choix Documentation

Release 0.3.5

Lucas Maystre

Jan 11, 2022

Contents

`choix` is a Python library that provides inference algorithms for models based on Luce's choice axiom. These probabilistic models can be used to explain and predict outcomes of comparisons between items.

- **Pairwise comparisons**: when the data consists of comparisons between two items, the model variant is usually referred to as the *Bradley-Terry* model. It is closely related to the Elo rating system used to rank chess players.
- **Partial rankings**: when the data consists of rankings over (a subset of) the items, the model variant is usually referred to as the *Plackett-Luce* model.
- **Top-1 lists**: another variation of the model arises when the data consists of discrete choices, i.e., we observe the selection of one item out of a subset of items.
- **Choices in a network**: when the data consists of counts of the number of visits to each node in a network, the model is known as the *Network Choice Model*.

`choix` makes it easy to infer model parameters from these different types of data, using a variety of algorithms:

- Luce Spectral Ranking
- Minorization-Maximization
- Rank Centrality
- Approximate Bayesian inference with expectation propagation

An easy way to get started is by exploring the notebooks!

CHAPTER 1

Contents

1.1 Installing

Most users will probably want to install the latest version hosted on PyPI:

```
pip install choix
```

Developers might want to install the latest version from GitHub:

```
git clone https://github.com/lucasmaystre/choix.git
cd choix
pip install -e .
```

The `-e` flag makes it possible to edit the code without needing to reinstall the library afterwards.

1.1.1 Dependencies

`choix` depends on `numpy`, `scipy`, and partially on `networkx` (only for network-related functions). Unit tests depend on `pytest`.

1.2 Types of Data

In order to simplify the code and speed up the implementation of algorithms, `choix` assumes that items are identified by consecutive integers ranging from `0` to `n_items - 1`.

Data processed by the inference algorithms in the library consist of outcomes of comparisons between subsets of items. Specifically, four types of observations are supported.

1.2.1 Pairwise comparisons

In the simplest (and perhaps the most widely-used) case, the data consist of outcomes of comparisons between *two* items. Mathematically, we represent the event "item i wins over item j" as

$$i \succ j.$$

In Python, we simply represent this event using a list with two integers:

```
[i, j]
```

By convention, the first element of the list represents the item which wins, and the second element the item which loses.

The statistical model that `choix` postulates for pairwise-comparison data is usually known as the *Bradley–Terry* model. Given parameters $\theta_1, \ldots, \theta_n$, and two items i and j, the probability of the outcome $i \succ j$ is

$$p(i \succ j) = \frac{e^{\theta_i}}{e^{\theta_i} + e^{\theta_j}}.$$

1.2.2 Top-1 lists

Another case arises when the data consist of choices of one item out of a set containing *several* other items. We call these *top-1 lists*. Compared to pairwise comparisons, this type of data is no longer restricted to comparing only two items: comparisons can involve sets of alternatives of any size between 2 and `n_items`. We denote the outcome "item i is chosen over items $j, \ldots, k$" as

$$i \succ \{j, \ldots, k\}.$$

In Python, we represent this event using a list with two elements:

```
[i, {j, ..., k}]
```

The first element of the list is an integer that represents the "winning" item, whereas the second element is a set containing the "losing" items. Note that this set does *not* include the winning item.

The statistical model that `choix` uses for these data is a straightforward extension of the Bradley–Terry model (see, e.g., Luce 1959). Given parameters $\theta_1, \ldots, \theta_n$, winning item i and losing alternatives $j, k, \ell, \ldots$, the probability of the corresponding outcome is

$$p(i \succ \{j, \ldots, k\}) = \frac{e^{\theta_i}}{e^{\theta_i} + e^{\theta_j} + \cdots + e^{\theta_k}}.$$

1.2.3 Rankings

Instead of observing a single choice, we might have observations that consist of a *ranking* over a set of alternatives. This leads to a third type of data. We denote the event "item i wins over item j … wins over item k" as

$$i \succ j \succ \ldots \succ k.$$

In Python, we represent this as a list:

```
[i, j, ..., k]
```

The list contains the subset of items in decreasing order of preference. For example, the list `[2, 0, 4]` corresponds to a ranking where `2` is first, `0` is second, and `4` is third.

In this case, the statistical model that `choix` uses is usually referred to as the *Plackett–Luce* model. Given parameters $\theta_1, \ldots, \theta_n$ and items $i, j, \ldots, k$, the probability of a given ranking is

$$p(i \succ j \succ \ldots \succ k) = \frac{e^{\theta_i}}{e^{\theta_i} + e^{\theta_j} + \cdots + e^{\theta_k}} \cdot \frac{e^{\theta_j}}{e^{\theta_j} + \cdots + e^{\theta_k}} \cdots .$$

The attentive reader will notice that this probability corresponds to that of an independent sequence of top-1 lists over the remaining alternatives.

1.2.4 Choices in a network

The fourth type of data is slightly more involved. It enables the processing of choices on networks based on marginal observations at the nodes of the network. The easiest way to get started is to follow this notebook.

We defer to [MG17] for a thorough presentation of the observed data and of the statistical model.

1.3 Notes on Regularization

In some cases, e.g., if the data is sparse, the iterative algorithms underlying the parameter inference functions might not converge. A pragmatic solution to this problem is to add a little bit of regularization.

Inference functions in `choix` provide a generic regularization argument: `alpha`. When $\alpha = 0$, regularization is turned off; setting $\alpha > 0$ turns it on. In practice, if regularization is needed, we recommend starting with small values (e.g., 10^{-4}) and increasing the value if necessary.

Below, we briefly how the regularization parameter is used inside the various parameter inference functions.

1.3.1 Markov-chain based algorithms

For Markov-chain based algorithms such Luce Spectral Ranking and Rank Centrality, α is used to initialize the transition rates of the Markov chain.

In the special case of pairwise-comparison data, this can be loosely understood as placing an independent Beta prior for each pair of items on the respective comparison outcome probability.

1.3.2 Minorization-maximization algorithms

In the case of Minorization-maximization algorithms, the exponentiated model parameters $e^{\theta_1}, \ldots, e^{\theta_n}$ are endowed each with an independent Gamma prior distribution, with scale $\alpha + 1$. See Caron & Doucet (2012) for details.

1.3.3 Other algorithms

The scipy-based optimization functions use an ℓ_2-regularizer on the parameters $\theta_1, \ldots, \theta_n$. In other words, the parameters are endowed each with an independent Gaussian prior with variance $1/\alpha$.

1.4 API Reference

Functions that *generate parameters and data*.

choix.generate_params	Generate random model parameters.
choix.generate_pairwise	Generate pairwise comparisons from a Bradley–Terry model.
choix.generate_rankings	Generate rankings according to a Plackett–Luce model.
choix.compare	Generate a comparison outcome that follows Luce's axiom.

Various utilities such as distance functions, etc.

choix.probabilities	Compute the comparison outcome probabilities given a subset of items.
choix.footrule_dist	Compute Spearman's footrule distance between two models.
choix.kendalltau_dist	Compute the Kendall tau distance between two models.

Functions that *process pairwise comparisons*.

choix.lsr_pairwise	Compute the LSR estimate of model parameters.
choix.ilsr_pairwise	Compute the ML estimate of model parameters using I-LSR.
choix.lsr_pairwise_dense	Compute the LSR estimate of model parameters given dense data.
choix.ilsr_pairwise_dense	Compute the ML estimate of model parameters given dense data.
choix.rank_centrality	Compute the Rank Centrality estimate of model parameters.
choix.opt_pairwise	Compute the ML estimate of model parameters using `scipy.optimize`.
choix.ep_pairwise	Compute a distribution of model parameters using the EP algorithm.
choix.mm_pairwise	Compute the ML estimate of model parameters using the MM algorithm.
choix.log_likelihood_pairwise	Compute the log-likelihood of model parameters.

Functions that *process rankings*.

choix.lsr_rankings	Compute the LSR estimate of model parameters.
choix.ilsr_rankings	Compute the ML estimate of model parameters using I-LSR.
choix.opt_rankings	Compute the ML estimate of model parameters using `scipy.optimize`.
choix.mm_rankings	Compute the ML estimate of model parameters using the MM algorithm.
choix.log_likelihood_rankings	Compute the log-likelihood of model parameters.

Functions that *process top-1 lists*.

choix.lsr_top1	Compute the LSR estimate of model parameters.
choix.ilsr_top1	Compute the ML estimate of model parameters using I-LSR.
choix.opt_top1	Compute the ML estimate of model parameters using `scipy.optimize`.
choix.mm_top1	Compute the ML estimate of model parameters using the MM algorithm.
choix.log_likelihood_top1	Compute the log-likelihood of model parameters.

Functions that *process choices in a network*.

choix.choicerank	Compute the MAP estimate of a network choice model's parameters.
choix.log_likelihood_network	Compute the log-likelihood of model parameters.

1.4.1 Generators

`choix.`**`generate_params`**(*n_items*, *interval=5.0*, *ordered=False*)

Generate random model parameters.

This function samples a parameter independently and uniformly for each item. `interval` defines the width of the uniform distribution.

Parameters

- **`n_items`** (*`int`*) – Number of distinct items.
- **`interval`** (*`float`*) – Sampling interval.
- **`ordered`** (*`bool, optional`*) – If true, the parameters are ordered from lowest to highest.

Returns **params** – Model parameters.

Return type numpy.ndarray

`choix.`**`generate_pairwise`**(*params*, *n_comparisons=10*)

Generate pairwise comparisons from a Bradley–Terry model.

This function samples comparisons pairs independently and uniformly at random over the `len(params)` choose 2 possibilities, and samples the corresponding comparison outcomes from a Bradley–Terry model parametrized by `params`.

Parameters

- **`params`** (*`array_like`*) – Model parameters.
- **`n_comparisons`** (*`int`*) – Number of comparisons to be returned.

Returns **data** – Pairwise-comparison samples (see *Pairwise comparisons*).

Return type list of (int, int)

`choix.`**`generate_rankings`**(*params*, *n_rankings*, *size=3*)

Generate rankings according to a Plackett–Luce model.

This function samples subsets of items (of size `size`) independently and uniformly at random, and samples the correspoding partial ranking from a Plackett–Luce model parametrized by `params`.

Parameters

- **params** (*array_like*) – Model parameters.
- **n_rankings** (*int*) – Number of rankings to generate.
- **size** (*int, optional*) – Number of items to include in each ranking.

Returns **data** – A list of (partial) rankings generated according to a Plackett–Luce model with the specified model parameters.

Return type list of numpy.ndarray

choix.**compare**(*items*, *params*, *rank=False*)

Generate a comparison outcome that follows Luce's axiom.

This function samples an outcome for the comparison of a subset of items, from a model parametrized by `params`. If `rank` is True, it returns a ranking over the items, otherwise it returns a single item.

Parameters

- **items** (*list*) – Subset of items to compare.
- **params** (*array_like*) – Model parameters.
- **rank** (*bool, optional*) – If true, returns a ranking over the items instead of a single item.

Returns **outcome** – The chosen item, or a ranking over `items`.

Return type int or list of int

1.4.2 Utilities

choix.**probabilities**(*items*, *params*)

Compute the comparison outcome probabilities given a subset of items.

This function computes, for each item in `items`, the probability that it would win (i.e., be chosen) in a comparison involving the items, given model parameters.

Parameters

- **items** (*list*) – Subset of items to compare.
- **params** (*array_like*) – Model parameters.

Returns **probs** – A probability distribution over `items`.

Return type numpy.ndarray

choix.**footrule_dist**(*params1*, *params2=None*)

Compute Spearman's footrule distance between two models.

This function computes Spearman's footrule distance between the rankings induced by two parameter vectors. Let σ_i be the rank of item `i` in the model described by `params1`, and τ_i be its rank in the model described by `params2`. Spearman's footrule distance is defined by

$$\sum_{i=1}^{N} |\sigma_i - \tau_i|$$

By convention, items with the lowest parameters are ranked first (i.e., sorted using the natural order).

If the argument `params2` is `None`, the second model is assumed to rank the items by their index: item `0` has rank 1, item `1` has rank 2, etc.

Parameters

- **params1** (*array_like*) – Parameters of the first model.
- **params2** (*array_like, optional*) – Parameters of the second model.

Returns **dist** – Spearman's footrule distance.

Return type float

choix.**kendalltau_dist**(*params1*, *params2=None*)

Compute the Kendall tau distance between two models.

This function computes the Kendall tau distance between the rankings induced by two parameter vectors. Let σ_i be the rank of item `i` in the model described by `params1`, and τ_i be its rank in the model described by `params2`. The Kendall tau distance is defined as the number of pairwise disagreements between the two rankings, i.e.,

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \mathbf{1}\{\sigma_i > \sigma_j \wedge \tau_i < \tau_j\}$$

By convention, items with the lowest parameters are ranked first (i.e., sorted using the natural order).

If the argument `params2` is `None`, the second model is assumed to rank the items by their index: item `0` has rank 1, item `1` has rank 2, etc.

If some values are equal within a parameter vector, all items are given a distinct rank, corresponding to the order in which the values occur.

Parameters

- **params1** (*array_like*) – Parameters of the first model.
- **params2** (*array_like, optional*) – Parameters of the second model.

Returns **dist** – Kendall tau distance.

Return type float

1.4.3 Processing pairwise comparisons

choix.**lsr_pairwise**(*n_items*, *data*, *alpha=0.0*, *initial_params=None*)

Compute the LSR estimate of model parameters.

This function implements the Luce Spectral Ranking inference algorithm [MG15] for pairwise-comparison data (see *Pairwise comparisons*).

The argument `initial_params` can be used to iteratively refine an existing parameter estimate (see the implementation of *ilsr_pairwise()* for an idea on how this works). If it is set to *None* (the default), the all-ones vector is used.

The transition rates of the LSR Markov chain are initialized with `alpha`. When `alpha > 0`, this corresponds to a form of regularization (see *Notes on Regularization* for details).

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Pairwise-comparison data.
- **alpha** (*float, optional*) – Regularization parameter.
- **initial_params** (*array_like, optional*) – Parameters used to build the transition rates of the LSR Markov chain.

Returns **params** – An estimate of model parameters.

Return type numpy.ndarray

choix.**ilsr_pairwise**(*n_items*, *data*, *alpha=0.0*, *initial_params=None*, *max_iter=100*, *tol=1e-08*)

Compute the ML estimate of model parameters using I-LSR.

This function computes the maximum-likelihood (ML) estimate of model parameters given pairwise-comparison data (see *Pairwise comparisons*), using the iterative Luce Spectral Ranking algorithm [MG15].

The transition rates of the LSR Markov chain are initialized with `alpha`. When `alpha > 0`, this corresponds to a form of regularization (see *Notes on Regularization* for details).

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Pairwise-comparison data.
- **alpha** (*float, optional*) – Regularization parameter.
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Maximum L1-norm of the difference between successive iterates to declare convergence.

Returns **params** – The ML estimate of model parameters.

Return type numpy.ndarray

choix.**lsr_pairwise_dense**(*comp_mat*, *alpha=0.0*, *initial_params=None*)

Compute the LSR estimate of model parameters given dense data.

This function implements the Luce Spectral Ranking inference algorithm [MG15] for dense pairwise-comparison data.

The data is described by a pairwise-comparison matrix `comp_mat` such that `comp_mat[i,j]` contains the number of times that item `i` wins against item `j`.

In comparison to *lsr_pairwise()*, this function is particularly efficient for dense pairwise-comparison datasets (i.e., containing many comparisons for a large fraction of item pairs).

The argument `initial_params` can be used to iteratively refine an existing parameter estimate (see the implementation of *ilsr_pairwise()* for an idea on how this works). If it is set to *None* (the default), the all-ones vector is used.

The transition rates of the LSR Markov chain are initialized with `alpha`. When `alpha > 0`, this corresponds to a form of regularization (see *Notes on Regularization* for details).

Parameters

- **comp_mat** (*np.array*) – 2D square matrix describing the pairwise-comparison outcomes.
- **alpha** (*float, optional*) – Regularization parameter.
- **initial_params** (*array_like, optional*) – Parameters used to build the transition rates of the LSR Markov chain.

Returns **params** – An estimate of model parameters.

Return type np.array

choix.**ilsr_pairwise_dense**(*comp_mat*, *alpha=0.0*, *initial_params=None*, *max_iter=100*, *tol=1e-08*)

Compute the ML estimate of model parameters given dense data.

This function computes the maximum-likelihood (ML) estimate of model parameters given dense pairwise-comparison data.

The data is described by a pairwise-comparison matrix `comp_mat` such that `comp_mat[i, j]` contains the number of times that item `i` wins against item `j`.

In comparison to *ilsr_pairwise()*, this function is particularly efficient for dense pairwise-comparison datasets (i.e., containing many comparisons for a large fraction of item pairs).

The transition rates of the LSR Markov chain are initialized with `alpha`. When `alpha > 0`, this corresponds to a form of regularization (see *Notes on Regularization* for details).

Parameters

- **comp_mat** (*np.array*) – 2D square matrix describing the pairwise-comparison outcomes.
- **alpha** (*float, optional*) – Regularization parameter.
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Maximum L1-norm of the difference between successive iterates to declare convergence.

Returns **params** – The ML estimate of model parameters.

Return type numpy.ndarray

choix.**rank_centrality**(*n_items*, *data*, *alpha=0.0*)

Compute the Rank Centrality estimate of model parameters.

This function implements Negahban et al.'s Rank Centrality algorithm [NOS12]. The algorithm is similar to *ilsr_pairwise()*, but considers the *ratio* of wins for each pair (instead of the total count).

The transition rates of the Rank Centrality Markov chain are initialized with `alpha`. When `alpha > 0`, this corresponds to a form of regularization (see *Notes on Regularization* for details).

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Pairwise-comparison data.
- **alpha** (*float, optional*) – Regularization parameter.

Returns **params** – An estimate of model parameters.

Return type numpy.ndarray

choix.**opt_pairwise**(*n_items*, *data*, *alpha=1e-06*, *method='Newton-CG'*, *initial_params=None*, *max_iter=None*, *tol=1e-05*)

Compute the ML estimate of model parameters using `scipy.optimize`.

This function computes the maximum-likelihood estimate of model parameters given pairwise-comparison data (see *Pairwise comparisons*), using optimizers provided by the `scipy.optimize` module.

If `alpha > 0`, the function returns the maximum a-posteriori (MAP) estimate under an isotropic Gaussian prior with variance `1 / alpha`. See *Notes on Regularization* for details.

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Pairwise-comparison data.
- **alpha** (*float, optional*) – Regularization strength.
- **method** (*str, optional*) – Optimization method. Either "BFGS" or "Newton-CG".
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Tolerance for termination (method-specific).

Returns **params** – The (penalized) ML estimate of model parameters.

Return type numpy.ndarray

Raises `ValueError` – If the method is not "BFGS" or "Newton-CG".

`choix.`**ep_pairwise**(*n_items*, *data*, *alpha*, *model='logit'*, *max_iter=100*, *initial_state=None*)

Compute a distribution of model parameters using the EP algorithm.

This function computes an approximate Bayesian posterior probability distribution over model parameters, given pairwise-comparison data (see *Pairwise comparisons*). It uses the expectation propagation algorithm, as presented, e.g., in [CG05].

The prior distribution is assumed to be isotropic Gaussian with variance `1 / alpha`. The posterior is approximated by a a general multivariate Gaussian distribution, described by a mean vector and a covariance matrix.

Two different observation models are available. `logit` (default) assumes that pairwise-comparison outcomes follow from a Bradley-Terry model. `probit` assumes that the outcomes follow from Thurstone's model.

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Pairwise-comparison data.
- **alpha** (*float*) – Inverse variance of the (isotropic) prior.
- **model** (*str, optional*) – Observation model. Either "logit" or "probit".
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **initial_state** (*tuple of array_like, optional*) – Natural parameters used to initialize the EP algorithm.

Returns

- **mean** (*numpy.ndarray*) – The mean vector of the approximate Gaussian posterior.
- **cov** (*numpy.ndarray*) – The covariance matrix of the approximate Gaussian posterior.

Raises `ValueError` – If the observation model is not "logit" or "probit".

`choix.`**mm_pairwise**(*n_items*, *data*, *initial_params=None*, *alpha=0.0*, *max_iter=10000*, *tol=1e-08*)

Compute the ML estimate of model parameters using the MM algorithm.

This function computes the maximum-likelihood (ML) estimate of model parameters given pairwise-comparison data (see *Pairwise comparisons*), using the minorization-maximization (MM) algorithm [Hun04], [CD12].

If `alpha > 0`, the function returns the maximum a-posteriori (MAP) estimate under a (peaked) Dirichlet prior. See *Notes on Regularization* for details.

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Pairwise-comparison data.
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **alpha** (*float, optional*) – Regularization parameter.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Maximum L1-norm of the difference between successive iterates to declare convergence.

Returns **params** – The ML estimate of model parameters.

Return type numpy.ndarray

choix.**log_likelihood_pairwise**(*data*, *params*)
Compute the log-likelihood of model parameters.

1.4.4 Processing rankings

choix.**lsr_rankings**(*n_items*, *data*, *alpha=0.0*, *initial_params=None*)
Compute the LSR estimate of model parameters.

This function implements the Luce Spectral Ranking inference algorithm [MG15] for ranking data (see *Rankings*).

The argument `initial_params` can be used to iteratively refine an existing parameter estimate (see the implementation of `ilsr_rankings()` for an idea on how this works). If it is set to *None* (the default), the all-ones vector is used.

The transition rates of the LSR Markov chain are initialized with `alpha`. When `alpha > 0`, this corresponds to a form of regularization (see *Notes on Regularization* for details).

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Ranking data.
- **alpha** (*float, optional*) – Regularization parameter.
- **initial_params** (*array_like, optional*) – Parameters used to build the transition rates of the LSR Markov chain.

Returns **params** – An estimate of model parameters.

Return type numpy.ndarray

choix.**ilsr_rankings**(*n_items*, *data*, *alpha=0.0*, *initial_params=None*, *max_iter=100*, *tol=1e-08*)
Compute the ML estimate of model parameters using I-LSR.

This function computes the maximum-likelihood (ML) estimate of model parameters given ranking data (see *Rankings*), using the iterative Luce Spectral Ranking algorithm [MG15].

The transition rates of the LSR Markov chain are initialized with `alpha`. When `alpha > 0`, this corresponds to a form of regularization (see *Notes on Regularization* for details).

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Ranking data.

- **alpha** (*float, optional*) – Regularization parameter.
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Maximum L1-norm of the difference between successive iterates to declare convergence.

Returns **params** – The ML estimate of model parameters.

Return type numpy.ndarray

choix.**opt_rankings**(*n_items*, *data*, *alpha=1e-06*, *method='Newton-CG'*, *initial_params=None*, *max_iter=None*, *tol=1e-05*)

Compute the ML estimate of model parameters using `scipy.optimize`.

This function computes the maximum-likelihood estimate of model parameters given ranking data (see *Rankings*), using optimizers provided by the `scipy.optimize` module.

If `alpha > 0`, the function returns the maximum a-posteriori (MAP) estimate under an isotropic Gaussian prior with variance `1 / alpha`. See *Notes on Regularization* for details.

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Ranking data.
- **alpha** (*float, optional*) – Regularization strength.
- **method** (*str, optional*) – Optimization method. Either "BFGS" or "Newton-CG".
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Tolerance for termination (method-specific).

Returns **params** – The (penalized) ML estimate of model parameters.

Return type numpy.ndarray

Raises `ValueError` – If the method is not "BFGS" or "Newton-CG".

choix.**mm_rankings**(*n_items*, *data*, *initial_params=None*, *alpha=0.0*, *max_iter=10000*, *tol=1e-08*)

Compute the ML estimate of model parameters using the MM algorithm.

This function computes the maximum-likelihood (ML) estimate of model parameters given ranking data (see *Rankings*), using the minorization-maximization (MM) algorithm [Hun04], [CD12].

If `alpha > 0`, the function returns the maximum a-posteriori (MAP) estimate under a (peaked) Dirichlet prior. See *Notes on Regularization* for details.

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Ranking data.
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **alpha** (*float, optional*) – Regularization parameter.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.

- **tol** (*float, optional*) – Maximum L1-norm of the difference between successive iterates to declare convergence.

Returns **params** – The ML estimate of model parameters.

Return type numpy.ndarray

choix.**log_likelihood_rankings**(*data*, *params*)
Compute the log-likelihood of model parameters.

1.4.5 Processing top-1 lists

choix.**lsr_top1**(*n_items*, *data*, *alpha=0.0*, *initial_params=None*)
Compute the LSR estimate of model parameters.

This function implements the Luce Spectral Ranking inference algorithm [MG15] for top-1 data (see *Top-1 lists*).

The argument `initial_params` can be used to iteratively refine an existing parameter estimate (see the implementation of `ilsr_top1()` for an idea on how this works). If it is set to *None* (the default), the all-ones vector is used.

The transition rates of the LSR Markov chain are initialized with `alpha`. When `alpha > 0`, this corresponds to a form of regularization (see *Notes on Regularization* for details).

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Top-1 data.
- **alpha** (*float*) – Regularization parameter.
- **initial_params** (*array_like*) – Parameters used to build the transition rates of the LSR Markov chain.

Returns **params** – An estimate of model parameters.

Return type numpy.ndarray

choix.**ilsr_top1**(*n_items*, *data*, *alpha=0.0*, *initial_params=None*, *max_iter=100*, *tol=1e-08*)
Compute the ML estimate of model parameters using I-LSR.

This function computes the maximum-likelihood (ML) estimate of model parameters given top-1 data (see *Top-1 lists*), using the iterative Luce Spectral Ranking algorithm [MG15].

The transition rates of the LSR Markov chain are initialized with `alpha`. When `alpha > 0`, this corresponds to a form of regularization (see *Notes on Regularization* for details).

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Top-1 data.
- **alpha** (*float, optional*) – Regularization parameter.
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Maximum L1-norm of the difference between successive iterates to declare convergence.

Returns **params** – The ML estimate of model parameters.

Return type numpy.ndarray

choix.**opt_top1**(*n_items*, *data*, *alpha=1e-06*, *method='Newton-CG'*, *initial_params=None*, *max_iter=None*, *tol=1e-05*)

Compute the ML estimate of model parameters using `scipy.optimize`.

This function computes the maximum-likelihood estimate of model parameters given top-1 data (see *Top-1 lists*), using optimizers provided by the `scipy.optimize` module.

If `alpha > 0`, the function returns the maximum a-posteriori (MAP) estimate under an isotropic Gaussian prior with variance `1 / alpha`. See *Notes on Regularization* for details.

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Top-1 data.
- **alpha** (*float, optional*) – Regularization strength.
- **method** (*str, optional*) – Optimization method. Either "BFGS" or "Newton-CG".
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Tolerance for termination (method-specific).

Returns **params** – The (penalized) ML estimate of model parameters.

Return type numpy.ndarray

Raises `ValueError` – If the method is not "BFGS" or "Newton-CG".

choix.**mm_top1**(*n_items*, *data*, *initial_params=None*, *alpha=0.0*, *max_iter=10000*, *tol=1e-08*)

Compute the ML estimate of model parameters using the MM algorithm.

This function computes the maximum-likelihood (ML) estimate of model parameters given top-1 data (see *Top-1 lists*), using the minorization-maximization (MM) algorithm [Hun04], [CD12].

If `alpha > 0`, the function returns the maximum a-posteriori (MAP) estimate under a (peaked) Dirichlet prior. See *Notes on Regularization* for details.

Parameters

- **n_items** (*int*) – Number of distinct items.
- **data** (*list of lists*) – Top-1 data.
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **alpha** (*float, optional*) – Regularization parameter.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Maximum L1-norm of the difference between successive iterates to declare convergence.

Returns **params** – The ML estimate of model parameters.

Return type numpy.ndarray

choix.**log_likelihood_top1**(*data*, *params*)

Compute the log-likelihood of model parameters.

1.4.6 Processing choices in a network

choix.**choicerank**(*digraph*, *traffic_in*, *traffic_out*, *weight=None*, *initial_params=None*, *alpha=1.0*, *max_iter=10000*, *tol=1e-08*)

Compute the MAP estimate of a network choice model's parameters.

This function computes the maximum-a-posteriori (MAP) estimate of model parameters given a network structure and node-level traffic data (see *Choices in a network*), using the ChoiceRank algorithm [MG17], [KTVV15].

The nodes are assumed to be labeled using consecutive integers starting from 0.

Parameters

- **digraph** (*networkx.DiGraph*) – Directed graph representing the network.
- **traffic_in** (*array_like*) – Number of arrivals at each node.
- **traffic_out** (*array_like*) – Number of departures at each node.
- **weight** (*str, optional*) – The edge attribute that holds the numerical value used for the edge weight. If None (default) then all edge weights are 1.
- **initial_params** (*array_like, optional*) – Parameters used to initialize the iterative procedure.
- **alpha** (*float, optional*) – Regularization parameter.
- **max_iter** (*int, optional*) – Maximum number of iterations allowed.
- **tol** (*float, optional*) – Maximum L1-norm of the difference between successive iterates to declare convergence.

Returns **params** – The MAP estimate of model parameters.

Return type numpy.ndarray

Raises ImportError – If the NetworkX library cannot be imported.

choix.**log_likelihood_network**(*digraph*, *traffic_in*, *traffic_out*, *params*, *weight=None*)

Compute the log-likelihood of model parameters.

If weight is not None, the log-likelihood is correct only up to a constant (independent of the parameters).

1.5 References

A list of references.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Bibliography

[ACPX13] H. Azari Soufiani, W. Z. Chen, D. C. Parkes, and L. Xia, Generalized Method-of-Moments for Rank Aggregation, NIPS 2013. PDF

[CD12] F. Caron and A. Doucet, "Efficient Bayesian Inference for Generalized Bradley-Terry models". Journal of Computational and Graphical Statistics, 21(1):174-196, 2012. PDF

[CG05] W. Chu and Z. Ghahramani, "Extensions of Gaussian processes for ranking: semi-supervised and active learning", NIPS 2005 Workshop on Learning to Rank. PDF

[Hun04] D. R. Hunter, "MM algorithms for generalized Bradley-Terry models", The Annals of Statistics 32(1):384-406, 2004. PDF

[KTVV15] R. Kumar, A. Tomkins, S. Vassilvitskii and E. Vee, "Inverting a Steady-State", WSDM 2015. PDF

[MG15] L. Maystre and M. Grossglauser, "Fast and Accurate Inference of Plackett-Luce Models", NIPS 2015. PDF

[MG17] L. Maystre and M. Grossglauser, "ChoiceRank: Identifying Preferences from Node Traffic in Networks", ICML 2017. PDF

[NOS12] S. Negahban, S. Oh, and D. Shah, "Iterative Ranking from Pair-wise Comparison", NIPS 2012. PDF

Index