Pied PyPler

Why packaging is important for both open and close data science projects
Motivation
How many times did you...

- Looked for an old piece of code
- Searched Dropbox & private repos
- Started Jupyter to search notebooks
- Gave up and rewritten it
How many times did you...

- Looked for an old piece of code
- Searched Dropbox & private repos
- Started Jupyter to search notebooks
- Gave up and rewritten it
How many times did you...

- Looked for an old piece of code
- Found it!
- Copied it over
- What does this line do?!
- Re-written half of it
How many times did you...

- Looked for an old piece of code
- Found it!
- Copied it over
- What does this line do?!
- Re-written half of it
The solution?
Packaging!
What’s in it for me?

- Write **less code** (over time)
- Code is **used more**
- All shall love you and **despair**
What’s in it for me?

- - -

- Write
- Code
- All

YOU
What’s in it for me?

- less code
- used more
- despair

YOU

ALSO, NO MATH!
You will learn:

- Benefits to your code
- Benefits to your colleagues
- Benefits to you
- Which DS code fits packaging
- Designing a simple Python package
You will learn:

- Benefits to your code
- Benefits to your colleagues
- **Benefits to you**
- What DS codes fits packaging
- Designing a simple Python package
Benefits
Benefits to your code

- Built-in incentive to:
  - Simplify & clarify
  - Document
  - Test
- Gains developer scrutiny
Benefits to your code

● Built-in incentive to:
  ○ Simplify & clarify
  ○ Document
  ○ Test
● Gains developer scrutiny

AKA,

IT WORKS...
Benefits to people

- Code used **more often**, in more places
- **Other people** fixing & improving
- Manage **technical debt**
- Easier to **deploy**
- Save others writing it
Benefits to people

- Code used **more often**, in more places
- **Other people** fixing & improving
- Manage **technical debt**
- Easier to **deploy**
- Save others writing it
Benefits to people

- Code used **more often**, in more places
- **Other people** fixing & improving
- Manage **technical debt**
- Easier to **deploy**
- Save others writing it
Benefits to people

- Code used **more often**, in more places
- **Other people** fixing & improving
- Manage **technical debt**
- Easier to **deploy**
- Save others writing it
What to package?
Types of code worth packaging

- Extensions to existing libraries
- Adapting existing code to a common API
- Automatizing your flow
- Data science infrastructure
- Specific technique implementation
Types of code worth packaging

- Extensions to existing libraries
- Adapting existing code to a common API
- Automatizing your flow
- Data science infrastructure
- Specific technique implementation
Additions to the `imbalance-learn` package.

```python
from imbulit.combine import MinMaxRandomSampler; from imblearn import pipeline;
# oversampling minority classes to 100 and undersampling majority classes to 800
sampler = MinMaxRandomSampler(min_freq=100, max_freq=800)
sampling_clf = pipeline.make_pipeline(sampler, inner_clf)
```

Contents

1. Installation
2. Basic Use
   2.1 combine
3. Contributing
   3.1 Installing for development
""""Randomly samples data to bring all class frequencies into a range.""""

```python
import numpy as np
from sklearn.utils import check_X_y

from imblearn.base import SamplerMixin
from imblearn.utils import check_target_type, hash_X_y
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

class MinMaxRandomSampler(SamplerMixin):
    """"Randomly samples data to bring all class frequencies into a range.

Parameters
```
class MinMaxRandomSampler(SamplerMixIn):

    """Random samples data to bring all class frequencies into a range.

    Parameters
    *********

    min_freq : int
        The minimum frequency for a class after sampling. All classes with fewer samples are over-sampled to have this number of samples.

    max_freq : int
        The maximum frequency for a class after sampling. All classes with more samples are under-sampled to have this number of samples.

    random_state : int, RandomState instance or None, optional (default=None)
        If int, random_state is the seed used by the random number generator;
        If RandomState instance, random_state is the random number generator;
        If None, the random number generator is the RandomState instance used by np.random.
Types of code worth packaging

- Extensions to existing libraries
- Adapting existing code to a common API
- Automatizing your flow
- Data science infrastructure
- Specific technique implementation
scikit-learn wrappers for Python fastText.

```python
>>> from skift import FirstColFtClassifier
>>> df = pandas.DataFrame([['woof', 0], ['meow', 1]], columns=['txt', 'lbl'])
>>> sk_clf = FirstColFtClassifier(lr=0.3, epoch=10)
>>> sk_clf.fit(df[['txt']], df['lbl'])
>>> sk_clf.predict([['woof']])
[0]
```
""""scikit-learn classifier wrapper for fasttext."""

import os
import abc

import numpy as np
from fastText import train_supervised
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils.multiclass import unique_labels
from sklearn.exceptions import NotFittedError

from .util import (temp_dataset_fpath,
dump_xy_to_fasttext_format,
python_fasttext_model_to_bytes,
bytes_to_python_fasttext_model,
)

class FtClassifierABC(BaseEstimator, ClassifierMixin, metaclass=abc.ABCMeta):
Return ndarrays instead of lists while predicting #2

Closed  uniaz opened this issue on Mar 13 · 4 comments

uniaz commented on Mar 13

The functions `predict`, `predict_proba` return lists instead of `numpy` arrays which makes them unusable with classifiers like `sklearn.multiclass.OneVsRestClassifier`, `GridSearch` and other similar functionality also don't work.

This is a quick fix.
@def predict(self, X):

    y : array of int of shape = [n_samples]
    Predicted labels for the given input samples.

    return [np.array([self._clean_label(res[0][0])
    for res in self._predict(X)]), dtype=np.float_]

@def _format_probas(self, result):

    lbl_prob_pairs = zip(result[0], result[1])

@def predict_proba(self, X):

    The class probabilities of the input samples. The order of the
classes corresponds to that in the attribute classes_.

    return [np.array([self._format_probas(res)
    for res in self._predict(X, self.num_classes_)])
    ], dtype=np.float_]
Types of code worth packaging

- Extensions to existing libraries
- Adapting existing code to a common API
- Automatizing your flow
- Data science infrastructure
- Specific technique implementation
Easy pipelines for pandas DataFrames.

```python
>>> df = pd.DataFrame(
      data=[[4, 165, 'USA'], [2, 180, 'UK'], [2, 170, 'Greece']],
      index=['Dana', 'Jane', 'Nick'],
      columns=['Medals', 'Height', 'Born']
    )
>>> pipeline = pdp.ColDrop('Medals').Binarize('Born')
>>> pipeline(df)
```

<table>
<thead>
<tr>
<th></th>
<th>Height</th>
<th>Born_UK</th>
<th>Born_USA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dana</td>
<td>165</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Jane</td>
<td>180</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Nick</td>
<td>170</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
3.2.2 Printing Pipelines

A pipeline structure can be clearly displayed by printing the object:

```python
>>> drop_name = pdp.ColDrop("Name")
>>> binar_label = pdp.Binarize("Label")
>>> map_job = pdp.MapColVals("Job", {"Part": True, "Full":True, "No": False})
>>> pipeline = pdp.PdPipeline([drop_name, binar_label, map_job])
>>> print(pipeline)
A pdpipe pipeline:
[ 0] Drop column Name
[ 1] Binarize Label
[ 2] Map values of column Job with {'Part': True, 'Full': True, 'No': False}.
```
Types of code worth packaging

- Extensions to existing libraries
- Adapting existing code to a common API
- Automatizing your flow
- Data science infrastructure
- Specific technique implementation
barn

Simple local/remote dataset store for Python.

```
from barn import Dataset
twitter_usa = Dataset(name='twitter_usa', task='NER')
# download from an azure block blob storage and load into a dataframe
twitter_usa.download(tags=['preprocessed'], version='20180305')
df = twitter_usa.df(tags=['preprocessed'], version='20180305')
```

Contents

- 1 Installation
- 2 Features
Types of code worth packaging

- Extensions to existing libraries
- Adapting existing code to a common API
- Automatizing your flow
- Data science infrastructure
- Specific technique implementation
Build text classification models

See tests/ folder for usage.

Word based models

When dataset represented as (docs, words) word based models can be created using

```python
from keras_text.models import TokenModelFactory
from keras_text.models import YoonKimCNN, AttentionRNN, StackedRNN

# RNN models can use `max_tokens=None` to indicate variable length words per mini-batch
factory = TokenModelFactory(1, tokenizer.token_index, max_tokens=100, embedding_type='default',
word_encoder_model = YoonKimCNN()
model = factory.build_model(token_encoder_model=word_encoder_model)
model.compile(optimizer='adam', loss='categorical_crossentropy')
model.summary()
```
Types of code worth packaging

- Extensions to existing libraries
- Adapting existing code to a common API
- Automatizing your flow
- Data science infrastructure
- Specific technique implementation
How to package?
Good practices

- Do one thing
- Generalize the use case
- Keep it simple
- Advertise professionalism
Do one thing

- NO: Data processing + model deployment + feature selection
- Don’t recreate a flow; rather, a slice.
  (horizontal, not vertical)
Generalize the use case

- Parameterize
- Split when X ways forward
  - If it’s not too much work...
3.1 Standard wrappers

These wrappers do not make additional assumptions on input besides those commonly made by scikit-learn classifies; i.e. that input is a 2d ndarray object and such.

- **FirstColFtClassifier** - An sklearn classifier adapter for fasttext that takes the first column of input ndarray objects as input.

- **IdxBasedFtClassifier** - An sklearn classifier adapter for fasttext that takes input by column index. This is set on object construction by providing the `input_ix` parameter to the constructor.

3.2 pandas-dependent wrappers

These wrappers assume the `X` parameter given to `fit`, `predict`, and `predict_proba` methods is a pandas.DataFrame object:

- **FirstObjFtClassifier** - An sklearn adapter for fasttext using the first column of `dtype == object` as input.

- **ColLblBasedFtClassifier** - An sklearn adapter for fasttext taking input by column label. This is set on object construction by providing the `input_col_lbl` parameter to the constructor.
@abc.abstractmethod
def _input_col(self, X):
    pass  # pragma: no cover

**FirstColFtClassifier**

class FirstColFtClassifier(FtClassifierABC):
    """An sklearn classifier adapter for fast Parameters
    """

**kwargs
    Additional keyword arguments will be
    fasttext.train_supervised.
    """

def _input_col(self, X):
    return np.array(X)[;, 0]

**FirstObjFtClassifier**

def _input_col(self, X):
    input_col_name = None
    for col_name, dtype in X.dtypes.items():
        if dtype == object:
            input_col_name = col_name
            break
    if input_col_name is not None:
        return X[input_col_name]
    raise ValueError("No object dtype column in input param X.")
Keep it simple

- Informative class/function names
- Minimal API
- A clear example with one/two imports
```python
from skift import FirstColFtClassifier

df = pandas.DataFrame([["woof", 0], ["meow", 1]], columns=['txt', 'lbl'])

sk_clf = FirstColFtClassifier(lr=0.3, epoch=10)
sk_clf.fit(df[['txt']], df[['lbl']])
sk_clf.predict([["woof"]])
```

```python
from skutil.estimators import ColumnIgnoringClassifier

# use a classifier that can't handle string data as an inner classifier in some stacked model, for example
```

```python
>>> df = pd.DataFrame(
    data=[[4, 165, 'USA'], [2, 180, 'UK'], [2, 170, 'Greece']],
    index=['Dana', 'Jane', 'Nick'],
    columns=['Medals', 'Height', 'Born'])

>>> pipeline = pdp.ColDrop('Medals'), Binarize('Born')

>>> pipeline(df)
```

<table>
<thead>
<tr>
<th></th>
<th>Height</th>
<th>Born_UK</th>
<th>Born_USA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dana</td>
<td>165</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Jane</td>
<td>180</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Nick</td>
<td>170</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Advertise professionalism

- Continuous testing
- Test coverage
- Permissive License
- Documentation
- Encourage contributions
Easy pipelines for pandas DataFrames.
Easy pipelines for pandas DataFrames.
pdpipe

Easy pipelines for pandas DataFrames.
6 Contributing

Package author and current maintainer is Shay Palachy (shay.palachy@gmail.com); You are more than welcome to approach him for help. Contributions are very welcomed, especially since this package is very much in its infancy and many other pipeline stages can be added.

6.1 Installing for development

Clone:

```
git clone git@github.com:shypal5/pdpipe.git
```

Install in development mode with test dependencies:

```
cd pdpipe
pip install -e ".[test]"
```

6.2 Running the tests

To run the tests, use:

```
python -m pytest --cov=pdpipe
```

6.3 Adding documentation

This project is documented using the numpy docstring conventions, which were chosen as they are perhaps the most widely-spread conventions that are both supported by common tools such as Sphinx and result in human-readable docstrings (in my personal opinion, of course). When documenting code you add to this project, please follow these conventions.
Good practices

- Do one thing
- Generalize the use case
- Keep it simple
- Advertise professionalism
What’s next?
Homework

- Go home
- Find code you can package
- Create & upload a first version
- **Email me** at shay.palachy@gmail.com
- I’ll help (if needed)

Example repo: https://github.com/shaypal5/catlolzer
Homework

- Go to your company
- Deploy a private PyPI server
- Create & upload a first version
- People catch up surprisingly fast
- Email me at shay.palachy@gmail.com
That’s it!

Email me at shay.palachy@gmail.com