Frontier Documentation

Release 0.1.3-dev

Sam Nicholls

August 14, 2014

Contents

Contents:

CHAPTER 1

Frontier

A Python package providing interfaces for the reading, storage and retrieval of large machine learning data sets for use with scikit-learn.

- Documentation: http://frontier.readthedocs.org.

1.1 Requirements

To use;

- numpy

To test;

- tox
- pytest

For coverage;

- nose
- python-coveralls

1.2 Installation

```
$ pip install frontier
```

CHAPTER 2

Installation

At the command line:

```
$ pip install frontier
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv frontier
$ pip install frontier
```

CHAPTER 3

Usage

3.1 Classes and Readers

3.1.1 Define Classes

Frontier is used for classification based machine learning problems, to be useful you must inform Frontier of the classes that define your problem.

For example, if you were to conduct an experiment to classify things that were and were not owls, you might construct a dictionary akin to:

```
CLASSES = {
        "hoot": {
            "names": ["owl", "owls"],
            "code": 1,
        },
        "unhoot": {
            "names": ["cat", "dog", "pancake"],
            "code": 0,
        },
}
```

Each key in the `CLASSES` dictionary names a particular class in your classification problem. The values to the `names` and `code` keys refer to a list of labels that belong to a particular class and the encoding used to refer to that class respectively.

3.1.2 Define Readers

Both data and targets are read in to Frontier via a `Reader` which you will often need to implement yourself by inheriting from `frontier.IO.AbstractReader`.

```
from frontier.IO.AbstractReader import AbstractReader

class MyReader(AbstractReader):

    def __init__(self, filepath, CLASSES=None, auto_close=True):
        """Initialise the structures for storing data and construct the reader."""
        self.myid = "something" # Could use os.path.basename(filepath)
        self.mydata = {} # Some structure for your data
        header_skip = 1 # Numer of initial lines to ignore
        super(MyReader, self).__init__(filepath, CLASSES, auto_close, header_skip)
```

```
    def process_line(self, line):
        """Process a line record in your file."""
        fields = line.split("\t")

        key = fields[0]
        value = fields[1]

        self.mydata[key] = value

    def get_id(self):
        """Interface to return record ID."""
        return self.myid

    def get_data(self):
        """Interface to return read data."""
        return self.mydata
```

The name of the structure used to hold data is irrelevant, just that it is returned sensibly by `get_data`. It is expected (and required) that data readers will return some unique identifier via `get_id` that corresponds to a key in the structure returned by `get_data` of your chosen target reader (thus linking a record of data to its target).

3.1.3 Import Readers

These readers can then be used to automatically process data and targets from single files or recursively through a directory. Currently it is expected that targets will be enumerated in one file and data will be stored in some directory structure.

```
from frontier import frontier
from myreaders import DataReader, TargetReader

data_dir = "/home/sam/Projects/owl_classifier/data/"
target_path = "/home/sam/Projects/owl_classifier/targets.txt"

statplexer = frontier.Statplexer(data_dir,
                                 target_path,
                                 CLASSES,
                                 DataReader,
                                 TargetReader)
```

The `frontier.frontier.Statplexer` class will then read in data and target inputs using the specified DataReader and TargetReader from the given paths.

The Statplexer can then be used to query the data and targets.

3.2 The Statplexer

3.2.1 Query Parameters or Features

`frontier.frontier.Statplexer.list_parameters()` Return a sorted list of all parameters

```
...
parameters = statplexer.list_parameters()
```

`frontier.frontier.Statplexer.find_parameters()` Given a list of input strings, return a list of parameters which contain any of those strings as a substring

frontier.frontier.Statplexer.exclude_parameters() Given a list of input strings, return a list of parameters which do not contain any of the input strings as a substring, or if needed an exact match

```
...
parameters = statplexer.exclude_parameters(["owl-ratio", "hoot"])
```

3.2.2 Retrieve Data and Target Pairs

frontier.frontier.Statplexer.get_data_by_parameters() Return data for each observation, but only include columns for each parameter in the given list

frontier.frontier.Statplexer.get_data_by_target() Return data for each observation that have been classified in one of the targets specified and additionally only return columns for the parameters in the given list

```
...
# Using the CLASSES above this would return data and targets for all data records
# classified with code 1 (ie. in the "hoot" classification), limited to just the
# "owl-ratio" and "hoot" parameters.
data, target, levels = statplexer.get_data_by_target(["owl-ratio", "hoot"], 1)
```

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/samstudio8/frontier/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/samstudio8/frontier/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 :)

4.2 Get Started!

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

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

   ```
   $ git clone git@github.com:your_name_here/frontier.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 frontier
   $ cd frontier/
   $ 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 frontier 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.

4.3 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 2.6, 2.7, and 3.3, 3.4, and for PyPy. Check https://travis-ci.org/samstudio8/frontier/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

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

CHAPTER 5

Credits

5.1 Development Lead

- Sam Nicholls <sam@samnicholls.net>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.3-dev

- *_test_variance* function of the *Statplexer* will now test the range of variance magnitudes across each parameter of read in data, issuing a warning and producing a table if a magnitude difference greater than ± 1 is discovered.

6.2 0.1.2 (2014-08-12)

- **Fix #2**
  - Add *get_id* to data readers to prevent cluttering parameter space.
  - Update *TestBamcheckReader.test_id_key* to use *get_id()* instead of *get_data()["_id"]*

6.3 0.1.1 (2014-06-30)

- Documentation now exists.
- Required data readers to specify an _id instead of forcing use of data file basename in the Statplexer.

6.4 0.1.0 (2014-06-28)

- First release on PyPI.

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*