pysarpu Documentation

Release 0.1.0

Olivier COUDRAY

Oct 19, 2022

CONTENTS:

1 pysarpu **1**
- 1.1 Features . . . 1
- 1.2 Credits . . . 1

2 Installation **3**
- 2.1 Stable release . . . 3
- 2.2 From sources . . . 3

3 Usage **5**
- 3.1 Import . . . 5

4 Modules **7**
- 4.1 PU learning model . . . 7
- 4.2 Classification models . . . 11
- 4.3 Propensity models . . . 14

5 Contributing **15**
- 5.1 Types of Contributions . . . 15
- 5.2 Get Started! . . . 16
- 5.3 Pull Request Guidelines . . . 17
- 5.4 Tips . . . 17
- 5.5 Deploying . . . 17

6 Credits **19**
- 6.1 Development Lead . . . 19
- 6.2 Contributors . . . 19

7 History **21**
- 7.1 0.1.0 (2022-10-07) . . . 21

8 Indices and tables **23**

Index **25**

CHAPTER

ONE

PYSARPU

PU learning under SAR assumption with unknown propensity. Implementation of the general SAR-EM algorithm.

- Free software: MIT license
- Documentation: https://pysarpu.readthedocs.io.

1.1 Features

- TODO

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install pysarpu, run this command in your terminal:

```
$ pip install pysarpu
```

This is the preferred method to install pysarpu, 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 pysarpu can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/ocoudray/pysarpu
```

Or download the tarball:

```
$ curl -OJL https://github.com/ocoudray/pysarpu/tarball/master
```

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

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

CHAPTER

THREE

USAGE

3.1 Import

To use pysarpu in a project:

```
import pysarpu
```

PU learning classification model can be imported as follows:

```
from pysarpu import PU
```

The definition of a PU model requires the specification of a classification model and of a propensity model. Implementations can be found in sub-modules *pysarpu.classification* and *pysarpu.propensity*:

```
from pysarpu.classification import LinearLogisticRegression, LinearDiscriminantClassifier
from pysarpu.propensity import LogisticPropensity, LogProbitPropensity, GumbelPropensity
```

CHAPTER

FOUR

MODULES

4.1 PU learning model

class PUClassifier(*cmodel*, *emodel*, *da=False*)

PU learning classification model under unknown propensity. This model works by specifying a model on the classification and on the propensity and estimates parameters using EM algorithm (SAR-EM, Bekker et al.)

Parameters

- **cmodel** (pysarpu.classification.Classifier) – an instance of class :class: *Classifier <pysarpu.classification.Classifier>* representing the classification model. This package includes two types of classification models: logistic regression (accessible through *pysarpu.classification.LinearLogisticRegression*) and linear discriminant analysis (accessible through *pysarpu.classification.LinearDiscriminantClassifier*)
- **emodel** (pysarpu.propensity.Propensity) – an instance of class pysarpu.propensity.Propensity representing the propensity model. This package includes multiple pre-implemented propensity models: logistic propensity (*pysarpu.propensity.LogisticPropensity*), log-normal propensity (*pysarpu.propensity.LogProbitPropensity*) and Gumbel propensity (*pysarpu.propensity.GumbelPropensity*)
- **da** (bool, optional) – whether the classification model is a discriminant analysis type model (*True*) or not (*False*). Indeed, the likelihood maximized is not the same in these two settings. Default: *False*.

Returns

Return an instance of PU learning model (not yet initialized).

Return type

pysarpu.PUClassifier

initialization(*Xc*, *Xe*, *Y*, *w=1.0*)

Initialization of parameters for both classification and propensity models before running EM algorithm. The parameters of each models are initialized following their respective method: see *initialization* methods for *cmodel* and *emodel*.

Parameters

- **Xc** (*numpy.array* of shape (n, d_1)) – covariate matrix for classification. The parameters of *cmodel* will be initialized in agreement with the dimension of the entry data d_1.
- **Xe** (*numpy.array* of shape (n, d_2)) – covariate matrix for propensity. The parameters of *emodel* will be initialized in agreement with the dimension of the entry data d_2.

- **Y** (*numpy.array* vector of size n.) – observed labels. Only used in the computation of the initial log-likelihood.

Returns
None

e(*Xe*)

Propensity function using the current parameters of propensity model *emodel*.

Parameters
Xe (*numpy.array* of shape (n, d_2)) – covariate matrix for propensity.

Returns
vector of propensity scores.

Return type
numpy.array of size n

loge(*Xe*)

Logarithm of propensity function using the current parameters of propensity model *emodel*.

Parameters
Xe (*numpy.array* of shape (n, d_2)) – covariate matrix for propensity.

Returns
vector of log-propensity scores.

Return type
numpy.array of size n

predict_cproba(*Xc*)

Class probability predictions using the parameters of the classification model.

Parameters
Xc (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.

Returns
posterior class probabilities.

Return type
numpy.array vector of size n

predict_clogproba(*Xc*)

Class log-probability predictions using the parameters of the classification model.

Parameters
Xc (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.

Returns
posterior class log-probabilities.

Return type
numpy.array vector of size n

predict_c(*Xc*, *threshold=0.5*)

Class binary predictions using the parameters of the classification model.

Parameters

- **Xc** (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.
- **threshold** (*float*, optional (in $[0, 1]$)) – decision threshold defining the decision rule.

Returns
class predictions.

Return type
numpy.array binary vector of size n

`predict_proba`(*Xc*, *Xe*)

Label probability predictions based on the classification model *cmodel* and the propensity model *emodel*. Note that this is different from method *predict_cproba* which returns class probabilities instead.

Parameters

- **Xc** (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.
- **Xe** (*numpy.array* with shape (n, d_2).) – covariate matrix for propensity.

Returns
posterior label probabilities.

Return type
numpy.array vector of size n

`predict_logproba`(*Xc*, *Xe*)

Label log-probability predictions based on the classification model *cmodel* and the propensity model *emodel*. Note that this is different from method *predict_clogproba* which returns class log-probabilities instead.

Parameters

- **Xc** (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.
- **Xe** (*numpy.array* with shape (n, d_2).) – covariate matrix for propensity.

Returns
posterior label log-probabilities.

Return type
numpy.array vector of size n

`predict`(*Xc*, *Xe*, *threshold=0.5*)

Label binary predictions based on the classification model *cmodel* and the propensity model *emodel*. Note that this is different from method *predict_c* which returns class predictions instead.

Parameters

- **Xc** (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.
- **Xe** (*numpy.array* with shape (n, d_2).) – covariate matrix for propensity.
- **threshold** (*float*, optional (in $[0, 1]$)) – decision threshold defining the decision rule.

Returns
label binary predictions.

Return type
numpy.array binary vector of size n

`loglikelihood`(*Xc*, *Xe*, *Y*, *w=1.0*)

Log-likelihood function given the current parameters of classification and propensity models. Note that the funciton returns the mean of individual dlog-likelihoods (instead of the usual sum).

Parameters

- **Xc** (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.

- **Xe** (*numpy.array* with shape (n, d_2).) – covariate matrix for propensity.
- **Y** (*numpy.array* vector of size n.) – observed labels. Only used in the computation of the initial log-likelihood.
- **w** (either *float* (*1.*, default) or *numpy.array* of size n, optional.) – individual weights (experimental, not tested). Apply weights to observations in the computation of the likelihood.

Returns
log-likelihood.

Return type
float

expectation(*Xc*, *Xe*, *Y*)

Compute the expectation step of EM algorithm, return the probabilities for every instance to be of positive class given the observed labels.

Parameters

- **Xc** (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.
- **Xe** (*numpy.array* with shape (n, d_2).) – covariate matrix for propensity.
- **Y** (*numpy.array* vector of size n.) – observed labels. Only used in the computation of the initial log-likelihood.

Returns
posterior probabilities

Return type
np.array vector of size n

maximisation(*Xc*, *Xe*, *Y*, *gamma*, *w=1.0*, *warm_start=True*, *balance=False*)

Compute the maximisation step of EM algorithm, update the model parameters in both classification and propensity models.

Parameters

- **Xc** (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.
- **Xe** (*numpy.array* with shape (n, d_2).) – covariate matrix for propensity.
- **Y** (*numpy.array* vector of size n.) – observed labels. Only used in the computation of the initial log-likelihood.
- **gamma** (*numpy.array* of size n) – posterior probabilities obtained in the expectation step.
- **w** (either *float* (*1.*, default) or *numpy.array* of size n, optional.) – individual weights (experimental, not tested). Apply weights to observations in the computation of the likelihood.

Returns
None

fit(*Xc*, *Xe*, *Y*, *w=1.0*, *tol=1e-06*, *max_iter=10000.0*, *warm_start=False*, *balance=False*, *n_init=20*, *iter_init=20*)

Estimation of PU learning model parameters (classifier and propensity) through EM algorithm. Multiple random initialization are considered and trained over a few iterations. Then, only the one achieving the best log-likelihood is considered and trained until convergence.

Parameters

- **Xc** (*numpy.array* with shape (n, d_1).) – covariate matrix for classification.
- **Xe** (*numpy.array* with shape (n, d_2).) – covariate matrix for propensity.

- **Y** (*numpy.array* vector of size n.) – observed labels. Only used in the computation of the initial log-likelihood.
- **w** (either *float* (*1.*, default) or *numpy.array* of size n, optional.) – individual weights (experimental, not tested). Apply weights to observations in the computation of the likelihood.
- **tol** (*float, optional*) – tolerance parameter. Once the increase in the log-likelihood is below *tol*, the algorithm stops (default *1e-6*).
- **max_iter** (*int, optional*) – maximum number of iterations (default: *1e4*)
- **warm_start** (*bool, optional*) – indicates whether current parameters can be used for initialization (*True*) or if they should be re-initialized before estimation (default *False*).
- **balance** (*bool, optional*) – re-balance weights when fitting the propensity model in the maximization (experimental, potentially interesting in highly unbalanced situations). Default: *False*.
- **n_init** (*int, optional*) – number of initialization to consider in the Small EM initialization strategy (default: *n_init=20*)
- **iter_init** (*int, optional*) – maximum number of iterations to consider for each initialization (default: *20*).

Returns
None

save(*path*)

Saving PU learning model with current parameters as a binary file (rely on *pickle* library).

Parameters
path (*str*) – path at which the model should be saved.

Returns
None

4.2 Classification models

Two classification models can be found in submodule *pysarpu.classification*:

- a Linear Logistic Regression model
- a Linear Discriminant Analysis model

These two models inherit from the general class *sklearn.classification.Classifier*.

class LinearLogisticRegression

Linear logistic regression model for classification.

Parameters
params (*numpy.array* vector of size $d_1 + 1$) – current parameter vector.

initialization(*Xc*, *w=1.0*)

Initialization of the parameters of the model. Initial parameters are chosen randomly and the dimension of parameter vector is the dimension of the covariates + 1 (intercept).

Parameters
Xc (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.

Returns
None

fit(*Xc*, *gamma*, *w=1.0*, *warm_start=True*)

Estimation of the parameters of the model given the covariates and the observed output. Note that the output does not need to be binary classes, it can consist in probability values.

Parameters

- **Xc** (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.
- **gamma** (*numpy.array* of size n) – posterior probabilities obtained in the expectation step.
- **w** (either *float* (*1.*, default) or *numpy.array* of size n, optional.) – individual weights (experimental, not tested). Apply weights to observations in the computation of the likelihood.
- **warm_start** (*bool, optional*) – indicates whether current parameters can be used for initialization (*True*) or if they should be re-initialized before estimation (default *False*).

Returns
None

eta(*Xc*)

Class probability predictions given the current parameters.

Parameters
Xc (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.

Returns
class probabilities.

Return type
numpy.array vector of size n

logeta(*Xc*)

Class log-probability predictions given the current parameters.

Parameters
Xc (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.

Returns
class log-probabilities.

Return type
numpy.array vector of size n

class LinearDiscriminantClassifier

Linear Discriminant Analysis model for classification.

Parameters
params (*dict*) – current parameters: *pi* is the class prior, *mu_0* the mean vector for negative class, *mu_1* the mean vector for positive class, *Sigma* the covariance matrix.

initialization(*Xc*, *w=1.0*)

Initialization of the parameters of the model:

- the class prior *pi* is randomly and uniformly drawn in $[0, 1]$
- the mean vectors *mu_0* and *mu_1* are drawn as standardized gaussian variables
- the covariance matrix *Sigma* is initialized as the empirical covariance matrix of the whole data set.

Parameters
Xc (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.

fit(*Xc*, *gamma*, *w=1.0*, *warm_start=True*)

Estimation of the parameters of the model given the covariates and the observed output. Note that the output does not need to be binary classes, it can consist in probability values.

Parameters

- **Xc** (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.
- **gamma** (*numpy.array* of size n) – posterior probabilities obtained in the expectation step.
- **w** (either *float* (*1.*, default) or *numpy.array* of size n, optional.) – individual weights (experimental, not tested). Apply weights to observations in the computation of the likelihood.
- **warm_start** (*bool, optional*) – indicates whether current parameters can be used for initialization (*True*) or if they should be re-initialized before estimation (default *False*). Not important here as the maximization is straightforward and does not depend on the initialization.

Returns
None

eta(*Xc*)

Class probability predictions given the current parameters.

Parameters
Xc (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.

Returns
class probabilities.

Return type
numpy.array vector of size n

logeta(*Xc*)

Class log-probability predictions given the current parameters.

Parameters
Xc (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.

Returns
class log-probabilities.

Return type
numpy.array vector of size n

pdf_pos(*Xc*)

Individual likelihood for the positive distribution $\mathbb{P}(x|Z = 1)$ and for the current parameters.

Parameters
Xc (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.

Returns
individual likelihood values.

Return type
numpy.array vector of size n

pdf_neg(*Xc*)

Individual likelihood for the positive distribution $\mathbb{P}(x|Z = 0)$ and for the current parameters.

Parameters
Xc (*numpy.array* of shape (n, d_1)) – covariate matrix for classification.

Returns
individual likelihood values.

Return type
numpy.array vector of size n

4.3 Propensity models

Three propensity models are provided in submodule *pysarpu.propensity*:

- a Logistic Regression model
- a logistic function with log-normal link function
- a logistic function with Weibull link function

class LogisticPropensity

Logistic Propensity : the feature vector Xe is assumed to contain an intercept as its first column

class LogProbitPropensity

class GumbelPropensity

CHAPTER

FIVE

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/ocoudray/pysarpu/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

pysarpu could always use more documentation, whether as part of the official pysarpu 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/ocoudray/pysarpu/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 *pysarpu* for local development.

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

```
$ git clone git@github.com:your_name_here/pysarpu.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 pysarpu
$ cd pysarpu/
$ 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 pysarpu tests
$ python setup.py test or pytest
$ 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 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/ocoudray/pysarpu/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_pysarpu
```

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:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Olivier COUDRAY <olivier.coudray.15@polytechnique.org>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

HISTORY

7.1 0.1.0 (2022-10-07)

- First release on PyPI.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

E

e() (*PUClassifier method*), 8
eta() (*LinearDiscriminantClassifier method*), 13
eta() (*LinearLogisticRegression method*), 12
expectation() (*PUClassifier method*), 10

F

fit() (*LinearDiscriminantClassifier method*), 12
fit() (*LinearLogisticRegression method*), 11
fit() (*PUClassifier method*), 10

G

GumbelPropensity (*class in pysarpu.propensity*), 14

I

initialization() (*LinearDiscriminantClassifier method*), 12
initialization() (*LinearLogisticRegression method*), 11
initialization() (*PUClassifier method*), 7

L

LinearDiscriminantClassifier (*class in pysarpu.classification*), 12
LinearLogisticRegression (*class in pysarpu.classification*), 11
loge() (*PUClassifier method*), 8
logeta() (*LinearDiscriminantClassifier method*), 13
logeta() (*LinearLogisticRegression method*), 12
LogisticPropensity (*class in pysarpu.propensity*), 14
loglikelihood() (*PUClassifier method*), 9
LogProbitPropensity (*class in pysarpu.propensity*), 14

M

maximisation() (*PUClassifier method*), 10

P

pdf_neg() (*LinearDiscriminantClassifier method*), 13
pdf_pos() (*LinearDiscriminantClassifier method*), 13
predict() (*PUClassifier method*), 9
predict_c() (*PUClassifier method*), 8
predict_clogproba() (*PUClassifier method*), 8
predict_cproba() (*PUClassifier method*), 8
predict_logproba() (*PUClassifier method*), 9
predict_proba() (*PUClassifier method*), 9
PUClassifier (*class in pysarpu*), 7

S

save() (*PUClassifier method*), 11