Open-Source Software's Responsibility to Science

Joel Nothman

20th July 2018

THE UNIVERSITY OF SYDNEY

Me in open source

- Mostly contributed to popular Scientific Python libraries: scikit-learn, nltk, scipy.sparse, pandas, ipython, numpydoc
- Also information extraction evaluation (neleval), etc.

- Community service
- "Volunteer software development"
- With thanks to our financial sponsors

- Caretakers aren't always founders
- Founders aren't always caretakers

Overheard at ICML

Don't worry about how tricky it is to implement . . .

Someone will put it in Scikit-learn and you can just use it.

Thoughts on an arrogant ML researcher

- Scientists think software maintenance is no big deal

Thoughts on an arrogant ML researcher

- Scientists think software maintenance is no big deal

- Science and engineering rely heavily on open-source infrastructure
- Popular tools become de-facto standards

- Most users are uncomfortable building their own tools
- Many will only use what's provided in a popular library
- Many will not inspect how it works on the inside

- Volunteer maintainers act as gatekeepers

The power of the gatekeeper

- decides which algorithms are available
- decides how to ensure correctness and stability
- decides how to name or describe the algorithm
- decides whether to be faithful to a published description
- decides on an API that may facilitate good science/engineering

The power of the gatekeeper

- decides which algorithms are available
- decides how to ensure correctness and stability
- decides how to name or describe the algorithm
- decides whether to be faithful to a published description
- decides on an API that may facilitate good science/engineering

OSS maintainers can enable or inhibit scientific best practices

But you can't blame the gatekeeper

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS **"AS IS"** AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

This presentation is a series of examples

- Risks to good science and engineering related to software design

- Some things we do to help science
- Some things we have changed to help science
- Some things we have yet to solve

- There's not a great deal of NLP in here
  but there's a lot of ML and software engineering in NLP
  (so I hope it is relevant, interesting and accessible)

Scikit-learn preliminaries

- An ecosystem of estimators
- Fit an estimator on some data, so that it can:
  - describe the training data
  - transform unseen data
  - predict a target for unseen data
- Data is usually a numeric matrix `X` (samples $\times$ features)
- May provide a target vector or matrix `y` at training time
  - real valued for regression
  - categories for multiclass classification
  - multiple columns of binary targets for multilabel classification

Methods and results and are indecipherable if researchers publish an inappropriate, or underspecified, algorithm name

A simple example of a bad name

- `sklearn.covariance.GraphLasso` for sparse inverse covariance estimation
- but *Graph Lasso* is sparse *regression* where the features lie on a graph
- the paper for covariance estimation named it *Graphical Lasso*

A simple example of a bad name

- `sklearn.covariance.GraphLasso` for sparse inverse covariance estimation
- but *Graph Lasso* is sparse *regression* where the features lie on a graph
- the paper for covariance estimation named it *Graphical Lasso*

Solution deprecate `GraphLasso` and rename it `GraphicalLasso`

Tripping over hidden parameters

- `precision_score(['a','a','b','b','c'], ['a','b','b','c','c'])`
- For multi-class or multi-label, how should you average across classes?
  - $P_a = \frac{1}{1}$, $P_b = \frac{1}{2}$, $P_c = \frac{1}{2}$
  - average='micro' $\Rightarrow \frac{3}{5}$
  - average='macro' $\Rightarrow (1 + \frac{1}{2} + \frac{1}{2})/3 = \frac{2}{3}$
  - average='weighted' $\Rightarrow (2 \times 1 + 2 \times \frac{1}{2} + 1 \times \frac{1}{2})/5 = \frac{7}{10}$
- for a long time, prevalence-weighted macro average was the default

  $\therefore$ papers say "We achieved a precision of ..."

Tripping over hidden parameters

- `precision_score(['a','a','b','b','c'], ['a','b','b','c','c'])`
- For multi-class or multi-label, how should you average across classes?
  - $P_a = \frac{1}{1}$, $P_b = \frac{1}{2}$, $P_c = \frac{1}{2}$
  - average='micro' $\Rightarrow \frac{3}{5}$
  - average='macro' $\Rightarrow (1 + \frac{1}{2} + \frac{1}{2})/3 = \frac{2}{3}$
  - average='weighted' $\Rightarrow (2 \times 1 + 2 \times \frac{1}{2} + 1 \times \frac{1}{2})/5 = \frac{7}{10}$
- for a long time, prevalence-weighted macro average was the default

  $\therefore$ papers say "We achieved a precision of ..."

Solution `precision_score` raises an error if the data is not binary, unless the user specifies `average`.

Use the API to force literacy & awareness

What's in a name?

- What makes an implementation of some named algorithm correct?

- Faithful to a published research paper?
- Faithful to a reference implementation?
- Faithful to *some* community of practice?
- Consistent with other components of our software library?
- Consistent with previous versions of the library?

Experimenters report sub-optimal results
because they assume our implementation is nicely behaved

The fit you thought was finished

- Many optimisations are iterative
- have criteria to test if it has converged on an optimum
- Predictions and inferences may be poor if parameters did not converge

The fit you thought was finished

- Many optimisations are iterative
- have criteria to test if it has converged on an optimum
- Predictions and inferences may be poor if parameters did not converge

Solution Warn if we did not detect convergence
but if we have too many warnings, users ignore them. . .

The words you didn't mean to stop

- `CountVectorizer` turns text into a term-document matrix
- can choose stop words: None, 'english' or BYO
- 'english' will remove *system* (and used to remove *computer*)
- 'english' will remove *five*, *six*, *eight* but not *seven*
- 'english' will remove *we have* but treat *we've* as *ve*
- This is documented nowhere.
- See my NLP-OSS paper with Hanmin Qin and Roman Yurchak

The words you didn't mean to stop

- `CountVectorizer` turns text into a term-document matrix
- can choose stop words: None, 'english' or BYO
- 'english' will remove *system* (and used to remove *computer*)
- 'english' will remove *five*, *six*, *eight* but not *seven*
- 'english' will remove *we have* but treat *we've* as *ve*
- This is documented nowhere.
- See my NLP-OSS paper with Hanmin Qin and Roman Yurchak

Solution Deprecate 'english'

The words you didn't mean to stop

- `CountVectorizer` turns text into a term-document matrix
- can choose stop words: None, 'english' or BYO
- 'english' will remove *system* (and used to remove *computer*)
- 'english' will remove *five*, *six*, *eight* but not *seven*
- 'english' will remove *we have* but treat *we've* as *ve*
- This is documented nowhere.
- See my NLP-OSS paper with Hanmin Qin and Roman Yurchak

Solution Deprecate 'english' and add another perfect stop list . . .

The intercept you didn't mean to regularise

- In Logistic Regression, we learn a weight vector β
- and a bias term β_0 which corresponds to a feature $\mathbf{x}_0$ of all-1s
- Regularisation: minimise $\sqrt{\sum_i \beta_i^2}$ to ensure small weights as well as small loss
- liblinear regularises β_0. You probably never want to do this.
- All our other linear estimators do not regularise the intercept.

The intercept you didn't mean to regularise

- In Logistic Regression, we learn a weight vector β
- and a bias term β_0 which corresponds to a feature $\mathbf{x}_0$ of all-1s
- Regularisation: minimise $\sqrt{\sum_i \beta_i^2}$ to ensure small weights as well as small loss
- liblinear regularises β_0. You probably never want to do this.
- All our other linear estimators do not regularise the intercept.

Sol'n 1 `intercept_scaling`: also need to optimise $\mathbf{x}_0$'s fill value

The intercept you didn't mean to regularise

- In Logistic Regression, we learn a weight vector β
- and a bias term β_0 which corresponds to a feature $\mathbf{x}_0$ of all-1s
- Regularisation: minimise $\sqrt{\sum_i \beta_i^2}$ to ensure small weights as well as small loss
- liblinear regularises β_0. You probably never want to do this.
- All our other linear estimators do not regularise the intercept.

Sol'n 1 `intercept_scaling`: also need to optimise $\mathbf{x}_0$'s fill value
...but most users don't see/do this

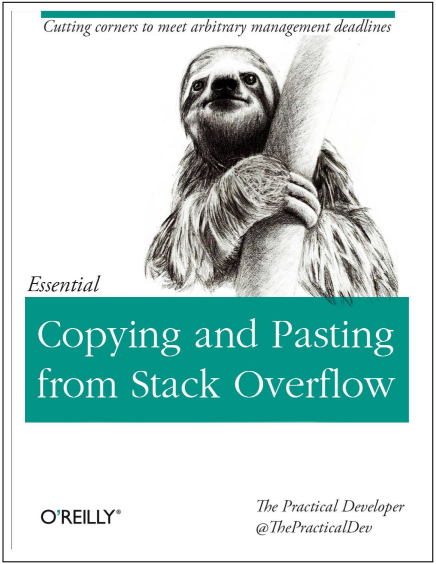
Sol'n 2 Implement alternative optimisers, and deprecate liblinear as default LogisticRegression solver

Analysis of code on GitHub shows that people use default parameters when they shouldn't

Andreas Müller

Most users are lazy

- Users don't explore alternatives
  - alternative parameters values
  - alternative software libraries
- My students tend to use a `CountVectorizer` even when they're counting non-words (e.g. synsets)
- We try to provide sensible default parameters



Sensible default fail

- Ten-tree forests
- Three-fold cross validation
- ?? A tokeniser that splits on word-internal punctuation

What makes a default value good?

- Good defaults should give good predictive models and reliable statistics
- ? Good defaults should behave how users expect
  - but different communities of practice
- Good defaults should be invariant to:
  - sample size (for stability in cross validation)
  - number of features (for stability in model selection)
  - ?feature scaling (for stability in different tasks/datasets)
- Example: finding a good default γ for an RBF kernel (#779, #10331)

Good parametrisation

- We choose the defaults, but also how parameters are expressed
- (and whether they can be changed at all)

- Should number of nearest neighbors be specified as:
  - an absolute value (e.g. 10)?
  - a proportion of training samples (e.g. 2%)?
  - an arbitrary function of training data shape?

- Algorithms and optimisation research often don't report on this

Scientific software should make it easy for users to do good science.

Have you ever tried to de-tokenise the Penn TreeBank?

- PTB is delivered with each token POS tagged or bracketed
- We wanted to know how paragraph structure, etc. informed parsing
- The source text before tokenisation is available
- An easy case:

```
 [ Imports/NNS ]
were/VBD at/IN
[ $/$ 50.38/CD billion/CD ]
,/, up/RB
[ 19/CD %/NN ]
./.
```

Imports were at $50.38 billion, up 19%.

Imports	1–7
were	9–12
at	14–15
$	16–16
50.38	17–21

Have you ever tried to de-tokenise the Penn TreeBank?

- PTB is delivered with each token POS tagged or bracketed
- We wanted to know how paragraph structure, etc. informed parsing
- The source text before tokenisation is available

$\Rightarrow$ alignment hell due to typos corrected/inserted, reorderings, missed text, etc.

- Hindsight: delivering annotations on tokenised text is a bad idea
- It restricts what you can do with it later
- NLP software should always provide stand-off markup
- Or store the whitespace/non-token data as spaCy does

Avoiding leakage in cross validation

Bad

```
X_preprocessed = preprocessor.fit_transform(X)
result = cross_validate(classifier, X_preprocessed, y)
```

Test data statistics leak into preprocessing
$\Rightarrow$ inflated cross validation results

Good

```
pipeline = make_pipeline(preprocessor, classifier)
result = cross_validate(pipeline, X, y)
```

Avoiding leakage in cross validation

Bad

```
X_preprocessed = preprocessor.fit_transform(X)
result = cross_validate(classifier, X_preprocessed, y)
```

Test data statistics leak into preprocessing
$\Rightarrow$ inflated cross validation results

Good

```
pipeline = make_pipeline(preprocessor, classifier)
result = cross_validate(pipeline, X, y)
```

Solution De-emphasise `fit_transform`.
And make sure `Pipeline` works with everything;
and make sure `cross_validate` works with everything.

Maintainers of large projects
can't be experts in *all* the things they maintain

Scientists can (and do) help us:

- make sure the implementation matches the name
- make users aware of or avoid unexpected behaviour
- parametrise algorithms and set defaults helpfully
- understand how our design choices lead to flawed experiments

Users trust popular OSS.
Thank you for helping us make OSS trustworthy.