saved. If this is not specified then the saved directory will have the same name as the matlab directory with _rda appended to the end of the directory name.

`save_file_type`
: A character string specifying the format that the raster data should be saved as. This must be set to a string that is either "rda", "rds", or "csv", and files will be saved to the corresponding format.

`sampling_interval_width`
: A number specifying how successive time bins will be labeled The default value of 1 means that points will be labeled as successive integers; i.e., time.1_2, time.2_3, etc. If this value was set to a larger number, then time points will be specified at the given sampling width. From example, if sampling_width is set to 10, then the time labels would be time.1_10, time.10_20, etc. This is useful if the data is sampled at a particular rate (e.g., if the data is sampled at 500Hz, one might want to use sampling_interval_width = 2, so that the times listed on the raster column names are in milliseconds).

`zero_time_bin`
: A number specifying the time bin that should be marked as time 0. The default (NULL value) is to use the first bin as time 1.

`files_contain`
: A string specifying that only a subset of the MATLAB raster data should be converted based on .mat files that contain this string.

`add_sequential_trial_numbers`
: A Boolean specifying one should add a variable to the data called 'trial_number' that has sequential trial. These trials numbers are needed for data that was recorded simultaneously so that trials can be aligned across different sites.

Value

Returns a string with the name of the directory that the .rda raster files have been saved to.

Examples

```
matlab_raster_dir_name <- file.path(
  system.file("extdata", package = "NeuroDecodeR"),
  "Zhang_Desimone_7object_raster_data_small_mat"
)

# create temporary directory to hold converted data
r_raster_dir_name <- tempdir()
r_raster_dir_name <- convert_matlab_raster_data(matlab_raster_dir_name,
  r_raster_dir_name,
  files_contain = "bp1001spk"
)
```

`create_binned_data` *Convert data from raster format to binned format*

Description

This function takes the name of a directory that contains files in raster format and averages the data within a specified bin width at specified sampling interval increments to create data in binned format used for decoding.

Usage

```
create_binned_data(
  raster_dir_name,
  save_prefix_name,
  bin_width,
  sampling_interval,
  start_time = NULL,
  end_time = NULL,
  files_contain = "",
  num_parallel_cores = NULL
)
```

Arguments

`raster_dir_name`
: A string that contains the path to a directory that has files in raster format. These files will be combined into binned format data.

`save_prefix_name`
: A string with a prefix that will be used name of file that contains the saved binned format data.

`bin_width`
: A number that has the number of data samples that data will be averaged over.

`sampling_interval`
: A number that has the specifies the sampling interval between successive binned data points.

`start_time`
: A number that specifies the time to start binning the data. This needs to be set to one of the start times in the raster data; i.e., if data columns are in the format time.XXX_YYY, then the start_time must be one of the XXX values. By default, the start_time is the first time in the raster data.

`end_time`
: A number that specifies the time to end the binning of the data. This needs to be set to one of the end times in the raster data; i.e., if data columns are in the format time.XXX_YYY, then the start_time must be one of the YYY values. By default, the end_time is the last time in the raster data.

`files_contain`
: A string that specifies that only raster files that contain this string should be included in the binned format data.

num_parallel_cores
: An integer specifying the number of parallel cores to use. The default (NULL) value is to use half of the cores detected on the system. If this value is set to a value of less than 1, then the code will be run serially.

Value

Returns a string with the name of the file that was created which has the data in binnned format.

Examples

```
# create binned data with 150 ms bin sizes sampled at 10 ms intervals
raster_dir_name <- file.path(
  "..", "data-raw", "raster",
  "Zhang_Desimone_7objects_raster_data_rda", ""
)

raster_dir_name <- trimws(file.path(system.file("extdata", package = "NeuroDecodeR"),
                          "Zhang_Desimone_7object_raster_data_small_rda", " "))


# The code could potentially run faster by using more parallel cores
# (e.g., by not setting the num_parallel_cores argument, half the cores available
#  will be used)
binned_file_name <- create_binned_data(raster_dir_name,
                                       file.path(tempdir(), "ZD"),
                                       150, 50,
                                       num_parallel_cores = 2)
```

---

cv_standard *The standard cross-validator (CV)*

Description

This object runs a decoding analysis where a classifier is repeatedly trained and tested using cross-validation.

Usage

```
cv_standard(
  ndr_container = NULL,
  datasource = NULL,
  classifier = NULL,
  feature_preprocessors = NULL,
  result_metrics = NULL,
  num_resample_runs = 50,
  run_TCD = TRUE,
```

```
  num_parallel_cores = NULL,
  parallel_outfile = NULL
)
```

Arguments

ndr_container	The purpose of this argument is to make the constructor of the cv_standard cross-validator work with the magrittr pipe (|>) operator. This argument would almost always be set at the end of a sequence of piping operators that include a datasource and a classifier. Alternatively, one can keep this set to NULL and directly use the datasource and classifier arguments (one would almost never use both types of arguments). See the examples.
datasource	A datasource (DS) object that will generate the training and test data.
classifier	A classifier (CS) object that will learn parameters based on the training data and will generate predictions based on the test data.
feature_preprocessors	A list of feature preprocessor (FP) objects that learn preprocessing parameters from the training data and apply preprocessing of both the training and test data based on these parameters.
result_metrics	A list of result metric (RM) objects that are used to evaluate the classification performance. If this is set to NULL then the rm_main_results(), rm_confusion_matrix() results metrics will be used.
num_resample_runs	The number of times the cross-validation should be run (i.e., "resample runs"), where on each run, new training and test sets are generated. If pseudo-populations are used (e.g., with the ds_basic), then new pseudo-populations will be generated on each resample run as well.
run_TCD	A Boolean indicating whether a Temporal Cross-Decoding (TCD) analysis should be run where the the classifier is trained and tested at all points in time. Setting this to FALSE causes the classifier to only be tested at same time it is trained on which can speed up the analysis run time and save memory at the cost of not calculated the temporal cross decoding results.
num_parallel_cores	An integers specifying the number of parallel cores to use when executing the resample runs in the analysis. The default (NULL) value is to use half of the cores detected on the system. If this value is set to a value of less than 1, then the code will be run serially and messages will be printed showing how long each CV split took to run which is useful for debugging.
parallel_outfile	A string specifying the name of a file where the output from running the code in parallel is written (this argument is ignored if num_parallel_cores < 1). By default the parallel output is written to dev/null so it is not accessible. If this is changed to an empty string the output will be written to the screen, otherwise it will be written to a file name specified. See parallel::makeCluster for more details.

Details

A cross-validator object takes a datasource (DS), a classifier (CL), feature preprocessors (FP) and result metric (RM) objects, and runs multiple cross-validation cycles where:

1. A datasource (DS) generates training and test data splits of the data
2. Feature preprocessors (FPs) do preprocessing of the data
3. A classifier (CL) is trained and predictions are generated on a test set
4. Result metrics (RMs) assess the accuracy of the predictions and compile the results.

Value

This constructor creates an NDR cross-validator object with the class `cv_standard`. Like all NDR cross-validator objects, one should use `run_decoding` method to run a decoding analysis.

Examples

```
data_file <- system.file("extdata/ZD_150bins_50sampled.Rda",
  package = "NeuroDecodeR")

ds <- ds_basic(data_file, "stimulus_ID", 18)
fps <- list(fp_zscore())
cl <- cl_max_correlation()

cv <- cv_standard(datasource = ds,
                  classifier = cl,
                  feature_preprocessors = fps,
                 num_resample_runs = 2)  # better to use more resample runs (default is 50)




# alternatively, one can also use the pipe (|>) to do an analysis
data_file2 <- system.file("extdata/ZD_500bins_500sampled.Rda",
  package = "NeuroDecodeR")

DECODING_RESULTS <- data_file2 |>
    ds_basic('stimulus_ID', 18) |>
    cl_max_correlation() |>
    fp_zscore() |>
    rm_main_results() |>
    rm_confusion_matrix() |>
    cv_standard(num_resample_runs = 2) |>
    run_decoding()
```

ds_basic *A basic datasource (DS)*

Description

The standard datasource used to get training and test splits of data.

Usage

```
ds_basic(
  binned_data,
  labels,
  num_cv_splits,
  use_count_data = FALSE,
  num_label_repeats_per_cv_split = 1,
  label_levels = NULL,
  num_resample_sites = NULL,
  site_IDs_to_use = NULL,
  site_IDs_to_exclude = NULL,
  randomly_shuffled_labels = FALSE,
  create_simultaneous_populations = 0
)
```

Arguments

binned_data	A string that list a path to a file that has data in binned format, or a data frame of binned_data that is in binned format.
labels	A string specifying the name of the labels that should be decoded. This label must be one of the columns in the binned data that starts with 'label.'. For example, if there was a column name in a binned data file called labels.stimulus_ID that you wanted to decode, then you would set this argument to be "stimulus_ID".
num_cv_splits	A number specifying how many cross-validation splits should be used.
use_count_data	If the binned data is neural spike counts, then setting use_count_data = TRUE will convert the data into spike counts. This is useful for classifiers that work on spike count data, e.g., the poisson_naive_bayes_CL.
num_label_repeats_per_cv_split	A number specifying how many times each label should be repeated in each cross-validation split.
label_levels	A vector of strings specifying specific label levels that should be used. If this is set to NULL then all label levels available will be used.
num_resample_sites	The number of sites that should be randomly selected when constructing training and test vectors. This number needs to be less than or equal to the number of sites available that have num_cv_splits * num_label_repeats_per_cv_split repeats.

`site_IDs_to_use`
: A vector of integers specifying which sites should be used. If this is NULL (default value), then all sites that have num_cv_splits * num_label_repeats_per_cv_split repeats will be used, and a message about how many sites are used will be displayed.

`site_IDs_to_exclude`
: A vector of integers specifying which sites should be excluded.

`randomly_shuffled_labels`
: A Boolean specifying whether the labels should be shuffled prior to running an analysis (i.e., prior to the first call to the the get_data() method). This is used when one wants to create a null distribution for comparing when decoding results are above chance.

`create_simultaneous_populations`
: If the data from all sites was recorded simultaneously, then setting this variable to 1 will cause the get_data() function to return simultaneous populations rather than pseudo-populations.

Details

This 'basic' datasource is the datasource that will most commonly be used for most analyses. It can generate training and tests sets for data that has been recorded simultaneously or pseudo-populations for data that was not recorded simultaneously.

Like all datasources, this datasource takes binned format data and has a `get_data()` method that is never explicitly called by the user of the package, but rather it is called internally by a cross-validation object to get training and testing splits of data that can be passed to a classifier.

Value

This constructor creates an NDR datasource object with the class `ds_basic`. Like all NDR datasource objects, this datasource will be used by the cross-validator to generate training and test data sets.

See Also

Other datasource: `ds_generalization()`

Examples

```
# A typical example of creating a datasource to be passed cross-validation object
data_file <- system.file(file.path("extdata", "ZD_150bins_50sampled.Rda"), package = "NeuroDecodeR")
ds <- ds_basic(data_file, "stimulus_ID", 18)

# If one has many repeats of each label, decoding can be faster if one
# uses fewer CV splits and repeats each label multiple times in each split.
ds <- ds_basic(data_file, "stimulus_ID", 6,
  num_label_repeats_per_cv_split = 3
)

# One can specify a subset of labels levels to be used in decoding. Here
#  we just do a three-way decoding analysis between "car", "hand" and "kiwi".
```

```
ds <- ds_basic(data_file, "stimulus_ID", 18,
  label_levels = c("car", "hand", "kiwi")
)

# One never explicitly calls the get_data() function, but rather this is
# called by the cross-validator. However, to illustrate what this function
# does, we can call it explicitly here to get training and test data:
all_cv_data <- get_data(ds)
names(all_cv_data)
```

`ds_generalization` *A datasource (DS) that allows training and testing on different but related labels*

Description

This datasource is useful for assessing whether information is invariant/abstract to particular conditions.

Usage

```
ds_generalization(
  binned_data,
  labels,
  num_cv_splits,
  train_label_levels,
  test_label_levels,
  use_count_data = FALSE,
  num_label_repeats_per_cv_split = 1,
  num_resample_sites = NULL,
  site_IDs_to_use = NULL,
  site_IDs_to_exclude = NULL,
  randomly_shuffled_labels = FALSE,
  create_simultaneous_populations = 0
)
```

Arguments

`binned_data`	A string that list a path to a file that has data in binned format, or a data frame of binned_data that is in binned format.
`labels`	A string specifying the name of the labels that should be decoded. This label must be one of the columns in the binned data that starts with 'label.'
`num_cv_splits`	A number specifying how many cross-validation splits should be used.
`train_label_levels`	A list that contains vectors specifying which label levels belong to which training class. Each element in the list corresponds to a class that the specified training labels will be mapped to. For example, values in the vector in the first element in the list will be mapped onto the first training class, etc.

`test_label_levels`
: A list that contains vectors specifying which label levels belong to which test class. Each element in the list corresponds to a class that the specified test labels will be mapped to. For example, values in the vector in the first element in the list will be mapped onto the first test class, etc. The number of elements in this list must be the same as the number of elements in `train_label_levels`.

`use_count_data`
: If the binned data is neural spike counts, then setting use_count_data = TRUE will convert the data into spike counts. This is useful for classifiers that work on spike count data, e.g., the poisson_naive_bayes_CL.

`num_label_repeats_per_cv_split`
: A number specifying how many times each label level should be repeated in each cross-validation split.

`num_resample_sites`
: The number of sites that should be randomly selected when constructing training and test vectors. This number needs to be less than or equal to the number of sites available that have num_cv_splits * num_label_repeats_per_cv_split repeats.

`site_IDs_to_use`
: A vector of integers specifying which sites should be used.

`site_IDs_to_exclude`
: A vector of integers specifying which sites should be excluded.

`randomly_shuffled_labels`
: A Boolean specifying whether the labels should be shuffled prior to running an analysis (i.e., prior to the first call to the the get_data() method). This is used when one wants to create a null distribution for comparing when decoding results are above chance.

`create_simultaneous_populations`
: If the data from all sites were recorded simultaneously, then setting this variable to 1 will cause the get_data() function to return simultaneous populations rather than pseudo-populations.

Details

Like all datasources, this datasource takes binned format data and has a get_data() method that is called by a cross-validation object to get training and testing splits of data that can be passed to a classifier.

Value

This constructor creates an NDR datasource object with the class `ds_generalization`. Like all NDR datasource objects, this datasource will be used by the cross-validator to generate training and test data sets.

See Also

Other datasource: `ds_basic()`

Examples

```
# One can test if a neural population contains information that is position
# invariant by generating training data for objects presented at 'upper' and 'middle'
# locations, and generating test data at a 'lower' location.

id_levels <- c("hand", "flower", "guitar", "face", "kiwi", "couch", "car")
train_label_levels <- NULL
test_label_levels <- NULL
for (i in seq_along(id_levels)) {
  train_label_levels[[i]] <- c(
    paste(id_levels[i], "upper", sep = "_"),
    paste(id_levels[i], "middle", sep = "_")
  )
  test_label_levels[[i]] <- list(paste(id_levels[i], "lower", sep = "_"))
}


data_file <- system.file("extdata/ZD_150bins_50sampled.Rda", package = "NeuroDecodeR")
ds <- ds_generalization(
  data_file,
  "combined_ID_position", 18,
  train_label_levels,
  test_label_levels
)
```

---

fp_select_k_features *A feature preprocessor (FP) that reduces data to the k most selective features*

Description

This feature preprocessor object applies an ANOVA to the training data to find the p-value of all features. It then either uses the top k features with the smallest p-values, or it removes the features with the smallest k p-values. Additionally, this function can be used to remove the top k p-values and then use only the following j next smallest p-values (for example, this can be useful if one is interesting in comparing the performance using the most selective 10 neurons to using the next 10 most selective neurons, etc.).

Usage

```
fp_select_k_features(
  ndr_container_or_object = NULL,
  num_sites_to_use = NA,
  num_sites_to_exclude = NA
)
```

Arguments

`ndr_container_or_object`
: The purpose of this argument is to make the constructor of the fp_select_k_features feature preprocessor work with the pipe (|>) operator. This argument should almost never be directly set by the user to anything other than NULL. If this is set to the default value of NULL, then the constructor will return a fp_select_k_features object. If this is set to an ndr container, then a fp_select_k_features object will be added to the container and the container will be returned. If this argument is set to another ndr object, then both that ndr object as well as a new fp_select_k_features object will be added to a new container and the container will be returned.

`num_sites_to_use`
: The number of features with the smallest p-values to use.

`num_sites_to_exclude`
: The number of features with the smallest p-values that should be excluded.

Value

This constructor creates an NDR feature preprocessor object with the class `fp_select_k_features`. Like all NDR feature preprocessor objects, this feature preprocessor will be used by the cross-validator to pre-process the training and test data sets.

See Also

Other feature_preprocessor: `fp_zscore()`

Examples

```
# This will cause the cross-validator use only the 50 most selective sites
fp <- fp_select_k_features(num_sites_to_use = 50)

# This will cause the cross-validator to remove the 20 most selective sites
fp <- fp_select_k_features(num_sites_to_exclude = 20)

# This will cause the cross-validator to remove the 20 most selective sites
# and then use only the 50 most selective sites that remain after the 20 are
# eliminated
fp <- fp_select_k_features(num_sites_to_use = 50, num_sites_to_exclude = 20)
```

`fp_zscore` *A feature preprocessor (FP) that z-score normalizes the data*

Description

This feature preprocessor object finds the mean and standard deviation using the training data. The preprocessor then z-score transforms the training and test data using this mean and standard deviation by subtracting the mean and dividing by the standard deviation.

Usage

```
fp_zscore(ndr_container_or_object = NULL)
```

Arguments

`ndr_container_or_object`
: The purpose of this argument is to make the constructor of the fp_zscore feature preprocessor work with the pipe (|>) operator. This argument should almost never be directly set by the user to anything other than NULL. If this is set to the default value of NULL, then the constructor will return a fp_zscore object. If this is set to an ndr container, then a fp_zscore object will be added to the container and the container will be returned. If this argument is set to another ndr object, then both that ndr object as well as a new fp_zscore object will be added to a new container and the container will be returned.

Details

This feature preprocessor object applies z-score normalization to each feature by calculating the mean and the standard deviation for each feature using the training data, and then subtracting the mean and dividing by the standard deviation for each feature in the training and test sets. This function is useful for preventing some classifiers from relying too heavily on particular features when different features can have very different ranges of values (for example, it is useful when decoding neural data because different neurons can have different ranges of firing rates).

Value

This constructor creates an NDR feature preprocessor object with the class `fp_zscore`. Like all NDR feature preprocessor objects, this feature preprocessor will be used by the cross-validator to pre-process the training and test data sets.

See Also

Other feature_preprocessor: `fp_select_k_features()`

Examples

```
# The fp_zscore() constructor does not take any parameters. This object
# just needs to added to a list and passed to the cross-validator applied
fp <- fp_zscore()
```

---

get_num_label_repetitions

Get the number of sites have at least k trials of each label level

---

Description

Calculates number of sites that have at least k label level repetitions for all values k. This information is useful for assessing how to set the number of cross-validation splits (and repeats of labels per cross-validation split) to use in a datasource. One can also assess the number of label level repetitions separately conditioned on another site_info variable. For example, if one has recordings from different brain regions, and the brain region information is contained in a site_info variable, then one could calculate how many sites have at least k repetitions for each stimulus in each brain region.

Usage

```
get_num_label_repetitions(
  binned_data,
  labels,
  site_info_grouping_name = NULL,
  label_levels = NULL
)
```

Arguments

`binned_data`	A string that list a path to a file that has data in binned format, or a data frame of binned_data that is in binned format.
`labels`	A string specifying which label variable should be used for calculating the minimum number of level repetitions.
`site_info_grouping_name`	A character string that specifies if the number of sites that have k repetitions should be computed separately based on the levels of a site_info variable.
`label_levels`	A character vector specifying which levels to include. If not set, all levels will be used.

Value

A data frame with the class `label_repetition` which allows the results to be plotted. The returned data frame has a row for each label level, and columns with sequential integer values k = 0, 1, ... The values in the data frame show the number of sites that have at least k repetitions of a given stimulus.

Note

The returned value is an S3 object that inherits from data.frame that has an associated plot() method.

Examples

```
data_file <- system.file("extdata/ZD_150bins_50sampled.Rda", package = "NeuroDecodeR")
label_rep_info <- get_num_label_repetitions(data_file, "stimulus_ID")
plot(label_rep_info)
```

get_num_label_repetitions_each_site

Get the number of trial repetitions for a given label for each site

Description

Calculates how many repeated trials there are for each label level for each site. This can be useful for selecting sites that have a minimum number of repetitions of each stimulus or other experimental condition.

Usage

```
get_num_label_repetitions_each_site(binned_data, labels, label_levels = NULL)
```

Arguments

binned_data	A string that list a path to a file that has data in binned format, or a data frame of binned_data that is in binned format.
labels	A string specifying which label variable should be used for calculating the minimum number of level repetitions.
label_levels	A character vector specifying which levels to include. If not set, all levels will be used.

Value

A data frame where each row corresponds to a recording site. The columns in the data frame are:

- *siteID*: The siteID each row in the data frame corresponds to
- *min_repeats*: minimum number of repeats across all label levels
- *level_XXX*: The number or repeats for a specific label level
- *site_info.XXX*: The site_info for each site

get_parameters.cv_standard

Get parameters of an NeuroDecodeR object

Description

Returns the parameters set in an NDR object to enable reproducible analyses.

Usage

```
## S3 method for class 'cv_standard'
get_parameters(ndr_obj)
```

Arguments

ndr_obj	An object from the NeuroDecodeR package to get the parameters from.

Details

This function that returns a data frame with the parameters of an NeuroDecodeR (NDR) object. All NDR objects (i.e., DS, FP, CL, RM and CV) need to define a method that implements this generic function. The CV object's `get_parameters()` method usually will call all the DS, FP, CL, RM and CV `get_parameters()` methods and aggregate and return all the parameters aggregated from these objects. These aggregated parameters can then be used to save the results of a particular analysis based on the parameters using the `log_save_results()` function. This method is most frequently used privately by other NDR objects to save all the parameters that were used in an analysis.

Value

Returns a data frame with a single row that contains all the NDR object's parameter values (e.g., values that were set in the object's constructor).

---

get_siteIDs_with_k_label_repetitions

Get the sitesIDs that have at least k trials for all label level

---

Description

This function gets the siteIDs that have at least k label level repetitions. These siteIDs can be used in a datasource to only get data from sites that have enough label repetitions. For example, one could use these siteIDs in conjunction with the ds_basic's site_IDs_to_use argument to only get data from sites that have enough repetitions of each stimulus.

Usage

```
get_siteIDs_with_k_label_repetitions(
  binned_data,
  labels,
  k,
  label_levels = NULL
)
```

Arguments

binned_data	A string that list a path to a file that has data in binned format, or a data frame of binned_data that is in binned format.
labels	A string specifying which label variable should be used when calculating the minimum number of level repetitions.
k	A number specifying that all sitesIDs returned should have at least k repetitions of all label levels.
label_levels	A character vector specifying which levels to include. If not set, all levels will be used.

Value

A vector of integers that specific which siteIDs have at least k repetitions of each label level (from the label levels that are used).

Examples

```
data_file <- system.file("extdata/ZD_150bins_50sampled.Rda", package = "NeuroDecodeR")
get_siteIDs_with_k_label_repetitions(data_file, "stimulus_ID", 5)
```

---

log_check_results_already_exist

A function that checks if a decoding analysis has already been run

---

Description

A function that checks if a decoding analysis has already been run

Usage

```
log_check_results_already_exist(decoding_params, manifest_df)
```

Arguments

`decoding_params`	A data frame of decoding parameters that can be created by calling the cross-validator's `get_parameters()` method.
`manifest_df`	A manifest data frame that has the list of parameters for which decoding analyses have already been run.

Value

returns a Boolean indicating if results with a given set of parameters already exist in the manifest data frame.

`log_load_results_from_params`

A function that loads DECODING_RESULTS based on decoding_parameters

---

Description

A function that loads DECODING_RESULTS based on decoding_parameters

Usage

```
log_load_results_from_params(decoding_params, results_dir_name)
```

Arguments

`decoding_params`
: A data frame of decoding parameters that can be created by calling the cross-validator's `get_parameters()` method.

`results_dir_name`
: A string containing the path to a directory that contains all the decoding results.

Value

A list that has all the DECODING_RESULTS that match the parameters that were specified. If only a single result matches the parameters specified, then this DECODING_RESULTS is returned rather than a list of DECODING_RESULTS.

`log_load_results_from_result_name`

A function that loads DECODING_RESULTS based on the result_name

---

Description

A function that loads DECODING_RESULTS based on the result_name

Usage

```
log_load_results_from_result_name(result_name, results_dir_name)
```

Arguments

`result_name`
: A string a specifying the result that should be loaded based on the name given. This result_name can be a regular expression in which all result_name values that match the regular expression will be returned as a list.

`results_dir_name`
: A string containing the path to a directory that contains all the decoding results.

Value

A named list that has all the DECODING_RESULTS that match the `result_name` argument value in the manifest file's `result_name` column. The names on the list that are returned correspond to the result_names for each result in the manifest file. If `result_name` argument matches only one result, then this DECODING_RESULTS is returned rather than a list of DECODING_RESULTS.

`log_save_results` *Saves the DECODING_RESULTS and logs the parameters used in the analysis*

Description

This function takes results returned by the cross-validator's `run_decoding()` method and uses the cross-validator's `get_properties()` method to save a log of the results that be used to reload the results.

Usage

```
log_save_results(
  DECODING_RESULTS,
  save_directory_name,
  result_name = "No result name set"
)
```

Arguments

`DECODING_RESULTS`	A list of results returned by the cross-validator's run_decoding method.
`save_directory_name`	A string specifying the directory name where the decoding results should be saved.
`result_name`	A string that gives a human readable name for the results that are to be saved. This name can be used to load the results later. The default value is "No result name set".

Value

Does not return a value but instead creates a directory that stores an .rda file with the decoding results and either creates or updates a manifest files that has information about the decoding results.

plot.label_repetition *A plot function for label_repetition object*

Description

This function plots the number of sites have at least k label repetitions. Creating this plot is useful for assessing how to set the number of cross-validation splits (and repeats of labels per cross-validation split) to use in a datasource. This function returns a ggplot2 object which can be further modified as needed.

Usage

```
## S3 method for class 'label_repetition'
plot(x, ..., show_legend = TRUE)
```

Arguments

x	A label_repetition object that was generated from calling the get_num_label_repetitions() function.
...	This is needed to conform to the plot generic interface.
show_legend	A Boolean specifying whether to show a legend that list which label each color in the plot corresponds to.

Value

. Returns a ggplot object that plots the number of sites that that have at least k label repetitions as a function of k.

plot.raster_data *A plot function for data in raster format*

Description

This function will plot data that is in raster format. If the data is a spike train consisting of only 0's and 1's then it will create a plot of spikes as black tick marks on a white background. If the raster data contains continuous data, then the plot will be color coded.

Usage

```
## S3 method for class 'raster_data'
plot(x, ..., facet_label = NULL)
```

Arguments

x	Either data that is in raster format, or a string containing the name of a file that has data in raster format.
...	This is needed to conform to the plot generic interface.
facet_label	If this is set to a string that is the name of one of the labels in the raster data, then the raster plots will be faceted by this label.

Value

Returns a ggplot object that plots the raster data.

---

plot.rm_confusion_matrix

A plot function for the rm_confusion_matrix object

Description

This function plots confusion matrices after the decoding analysis has been run (and all results have been aggregated). This function can also plot mutual information calculated from the confusion matrix.

Usage

```
## S3 method for class 'rm_confusion_matrix'
plot(
  x,
  ...,
  results_to_show = "zero_one_loss",
  plot_TCD_results = FALSE,
  plot_only_one_train_time = NULL
)
```

Arguments

x	A rm_confusion_matrix object that has aggregated runs from a decoding analysis, e.g., if DECODING_RESULTS are the output from the run_decoding(cv) then this argument should be `DECODING_RESULTS$rm_confusion_matrix`.
...	This is needed to conform to the plot generic interface.
results_to_show	A string specifying the type of result to plot that can take the following values: • "zero_one_loss": plot a regular confusion matrix. • "decision_vals": plot a confusion matrix with the average decision values. • "mutual_information": plot the mutual information calculated from the zero-one loss confusion matrix.

plot_TCD_results
: A Boolean indicating whether the a cross-temporal decoding of the confusion matrices should only be plotted. If the results_to_show == "mutual_information" setting this to TRUE will plot a TCD plot of the mutual information otherwise it will plot a line plot of the mutual information for training and testing at the same time.

plot_only_one_train_time
: If this is set to a numeric value the the confusion matrix will only be plotted for the training time *start time* that is specified. If the number passed is not equal to an exact start training time, then the closest training time will be used and a message saying that the time specified does not exist will be printed.

Value

Returns a ggplot object that plots the confusion matrix results.

See Also

Other result_metrics: plot.rm_main_results(), plot_main_results(), rm_confusion_matrix(), rm_main_results()

---

plot.rm_main_results *A plot function for the rm_main_results object*

Description

This function can create a line plot of the results or temporal cross-decoding results for the the zero-one loss, normalized rank and/or decision values after the decoding analysis has been run (and all results have been aggregated).

Usage

```
## S3 method for class 'rm_main_results'
plot(x, ..., results_to_show = "zero_one_loss", errorbar = NULL, type = "TCD")
```

Arguments

x
: A rm_main_result object that has aggregated runs from a decoding analysis, e.g., if DECODING_RESULTS are the out from the run_decoding(cv) then this argument should be DECODING_RESULTS$rm_main_results.

...
: This is needed to conform to the plot generic interface.

results_to_show
: A string specifying the types of results to plot. Options are: 'zero_one_loss', 'normalized_rank', 'decision_values', or 'all'.

errorbar — A string specifying if error bars should be plotted. Options are: 'sd', 'se', or '2se'. If this is set to NULL, then no error bars will be plotted. If this is set to 'sd', then the standard deviation of the results will be plotted. If this is set to 'se', then the standard error of the results will be plotted. If this is set to '2se', then two times the standard error of the results will be plotted (which is often used to represent a 95% confidence interval). Note, these error bars are slight underestimates of the sd and sderr because when using cross-validation the test data is not independent of the training data. Also, note that error bars can only be plotted for line plots and not for TCD plots.

type — A string specifying the type of results to plot. Options are 'TCD' to plot a temporal cross decoding matrix or 'line' to create a line plot of the decoding results as a function of time.

Value

Returns a ggplot object that plots the main results.

See Also

Other result_metrics: `plot.rm_confusion_matrix()`, `plot_main_results()`, `rm_confusion_matrix()`, `rm_main_results()`

---

plot_main_results *A plot function to plot multiple rm_main_results*

Description

This function can create a line plot of the results or temporal cross-decoding results for the the zero-one loss, normalized rank and/or decision values after the decoding analysis has been run (and all results have been aggregated).

Usage

```
plot_main_results(
  results_dir_name,
  results_to_plot,
  results_to_show = "zero_one_loss",
  type = "line",
  errorbar = NULL,
  display_names = NULL
)
```

Arguments

results_dir_name — A string specifying the directory name that contains files with DECODING_RESULTS that have rm_main_results as one of the result metrics.

`results_to_plot`
: This can be set to a vector of strings specifying result_names for the results to plot, or a vector of numbers that contain the rows in the results_manifest file of the results that should be compared. The results_manifest file should be created from saving results using the log_save_results() function. Finally, if this is set to a single string that is a regular expression, all results in the results_manifest file result_name variable that match the regular expression will be plotted.

`results_to_show`
: A string specifying the types of results to plot. Options are: 'zero_one_loss', 'normalized_rank', 'decision_values', or 'all'.

`type`
: A string specifying the type of results to plot. Options are 'TCD' to plot a temporal cross decoding matrix or 'line' to create a line plot of the decoding results as a function of time.

`errorbar`
: A string specifying if error bars should be plotted. Options are: 'sd', 'se', or '2se'. If this is set to NULL, then no error bars will be plotted. If this is set to 'sd', then the standard deviation of the results will be plotted. If this is set to 'se', then the standard error of the results will be plotted. If this is set to '2se', then two times the standard error of the results will be plotted (which is often used to represent a 95% confidence interval). Note, these error bars are slight underestimates of the sd and sderr because when using cross-validation the test data is not independent of the training data. Also, note that error bars can only be plotted for line plots and not for TCD plots.

`display_names`
: A vector of strings specifying what the labels on the plots should say for each result. If this is NULL, the result names will be the names from the manifest file's result_name column, or if these are set to "No result name set" then the analysisID will be the label.

Value

Returns a ggplot object that a comparison of main decoding results.

See Also

Other result_metrics: `plot.rm_confusion_matrix()`, `plot.rm_main_results()`, `rm_confusion_matrix()`, `rm_main_results()`

---

read_raster_data *Read a csv, rda, rds or mat file in raster format*

Description

Reads a csv, rda, rds or mat file that has the appropriate raster_data column names (i.e., columns that start with site.info, labels. and time.), and returns data in raster_data format (i.e., a data frame with the raster.data class attribute).

Usage

```
read_raster_data(raster_file_name)
```

Arguments

raster_file_name
: A string specifying the name (and path) to a csv, rda, rds or mat raster data file that has the appropriate raster data column names (i.e., columns that start with site.info, labels. and time.)

Value

Returns a data frame of data in raster format (i.e., with class raster_data). Data that is in raster format as the following variables:

1. labels.XXX These variables contain labels of which experimental conditions were shown on a given trial.
2. time.XXX_YYY These variables contain the data for a given time, XXX is the start time of the data in a particular bin and YYY is the end time.
3. site_info.XXX These variables contain additional meta data about the site.
4. trial_number This variable specifies a unique number for each row indicating which trial a given row of data came from.

For more details on raster format data see the vignette: vignette("data_formats", package = "NeuroDecodeR")

Examples

```
# reading in a csv file in raster format
csv_raster_file_name <- file.path(
  system.file("extdata", package = "NeuroDecodeR"),
  "Zhang_Desimone_7object_raster_data_small_csv",
  "bp1001spk_01A_raster_data.csv"
)

# read the csv file into a raster_data data frame
raster_data <- read_raster_data(csv_raster_file_name)
```

---

rm_confusion_matrix *A result metric (RM) that calculates confusion matrices*

Description

This result metric calculate a confusion matrices from all points in time.

Usage

```
rm_confusion_matrix(
  ndr_container_or_object = NULL,
  save_TCD_results = FALSE,
  create_decision_vals_confusion_matrix = TRUE
)
```

Arguments

`ndr_container_or_object`
: The purpose of this argument is to make the constructor of the rm_confusion_matrix feature preprocessor work with the magrittr pipe (|>) operator. This argument should almost never be directly set by the user to anything other than NULL. If this is set to the default value of NULL, then the constructor will return a rm_confusion_matrix object. If this is set to an ndr container, then a rm_confusion_matrix object will be added to the container and the container will be returned. If this argument is set to another ndr object, then both that ndr object as well as a new rm_confusion_matrix object will be added to a new container and the container will be returned.

`save_TCD_results`
: A Boolean specifying whether one wants to save results to allow one to create temporal cross decoding confusion matrices; i.e., confusion matrices when training at one point in time and testing a different point in time. Setting this to FALSE can save memory.

`create_decision_vals_confusion_matrix`
: A boolean specifying whether one wants to create a confusion matrix of the decision values. In this confusion matrix, each row corresponds to the correct class (like a regular confusion matrix) and each column corresponds to the mean decision value of the predictions for each class.

Details

Like all result metrics, this result metric has functions to aggregate results after completing each set of cross-validation classifications, and also after completing all the resample runs. The results should then be available in the DECODING_RESULTS object returned by the cross-validator.

Value

This constructor creates an NDR result metric object with the class `rm_confusion_matrix`. Like all NDR result metric objects, this result metric will be used by a cross-validator to create a measure of decoding accuracy by aggregating the results after all cross-validation splits have been run, and after all resample runs have completed.

See Also

Other result_metrics: `plot.rm_confusion_matrix()`, `plot.rm_main_results()`, `plot_main_results()`, `rm_main_results()`

Examples

```
# If you only want to use the rm_confusion_matrix(), then you can put it in a
# list by itself and pass it to the cross-validator.
the_rms <- list(rm_confusion_matrix())
```

`rm_main_results` *A result metric (RM) that calculates main decoding accuracy measures*

Description

This result metric calculate the zero-one loss, the normalized rank, and the mean of the decision values. This is also an S3 object which has an associated plot function to display the results.

Usage

```
rm_main_results(
  ndr_container_or_object = NULL,
  include_norm_rank_results = TRUE
)
```

Arguments

`ndr_container_or_object`
: The purpose of this argument is to make the constructor of the rm_main_results feature preprocessor work with the magrittr pipe (%>%) operator. This argument should almost never be directly set by the user to anything other than NULL. If this is set to the default value of NULL, then the constructor will return a rm_main_results object. If this is set to an ndr container, then a rm_main_results object will be added to the container and the container will be returned. If this argument is set to another ndr object, then both that ndr object as well as a new rm_main_results object will be added to a new container and the container will be returned.

`include_norm_rank_results`
: An argument specifying if the normalized rank and decision value results should be saved. If this is a Boolean set to TRUE, then the normalized rank and decision values for the correct category will be calculated. If this is a Boolean set to FALSE then the normalized rank and decision values will not be calculated. If this is a string set to "only_same_train_test_time", then the normalized rank and decision values will only be calculated when for results when training and testing at the same time. Not returning the full results can speed up the run-time of the code and will use less memory so this can be useful for large data sets.

Details

Like all result metrics, this result metric has functions to aggregate results after completing each set of cross-validation classifications, and also after completing all the resample runs. The results should then be available in the DECODING_RESULTS object returned by the cross-validator.

Value

This constructor creates an NDR result metric object with the class `rm_main_results`. Like all NDR result metric objects, this result metric will be used by a cross-validator to create a measure of decoding accuracy by aggregating the results after all cross-validation splits have been run, and after all resample runs have completed.

See Also

Other result_metrics: `plot.rm_confusion_matrix()`, `plot.rm_main_results()`, `plot_main_results()`, `rm_confusion_matrix()`

Examples

```
# If you only want to use the rm_main_results(), then you can put it in a
# list by itself and pass it to the cross-validator.
the_rms <- list(rm_main_results())
```

`run_decoding.cv_standard`

A cross-validator (CV) method to run a decoding analysis

Description

This method runs a full decoding analysis based on the DS, FP, CL, and RM objects that are passed to the cross-validator constructor.

Usage

```
## S3 method for class 'cv_standard'
run_decoding(cv_obj)
```

Arguments

`cv_obj`	A CV object. Parameters that affect the decoding analyses are set in the CV's constructor.

Value

A list, usually called `DECODING_RESULTS`, that contains the results from the decoding analysis. This `DECODING_RESULTS` list should contain the result compiled by the result metric objects, as well as a list in `DECODING_RESULTS$cross_validation_paramaters$parameter_df` contains data on all that DS, FP, CL and RM parameters that were used in the decoding analysis that can be used to store and retrieve the results. Additionally, the DS, FP, CL and RM objects used in the analysis can be saved in the `DECODING_RESULTS$cross_validation_paramaters`.

See Also

cv_standard()

---

test_valid_raster_format *Tests if a data frame is in valid raster format*

Description

This function takes a data frame and tests that the data frame is in valid raster format by checking that the data frame contains variables with the appropriate names. If the data frame is not in correct raster format, an error will be thrown that contains a message why the data is not in valid raster format.

Usage

```
test_valid_raster_format(raster_data)
```

Arguments

raster_data	A data frame or string specifying a file that will be checked to see if it is in valid raster format.

Value

Returns NULL if object is in valid raster format. Otherwise it will give an error message.

Examples

```
# This is valid raster data so the function will return no error message
raster_dir_name <- file.path(
  system.file("extdata", package = "NeuroDecodeR"),
  "Zhang_Desimone_7object_raster_data_small_rda"
)
file_name <- "bp1001spk_01A_raster_data.rda"
raster_full_path <- file.path(raster_dir_name, file_name)

test_valid_raster_format(raster_full_path)


# Binned data is not in raster format (it has an extra column called siteID) so
# checking if it is in raster format should return an error.

binned_file_name <- system.file(file.path("extdata", "ZD_150bins_50sampled.Rda"),
                                package = "NeuroDecodeR")
try(test_valid_raster_format(binned_file_name))
```

Index