Pulsar Feature Lab Version 1.0

R. J. Lyon [*]
School of Computer Science &
Jodrell Bank Centre for Astrophysics
University of Manchester
Kilburn Building
Oxford Road
Manchester M13 9PL

Thursday 18th June, 2015

Abstract

The PULSAR FEATURE LAB application is a collection of python scripts useful for extracting machine learning features (otherwise known as scores or variables) from pulsar candidate files. The code was written in order to provide a tool-kit useful for designing and extracting new candidate features, whilst retaining the ability to extract existing features developed by the community at large. This enables newly conceived features to be evaluated with respect to existing features allowing an objective decision on their utility to be reached. This document therefore describes i) how to use the PULSAR FEATURE LAB application to achieve this, ii) its basic system requirements and iii) its capabilities. It also provides a guide on how to update the PULSAR FEATURE LAB source code, making it easier for the pulsar search community to add and thereby share their own features, improving the techniques we also use to isolate signals of interest - be they periodic or single pulse.

It is hoped this code base will be used by the radio astronomy community. By sharing features and the source code implementations used to extract them, existing and newly devised features can be evaluated together. A statistically optimal feature set can then be produced which maximises the performance of learning algorithms on observational data. This will assist all in isolating legitimate pulsar/transient/single-pulse detections in data collected around the world. Given the proliferation of observational data and the increase in data volumes to be expected from next generation radio telescopes such as the Square Kilometre Array (SKA), such collaboration is important if we are to avoid the 'big data' problems associated with other large science projects such as the Atlas Experiment at the Large Hadron Collider (LHC).

[*] Email: robert.lyon@postgrad.manchester.ac.uk , Web: `www.scienceguyrob.com` .

Contents

1 Acknowledgements

Thank you to Dan Thornton and Sam Bates for providing their feature generation code, and for helping to review it for bugs. Thanks to Ben Stappers and Sally Cooper for spending many hours tuning and testing the features designed by Dan and Sam for PFD files (especially the curve fitting procedures!). Thanks to Scott Ransom for making available his PRESTO code via github, from which I gratefully borrowed the PFD reading routines. Thanks to Aristeidis Noutsos for the sub-band processing code, which I found hidden away in a source code file.

Development work was supported by grant EP/I028099/1 for the University of Manchester Centre for Doctoral Training in Computer Science, from the UK Engineering and Physical Sciences Research Council (EPSRC). Provided feature files were extracted from data obtained by the Parkes Observatory, funded by the Commonwealth of Australia and managed by the CSIRO, and by the LOFAR array funded by Astron in the Netherlands. We acknowledge help from the DRAGNET team, which is supported by ERC Starting Grant 337062 (PI Hessels).

2 Overview

Pulsar candidates are signal detections made be radio telescopes exhibiting characteristics reminiscent of pulsar emission (pulsars are a rare form of neutron star). The goal of candidate selection is to choose signal detections worthy of further investigation, and ignore those likely arising from noise or RFI. In order to build automated methods that undertake such decision making for us, candidates have to be described using a number of variables or **features**. Ideally these features will discriminate between pulsar and non-pulsar candidates reasonably well. These can then be used to train machine learning algorithms [1] to build highly accurate candidate classifiers that choose interesting candidates for us automatically. The pulsar search community has developed many such features [2, 3, 4, 5, 6, 7], and likely continues to develop more. The key question is which of these are better discriminators, and under what circumstances (i.e. in isolation or when combined). For example some features may work well for single pulse candidates, but be poor separators for periodic ones (and vice versa). Whilst it is possible that feature(s) developed by one group, may enhance the separability of features designed by another. In order to begin building a set of highly separable features as a community, we require a data independent method of generating and evaluating them. That way statistically informed conclusions on their utility can be reached, directly leading to improved candidate selection mechanisms.

The Pulsar Feature Lab code aims to provide an independent and extendible code base for feature **implementation and extraction only**. It enables the features designed by one group, to be generated by another, in a reproducible way with no implementation ambiguity.

2.1 Evaluation

Since the code described herein concerns feature extraction, how exactly are features to be evaluated in a concise and consistent way? Appropriate feature evaluation techniques were outlined in [7], and it is recommended that these be adopted by the community at large. These include:

- Computation of point-biserial correlation coefficients, which test for the presence of linear correlations.
- Information theoretic analysis using tools including the FEAST (`http://www.cs.man.ac.uk/~gbrown/fstoolbox`) and MITOOLBOX (`http://www.cs.man.ac.uk/~pococka4/MIToolbox.html`) toolboxes [8]. Such tools help test for non-linear correlations, and rank features according to how much information they contain.
- Recording classification results for multiple classifiers, appropriate use of evaluation metrics, cross-validation, and statistical significance testing. The WEKA tool is a good choice for undertaking repeatable experiments using multiple algorithms.

To summarise, **evaluation is to be undertaken externally**.

3 System Requirements

PULSAR FEATURE LAB scripts have the following system requirements:

- Python 2.4 or later.
- SciPy (`http://www.scipy.org/`)
- NumPy (`http://www.numpy.org/`)
- MATPLOTLIB library (`http://matplotlib.org/`)

4 Compatible Candidates

The code can extract features from three separate types of pulsar candidate file. These include:

1. The Pulsar Hunter Candidate XML (PHCX) candidates output by the pipeline described by Morello et al.[6].
2. The gnuzipped ('.gz') PHCX candidates produced by the pipeline described by Thornton [4].
3. The prepfold (PFD) files output by the PRESTO tool, and the LOTAAS survey pipeline.

5 Extractable Features

At present the code can extract the following features from the candidate files described above:

- The 22 features described by Thornton. [4].
- The 8 features described by Lyon et al. [7].
- Integrated (folded) profile data (intensity bin values, scaled to the range [0,255]).
- DM-SNR Curve data points (varying number of points per candidate).

The following features **cannot** be extracted:

- 12 features from Eatough et al. [2].
- 22 features from Bates et al. [3].
- 6 features from Lee et al. [5].
- 6 features from Morello et al. [6].

These have not be included as some of their implementation details are unclear, or are open to interpretation. It is hoped that the authors of these features will add them to the Pulsar Feature Lab code base over time.

6 Basic Usage

The application script PulsarFeatureLab.py can be executed via:

```
python PulsarFeatureLab.py
```

The script accepts a number of arguments. It requires four of these to execute, and accepts another three as optional.

6.1 Required Arguments

Flag	Type	Description
-c	integer	Candidate file type. 1. The Pulsar Hunter Candidate XML (PHCX) candidates output by the pipeline described by Morello et al.[6]. 2. The gnuzipped ('.gz') PHCX candidates produced by the pipeline described by Thornton [4]. 3. The prepfold (PFD) files output by the LOTAAS and similar surveys.
-d	string	Path to the directory containing candidates.
-f	string	Path to the output file to create or append to.
-t	integer	Type of features to generate. 1. 12 features from Eatough et al. [2]. 2. 22 features from Bates et al. [3]. 3. 22 features from Thornton. [4]. 4. 6 features from Lee et al. [5]. 5. 6 features from Morello et al. [6]. 6. 8 features from Lyon et al. [7]. 7. Integrated (folded) profile data. 8. DM-SNR Curve data.

Table 1: Required parameters.

6.2 Optional Arguments

Flag	Type	Description
−−arff	boolean	Flag that when provided, writes feature output in the WEKA ARFF file format.
−−meta	boolean	Flag that when provided, writes meta information, i.e. the candidate file name, to the output file.
-v	boolean	Verbose debugging flag.

Table 2: Optional parameters.

6.3 Output

The code can output feature data in two formats, either i) comma separated value (CSV) format, or ii) ARFF format. The CSV format is straightforward. For each candidate processed, a row of feature data will be written to the output file. Thus for five candidates, using the 8 features described in [7], output data will resemble the following:

Listing 1: Feature data output in CSV format.

```
8,5,2,1,1,2,3,1
6,5,3,1,1,1,4,2
6,2,3,1,1,2,3,1
8,5,2,1,1,2,3,1
5,3,3,1,1,1,5,3
```

If the −−meta flag is supplied by the user, then additional detail will be provided in this file. Specifically, the full path to the candidate, enabling each row of data to be linked to the candidate it was extracted from. For example the output above will appear as follows when the −−meta flag is used:

Listing 2: Feature data output in CSV format, with meta information.

```
8,5,2,1,1,2,3,1%/home/cands/candidate_1.phcx
6,5,3,1,1,1,4,2%/home/cands/candidate_2.phcx
6,2,3,1,1,2,3,1%/home/cands/candidate_3.phcx
8,5,2,1,1,2,3,1%/home/cands/candidate_4.phcx
5,3,3,1,1,1,5,3%/home/cands/candidate_5.phcx
```

Here the % sign has been used to signal the start of the meta information, and candidates are assumed to be stored in the /home/cands folder.

6.3.1 ARFF Format

ARFF is the standard file format used by the WEKA data mining tool (`http://www.cs.waikato.ac.nz/ml/weka/`). The Pulsar Feature Lab provides the option to write out candidate feature data in ARFF format, for direct use with WEKA and other tools. The format itself is simple. It includes a header that describes the data it contains, and requires that each row of features be associated with a class label. The feature data shown in Listing 1 describes 5 candidates, by 8 individual features. To represent this data in ARFF format we describe:

1. the file itself with the @relation tag.
2. each feature used with the @attribute tag.
3. the possible class labels, also via the @attribute tag.
4. where the data starts using the @data tag.

Thus the data would be represented as follows in ARFF format:

Listing 3: Feature data output in ARFF format.

```
@relation 5_candidates_described_by_8_features

@attribute Feature_1 numeric
@attribute Feature_2 numeric
@attribute Feature_3 numeric
@attribute Feature_4 numeric
@attribute Feature_5 numeric
@attribute Feature_6 numeric
@attribute Feature_7 numeric
@attribute Feature_8 numeric
@attribute class {0,1}

@data
8,5,2,1,1,2,3,1,? % This is a comment.
6,5,3,1,1,1,4,2,?
6,2,3,1,1,2,3,1,?
8,5,2,1,1,2,3,1,?
5,3,3,1,1,1,5,3,?
```

Note that an extra class label (?) has been added to each row of data. This indicates that the true class of each row is unknown. In other words we do not know if a row belongs to pulsar or non-pulsar candidate. Though if a candidate was known to be positive (i.e. pulsar) then on its row, ? would be replaced with 1. Likewise if a candidate was known to be negative (i.e. non-pulsar) then ? would be replaced by 0.

ARFF files are most useful when the true class (correct label) of each row of data is known. Thus it is advisable to process known pulsar, and known non-pulsar candidates separately using the Pulsar Feature Lab. Then it is possible to correctly label an output ARFF using simple text processing. Simply **find and replace all ?** values with the appropriate label. Then merge the labelled positive and negative data into a single ARFF file.

If the −−meta flag is used in combination with −−arff flag, then the output ARFF file will contain meta information. An example of this is shown below.

Listing 4: Feature data output in ARFF format with meta information.

```
@relation 5_candidates_described_by_8_features

@attribute Feature_1 numeric
@attribute Feature_2 numeric
@attribute Feature_3 numeric
@attribute Feature_4 numeric
@attribute Feature_5 numeric
@attribute Feature_6 numeric
@attribute Feature_7 numeric
@attribute Feature_8 numeric
@attribute class {0,1}

@data
8,5,2,1,1,2,3,1,? %/home/cands/candidate_1.phcx
6,5,3,1,1,1,4,2,? %/home/cands/candidate_2.phcx
6,2,3,1,1,2,3,1,? %/home/cands/candidate_3.phcx
8,5,2,1,1,2,3,1,? %/home/cands/candidate_4.phcx
5,3,3,1,1,1,5,3,? %/home/cands/candidate_5.phcx
```

7 Data Mining Tool Compatibility

There are numerous data mining tools you can use to process extracted feature data. These include:

- MatLab / GNU Octave.
- SciKit learn (`http://scikit-learn.org/stable/`).
- WEKA.
- MOA (`http://moa.cms.waikato.ac.nz/`).
- Many, many more!

These data mining tools generally require labelled data in order to build classifiers, and evaluate features. Thus to use the features extracted from the Pulsar Feature Lab, each row of features describing a single candidate must be labelled. Either with the positive (pulsar) label, or the negative (non-pulsar) label. Thus it makes sense to process known pulsar, and known non-pulsar candidates **separately**. This will give you two files: a positive and a negative feature file. It is then possible to label the positive and negative file with simple text processing procedures. It is also advisable to not leave meta information in files you intend to use with external data mining tools. MatLab in particular will not process the data since it contains text information.

7.1 Labelling CSV Output

To label CSV output assuming a positive and negative output file, do the following:

- **Negative File:** append ',0' to the end of each data row, thereby labelling the data negative. The label must occur before any meta information.
- **Positive File:** append ',1' to the end of each data row, thereby labelling the data positive. The label must occur before any meta information.

Finally merge the two CSV files in to a single file containing positive and negative data, usable by the tools mentioned above.

7.2 Labelling ARFF Output

To label ARFF output assuming a positive and negative output file, do the following:

- **Negative File:** replace '?' with '0' thereby labelling the data negative. The label must occur before any meta information.
- **Positive File:** replace '?' with '1' thereby labelling the data positive. The label must occur before any meta information.

Finally merge the two ARFF files in to a single file (just one header required) containing positive and negative data, usable by the tools mentioned above.

8 Data

8.1 Candidate Data

This distribution includes Thornton processed [4] pulsar candidates (HTRU 2. data), labelled and processed by Lyon et al. [7]. These can be found in the 'Data' folder of this distribution, separated according to their positive and negative labels.

The HTRU 1. candidates obtained by Morello et al. [6] can be downloaded from:

`http://astronomy.swin.edu.au/~vmorello/`.

LOTAAS 1. candidates however are not yet publicly available for study.

8.2 Feature Data

Dataset	Examples	Non-pulsars	Pulsars
HTRU 1.	91,192	89,995	1,196
HTRU 2.	17,898	16,259	1,639
LOTAAS 1.	5,053	4,987	66

Table 3: Data sets from which supplied feature data was extracted.

The ARFF and CSV data provided with this distribution (36 files in total), contain different features (and combinations of features) extracted from the candidate sets described in Table 3. Versions of these data sets, which have been 10 bin discretised using the WEKA data mining tool, are also provided since these are directly usable with feature selection toolboxes.

9 Source Modification

There are 8 scripts that make up the source distribution:

Name	Description
Candidate.py	Describes an individuate candidate.
DataProcessor.py	Searches for candidate files, and initiates feature generation.
FeatureExtractor.py	Contains feature implementations shared by both PHCX and PFD files.
PFDFeatureExtractor.py	Contains feature implementations specific to PFD files.
PFDFile.py	Reads in and represents an individual PFD file.
PHCXFeatureExtractor.py	Contains feature implementations specific to PHCX files.
PHCXFile.py	Reads in and represents an individual PHCX file.
PulsarFeatureLab.py	Main class that processing command line arguments, and begins execution.
Utilities.py	Contains utility methods common to all files (i.e. file write etc).

Table 4: Source files in the distribution.

The code is designed to be as easy to use as possible, for those who have limited programming experience. Object orientated inheritance is used by most classes stored in the scripts. For example the 'FeatureExtractor.py' script contains feature implementation methods common to all types of candidate. The 'PFDFeatureExtractor.py' and 'PHCXFeatureExtractor.py' scripts then extend 'FeatureExtractor.py', and declare file specific feature implementation/extraction details. This means that these files can/will become larger as more feature extraction methods are added. However this is somewhat preferable, for the time being, than having many additional files for each individual feature.

9.1 Adding New Features

To add new features there are a number of specific places in the source code that need to be modified. Such changes are fiddly. But I decided it best to use this approach, since more advanced design patterns will likely be unfamiliar to most working in this domain, and make the code difficult to understand, and therefore unusable!

To add a new feature, the code needs to be modified in a number of places. These have been highlighted in the source using the following text, which you can search for using a text editor with 'find' functionality.

```
# ADDING A NEW FEATURE?
```

The changes which need to be made are as follows (please add comments to the code as appropriate):

1. PulsarFeatureLab.py - modify the command line processing code to accept an additional feature set. Currently there are 8 feature sets, thus yours will be numbered 9 (or higher). Changes must be made on line 167 onwards.
2. PulsarFeatureLab.py - modify the part which writes out the ARFF header, so that it produces a header consistent with your features. Line 245 onwards.
3. PFDFile.py and PHCXFile.py - modify the:
   def computeFeatures(self,feature_type)
   function to incorporate a call to your new feature generation code. Also add an additional function,
   def computeType_<your feature type> ()
   that calculates and extracts your feature values from FeatureExtractor.py, PFDFeatureExtractor.py, and PHCXFeatureExtractor.py.
4. FeatureExtractor.py - Implement the feature extraction code (called above) which works for both PFD and PHCX files.
5. PFDFeatureExtractor.py - Implement the feature extraction code which works only for PFD files.
6. PHCXFeatureExtractor.py- Implement the feature extraction code which works only for PHCX.

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, 2006.

[2] R. P. Eatough, N. Molkenthin, M. Kramer, A. Noutsos, M. J. Keith, B. W. Stappers, and A. Lyne, "Selection of radio pulsar candidates using artificial neural networks", *Monthly Notices of the Royal Astronomical Society*, vol. 407, no. 4, pp. 2443–2450, 2010.

[3] S. D. Bates, M. Bailes, B. R. Barsdell, N. D. R. Bhat, M. Burgay, S. Burke-Spolaor, D. J. Champion, P. Coster, N. D'Amico, A. Jameson, S. Johnston, M. J. Keith, M. Kramer, L. Levin, A. Lyne, S. Milia, C. Ng, C. Nietner, A. Possenti, B. Stappers, D. Thornton, and W. van Straten, "The high time resolution universe pulsar survey vi. an artificial neural network and timing of 75 pulsars", *Monthly Notices of the Royal Astronomical Society*, vol. 427, no. 2, pp. 1052–1065, 2012.

[4] D. Thornton, *The High Time Resolution Radio Sky.* PhD thesis, University of Manchester, Jodrell Bank Centre for Astrophysics School of Physics and Astronomy, 2013.

[5] K. J. Lee, K. Stovall, F. A. Jenet, J. Martinez, L. P. Dartez, A. Mata, G. Lunsford, S. Cohen, C. M. Biwer, M. Rohr, J. Flanigan, A. Walker, S. Banaszak, B. Allen, E. D. Barr, N. D. R. Bhat, S. Bogdanov, A. Brazier, F. Camilo, D. J. Champion, S. Chatterjee, J. Cordes, F. Crawford, J. Deneva, G. Desvignes, R. D. Ferdman, P. Freire, J. W. T. Hessels, R. Karuppusamy, V. M. Kaspi, B. Knispel, M. Kramer, P. Lazarus, R. Lynch, A. Lyne, M. McLaughlin, S. Ransom, P. Scholz, X. Siemens, L. Spitler, I. Stairs, M. Tan, J. van Leeuwen, and W. W. Zhu, "PEACE: pulsar evaluation algorithm for candidate extraction a software package for post-analysis processing of pulsar survey candidates", *Monthly Notices of the Royal Astronomical Society*, vol. 433, no. 1, pp. 688–694, 2013.

[6] V. Morello, E. D. Barr, M. Bailes, C. M. Flynn, E. F. Keane, and W. van Straten, "SPINN: a straightforward machine learning solution to the pulsar candidate selection problem", *Monthly Notices of the Royal Astronomical Society*, vol. 443, no. 2, pp. 1651–1662, 2014.

[7] R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles, "Machine Learning & Candidate Pulsar Selection", *Monthly Notices of the Royal Astronomical Society*, vol. 000, no. 0, pp. 0000–0000, 2015.

[8] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, "Conditional likelihood maximisation: A unifying framework for information theoretic feature selection", *Journal of Machine Learning Research*, vol. 13, pp. 27–66, 2012.