Bayes Logistic Regression Documentation

Release 0.1.0

Rob Haslinger

June 24, 2016

Contents

1 Bayes Logistic Regression **3**
1.1 Demo . . . 3

2 Installation **5**

3 Usage **7**
3.1 Methods . . . 7

4 Contributing **9**
4.1 Types of Contributions . . . 9
4.2 Get Started! . . . 10
4.3 Pull Request Guidelines . . . 10
4.4 Tips . . . 11

5 Credits **13**
5.1 Development Lead . . . 13
5.2 Contributors . . . 13

6 History **15**

7 0.2.0 (2015-09-02) **17**

8 Indices and tables **19**

Contents:

CHAPTER 1

Bayes Logistic Regression

This package will fit Bayesian logistic regression models with arbitrary prior means and covariance matrices, although we work with the inverse covariance matrix which is the log-likelihood Hessian.

Either the full Hessian or a diagonal approximation may be used.

Individual data points may be weighted in an arbitrary manner.

Finally, p-values on each fitted parameter may be calculated and this can be used for variable selection of sparse models.

- Free software (BSD):
- Documentation: https://bayes_logistic.readthedocs.org.
- See related presentation video here.

1.1 Demo

Example Notebook

CHAPTER 2

Installation

At the command line:

```
$ easy_install bayes_logistic
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv bayes_logistic
$ pip install bayes_logistic
```

CHAPTER 3

Usage

To use Bayes Logistic Regression in a project:

```
import bayes_logistic
```

3.1 Methods

bayes_logistic.**bayes_logistic_prob**(*X*, *w*, *H*)

Posterior predictive logistic regression probability. Uses probit approximation to the logistic regression sigmoid. Also has overflow prevention via exponent truncation.

X [array-like, shape (N, p)] array of covariates

w [array-like, shape (p,)] array of fitted MAP parameters

H [array-like, shape (p, p) or (p,)] array of log posterior Hessian (covariance matrix of fitted MAP parameters)

pr [array-like, shape (N,)] moderated (by full distribution) logistic probability

Chapter 8 of Murphy, K. 'Machine Learning a Probabilistic Perspective', MIT Press (2012) Chapter 4 of Bishop, C. 'Pattern Recognition and Machine Learning', Springer (2006)

bayes_logistic.**get_pvalues**(*w*, *H*)

Calculates p-values on fitted parameters. This can be used for variable selection by, for example, discarding every parameter with a p-value less than 0.05 (or some other cutoff)

w [array-like, shape (p,)] array of posterior means on the fitted parameters

H [array-like, shape (p, p) or (p,)] array of log posterior Hessian

pvals [array-like, shape (p,)] array of p-values for each of the fitted parameters

Chapter 2 of Pawitan, Y. 'In All Likelihood', Oxford University Press (2013) Also see: Gerhard, F. 'Extraction of network topology from multi-electrode recordings: is there a small world effect', Frontiers in Computational Neuroscience (2011) for a use case of p-value based variable selection.

bayes_logistic.**fit_bayes_logistic**(*y*, *X*, *wprior*, *H*, *weights=None*, *solver='Newton-CG'*, *bounds=None*, *maxiter=100*)

Bayesian Logistic Regression Solver. Assumes Laplace (Gaussian) Approximation to the posterior of the fitted parameter vector. Uses scipy.optimize.minimize

y [array-like, shape (N,)] array of binary {0,1} responses

X [array-like, shape (N, p)] array of features

wprior [array-like, shape (p,)] array of prior means on the parameters to be fit

H [array-like, shape (p, p) or (p,)] array of prior Hessian (inverse covariance of prior distribution of parameters)

weights [array-like, shape (N,)] array of data point weights. Should be within [0,1]

solver [string] scipy optimize solver used. this should be either 'Newton-CG', 'BFGS' or 'L-BFGS-B'. The default is Newton-CG.

bounds [iterable of length p] a length p list (or tuple) of tuples each of length 2. This is only used if the solver is set to 'L-BFGS-B'. In that case, a tuple (lower_bound, upper_bound), both floats, is defined for each parameter. See the scipy.optimize.minimize docs for further information.

maxiter [int] maximum number of iterations for scipy.optimize.minimize solver.

w_fit [array-like, shape (p,)] posterior parameters (MAP estimate)

H_fit [array-like, shape like *H*] posterior Hessian (Hessian of negative log posterior evaluated at MAP parameters)

Chapter 8 of Murphy, K. 'Machine Learning a Probabilistic Perspective', MIT Press (2012) Chapter 4 of Bishop, C. 'Pattern Recognition and Machine Learning', Springer (2006)

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/maxpoint/bayes_logistic/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

Bayes Logistic Regression could always use more documentation, whether as part of the official Bayes Logistic Regression 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/maxpoint/bayes_logistic/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 *bayes_logistic* for local development.

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

```
$ git clone git@github.com:your_name_here/bayes_logistic.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 bayes_logistic
$ cd bayes_logistic/
$ 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 bayes_logistic 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, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/MaxPoint/bayes_logistic/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_bayes_logistic
```

CHAPTER 5

Credits

5.1 Development Lead

- Rob Haslinger <rob.haslinger@maxpoint.com>

5.2 Contributors

- Marius van Niekerk

CHAPTER 6

History

CHAPTER 7

0.2.0 (2015-09-02)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

B

bayes_logistic_prob() (in module bayes_logistic), 7

F

fit_bayes_logistic() (in module bayes_logistic), 7

G

get_pvalues() (in module bayes_logistic), 7