heamy Documentation

Release 0.0.7

Artem Golubin

Nov 15, 2017

Contents

Contents:

CHAPTER 1

Usage

Using heamy in a project:

```
from heamy.dataset import Dataset
from heamy.estimator import Regressor, Classifier
from heamy.pipeline import ModelsPipeline
```

1.1 Stacking

```
# load boston dataset from sklearn
from sklearn.datasets import load_boston
data = load_boston()
X, y = data['data'], data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_
↪state=111)

# create dataset
dataset = Dataset(X_train,y_train,X_test)

# initialize RandomForest & LinearRegression
model_rf = Regressor(dataset=dataset, estimator=RandomForestRegressor, parameters={'n_
↪estimators': 50},name='rf')
model_lr = Regressor(dataset=dataset, estimator=LinearRegression, parameters={
↪'normalize': True},name='lr')

# Stack two models
# Returns new dataset with out-of-fold predictions
pipeline = ModelsPipeline(model_rf,model_lr)
stack_ds = pipeline.stack(k=10,seed=111)

# Train LinearRegression on stacked data (second stage)
stacker = Regressor(dataset=stack_ds, estimator=LinearRegression)
results = stacker.predict()
```

```
# Validate results using 10 fold cross-validation
results = stacker.validate(k=10,scorer=mean_absolute_error)
```

1.2 Blending

```
# load boston dataset from sklearn
from sklearn.datasets import load_boston
data = load_boston()
X, y = data['data'], data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_
↪state=111)

# create dataset
dataset = Dataset(X_train,y_train,X_test)

# initialize RandomForest & LinearRegression
model_rf = Regressor(dataset=dataset, estimator=RandomForestRegressor, parameters={'n_
↪estimators': 50},name='rf')
model_lr = Regressor(dataset=dataset, estimator=LinearRegression, parameters={
↪'normalize': True},name='lr')

# Stack two models
# Returns new dataset with out-of-fold predictions
pipeline = ModelsPipeline(model_rf,model_lr)
stack_ds = pipeline.blend(proportion=0.2,seed=111)

# Train LinearRegression on stacked data (second stage)
stacker = Regressor(dataset=stack_ds, estimator=LinearRegression)
results = stacker.predict()
# Validate results using 10 fold cross-validation
results = stacker.validate(k=10,scorer=mean_absolute_error)
```

1.3 Weighted average

```
dataset = Dataset(preprocessor=boston_dataset)

model_rf = Regressor(dataset=dataset, estimator=RandomForestRegressor, parameters={'n_
↪estimators': 151},name='rf')
model_lr = Regressor(dataset=dataset, estimator=LinearRegression, parameters={
↪'normalize': True},name='lr')
model_knn = Regressor(dataset=dataset, estimator=KNeighborsRegressor, parameters={'n_
↪neighbors': 15},name='knn')

pipeline = ModelsPipeline(model_rf,model_lr,model_knn)

weights = pipeline.find_weights(mean_absolute_error)
result = pipeline.weight(weights)
```

CHAPTER 2

Installation

2.1 Stable release

To install heamy, run this command in your terminal:

```
$ pip install heamy
```

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

2.2 From sources

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

You can either clone the public repository:

```
$ git clone git://github.com/rushter/heamy
```

Or download the tarball:

```
$ curl  -OL https://github.com/rushter/heamy/tarball/master
```

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

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

CHAPTER 3

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:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/rushter/heamy/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.

3.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

3.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

3.1.4 Write Documentation

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

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/rushter/heamy/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 :)

3.2 Get Started!

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

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

   ```
   $ git clone git@github.com:your_name_here/heamy.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 heamy
   $ cd heamy/
   $ 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 heamy tests
   $ python setup.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.

3.3 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 and 3.5. Check https://travis-ci.org/rushter/heamy/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 4

API

4.1 heamy.dataset module

class `heamy.dataset.`**`Dataset`**(*X_train=None*, *y_train=None*, *X_test=None*, *y_test=None*, *preprocessor=None*, *use_cache=True*)

Dataset wrapper.

Parameters **X_train** : pd.DataFrame or np.ndarray, optional

y_train : pd.DataFrame, pd.Series or np.ndarray, optional

X_test : pd.DataFrame or np.ndarray, optional

y_test : pd.DataFrame, pd.Series or np.ndarray, optional

preprocessor : function, optional

A callable function that returns preprocessed data.

use_cache : bool, default True

If *use_cache=True* then preprocessing step will be cached until function code is changed.

Examples

```
>>> # function-based definition
>>> from sklearn.datasets import load_boston
>>> def boston_dataset():
>>>     data = load_boston()
>>>     X, y = data['data'], data['target']
>>>     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,␣
↪random_state=111)
>>>     return X_train, y_train, X_test, y_test
>>> dataset = Dataset(preprocessor=boston_dataset)
>>> # class-based definition
>>> class BostonDataset(Dataset):
>>> def preprocess(self):
>>>     data = load_boston()
>>>     X, y = data['data'], data['target']
>>>     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,␣
↪random_state=111)
>>>     return X_train, y_train, X_test, y_test
```

hash

Return md5 hash for current dataset.

kfold(*k=5*, *stratify=False*, *shuffle=True*, *seed=33*)

K-Folds cross validation iterator.

Parameters **k** : int, default 5

stratify : bool, default False

shuffle : bool, default True

seed : int, default 33

Yields X_train, y_train, X_test, y_test, train_index, test_index

merge(*ds*, *inplace=False*, *axis=1*)

Merge two datasets.

Parameters **axis** : {0,1}

ds : *Dataset*

inplace : bool, default False

Returns *Dataset*

split(*test_size=0.1*, *stratify=False*, *inplace=False*, *seed=33*, *indices=None*)

Splits train set into two parts (train/test).

Parameters **test_size** : float, default 0.1

stratify : bool, default False

inplace : bool, default False

If *True* then dataset's train/test sets will be replaced with new data.

seed : int, default 33

indices : list(np.ndarray, np.ndarray), default None

Two numpy arrays that contain indices for train/test slicing.

Returns **X_train** : np.ndarray

y_train : np.ndarray

X_test : np.ndarray

y_test : np.ndarray

Examples

```
>>> train_index = np.array(range(250))
>>> test_index = np.array(range(250,333))
>>> res = dataset.split(indices=(train_index,test_index))
```

```
>>> res = dataset.split(test_size=0.3,seed=1111)
```

to_csc()

Convert Dataset to scipy's Compressed Sparse Column matrix.

to_csr()

Convert Dataset to scipy's Compressed Sparse Row matrix.

to_dense()

Convert sparse Dataset to dense matrix.

4.2 heamy.estimator module

4.2.1 Regressor

class heamy.estimator.**Regressor**(*dataset*, *estimator=None*, *parameters=None*, *name=None*, *use_cache=True*)

Bases: heamy.estimator.BaseEstimator

Wrapper for regression problems.

Parameters **dataset** : *Dataset* object

estimator : a callable scikit-learn like interface, custom function/class, optional

parameters : dict, optional

Arguments for *estimator* object.

name : str, optional

The unique name of *Estimator* object.

use_cache : bool, optional

if *True* then validate/predict/stack/blend results will be cached.

blend(*proportion=0.2*, *stratify=False*, *seed=100*, *indices=None*)

Blend a single model. You should rarely be using this method. Use *ModelsPipeline.blend* instead.

Parameters **proportion** : float, default 0.2

Test size holdout.

stratify : bool, default False

seed : int, default 100

indices : list(np.ndarray,np.ndarray), default None

Two numpy arrays that contain indices for train/test slicing. (train_index,test_index)

Returns *Dataset*

stack(*k=5*, *stratify=False*, *shuffle=True*, *seed=100*, *full_test=True*)

Stack a single model. You should rarely be using this method. Use *ModelsPipeline.stack* instead.

Parameters **k** : int, default 5

stratify : bool, default False

shuffle : bool, default True

seed : int, default 100

full_test : bool, default True

If *True* then evaluate test dataset on the full data otherwise take the mean of every fold.

Returns *Dataset* with out of fold predictions.

validate(*scorer=None*, *k=1*, *test_size=0.1*, *stratify=False*, *shuffle=True*, *seed=100*, *indices=None*)

Evaluate score by cross-validation.

Parameters **scorer** : function(y_true,y_pred), default None

Scikit-learn like metric that returns a score.

k : int, default 1

The number of folds for validation.

If k=1 then randomly split X_train into two parts otherwise use K-fold approach.

test_size : float, default 0.1

Size of the test holdout if k=1.

stratify : bool, default False

shuffle : bool, default True

seed : int, default 100

indices : list(np.array,np.array), default None

Two numpy arrays that contain indices for train/test slicing. (train_index,test_index)

Returns y_true: list

Actual labels.

y_pred: list

Predicted labels.

Examples

```
>>> # Custom indices
>>> train_index = np.array(range(250))
>>> test_index = np.array(range(250,333))
>>> res = model_rf.validate(mean_absolute_error,indices=(train_index,test_
↪index))
```

4.2.2 Classifier

class heamy.estimator.**Classifier**(*dataset*, *estimator=None*, *parameters=None*, *name=None*, *use_cache=True*, *probability=True*)

Bases: heamy.estimator.BaseEstimator

Wrapper for classification problems.

Parameters **dataset** : *Dataset* object

estimator : a callable scikit-learn like interface, custom function/class, optional

parameters : dict, optional

Arguments for *estimator* object.

name : str, optional

The unique name of *Estimator* object.

use_cache : bool, optional

if *True* then validate/predict/stack/blend results will be cached.

blend (*proportion=0.2*, *stratify=False*, *seed=100*, *indices=None*)

Blend a single model. You should rarely be using this method. Use *ModelsPipeline.blend* instead.

Parameters **proportion** : float, default 0.2

Test size holdout.

stratify : bool, default False

seed : int, default 100

indices : list(np.ndarray,np.ndarray), default None

Two numpy arrays that contain indices for train/test slicing. (train_index,test_index)

Returns *Dataset*

stack (*k=5*, *stratify=False*, *shuffle=True*, *seed=100*, *full_test=True*)

Stack a single model. You should rarely be using this method. Use *ModelsPipeline.stack* instead.

Parameters **k** : int, default 5

stratify : bool, default False

shuffle : bool, default True

seed : int, default 100

full_test : bool, default True

If *True* then evaluate test dataset on the full data otherwise take the mean of every fold.

Returns *Dataset* with out of fold predictions.

validate (*scorer=None*, *k=1*, *test_size=0.1*, *stratify=False*, *shuffle=True*, *seed=100*, *indices=None*)

Evaluate score by cross-validation.

Parameters **scorer** : function(y_true,y_pred), default None

Scikit-learn like metric that returns a score.

k : int, default 1

The number of folds for validation.

If k=1 then randomly split X_train into two parts otherwise use K-fold approach.

test_size : float, default 0.1

Size of the test holdout if k=1.

stratify : bool, default False

shuffle : bool, default True

seed : int, default 100

indices : list(np.array,np.array), default None

Two numpy arrays that contain indices for train/test slicing. (train_index,test_index)

Returns y_true: list

Actual labels.

y_pred: list

Predicted labels.

Examples

```
>>> # Custom indices
>>> train_index = np.array(range(250))
>>> test_index = np.array(range(250,333))
>>> res = model_rf.validate(mean_absolute_error,indices=(train_index,test_
↪index))
```

4.3 heamy.pipeline module

class heamy.pipeline.**ModelsPipeline**(**args*)

Combines sequence of models.

add(*model*)

Adds a single model.

Parameters **model** : *Estimator*

apply(*func*)

Applies function along models output.

Parameters **func** : function

Arbitrary function with one argument.

Returns *PipeApply*

Examples

```
>>> pipeline = ModelsPipeline(model_rf,model_lr)
>>> pipeline.apply(lambda x: np.max(x,axis=0)).execute()
```

blend(*proportion=0.2*, *stratify=False*, *seed=100*, *indices=None*, *add_diff=False*)

Blends sequence of models.

Parameters **proportion** : float, default 0.2

stratify : bool, default False

seed : int, default False

indices : list(np.ndarray,np.ndarray), default None

Two numpy arrays that contain indices for train/test slicing.

add_diff : bool, default False

Returns *DataFrame*

Examples

```
>>> pipeline = ModelsPipeline(model_rf,model_lr)
>>> pipeline.blend(seed=15)
```

```
>>> # Custom indices
>>> train_index = np.array(range(250))
>>> test_index = np.array(range(250,333))
>>> res = model_rf.blend(indicies=(train_index,test_index))
```

find_weights (*scorer*, *test_size=0.2*, *method='SLSQP'*)

Finds optimal weights for weighted average of models.

Parameters **scorer** : function

Scikit-learn like metric.

test_size : float, default 0.2

method : str

Type of solver. Should be one of:

- 'Nelder-Mead'
- 'Powell'
- 'CG'
- 'BFGS'
- 'Newton-CG'
- 'L-BFGS-B'
- 'TNC'
- 'COBYLA'
- 'SLSQP'
- 'dogleg'
- 'trust-ncg'

Returns list

gmean ()

Returns the gmean of the models predictions.

Returns *PipeApply*

max ()

Returns the max of the models predictions.

Returns *PipeApply*

mean()

Returns the mean of the models predictions.

Returns *PipeApply*

Examples

```
>>> # Execute
>>> pipeline = ModelsPipeline(model_rf,model_lr)
>>> pipeline.mean().execute()
```

```
>>> # Validate
>>> pipeline = ModelsPipeline(model_rf,model_lr)
>>> pipeline.mean().validate()
```

min()

Returns the min of the models predictions.

Returns *PipeApply*

stack(*k=5*, *stratify=False*, *shuffle=True*, *seed=100*, *full_test=True*, *add_diff=False*)

Stacks sequence of models.

Parameters **k** : int, default 5

Number of folds.

stratify : bool, default False

shuffle : bool, default True

seed : int, default 100

full_test : bool, default True

If True then evaluate test dataset on the full data otherwise take the mean of every fold.

add_diff : bool, default False

Returns *DataFrame*

Examples

```
>>> pipeline = ModelsPipeline(model_rf,model_lr)
>>> stack_ds = pipeline.stack(k=10, seed=111)
```

weight(*weights*)

Applies weighted mean to models.

Parameters **weights** : list

Returns np.ndarray

Examples

```
>>> pipeline = ModelsPipeline(model_rf,model_lr)
>>> pipeline.weight([0.8,0.2])
```

4.4 heamy.feature module

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index