pypsy Documentation

Release 0.0.1

chris dai

Sep 27, 2018

Contents:

CHAPTER 1

pypsy

psychometrics package, including structural equation model, confirmatory factor analysis, unidimensional item response theory, multidimensional item response theory, cognitive diagnosis model, factor analysis and adaptive testing. The package is still a doll. will be finished in future.

1.1 unidimensional item response theory

1.1.1 models

- binary response data IRT (two parameters, three parameters).
- grade respone data IRT (GRM model)

1.2 Parameter estimation algorithm

- EM algorithm (2PL, GRM)
- MCMC algorithm (3PL

1.3 Multidimensional item response theory (full information item factor analysis)

1.3.1 Parameter estimation algorithm

The initial value

The approximate polychoric correlation is calculated, and the slope initial value is obtained by factor analysis of the polychoric correlation matrix.

EM algorithm

- E step uses GH integral.
- M step uses Newton algorithm (sparse matrix is divided into non sparse matrix).

Factor rotation

Gradient projection algorithm

1.3.2 The shortcomings

GH integrals can only estimate low dimensional parameters.

1.4 Cognitive diagnosis model

1.4.1 models

- Dina
- ho-dina

1.4.2 parameter estimation algorithms

- EM algorithm
- MCMC algorithm
- maximum likelihood estimation (only for estimating skill parameters of subjects)

1.5 Structural equation model

- contains three parameter estimation methods(ULS, ML and GLS).
- based on gradient descent

1.6 Confirmatory factor analysis

- can be used for continuous data, binary data and ordered data.
- based on gradient descent
- binary and ordered data based on Polychoric correlation matrix.

1.7 Factor analysis

For the time being, only for the calculation of full information item factor analysis, it is very simple.

1.7.1 The algorithm

principal component analysis

1.7.2 The rotation algorithm

gradient projection

1.8 Adaptive test

1.8.1 model

Thurston IRT model (multidimensional item response theory model for personality test)

1.8.2 Algorithm

Maximum information method for multidimensional item response theory

---

1.9 Require

- numpy
- progressbar2

1.10 How to use it

1.10.1 install

```
pip install psy
```

See demo

1.11 TODO LIST

- theta parameterization of CCFA
- parameter estimation of structural equation models for multivariate data
- Bayesin knowledge tracing (Bayesian knowledge tracking)
- multidimensional item response theory (full information item factor analysis)
- high dimensional computing algorithm (adaptive integral, etc.)
- various item response models
- cognitive diagnosis model
- G-DINA model
- Q matrix correlation algorithm
- Factor analysis
- maximum likelihood estimation
- various factor rotation algorithms
- adaptive
- adaptive cognitive diagnosis
- other adaption model
- standard error and P value
- code annotation, testing and documentation.

1.12 Reference

- DINA Model and Parameter Estimation: A Didactic
- Higher-order latent trait models for cognitive diagnosis
- Full-Information Item Factor Analysis.
- Multidimensional adaptive testing
- Derivative free gradient projection algorithms for rotation

CHAPTER 2

Installation

2.1 Stable release

To install pypsy, run this command in your terminal:

```
$ pip install pypsy
```

This is the preferred method to install pypsy, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for pypsy can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/inuyasha2012/pypsy
```

Or download the tarball:

```
$ curl  -OL https://github.com/inuyasha2012/pypsy/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use pypsy in a project:

```
import pypsy
```

CHAPTER 4

API

4.1 psy package

4.1.1 Subpackages

psy.cat package

Submodules

psy.cat.tirt module

class psy.cat.tirt.**BaseModel**(*slop*, *threshold*, *init_theta=None*, *score=None*, *iter_method=u'newton'*, *sigma=None*)

Bases: object

gradient_ascent

newton

:return: ndarray(int|float),

prob(*theta*)

score

solve

z(*theta*)

class psy.cat.tirt.**BaseProbitModel**(*slop*, *threshold*, *init_theta=None*, *score=None*, *iter_method=u'newton'*, *sigma=None*)

Bases: *psy.cat.tirt.BaseModel*

info(*theta*)

class psy.cat.tirt.**BaseSimTirt**(*subject_nums*, *trait_size*, *model=u'bayes_probit'*, *sigma=None*, *iter_method=u'newton'*, *block_size=3*, *lower=1*, *upper=4*, *avg=0*, *std=1*)

Bases: object

MODEL = {u'bayes_probit': <class 'psy.cat.tirt.BayesProbitModel'>}

random_thetas

:return: ndarray

class psy.cat.tirt.**BayesProbitModel**(*slop*, *threshold*, *init_theta=None*, *score=None*, *iter_method=u'newton'*, *sigma=None*)

Bases: *psy.cat.tirt.BaseProbitModel*

info(*theta*)

class psy.cat.tirt.**SimAdaptiveTirt**(*item_size*, *max_sec_item_size=10*, **args*, ***kwargs*)

Bases: *psy.cat.tirt.BaseSimTirt*

item_bank

scores

sim()

thetas

Module contents

psy.cdm package

Submodules

psy.cdm.irm module

class psy.cdm.irm.**BaseEmDina**(*attrs*, *score=None*)

Bases: *psy.cdm.irm.Dina*

class psy.cdm.irm.**BaseMcmcDina**(*thin=1*, *burn=3000*, *max_iter=10000*, **args*, ***kwargs*)

Bases: *psy.cdm.irm.Dina*

class psy.cdm.irm.**Dina**(*attrs*, *score=None*)

Bases: object

get_p(*yita*, *no_slip*, *guess*)

get_yita(*skills*)

item_size

class psy.cdm.irm.**EmDina**(*guess_init=None*, *no_slip_init=None*, *max_iter=100*, *tol=1e-05*, **args*, ***kwargs*)

Bases: *psy.cdm.irm.BaseEmDina*

em()

class psy.cdm.irm.**McmcDina**(*thin=1*, *burn=3000*, *max_iter=10000*, **args*, ***kwargs*)

Bases: *psy.cdm.irm.BaseMcmcDina*

mcmc()

class psy.cdm.irm.**McmcHoDina**(*thin=1*, *burn=3000*, *max_iter=10000*, **args*, ***kwargs*)
Bases: *psy.cdm.irm.BaseMcmcDina*

static get_skills_p(*lam0*, *lam1*, *theta*)

mcmc()

class psy.cdm.irm.**MlDina**(*guess*, *no_slip*, **args*, ***kwargs*)
Bases: *psy.cdm.irm.BaseEmDina*

solve()

Module contents

psy.ctt package

Submodules

psy.ctt.ctt module

class psy.ctt.ctt.**BaseCtt**(*scores*)
Bases: object

get_alpha_reliability()

get_composite_reliability()

class psy.ctt.ctt.**BivariateCtt**(*scores*)
Bases: *psy.ctt.ctt.BaseCtt*

get_difficulty()

get_discrimination()

Module contents

psy.data package

Submodules

psy.data.data module

Module contents

psy.exceptions package

Submodules

psy.exceptions.cat module

exception psy.exceptions.cat.**ItemParamError**
Bases: exceptions.Exception

Item Param Type Error

exception psy.exceptions.cat.**IterMethodError**
Bases: exceptions.Exception

iter method error

exception psy.exceptions.cat.**ScoreError**
Bases: exceptions.Exception

score error

exception psy.exceptions.cat.**ThetaError**
Bases: exceptions.Exception

theta error

exception psy.exceptions.cat.**UnknownModelError**
Bases: exceptions.Exception

unknown model

Module contents

exception psy.exceptions.**ConvergenceError**
Bases: exceptions.Exception

no convergence

psy.fa package

Submodules

psy.fa.factors module

class psy.fa.factors.**Factor**(*scores*, *factors_num*)
Bases: object

cor

loadings

mirt_loading

polycor

psy.fa.rotations module

class psy.fa.rotations.**GPForth**(*init_loadings*, *method='varimax'*)
Bases: object

solve()

static varimax(*l*)

Module contents

psy.irt package

Submodules

psy.irt.base module

class psy.irt.base.**GuessLogitMixin**
Bases: *psy.irt.base.LogitMixin*

p(*guess*, **args*, ***kwargs*)

class psy.irt.base.**GuessProbitMixin**
Bases: *psy.irt.base.ProbitMixin*

p(*guess*, **args*, ***kwargs*)

class psy.irt.base.**LogitMixin**
Bases: object

p(*z*)

class psy.irt.base.**ProbitMixin**
Bases: object

p(*z*)

class psy.irt.base.**RaschZMixin**
Bases: object

z(*threshold*, *theta*)

class psy.irt.base.**ZMixin**
Bases: object

z(*slop*, *threshold*, *theta*)

Parameters

- **slop** –
- **threshold** –
- **theta** –

Returns

psy.irt.grm module

class psy.irt.grm.**Grm**(*scores=None*, *init_slop=None*, *init_threshold=None*, *max_iter=1000*, *tol=1e-05*, *gp_size=11*)
Bases: object

em()

static get_gh_point(*gp_size*)

static p(*z*)

static z(*slop*, *thresholds*, *theta*)

psy.irt.irm module

class psy.irt.irm.**Irt**(*scores*, *link='logit'*, *params_type='2PL'*, *init_slop=None*, *init_threshold=None*, *max_iter=1000*, *tol=1e-05*, **args*, ***kwargs*)

Bases: object

LINK_DT = {'logit': <class 'psy.irt.irm._LogitIrt'>, 'probit': <class 'psy.irt.irm._

PARAMS_TYPE_TP = ('1PL', '2PL', '3PL')

fit()

class psy.irt.irm.**Mirt**(*dim_size*, *init_slop=None*, *init_threshold=None*, *max_iter=1000*, *tol=0.0001*, **args*, ***kwargs*)

Bases: psy.irt.irm._BaseEmIrt, *psy.irt.base.LogitMixin*

em()

static z(*slop*, *threshold*, *theta*)

Module contents

psy.sem package

Submodules

psy.sem.ccfa module

psy.sem.ccfa.**delta_i_ccfa**(*data*, *lam*, *step=0.01*, *max_iter=1000000*, *rdd=3*, *tol=1e-06*)

psy.sem.ccfa.**get_irt_parameter**(*lam*, *thresholds*, *theta*)

psy.sem.ccfa.**get_thresholds**(*data*)

psy.sem.cfa module

psy.sem.cfa.**cfa**(*data*, *lam*, *step=0.01*, *max_iter=10000*, *rdd=3*, *tol=1e-07*)

psy.sem.sem module

psy.sem.sem.**sem**(*data*, *y*, *x*, *lam_x*, *lam_y*, *beta*, *gamma*, *method=u'ml'*, *step=0.1*, *max_iter=50000*, *tol=1e-07*)

Module contents

psy.settings package

Submodules

psy.settings.cat module

psy.settings.mirt module

Module contents

psy.utils package

Submodules

psy.utils.probs module

psy.utils.probs.**get_log_beta_pd**(*no_slip*, *guess*)

psy.utils.probs.**get_log_lognormal_pd**(*x*)

psy.utils.probs.**get_log_normal_pd**(*x*)

psy.utils.probs.**get_nodes_weights**(*dim_size*)

psy.utils.probs.**inverse_logistic**(*y*)

psy.utils.probs.**r4beta**(*shape1*, *shape2*, *a*, *b*, *size*)

psy.utils.randoms module

psy.utils.randoms.**gen_item_bank**(*trait_size*, *item_size*, *block_size=3*, *lower=1*, *upper=4*, *avg=0*, *std=1*)

:param trait_size: int :param item_size: int :param block_size: int :param lower: int|float :param upper: int|float :param avg: int|float :param std: int|float :return:

psy.utils.randoms.**random_params**(*item_dt*, *trait_size*, *block_size=3*, *lower=1*, *upper=4*, *avg=0*, *std=1*)

:param item_dt: dict,30:1,1:0,2:2}11 02 :param trait_size: int :param block_size: int :param lower: int|float :param upper: int|float :param avg: int|float :param std: int|float :return:

psy.utils.tools module

class psy.utils.tools.**cached_property**(*func*, *name=None*)

Bases: object

django

Module contents

4.1.2 Module contents

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/inuyasha2012/pypsy/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

pypsy could always use more documentation, whether as part of the official pypsy docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/inuyasha2012/pypsy/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *pypsy* for local development.

1. Fork the *pypsy* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pypsy.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pypsy
$ cd pypsy/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pypsy tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/inuyasha2012/pypsy/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_pypsy
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

History

6.1 0.0.1 (2018-09-18)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

N

P

R

S

T

U

V

Z