Package: maxent.ot (via r-universe)

November 23, 2024

Title Perform Phonological Analyses using Maximum Entropy Optimality Theory

Version 1.0.0

Description Fit Maximum Entropy Optimality Theory models to data sets, generate the predictions made by such models for novel data, and compare the fit of different models using a variety of metrics. The package is described in Mayer, C., Tan, A., Zuraw, K. (in press) <https://sites.socsci.uci.edu/~cjmayer/papers/cmayer_et_al_maxent_ot_accepted.pdf>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports data.table

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/connormayer/maxent.ot

BugReports https://github.com/connormayer/maxent.ot/issues

VignetteBuilder knitr

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

RemoteUrl https://github.com/connormayer/maxent.ot

RemoteRef HEAD

RemoteSha a09eca6c4e582e584ec2c76ed1a279e3771a48d4

Contents

compare_models 2
cross_validate 5
monte_carlo_weights 8
optimize_weights 10
otsoft_bias_to_df 13
otsoft_tableaux_to_df 14
predict_probabilities 15

Index **18**

compare_models *Compare Maxent OT models using a variety of methods*

Description

Compares two or more model fit to the same data set to determine which provides the best fit, using a variety of methods.

Usage

```
compare_models(..., method = "lrt")
```

Arguments

`...` Two or more models objects to be compared. These objects should be in the same format as the objects returned by the `optimize_weights` function. Note that the likelihood ratio test applies to exactly two models, while the other comparison methods can be applied to arbitrarily many models.

`method` The method of comparison to use. This currently includes `lrt` (likelihood ratio test), `aic` (Akaike Information Criterion), `aic_c` (Akaike Information Criterion adjusted for small sample sizes), and `bic` (Bayesian Information Criterion).

Details

The available comparison methods are

- **lrt**: The likelihood ratio test. This method can be applied to a maximum of two models, and the parameters of these models (i.e., their constraints) *must be in a strict subset/superset relationship.* If your models do not meet these requirements, you should use a different method.

  The likelihood ratio is calculated as follows:

  $$LR = 2(LL_2 - LL_1)$$

  where LL_2 is log likelihood of the model with more parameters. A p-value is calculated by conducting a chi-squared test with $X^2 = LR$ and the degrees of freedom set to the difference in number of parameters between the two models. This p-value tells us whether the difference in likelihood between the two models is significant (i.e., whether the extra parameters in the full model are justified by the increase in model fit).

- **aic**: The Akaike Information Criterion. This is calculated as follows for each model:

  $$AIC = 2k - 2LL$$

  where k is the number of model parameters (i.e., constraints) and LL is the model's log likelihood.

- **aic_c**: The Akaike Information Criterion corrected for small sample sizes. This is calculated as follows:

  $$AIC_c = 2k - 2LL + \frac{2k^2 + 2k}{n - k - 1}$$

  where n is the number of samples and the other parameters are identical to those used in the AIC calculation. As n approaches infinity, the final term converges to 0, and so this equation becomes equivalent to AIC. Please see the note below for information about sample sizes.

- **bic**: The Bayesian Information Criterion. This is calculated as follows:

  $$BIC = k\ln(n) - 2LL$$

  As with `aic_c`, this calculation relies on the number of samples. Please see the discussion on sample sizes below before using this method.

A few caveats for several of the comparison methods:

- The likelihood ratio test (`lrt`) method applies to exactly two models, and assumes that the parameters of these models are *nested*: that is, the constraints in the smaller model are a strict subset of the constraints in the larger model. This function will verify this to some extent based on the number and names of constraints.
- The Akaike Information Criterion adjusted for small sample sizes (`aic_c`) and the Bayesian Information Criterion (`bic`) rely on sample sizes in their calculations. The sample size for a data set is defined as the sum of the column of surface form frequencies. If you want to apply these methods, it is important that the values in the column are token counts, not relative frequencies. Applying these methods to relative frequencies, which effectively ignore sample size, will produce invalid results.

The `aic`, `aic_c`, and `bic` comparison methods return raw AIC/AICc/BIC values as well as weights corresponding to these values. These weights are calculated similarly for each model:

$$W_i = \frac{\exp(-0.5\delta_i)}{\sum_{j=1}^{m}\exp(-0.5\delta_j)}$$

where δ_i is the difference in score (AIC, AICc, BIC) between model i and the model with the best score, and m is the number of models being compared. These weights provide the relative likelihood or conditional probability of this model being the best model (by whatever definition of "best" is assumed by the measurement type) given the data and the selection of models it is being compared to.

Value

A data frame containing information about the comparison. The contents and size of this data frame vary depending on the method used.

- `lrt`: A data frame with a single row and the following columns:
  - `description`: the names of the two models being compared. The name of the model with more parameters will be first.
  - `chi_sq`: the chi-squared value calculated during the test.
  - `k_delta`: the difference in parameters between the two models used as degrees of freedom in the chi-squared test.
  - `p_value`: the p-value calculated by the test
- `aic`: A data frame with as many rows as there were models passed in. The models are sorted in ascending order of AIC (i.e., best first). This data frame has the following columns:
  - `model`: The name of the model.
  - `k`: The number of parameters.
  - `aic`: The model's AIC value.
  - `aic.delta`: The difference between this model's AIC value and the AIC value of the model with the smallest AIC value.
  - `aic.wt`: The model's AIC weight: this reflects the relative likelihood (or conditional probability) that this model is the "best" model in the set.
  - `cum.wt`: The cumulative sum of AIC weights up to and including this model.
  - `ll`: The log likelihood of this model.
- `aicc`: The data frame returned here is analogous to the structure of the AIC data frame, with AICc values replacing AICs and accordingly modified column names. There is one additional column:
  - `n`: The number of samples in the data the model is fit to.
- `bic`: The data frame returned here is analogous to the structure of the AIC and AICc data frames. Like the AICc data frame, it contains the `n` column.

Examples

```
# Get paths to toy data files
# This file has two constraints
data_file_small <- system.file(
    "extdata", "sample_data_frame.csv", package = "maxent.ot"
)
# This file has three constraints
data_file_large <- system.file(
    "extdata", "sample_data_frame_large.csv", package = "maxent.ot"
)

# Fit weights to both data sets with no biases
tableaux_small <- read.csv(data_file_small)
small_model <- optimize_weights(tableaux_small)

tableaux_large <- read.csv(data_file_large)
large_model <- optimize_weights(tableaux_large)
```

```
# Compare models using likelihood ratio test. This is appropriate here
# because the constraints are nested.
compare_models(small_model, large_model, method='lrt')

# Compare models using AIC
compare_models(small_model, large_model, method='aic')

# Compare models using AICc
compare_models(small_model, large_model, method='aic_c')

# Compare models using BIC
compare_models(small_model, large_model, method='bic')
```

`cross_validate` *Cross-validate bias parameters for constraint weights.*

Description

Performs k-fold cross-validation of a data set and a set of input bias parameters. Cross-validation allows the space of bias parameters to be searched to find the settings that best support generalization to unseen data.

Usage

```
cross_validate(
  input,
  k,
  mu_values,
  sigma_values,
  grid_search = FALSE,
  output_path = NA,
  out_sep = ",",
  control_params = NA,
  upper_bound = DEFAULT_UPPER_BOUND,
  encoding = "unknown",
  model_name = NA,
  allow_negative_weights = FALSE
)
```

Arguments

`input` The input data frame/data table/tibble. This should contain one or more OT tableaux consisting of mappings between underlying and surface forms with observed frequency and violation profiles. Constraint violations must be numeric.

For an example of the data frame format, see inst/extdata/sample_data_frame.csv. You can read this file into a data frame using read.csv or into a tibble using dplyr::read_csv.

This function also supports the legacy OTSoft file format. You can use this format by passing in a file path string to the OTSoft file rather than a data frame. For examples of OTSoft format, see inst/extdata/sample_data_file.txt.

`k` The number of folds to use in cross-validation.

`mu_values` A vector or list of mu bias parameters to use in cross-validation. Parameters may either be scalars, in which case the same mu parameter will be applied to every constraint, or vectors/lists containing a separate mu bias parameter for each constraint.

`sigma_values` A vector or list of sigma bias parameters to use in cross-validation. Parameters may either be scalars, in which case the same sigma parameter will be applied to every constraint, or vectors/lists containing a separate sigma bias parameter for each constraint.

`grid_search` (optional) If TRUE, the Cartesian product of the values in `mu_values` and `sigma_values` will be validated. For example, if `mu_values = c(0, 1)` and `sigma_values = c(0.1, 1)`, cross-validation will be done on the mu/sigma pairs `(0, 0.1), (0, 1), (1, 0.1), (1, 1)`. If FALSE (default), cross-validation will be done on each pair of values at the same indices in `mu_values` and `sigma_values`. For example, if `mu_values = c(0, 1)` and `sigma_values = c(0.1, 1)`, cross-validation will be done on the mu/sigma pairs `(0, 0.1), (1, 1)`.

`output_path` (optional) A string specifying the path to a file to which the cross-validation results will be saved. If the file exists it will be overwritten. If this argument isn't provided, the output will not be written to a file.

`out_sep` (optional) The delimiter used in the output files. Defaults to tabs.

`control_params` (optional) A named list of control parameters that will be passed to the optim function. See the documentation of that function for details. Note that some parameter settings may interfere with optimization. The parameter `fnscale` will be overwritten with `-1` if specified, since this must be treated as a maximization problem.

`upper_bound` (optional) The maximum value for constraint weights. Defaults to 100.

`encoding` (optional) The character encoding of the input file. Defaults to "unknown".

`model_name` (optional) A name for the model. If not provided, the file name will be used if the input is a file path. If the input is a data frame the name of the variable will be used.

`allow_negative_weights`
(optional) Whether the optimizer should allow negative weights. Defaults to FALSE.

Details

The cross-validation procedure is as follows:

1. Randomly divide the data into k partitions.
2. Iterate through every combination of mu and sigma specified in the input arguments (see the documentation for the `grid_search` argument for details on how this is done).
3. For each combination, for each of the k partitions, train a model on the other (k-1) partitions using `optimize_weights` and then run `predict_probabilities` on the remaining partition.
4. Record the mean log likelihood the models apply to the held-out partitions.

Value

A data frame with the following columns:

- `model_name`: the name of the model
- `mu`: the value(s) of mu tested
- `sigma`: the value(s) of sigma tested
- `folds`: the number of folds
- `mean_ll`: the mean log likelihood of k-fold cross-validation using these bias parameters

Examples

```
# Get paths to OTSoft file. Note that you can also pass dataframes into
# this function, as described in the documentation for `optimize`.
data_file <- system.file(
    "extdata", "amp_demo_grammar.csv", package = "maxent.ot"
)
tableaux_df <- read.csv(data_file)

# Define mu and sigma parameters to try
mus <- c(0, 1)
sigmas <- c(0.01, 0.1)

# Do 2-fold cross-validation
cross_validate(tableaux_df, 2, mus, sigmas)

# Do 2-fold cross-validation with grid search of parameters
cross_validate(tableaux_df, 2, mus, sigmas, grid_search=TRUE)

# You can also use vectors/lists for some/all of the bias parameters to set
# separate biases for each constraint
mus_v <- list(
  c(0, 1),
  c(1, 0)
)
sigmas_v <- list(
  c(0.01, 0.1),
  c(0.1, 0.01)
)

cross_validate(tableaux_df, 2, mus_v, sigmas_v)

# Save cross-validation results to a file
tmp_output <- tempfile()
cross_validate(tableaux_df, 2, mus, sigmas, output_path=tmp_output)
```

monte_carlo_weights *Create simulated data and learn weights for these data*

Description

Creates a simulated data set by picking an output for each instance of an input. The probability of picking a particular output is guided by its conditional probability given the input. Learns constraint weights for each simulated data set.

Usage

```
monte_carlo_weights(
  pred_prob,
  num_simul,
  bias_file = NA,
  mu = NA,
  sigma = NA,
  output_path = NA,
  out_sep = ",",
  control_params = NA,
  upper_bound = DEFAULT_UPPER_BOUND,
  allow_negative_weights = FALSE
)
```

Arguments

`pred_prob`	A data frame with a column for predicted probabilities. This object should be in the same format as the `predictions` attribute of the object returned by the `predict_probabilities` function.
`num_simul`	The number of simulations to run.
`bias_file`	(optional) The path to the file containing mus and sigma for constraint biases. If this argument is provided, the scalar and vector mu and sigma arguments will be ignored. Each row in this file should be the name of the constraint, followed by the mu, followed by the sigma (separated by whatever the relevant separator is; default is commas).
`mu`	(optional) A scalar or vector that will serve as the mu for each constraint in the bias term. Constraint weights will also be initialized to this value. If a vector, its length must equal the number of constraints in the input file. This value will not be used if `bias_file` is provided.
`sigma`	(optional) A scalar or vector that will serve as the sigma for each constraint in the bias term. If a vector, its length must equal the number of constraints in the input file. This value will not be used if `bias_file` is provided.
`output_path`	(optional) A string specifying the path to a file to which the output will be saved. If the file exists it will be overwritten. If this argument isn't provided, the output will not be written to a file.

`out_sep` (optional) The delimiter used in the output files. Defaults to commas.

`control_params` (optional) A named list of control parameters that will be passed to the optim function. See the documentation of that function for details. Note that some parameter settings may interfere with optimization. The parameter `fnscale` will be overwritten with `-1` if specified, since this must be treated as a maximization problem.

`upper_bound` (optional) The maximum value for constraint weights. Defaults to 100.

`allow_negative_weights`
(optional) Whether the optimizer should allow negative weights. Defaults to FALSE.

Details

This function creates multiple simulated data sets, and learns a set of weights that maximizes the likelihood of data for each simulated data set.

To create a simulated data set, one output is randomly chosen for each instance of an input. The probability of picking a particular output, O_i, which arises from input I_j depends on $Pr(O_i|I_j)$.

The function `optimize_weights()` is called to find a set of weights that maximize the likelihood of the simulated data. All optional arguments of `optimize_weights()` that were available for the user to specify biases and bounds are likewise available in this function, `monte_carlo_weights()`.

The process of simulating a data set and learning weights that optimize the likelihood of the simulated data is repeated as per the number of specified simulations.

Value

A data frame with the following structure:

- rows: As many rows as the number of simulations
- columns: As many columns as the number of constraints

Why use this function?

This function gives us a way to estimate constraint weights via a Monte Carlo process. For example we might be interested in the effect of temperature on polarizing predicted probabilities, and the resulting constraint weights. This function can produce a distribution of constraint weights for the simulated polarized data, as well as a distribution of constraint weights for the simulated non-polarized ones, thereby allowing a comparison of the two.

Examples

```
# Get paths to toy data file
data_file <- system.file(
    "extdata", "sample_data_frame.csv", package = "maxent.ot"
)

tableaux_df <- read.csv(data_file)

# Fit weights to data with no biases
```

```
 fit_model <- optimize_weights(tableaux_df)

 # Predict probabilities for the same input with temperature = 2
 pred_obj <- predict_probabilities(
     tableaux_df, fit_model$weights, temperature = 2
 )

# Run 5 monte carlo simulations
# based on predicted probabilities when temperature = 2,
# and learn weights for these 5 simulated data sets
monte_carlo_weights(pred_obj$predictions, 5)

# Save learned weights to a file
tmp_output <- tempfile()
monte_carlo_weights(pred_obj$predictions, 5, output_path=tmp_output)
```

optimize_weights *Optimize MaxEnt OT constraint weights*

Description

Optimizes constraint weights given a data set and optional biases. If no bias arguments are provided, the bias term(s) will not be included in the optimization.

Usage

```
optimize_weights(
  input,
  bias_input = NA,
  mu = NA,
  sigma = NA,
  control_params = NA,
  upper_bound = DEFAULT_UPPER_BOUND,
  encoding = "unknown",
  model_name = NA,
  allow_negative_weights = FALSE
)
```

Arguments

`input` The input data frame/data table/tibble. This should contain one or more OT tableaux consisting of mappings between underlying and surface forms with observed frequency and violation profiles. Constraint violations must be numeric.

For an example of the data frame format, see inst/extdata/sample_data_frame.csv. You can read this file into a data frame using read.csv or into a tibble using dplyr::read_csv.

This function also supports the legacy OTSoft file format. You can use this format by passing in a file path string to the OTSoft file rather than a data frame.

For examples of OTSoft format, see inst/extdata/sample_data_file.txt.

`bias_input` (optional) A data frame/data table/tibble containing the bias mus and sigmas. Each row corresponds to an individual constraint, and consists of three columns: `Constraint`, which contains the constraint name, `Mu`, which contains the mu, and `Sigma`, which contains the sigma. If this argument is provided, the mu and sigma arguments will be ignored. Like the `input` argument, this function also supports the legacy OTSoft file format for this argument. In this case, `bias_input` should be a path to the bias parameters in OTSoft format.

For examples of OTSoft bias format, see inst/extdata/sample_bias_file_otsoft.txt. Each row in this file should be the name of the constraint, followed by the mu, followed by the sigma (separated by tabs).

`mu` (optional) A scalar or vector that will serve as the mu for each constraint in the bias term. Constraint weights will also be initialized to this value. If a vector, its length must equal the number of constraints in the input file. This value will not be used if `bias_file` is provided.

`sigma` (optional) A scalar or vector that will serve as the sigma for each constraint in the bias term. If a vector, its length must equal the number of constraints in the input file. This value will not be used if `bias_file` is provided.

`control_params` (optional) A named list of control parameters that will be passed to the optim function. See the documentation of that function for details. Note that some parameter settings may interfere with optimization. The parameter `fnscale` will be overwritten with `-1` if specified, since this must be treated as a maximization problem.

`upper_bound` (optional) The maximum value for constraint weights. Defaults to 100.

`encoding` (optional) The character encoding of the input file. Defaults to "unknown".

`model_name` (optional) A name for the model. If not provided, the name of the variable will be used if the input is a data frame. If the input is a path to an OTSoft file, the filename will be used.

`allow_negative_weights`
(optional) Whether the optimizer should allow negative weights. Defaults to FALSE.

Details

The objective function $J(w)$ that is optimized is defined as

$$J(w) = \sum_{i=1}^{n} \ln P(y_i|x_i;w) - \sum_{k=1}^{m} \frac{(w_k - \mu_k)^2}{2\sigma_k^2}$$

The first term in this equation calculates the natural logarithm of the conditional likelihood of the training data under the weights w. n is the number of data points (i.e., the sample size or the sum of the frequency column in the input),x_i is the input form of the ith data point, and y_i is the observed surface form corresponding to x_i.$P(y_i|x_i;w)$ represents the probability of realizing underlying x_i as surface y_i given weights w. This probability is defined as

$$P(y_i|x_i;w) = \frac{1}{Z_w(x_i)} \exp(-\sum_{k=1}^{m} w_k f_k(y_i, x_i))$$

where $f_k(y_i, x_i)$ is the number of violations of constraint k incurred by mapping underlying x_i to surface y_i. $Z_w(x_i)$ is a normalization term defined as

$$Z(x_i) = \sum_{y \in \mathcal{Y}(x_i)} \exp(-\sum_{k=1}^{m} w_k f_k(y, x_i))$$

where $\mathcal{Y}(x_i)$ is the set of observed surface realizations of input x_i.

The second term of the equation for calculating the objective function is the optional bias term, where w_k is the weight of constraint k, and μ_k and σ_k parameterize a normal distribution that serves as a prior for the value of w_k. μ_k specifies the mean of this distribution (the expected weight of constraint k before seeing any data) and $sigma_k$ reflects certainty in this value: lower values of σ_k penalize deviations from μ_k more severely, and thus require greater amounts of data to move w_k away from mu_k. While increasing σ_k will improve the fit to the training data, it may result in overfitting, particularly for small data sets.

A general bias with $\mu_k = 0$ for all k is commonly used as a form of simple regularization to prevent overfitting (see, e.g., Goldwater and Johnson 2003). Bias terms have also been used to model proposed phonological learning biases; see for example Wilson (2006), White (2013), and Mayer (2021, Ch. 4). The choice of σ depends on the sample size. As the number of data points increases, σ must decrease in order for the effect of the bias to remain constant: specifically, $n\sigma^2$ must be held constant, where n is the number of tokens.

Optimization is done using the optim function from the R-core statistics library. By default it uses `L-BFGS-B` optimization, which is a quasi-Newtonian method that allows upper and lower bounds on variables. Constraint weights are restricted to finite, non-negative values.

If no bias parameters are specified (either the `bias_file` argument or the mu and sigma parameters), optimization will be done without the bias term.

Value

An object with the following named attributes:

- `weights`: A named list of the optimal constraint weights
- `log_lik`: the log likelihood of the data under the discovered weights
- `k`: the number of constraints
- `n`: the number of data points in the training set

Examples

```
# Get paths to toy data and bias files.
df_file <- system.file(
    "extdata", "sample_data_frame.csv", package = "maxent.ot"
)
bias_file <- system.file(
     "extdata", "sample_bias_data_frame.csv", package = "maxent.ot"
)
# Fit weights to data with no biases
tableaux_df <- read.csv(df_file)
optimize_weights(tableaux_df)
```

```
# Fit weights with biases specified in file
bias_df <- read.csv(bias_file)
optimize_weights(tableaux_df, bias_df)

# Fit weights with biases specified in vector form
optimize_weights(
    tableaux_df, mu = c(1, 2), sigma = c(100, 200)
)

# Fit weights with biases specified as scalars
optimize_weights(tableaux_df, mu = 0, sigma = 1000)

# Fit weights with mix of scalar and vector biases
optimize_weights(tableaux_df, mu = c(1, 2), sigma = 1000)

# Pass additional arguments to optim function
optimize_weights(tableaux_df, control_params = list(maxit = 500))
```

otsoft_bias_to_df *Converts an OTSoft bias file to a data frame*

Description

Loads an OTSoft bias file and converts it to the data frame format used by the maxent.ot functions.

Usage

```
otsoft_bias_to_df(input, output_path = NA)
```

Arguments

input The path to the input bias file. This should contain more OT tableaux consisting of mappings between underlying and surface forms with observed frequency and violation profiles. Constraint violations must be numeric.

The file should be in OTSoft format. For examples of OTSoft format, see inst/extdata/sample_bias_file_otsoft.txt.

output_path (optional) A string specifying the path to a file to which the data frame will be saved in CSV format. If the file exists it will be overwritten. If this argument isn't provided, the output will not be written to a file.

Value

A data frame corresponding to the input OTSoft bias file, containing the columns

- `Constraint`: The constraint name.
- `Mu`: The mu value for the regularization term.
- `Sigma`: The sigma value for the regularization term.

Examples

```
# Convert OTSoft bias file to data frame format
otsoft_file <- system.file(
    "extdata", "sample_bias_file_otsoft.txt", package = "maxent.ot"
)
df_output <- otsoft_bias_to_df(otsoft_file)

# Save data frame to a file
tmp_output <- tempfile()
otsoft_bias_to_df(otsoft_file, tmp_output)
```

`otsoft_tableaux_to_df` *Converts an OTSoft tableaux file to a data frame*

Description

Loads an OTSoft tableaux file and converts it to the data frame format used by the maxent.ot functions.

Usage

```
otsoft_tableaux_to_df(input, output_path = NA, encoding = "unknown")
```

Arguments

`input`	The path to the input data file. This should contain more OT tableaux consisting of mappings between underlying and surface forms with observed frequency and violation profiles. Constraint violations must be numeric. The file should be in OTSoft format. For examples of OTSoft format, see inst/extdata/sample_data_file.txt.
`output_path`	(optional) A string specifying the path to a file to which the data frame will be saved in CSV format. If the file exists it will be overwritten. If this argument isn't provided, the output will not be written to a file.
`encoding`	(optional) The character encoding of the input file. Defaults to "unknown".

Value

A data frame corresponding to the input OTSoft tableau, containing the columns

- `Input`: The input form.
- `Output`: The output form.
- `Frequency`: The frequency of the input/output mapping.
- One column for each constraint containing its violation counts.

Examples

```
# Convert OTSoft file to data frame format
otsoft_file <- system.file(
    "extdata", "sample_data_file_otsoft.txt", package = "maxent.ot"
)
df_output <- otsoft_tableaux_to_df(otsoft_file)

# Save data frame to a file
tmp_output <- tempfile()
otsoft_tableaux_to_df(otsoft_file, tmp_output)
```

`predict_probabilities` *Predict probabilities of OT candidates*

Description

Predict probabilities of candidates based on their violation profiles and constraint weights.

Usage

```
predict_probabilities(
  test_input,
  constraint_weights,
  output_path = NA,
  out_sep = ",",
  encoding = "unknown",
  temperature = DEFAULT_TEMPERATURE
)
```

Arguments

`test_input` The input data frame/data table/tibble. This should contain one or more OT tableaux consisting of mappings between underlying and surface forms with observed frequency and violation profiles. Constraint violations must be numeric.

For an example of the data frame format, see inst/extdata/sample_data_frame.csv. You can read this file into a data frame using read.csv or into a tibble using dplyr::read_csv.

This function also supports the legacy OTSoft file format. You can use this format by passing in a file path string to the OTSoft file rather than a data frame.

For examples of OTSoft format, see inst/extdata/sample_data_file.txt.

`constraint_weights` A vector of constraint weights to use. These are typically generated by the `optimize_weights` function.

`output_path` (optional) A string specifying the path to a file to which the predictions will be saved. If the file exists it will be overwritten. If this argument isn't provided, the output will not be written to a file.

`out_sep` (optional) The delimiter used in the output files. Defaults to commas.

`encoding` (optional) The character encoding of the input file. Defaults to "unknown".

`temperature` (optional) The temperature parameter, which should be a real number $>= 1$. Defaults to 1.

Details

For each input/output pair in the provided file this function will calculate the probability of that output given the input form and the provided weights. This probability is defined as

$$P(y|x; w) = \frac{1}{Z_w(x)} \exp(-\sum_{k=1}^{m} w_k f_k(y, x))$$

where $f_k(y, x)$ is the number of violations of constraint k incurred by mapping underlying x to surface y, w_k is the weight associated with constraint k, and $Z_w(x)$ is a normalization term defined as

$$Z_w(x) = \sum_{y \in \mathcal{Y}(x)} \exp(-\sum_{k=1}^{m} w_k f_k(y, x))$$

where $\mathcal{Y}(x)$ is the set of all output candidates for input x.

The resulting probabilities will be appended to a data frame object representing the input tableaux. This data frame can also be saved to a file if the `output_path` argument is provided.

Value

An object with the following named attributes:

- `log_lik`: the log likelihood of the data under the provided weights
- `predictions`: A data table containing all the tableaux, with probabilities assigned to each candidate and errors.

Using temperature

If the temperature parameter T is specified, $P(y|x; w)$ is calculated as

$$\frac{1}{Z_w(x)} \exp(-\sum_{k=1}^{m} (w_k f_k(y, x))/T)$$

and $Z_w(x)$ is similarly calculated as

$$\sum_{y \in \mathcal{Y}(x)} \exp(-\sum_{k=1}^{m} (w_k f_k(y, x))/T)$$

Larger values of T move the predicted probabilities of output candidates for a particular input towards equality with one another. For example, if a particular input has two candidate outputs, higher values of T will move the probability of each towards `0.5`.

The temperature parameter can be used to generate less categorical predictions in a way that is independent of the constraint weights. See Ackley, Hinton, and Sejnowski (1985, p. 150-152) for more detail, and Hayes et al. (2009) and Mayer (2021, Ch. 4) for examples of temperature used in practice. By default this parameter is set to 1, which renders the equations in this section equivalent to the standard calculations of probability.

Examples

```
# Get paths to toy data file
df_file <- system.file(
    "extdata", "sample_data_frame.csv", package = "maxent.ot"
)
# Fit weights to dataframe with no biases
tableaux_df <- read.csv(df_file)
fit_model <- optimize_weights(tableaux_df)
predict_probabilities(tableaux_df, fit_model$weights)

# Do so with a temperature parameter
predict_probabilities(tableaux_df, fit_model$weights, temperature = 2)

# Save predictions to a file
tmp_output <- tempfile()
predict_probabilities(tableaux_df, fit_model$weights, output_path=tmp_output)
```

Index

compare_models, 2
cross_validate, 5

monte_carlo_weights, 8

optim, *6*, *9*, *11*, *12*
optimize_weights, 10, *15*
otsoft_bias_to_df, 13
otsoft_tableaux_to_df, 14

predict_probabilities, 15