pyuplift

Release 0.4.3

Artem Kuchumov
contributors

Dec 02, 2019

Contents

pyuplift is a scientific uplift modeling library. It implements variable selection and transformation approaches. pyuplift provides API for work with such an uplift datasets as Hillstrom Email Marketing and Criteo Uplift Prediction.

CHAPTER 1

Contents

1.1 Installation Guide

1.1.1 Install from PyPI

```
pip install pyuplift
```

1.1.2 Install from source code

```
python setup.py install
```

1.2 Examples of Usage

This section contains official examples of usage pyuplift package.

1.2.1 Contents

1.3 Contribute to pyuplift

Everyone is more than welcome to contribute. It is a way to make the project better and more accessible to more users.

Guidelines

1.3.1 Submit Pull Request

- Before submit, please rebase your code on the most recent version of master, you can do it by

```
git remote add upstream https://github.com/duketemon/pyuplift
git fetch upstream
git rebase upstream/master
```

- If you have multiple small commits, it might be good to merge them together(use git rebase then squash) into more meaningful groups.
- Send the pull request!
  - Fix the problems reported by automatic checks
  - If you are contributing a new module, consider add a testcase

1.3.2 Git Workflow Howtos

How to resolve conflict with master

- First rebase to most recent master

```
# The first two steps can be skipped after you do it once.
git remote add upstream https://github.com/duketemon/pyuplift
git fetch upstream
git rebase upstream/master
```

- The git may show some conflicts it cannot merge, say `conflicted.py`.
  - Manually modify the file to resolve the conflict.
  - After you resolved the conflict, mark it as resolved by

```
git add conflicted.py
```

- Then you can continue rebase by

```
git rebase --continue
```

- Finally push to your fork, you may need to force push here.

```
git push --force
```

How to combine multiple commits into one

Sometimes we want to combine multiple commits, especially when later commits are only fixes to previous ones, to create a PR with set of meaningful commits. You can do it by following steps.

- Before doing so, configure the default editor of git if you haven't done so before.

```
git config core.editor the-editor-you-like
```

- Assume we want to merge last 3 commits, type the following commands

```
git rebase -i HEAD~3
```

- It will pop up an text editor. Set the first commit as `pick`, and change later ones to `squash`.
- After you saved the file, it will pop up another text editor to ask you modify the combined commit message.
- Push the changes to your fork, you need to force push.

```
git push --force
```

What is the consequence of force push

The previous two tips requires force push, this is because we altered the path of the commits. It is fine to force push to your own fork, as long as the commits changed are only yours.

1.3.3 Documents

- Documentation is built using sphinx.
- Each document is written in reStructuredText.
- You can build document locally to see the effect.

1.4 Base Model

The base class for all uplift estimators.

Note: This class should not be used directly. Use derived classes instead.

1.5 Variable Selection

The pyuplift.variable_selection module includes classes which belongs to variable selection group of approaches.

1.5.1 Two Model

The class which implements the two model approach [1].

Parameters	
	no_treatment_model : object, optional (default=sklearn.linear_model.LinearRegression) The regression model which will be used for predict uplift. **has_treatment_model : object, optional (default=sklearn.linear_model.LinearRegression)** The regression model which will be used for predict uplift.

Methods

fit(self, X, y, t)	Build a two model model from the training set (X, y, t).
predict(self, X, t=None)	Predict an uplift for X.

fit(self, X, y, t)

Build a model model model from the training set (X, y, t).

Parameters	
	X: numpy ndarray with shape = [n_samples, n_features] Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments.
Returns	**self : object**

predict(self, X, t=None)

Predict an uplift for X.

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **t: numpy array with shape = [n_samples,] or None** Array of treatments.
Returns	**self : object** The predicted values.

References

1. A Literature Survey and Experimental Evaluation of the State-of-the-Art in Uplift Modeling: A Stepping Stone Toward the Development of Prescriptive Analytics by Floris Devriendt, Darie Moldovan, and Wouter Verbeke

```
from pyuplift.variable_selection import TwoModel
...
model = TwoModel()
model.fit(X[train_indexes, :], y[train_indexes], t[train_indexes])
uplift = model.predict(X[test_indexes, :])
print(uplift)
```

1.5.2 Econometric

The class which implements the econometric approach [1].

Parameters	**model : object, optional (default=sklearn.linear_model.LinearRegression)** The regression model which will be used for predict uplift.

Methods

fit(self, X, y, t)	Build an econometric model from the training set (X, y, t).
predict(self, X, t=None)	Predict an uplift for X.

fit(self, X, y, t)

Build an econometric model from the training set (X, y, t).

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments.
Returns	**self : object**

predict(self, X, t=None)

Predict an uplift for X.

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **t: numpy array with shape = [n_samples,] or None** Array of treatments.
Returns	**self : object** The predicted values.

References

1. A Literature Survey and Experimental Evaluation of the State-of-the-Art in Uplift Modeling: A Stepping Stone Toward the Development of Prescriptive Analytics by Floris Devriendt, Darie Moldovan, and Wouter Verbeke

```
from pyuplift.variable_selection import Econometric
...
model = Econometric()
model.fit(X[train_indexes, :], y[train_indexes], t[train_indexes])
uplift = model.predict(X[test_indexes, :])
print(uplift)
```

1.5.3 Dummy

The class which implements the dummy approach [1].

Parameters	**model : object, optional (default=sklearn.linear_model.LinearRegression)** The regression model which will be used for predict uplift.

Methods

fit(self, X, y, t)	Build a dummy model from the training set (X, y, t).
predict(self, X, t=None)	Predict an uplift for X.

fit(self, X, y, t)

Build a dummy model from the training set (X, y, t).

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments.
Returns	**self : object**

predict(self, X, t=None)

Predict an uplift for X.

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **t: numpy array with shape = [n_samples,] or None** Array of treatments.
Returns	**self : object** The predicted values.

References

1. A Literature Survey and Experimental Evaluation of the State-of-the-Art in Uplift Modeling: A Stepping Stone Toward the Development of Prescriptive Analytics by Floris Devriendt, Darie Moldovan, and Wouter Verbeke

```
from pyuplift.variable_selection import Dummy
...
model = Dummy()
model.fit(X[train_indexes, :], y[train_indexes], t[train_indexes])
uplift = model.predict(X[test_indexes, :])
print(uplift)
```

1.5.4 Cadit

The class which implements the cadit approach [1].

Parameters	**model : object, optional (default=sklearn.linear_model.LinearRegression)** The regression model which will be used for predict uplift.

Methods

fit(self, X, y, t)	Build a model from the training set (X, y, t).
predict(self, X, t=None)	Predict an uplift for X.

fit(self, X, y, t)

Build a model from the training set (X, y, t).

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments.
Returns	**self : object**

predict(self, X, t=None)

Predict an uplift for X.

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **t: numpy array with shape = [n_samples,] or None** Array of treatments.
Returns	**self : object** The predicted values.

References

1. Weisberg HI, Pontes VP. Post hoc subgroups in clinical trials: Anathema or analytics? // Clinical trials. 2015 Aug;12(4):357-64.

```
from pyuplift.variable_selection import Cadit
...
model = Cadit()
model.fit(X[train_indexes, :], y[train_indexes], t[train_indexes])
uplift = model.predict(X[test_indexes, :])
print(uplift)
```

variable_selection.TwoModel([no_treatment_model, has_treatment_model])	A two model approach.
variable_selection.Econometric([model])	An econometric approach.
variable_selection.Dummy([model])	A dummy approach.
variable_selection.Cadit([model])	A cadit approach.

1.6 Transformation

The pyuplift.transformation module includes classes which belongs to a transformation group of approaches.

1.6.1 Transformation Base Model

The base class for a transformation uplift estimators.

Note: This class should not be used directly. Use derived classes instead.

1.6.2 Lai

The class which implements the Lai's approach [1].

Parameters	
	model : object, optional (default=sklearn.linear_model.LogisticRegression) The classification model which will be used for predict uplift. **use_weights : boolean, optional (default=False)** Use or not weights?

Methods

fit(self, X, y, t)	Build a the model from the training set (X, y, t).
predict(self, X, t=None)	Predict an uplift for X.

fit(self, X, y, t)

Build a the model from the training set (X, y, t).

Parameters	
	X: numpy ndarray with shape = [n_samples, n_features] Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments.
Returns	**self : object**

predict(self, X, t=None)

Predict an uplift for X.

Parameters	
	X: numpy ndarray with shape = [n_samples, n_features] Matrix of features. **t: numpy array with shape = [n_samples,] or None** Array of treatments.
Returns	**self : object** The predicted values.

References

1. A Literature Survey and Experimental Evaluation of the State-of-the-Art in Uplift Modeling: A Stepping Stone Toward the Development of Prescriptive Analytics by Floris Devriendt, Darie Moldovan, and Wouter Verbeke

```
from pyuplift.transformation import Lai
...
model = Lai()
model.fit(X[train_indexes, :], y[train_indexes], t[train_indexes])
uplift = model.predict(X[test_indexes, :])
print(uplift)
```

1.6.3 Kane

The class which implements the Kane's approach [1].

Parameters	**model : object, optional (default=sklearn.linear_model.LogisticRegression)** The classification model which will be used for predict uplift. **use_weights : boolean, optional (default=False)** Use or not weights?

Methods

fit(self, X, y, t)	Build the model from the training set (X, y, t).
predict(self, X, t=None)	Predict an uplift for X.

fit(self, X, y, t)

Build the model from the training set (X, y, t).

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments.
Returns	**self : object**

predict(self, X, t=None)

Predict an uplift for X.

Parameters	
	X: numpy ndarray with shape = [n_samples, n_features] Matrix of features. **t: numpy array with shape = [n_samples,] or None** Array of treatments.
Returns	
	self : object The predicted values.

References

1. A Literature Survey and Experimental Evaluation of the State-of-the-Art in Uplift Modeling: A Stepping Stone Toward the Development of Prescriptive Analytics by Floris Devriendt, Darie Moldovan, and Wouter Verbeke

```
from pyuplift.transformation import Kane
...
model = Kane()
model.fit(X[train_indexes, :], y[train_indexes], t[train_indexes])
uplift = model.predict(X[test_indexes, :])
print(uplift)
```

1.6.4 Jaskowski

The class which implements the Jaskowski's approach [1].

Parameters	
	model : object, optional (default=sklearn.linear_model.LogisticRegression) The classification model which will be used for predict uplift.

Methods

fit(self, X, y, t)	Build the model from the training set (X, y, t).
predict(self, X, t=None)	Predict an uplift for X.

fit(self, X, y, t)

Build the model from the training set (X, y, t).

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments.
Returns	**self : object**

predict(self, X, t=None)

Predict an uplift for X.

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **t: numpy array with shape = [n_samples,] or None** Array of treatments.
Returns	**self : object** The predicted values.

References

1. A Literature Survey and Experimental Evaluation of the State-of-the-Art in Uplift Modeling: A Stepping Stone Toward the Development of Prescriptive Analytics by Floris Devriendt, Darie Moldovan, and Wouter Verbeke

```
from pyuplift.transformation import Jaskowski
...
model = Jaskowski()
model.fit(X[train_indexes, :], y[train_indexes], t[train_indexes])
uplift = model.predict(X[test_indexes, :])
print(uplift)
```

1.6.5 Pessimistic

The class which implements the pessimistic approach [1].

Parameters	**model : object, optional (default=sklearn.linear_model.LogisticRegression)** The classification model which will be used for predict uplift.

Methods

fit(self, X, y, t)	Build the model from the training set (X, y, t).
predict(self, X, t=None)	Predict an uplift for X.

fit(self, X, y, t)

Build the model from the training set (X, y, t).

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments.
Returns	**self : object**

predict(self, X, t=None)

Predict an uplift for X.

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **t: numpy array with shape = [n_samples,] or None** Array of treatments.
Returns	**self : object** The predicted values.

References

1. A Literature Survey and Experimental Evaluation of the State-of-the-Art in Uplift Modeling: A Stepping Stone Toward the Development of Prescriptive Analytics by Floris Devriendt, Darie Moldovan, and Wouter Verbeke

```
from pyuplift.transformation import Pessimistic
...
model = Pessimistic()
model.fit(X[train_indexes, :], y[train_indexes], t[train_indexes])
uplift = model.predict(X[test_indexes, :])
print(uplift)
```

1.6.6 Reflective

The class which implements the reflective approach [1].

Parameters	**model : object, optional (default=sklearn.linear_model.LogisticRegression)** The classification model which will be used for predict uplift.

Methods

fit(self, X, y, t)	Build the model from the training set (X, y, t).
predict(self, X, t=None)	Predict an uplift for X.

fit(self, X, y, t)

Build the model from the training set (X, y, t).

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments.
Returns	**self : object**

predict(self, X, t=None)

Predict an uplift for X.

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **t: numpy array with shape = [n_samples,] or None** Array of treatments.
Returns	**self : object** The predicted values.

References

1. A Literature Survey and Experimental Evaluation of the State-of-the-Art in Uplift Modeling: A Stepping Stone Toward the Development of Prescriptive Analytics by Floris Devriendt, Darie Moldovan, and Wouter Verbeke

```
from pyuplift.transformation import Reflective
...
model = Reflective()
model.fit(X[train_indexes, :], y[train_indexes], t[train_indexes])
uplift = model.predict(X[test_indexes, :])
print(uplift)
```

transformation.TransformationBaseModel()	A base model of all classes which implements a transformation approaches.
transformation.Lai([model, use_weights])	A Lai's approach.
transformation.Kane([model, use_weights])	A Kane's approach.
transformation.Jaskowski([model])	A Jaskowski's approach.
transformation.Pessimistic([model])	A pessimistic approach.
transformation.Reflective([model])	A reflective approach.

1.7 Datasets

1.7.1 load_criteo_uplift_prediction

Loading the Criteo Uplift Prediction dataset from the local file.

Data description

This dataset is constructed by assembling data resulting from several incrementality tests, a particular randomized trial procedure where a random part of the population is prevented from being targeted by advertising. It consists of 25M rows, each one representing a user with 11 features, a treatment indicator and 2 labels (visits and conversions).

Privacy

For privacy reasons the data has been sub-sampled non-uniformly so that the original incrementality level cannot be deduced from the dataset while preserving a realistic, challenging benchmark. Feature names have been anonymized and their values randomly projected so as to keep predictive power while making it practically impossible to recover the original features or user context.

Features	11
Treatment	2
Samples total	25,309,483
Average visit rate	0.04132
Average conversion rate	0.00229

More information about dataset you can find in the official dataset description.

Parameters	
	data_home: str Specify another download and cache folder for the dataset. By default the dataset will be stored in the data folder in the same folder. **download_if_missing: bool, default=True** Download the dataset if it is not downloaded.
Returns:	**dataset**: dict Dictionary object with the following attributes: **dataset.description** : str Description of the Criteo Uplift Prediction dataset. **dataset.data**: numpy ndarray of shape (25309483, 11) Each row corresponding to the 11 feature values in order. **dataset.feature_names**: list, size 11 List of feature names. **dataset.treatment**: numpy ndarray, shape (25309483,) Each value corresponds to the treatment. **dataset.target**: numpy array of shape (25309483,) Each value corresponds to one of the outcomes. By default, it's *visit* outcome (look at *target_visit* below). **dataset.target_visit**: numpy array of shape (25309483,) Each value corresponds to whether a visit occurred for this user (binary, label). **dataset.target_exposure**: numpy array of shape (25309483,) Each value corresponds to treatment effect, whether the user has been effectively exposed (binary). **dataset.target_conversion**: numpy array of shape (25309483,) Each value corresponds to whether a conversion occurred for this user (binary, label).

Examples

```
from pyuplift.datasets import load_criteo_uplift_prediction
df = load_criteo_uplift_prediction()
print(df)
```

1.7.2 download_criteo_uplift_prediction

Downloading the Criteo Uplift Prediction dataset.

Data description

This dataset is constructed by assembling data resulting from several incrementality tests, a particular randomized trial procedure where a random part of the population is prevented from being targeted by advertising. It consists of 25M rows, each one representing a user with 11 features, a treatment indicator and 2 labels (visits and conversions).

Privacy

For privacy reasons the data has been sub-sampled non-uniformly so that the original incrementality level cannot be deduced from the dataset while preserving a realistic, challenging benchmark. Feature names have been anonymized and their values randomly projected so as to keep predictive power while making it practically impossible to recover the original features or user context.

Features	11
Treatment	2
Samples total	25,309,483
Average visit rate	0.04132
Average conversion rate	0.00229

More information about dataset you can find in the official dataset description.

Parameters:	**data_home**: str, default=None The URL to file with data. **url**: str, default=https://s3.us-east-2.amazonaws.com/criteo-uplift-dataset/criteo-uplift.csv.gz The URL to file with data.
Returns:	**dataset**: dict Dictionary object with the following attributes: **dataset.description** : str Description of the Criteo Uplift Prediction dataset. **dataset.data**: numpy ndarray of shape (25309483, 11) Each row corresponding to the 11 feature values in order. **dataset.feature_names**: list, size 11 List of feature names. **dataset.treatment**: numpy ndarray, shape (25309483,) Each value corresponds to the treatment. **dataset.target**: numpy array of shape (25309483,) Each value corresponds to one of the outcomes. By default, it's *visit* outcome (look at *target_visit* below). **dataset.target_visit**: numpy array of shape (25309483,) Each value corresponds to whether a visit occurred for this user (binary, label). **dataset.target_exposure**: numpy array of shape (25309483,) Each value corresponds to treatment effect, whether the user has been effectively exposed (binary). **dataset.target_conversion**: numpy array of shape (25309483,) Each value corresponds to whether a conversion occurred for this user (binary, label).

Examples

```
from pyuplift.datasets import download_criteo_uplift_prediction
download_criteo_uplift_prediction()
```

1.7.3 load_hillstrom_email_marketing

Loading the Hillstrom Email Marketing dataset from the local file.

Data description

This dataset contains 64,000 customers who last purchased within twelve months. The customers were involved in an e-mail test.

- 1/3 were randomly chosen to receive an e-mail campaign featuring Mens merchandise.
- 1/3 were randomly chosen to receive an e-mail campaign featuring Womens merchandise.
- 1/3 were randomly chosen to not receive an e-mail campaign.

During a period of two weeks following the e-mail campaign, results were tracked. Your job is to tell the world if the Mens or Womens e-mail campaign was successful.

Features	8
Treatment	3
Samples total	64,000
Average spend rate	1.05091
Average visit rate	0.14678
Average conversion rate	0.00903

More information about dataset you can find in the official paper.

Parameters:	**data_home**: str, default=None Specify another download and cache folder for the dataset. By default the dataset will be stored in the data folder in the same folder. **load_raw_data**: bool, default=False The loading of raw or preprocessed data? **download_if_missing**: bool, default=True Download the dataset if it is not downloaded.
Returns:	**dataset**: dict Dictionary object with the following attributes: **dataset.description** : str Description of the Hillstrom email marketing dataset. **dataset.data**: numpy ndarray of shape (64000, 8) Each row corresponding to the 8 feature values in order. **dataset.feature_names**: list, size 8 List of feature names. **dataset.treatment**: numpy ndarray, shape (64000,) Each value corresponds to the treatment. **dataset.target**: numpy array of shape (64000,) Each value corresponds to one of the outcomes. By default, it's *spend* outcome (look at *target_spend* below). **dataset.target_spend**: numpy array of shape (64000,) Each value corresponds to how much customers spent during a two-week outcome period. **dataset.target_visit**: numpy array of shape (64000,) Each value corresponds to whether people visited the site during a two-week outcome period. **dataset.target_conversion**: numpy array of shape (64000,) Each value corresponds to whether they purchased at the site ("conversion") during a two-week outcome period.

Examples

```
from pyuplift.datasets import load_hillstrom_email_marketing
df = load_hillstrom_email_marketing()
print(df)
```

1.7.4 download_hillstrom_email_marketing

Downloading the Hillstrom Email Marketing dataset.

Data description

This dataset contains 64,000 customers who last purchased within twelve months. The customers were involved in an e-mail test.

- 1/3 were randomly chosen to receive an e-mail campaign featuring Mens merchandise.
- 1/3 were randomly chosen to receive an e-mail campaign featuring Womens merchandise.
- 1/3 were randomly chosen to not receive an e-mail campaign.

During a period of two weeks following the e-mail campaign, results were tracked. Your job is to tell the world if the Mens or Womens e-mail campaign was successful.

Features	8
Treatment	3
Samples total	64,000
Average spend rate	1.05091
Average visit rate	0.14678
Average conversion rate	0.00903

More information about dataset you can find in the official paper.

Parameters	**data_home: str** Specify another download and cache folder for the dataset. By default the dataset will be stored in the data folder in the same folder. **url: str** The URL to file with data.
Returns	**None**

Examples

```
from pyuplift.datasets import download_hillstrom_email_marketing
download_hillstrom_email_marketing()
```

1.7.5 load_lalonde_nsw

Loading the Lalonde NSW dataset from the local file.

Data description

The dataset contains the treated and control units from the male sub-sample from the National Supported Work Demonstration as used by Lalonde in his paper.

Features	7
Treatment	2
Samples total	722

Features description

- **treat** - an indicator variable for treatment status.
- **age** - age in years.
- **educ** - years of schooling.
- **black** - indicator variable for blacks.
- **hisp** - indicator variable for Hispanics.
- **married** - indicator variable for martial status.
- **nodegr** - indicator variable for high school diploma.
- **re75** - real earnings in 1975.
- **re78** - real earnings in 1978.

More information about dataset you can find here.

Parameters:	
	data_home: str, default=None Specify another download and cache folder for the dataset. By default the dataset will be stored in the data folder in the same folder. **download_if_missing**: bool, default=True Download the dataset if it is not downloaded.
Returns:	**dataset**: dict Dictionary object with the following attributes: **dataset.description** : str Description of the Hillstrom email marketing dataset. **dataset.data**: numpy ndarray of shape (722, 7) Each row corresponding to the 7 feature values in order. **dataset.feature_names**: list, size 7 List of feature names. **dataset.treatment**: numpy ndarray, shape (722,) Each value corresponds to the treatment. **dataset.target**: numpy array of shape (722,) Each value corresponds to one of the outcomes. By default, it's *re78* outcome.

Examples

```
from pyuplift.datasets import load_lalonde_nsw
df = load_lalonde_nsw()
print(df)
```

1.7.6 download_lalonde_nsw

Downloading the Lalonde NSW dataset.

Data description

The dataset contains the treated and control units from the male sub-sample from the National Supported Work Demonstration as used by Lalonde in his paper.

Features	7
Treatment	2
Samples total	722

Features description

- **treat** - an indicator variable for treatment status.
- **age** - age in years.
- **educ** - years of schooling.
- **black** - indicator variable for blacks.
- **hisp** - indicator variable for Hispanics.
- **married** - indicator variable for martial status.
- **nodegr** - indicator variable for high school diploma.
- **re75** - real earnings in 1975.
- **re78** - real earnings in 1978.

More information about dataset you can find here.

Parameters	**data_home: str** Specify another download and cache folder for the dataset. By default the dataset will be stored in the data folder in the same folder. **control_data_url: str** The URL to file with data of the control group. **treated_data_url: str** The URL to file with data of the treated group. **separator: str** The separator which used in the data files. **column_names: list** List of column names of the dataset. **column_types: dict** List of types for columns of the dataset. **random_state: int** The random seed.
Returns	**None**

Examples

```
from pyuplift.datasets import download_lalonde_nsw
download_lalonde_nsw()
```

1.7.7 make_linear_regression

Generate data by formula.

Data description

Synthetic data generated by Generate data by formula:

```
Y' = X1 + X2 * T + E
Y = Y', if Y' - int(Y') > eps,
Y = 0, otherwise.
```

Statistics for default parameters and size equals 100,000:

Features	3
Treatment	2
Samples total	*size*
Y not equals 0	0.49438
Y values	0 to 555.93

Parameters:	**size**: integer The number of observations. **x1_params** : tuple(mu, sigma), default: (0, 1) The feature with gaussian distribution and mean=mu, sd=sigma. X1 ~ N(mu, sigma) **x2_params** : tuple(mu, sigma), default: (0, 0.1) The feature with gaussian distribution and mean=mu, sd=sigma. X2 ~ N(mu, sigma) **x3_params** : tuple(mu, sigma), default: (0, 1) The feature with gaussian distribution and mean=mu, sd=sigma. X3 ~ N(mu, sigma) **t_params** : tuple(mu, sigma), default: (0, 1) The treatment with uniform distribution. Min value=min, Max value=max-1 T ~ R(min, max) **e_params** : tuple(mu, sigma), default: (0, 1) The error with gaussian distribution and mean=mu, sd=sigma. E ~ N(mu, sigma) **eps** : tuple(mu, sigma), default: (0, 1) The border value. **random_state** : integer, default=777 random_state is the seed used by the random number generator.
Returns:	**dataset**: pandas DataFrame Generated data.

Examples

```
from pyuplift.datasets import make_linear_regression
df = make_linear_regression(10000)
print(df)
```

The pyuplift.datasets module includes utilities to load datasets, including methods to download and return popular datasets. It also features some artificial data generators.

1.7.8 Loaders

datasets.download_criteo_uplift_prediction([data_home, url])	Downloading the Criteo Uplift Prediction dataset.
datasets.load_criteo_uplift_prediction([data_home, download_if_missing])	Loading the Criteo Uplift Prediction dataset from the local file.
datasets.download_hillstrom_email_marketing([data_home, url])	Downloading the Hillstrom Email Marketing dataset.
datasets.load_hillstrom_email_marketing([data_home, load_raw_data, download_if_missing])	Loading the Hillstrom Email Marketing dataset from the local file.
datasets.download_lalonde_nsw([data_home, control_data_url, treated_data_url, separator, column_names, column_types, random_state])	Downloading the Lalonde NSW dataset.
datasets.load_lalonde_nsw([data_home, load_raw_data, download_if_missing])	Loading the Lalonde NSW dataset from the local file.

1.7.9 Generators

datasets.make_linear_regression(size, [x1_params, x2_params, x3_params, t_params, e_params, eps, seed])	Generate data by formula: Y' = X1+X2*T+E Y = Y', if Y' - int(Y') > eps, Y = 0, otherwise.

1.8 Model Selection

1.8.1 train_test_split

Split X, y, t into random train and test subsets.

Parameters	
	X: numpy ndarray with shape = [n_samples, n_features] Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments. **train_share: float, optional (default=0.7)** train_share represents the proportion of the dataset to include in the train split. **random_state: int, optional (default=None)** random_state is the seed used by the random number generator.
Return	
	X_train: numpy ndarray Train matrix of features. **X_test: numpy ndarray** Test matrix of features. **y_train: numpy array** Train array of target of feature. **y_test: numpy array** Test array of target of feature. **t_train: numpy array** Train array of treatments. **t_test: numpy array** Test array of treatments.

Examples

```
from pyuplift.model_selection import train_test_split
...
for seed in seeds:
    X_train, X_test, y_train, y_test, t_train, t_test = train_test_split(X, y, t,␣
↪train_share, seed)
    model.fit(X_train, y_train, t_train)
    score = get_average_effect(y_test, t_test, model.predict(X_test))
    scores.append(score)
```

1.8.2 treatment_cross_val_score

Evaluate a scores by cross-validation.

Parameters	**X: numpy ndarray with shape = [n_samples, n_features]** Matrix of features. **y: numpy array with shape = [n_samples,]** Array of target of feature. **t: numpy array with shape = [n_samples,]** Array of treatments. **train_share: float, optional (default=0.7)** train_share represents the proportion of the dataset to include in the train split. **random_state: int, optional (default=777)** random_state is the seed used by the random number generator.
Return	**scores: numpy array of floats** Array of scores of the estimator for each run of the cross validation.

Examples

```
from pyuplift.model_selection import treatment_cross_val_score
...
for model_name in models:
    scores = treatment_cross_val_score(X, y, t, models[model_name], cv, seeds=seeds)
```

The pyuplift.model_selection module includes model validation and splitter functions.

1.8.3 Splitter Functions

model_selection.train_test_split(X, y, t, [train_share, random_state])	Split X, y, t into random train and test subsets.

1.8.4 Model validation

model_selection.treatment_cross_val_score(X, y, t, model, [cv, train_share, seeds])	Evaluate a scores by cross-validation.

1.9 Metrics

1.9.1 get_average_effect

Estimating an average effect of the test set.

Parameters:	**y_test**: numpy array Actual y values. **t_test**: numpy array Actual treatment values. **y_pred**: numpy array Predicted y values by uplift model. **test_share**: float Share of the test data which will be taken for estimating an average effect.
Returns:	**average effect**: float Average effect on the test set.

Examples

```
from pyuplift.metrics import get_average_effect
...
model.fit(X_train, y_train, t_train)
y_pred = model.predict(X_test)
effect = get_average_effect(y_test, t_test, y_pred, test_share)
print(effect)
```

The pyuplift.metrics module includes score functions, performance metrics and pairwise metrics and distance computations.

metrics.get_average_effect(y_test, t_test, y_pred, [test_share])	Estimating an average effect of the test set.

1.10 Utilities

1.10.1 download_file

Download file from *url* to *output_path*.

Parameters	**url: string** Data's URL. **output_path: string** Path where file will be saved.
Returns	**None**

Examples

```
from pyuplift.utils import download_file
...
 if not os.path.exists(data_path):
     if not os.path.exists(archive_path):
         download_file(url, archive_path)
```

1.10.2 retrieve_from_gz

The retrieving gz-archived data from *archive_path* to *output_path*.

Parameters	**archive_path: string** The archive path. **output_path: string** The retrieved data path.
Returns	**None**

Examples

```
from pyuplift.utils import retrieve_from_gz
...
 if not os.path.exists(data_path):
     if not os.path.exists(archive_path):
         download_file(url, archive_path)
             retrieve_from_gz(archive_path, data_path)
```

The pyuplift.utils module includes various utilities.

utils.download_file(url, output_path)	Download file from *url* to *output_path*.
utils.retrieve_from_gz(archive_path, output_path)	The retrieving gz-archived data from *archive_path* to *output_path*