ExpAn Documentation

Release 0.5.0

Zalando SE

Apr 05, 2017
8 Indices and tables
Contents:
A/B tests (a.k.a. Randomized Controlled Trials or Experiments) have been widely applied in different industries to optimize business processes and user experience. ExpAn (Experiment Analysis) is a Python library developed for the statistical analysis of such experiments and to standardise the data structures used.

The data structures and functionality of ExpAn are generic such that they can be used by both data scientists optimizing a user interface and biologists running wet-lab experiments. The library is also standalone and can be imported and used from within other projects and from the command line.

Major statistical functionalities include:

- feature check
- delta
- subgroup analysis
- trend

**Installation**

To install ExpAn, run this command in your terminal:

```
$ pip install expan
```

**Usage**

To use ExpAn in a project:

```
import expan
```

Some mock-up data:
from expan.core.experiment import Experiment
from tests.tests_core.test_data import generate_random_data

exp = Experiment('B', *generate_random_data())
exp.delta()

Documentation

The latest stable version is 0.5.0.

ExpAn main documentation

ExpAn Description - details about the concept of the library and data structures.

ExpAn Introduction - a full jupyter (iPython) notebook. You can view it as slides with jupyter:

```
sh serve_intro_slides
```

License

The MIT License (MIT)

Copyright © [2016] Zalando SE, https://tech.zalando.com

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
CHAPTER 2

Installation

Stable release

To install ExpAn, run this command in your terminal:

$ pip install expan

This is the preferred method to install ExpAn, as it will always install the most recent stable release. If you don’t have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for ExpAn can be downloaded from the Github repo.

You can either clone the public repository:

$ git clone git://github.com/zalando/expan

Or download the tarball:

$ curl -OL https://github.com/zalando/expan/tarball/master

Once you have a copy of the source, you can install it with:

$ python setup.py install
To use ExpAn in a project:

```python
import expan

Some mock-up data

```from expan.core.experiment import Experiment
from tests.tests_core.test_data import generate_random_data

exp = Experiment('B', *generate_random_data())
exp.delta()

Per-entity ratio vs. ratio of totals

There are two different definitions of a ratio metric (think of e.g. conversion rate, which is the ratio between the number of orders and the number of visits): 1) one that is based on the entity level or 2) ratio between the total sums, and ExpAn supports both of them.

In a nutshell, one can reweight the individual per-entity ratio to calculate the ratio of totals. This enables to use the existing statistics.delta() function to calculate both ratio statistics (either using normal assumption or bootstrapping).

Calculating the conversion rate

As an example let’s look at how to calculate the conversion rate, which might be typically defined per-entity as the average ratio between the number of orders and the number of visits:

\[
\overline{CR}^{(pe)} = \frac{1}{n} \sum_{i=1}^{n} CR_i = \frac{1}{n} \sum_{i=1}^{n} \frac{O_i}{V_i}
\]
The ratio of totals is a reweighted version of $CR_i$ to reflect not the entities’ contributions (e.g. contribution per customer) but overall equal contributions to the conversion rate, which can be formulated as:

$$CR^{(rt)} = \frac{\sum_{i=1}^{n} O_i}{\sum_{i=1}^{n} V_i}$$

**Overall as reweighted Individual**

One can calculate the $CR^{(rt)}$ from the $CR^{(pe)}$ using the following weighting factor (easily proved by paper and pencile):

$$CR^{(rt)} = \frac{1}{n} \sum_{i=1}^{n} \frac{O_i}{V_i}$$

with

$$\alpha_i = \frac{n}{n} \frac{V_i}{\sum_{i=1}^{n} V_i}$$

**Weighted delta function**

To have such functionality as a more generic approach in ExpAn, we can introduce a *weighted delta* function. Its input are

- The per-entity metric, e.g. $O_i/V_i$
- A reference metric, on which the weighting factor is based, e.g. $V_i$

With this input it calculates $\alpha$ as described above and outputs the result of `statistics.delta()`.
Contributing

Style guide

We follow PEP8 standards with the following exceptions:

- Use tabs instead of spaces - this allows all individuals to have visual depth of indentation they prefer, without changing the source code at all, and it is simply smaller

Testing

Easiest way to run tests is by running the command tox from the terminal. The default Python environments for testing with are py27 and py34, but you can specify your own by running e.g. tox -e py35.

Branching / Release

We currently use the gitflow workflow. Feature branches are created from and merged back to the dev branch, and the master branch stores snapshots/releases of the dev branch.

See also the much simpler github flow here

Versioning

For the sake of reproducibility, always be sure to work with a release when doing the analysis!

We use semantic versioning (http://semver.org), and the current version of ExpAn is: v0.4.0.

The version is maintained in setup.cfg, and propagated from there to various files by the bumpversion program. The most important propagation destination is in version.py where it is held in the string __version__ with the form:
The __version__ string and a version() function is imported by core.__init__ and so is accessible to imported functions in expan.

The version(format_str) function generates version strings of any form. It can use git’s commit count and revision number to generate a long version string which may be useful for pip versioning? Examples: NB: caution using this... it won’t work if not in the original git repository.

```python
>>> import core.binning
>>> core.version()
'v0.4.0'
>>> core.version('{major}.{minor}..{commits}')
'0.0..176'
>>> core.version('{commit}')
a24730a42a4b5ae01bbdb05f6556dedd453c1767'
```

See: StackExchange 151558

**Bumping Version**

Can use bumpversion to maintain the __version__ in version.py:

```
$ bumpversion patch
```

or

```
$ bumpversion minor
```

This will update the version number, create a new tag in git, and commit the changes with a standard commit message.

When you have done this, you must push the commit and new tag to the repository with:

```
$ git push --tags
```

**Travis CI and PyPI deployment**

We use Travis CI for testing builds and deploying our PyPI package.

A build and test is triggered when a commit is pushed to either

- dev,
- master

- or a pull request branch to dev or master.

If you want to deploy to PyPI, then follow these steps:

- assuming you have a dev branch that is up to date, create a pull request from dev to master (a travis job will be started for the pull request)
- once the pull request is approved, merge it (another travis job will be started because a push to master happened)
- checkout master
• push tags to master (a third travis job will be started, but this time it will also push to PyPI because tags were pushed)

If you wish to skip triggering a CI task (for example when you change documentation), please include [ci skip] in your commit message.

**TODOs**

• parallelization, eg. for the bootstrapping code

• Bayesian updating/early stopping

• multiple comparison correction, definitely relevant for delta and SGA, have to think about how to correct for time dependency in the trend analysis

• implement from_json and to_json methods in the Binning class, in order to convert the Python object to a json format for persisting in the Results metadata and reloading from a script
CHAPTER 5

Related Projects

There may be alternative libraries providing similar functionality, and these should be collected here. Very incomplete list so far...

- **abba** ([https://github.com/thumbtack/abba](https://github.com/thumbtack/abba))
  - Mainly handles binomial distributions.

- **bootstrapped** ([https://github.com/facebookincubator/bootstrapped](https://github.com/facebookincubator/bootstrapped))
  - Calculates bootstrapped confidence intervals, with A/B test as an example.
CHAPTER 6

Credits

Development Lead

• Octopus McSquid <team-octopus@zalando.de>

Contributors

• Grigory Bordyugov <grigory.bordyugov@zalando.de>
• Dominic Heger <dominic.heger@zalando.de>
• Jie Bao <jie.bao@zalando.de>
• Marko Kolarek <marko.kolarek@zalando.de>
• Robert Muil <robert.muil@zalando.de>
• Till Riffert <till.riffert@zalando.de>
v0.4.5 (2017-02-10)

Full Changelog

Fixed bugs:
- Numbers cannot appear in variable names for derived metrics #58

Merged pull requests:
- Feature/results and to json refactor #74 (mkolarek)
- Merge to_json() and prob_uplift_over_zero changes #72 (mkolarek)
- regex fix, see https://github.com/zalando/expan/issues/58 #70 (gbordyugov)

v0.4.4 (2017-02-09)

Full Changelog

Implemented enhancements:
- Add argument assume_normal and treatment_cost to calculate_prob_uplift_over_zero() and prob_uplift_over_zero_single_metric() #26
- host intro slides (from the ipython notebook) somewhere for public viewing #10

Closed issues:
- migrate issues from github enterprise #20

Merged pull requests:
- Feature/results and to json refactor #71 (mkolarek)
- new to_json() functionality and improved vim support #67 (mkolarek)
v0.4.3 (2017-02-07)

Full Changelog

Closed issues:

• coverage % is misleading #23

Merged pull requests:

• Vim modelines #63 (gbordyugov)
• Feature/octo 1253 expan results in json #62 (mkolarek)
• 0.4.2 release #60 (mkolarek)

v0.4.2 (2016-12-08)

Full Changelog

Fixed bugs:

• frequency table in the chi square test doesn’t respect the order of categories #56

Merged pull requests:

• OCTO-1143 Review outlier filtering #59 (domheger)
• Workaround to fix #56 #57 (jbao)

v0.4.1 (2016-10-18)

Full Changelog

Merged pull requests:

• small doc cleanup #55 (jbao)
• Add comments to cli.py #54 (igusher)

v0.4.0 (2016-08-19)

Full Changelog

Closed issues:

• Support ‘overall ratio’ metrics (e.g. conversion rate/return rate) as opposed to per-entity ratios #44

v0.3.4 (2016-08-08)

Full Changelog

Closed issues:

• perform trend analysis cumulatively #31
• Python3 #21

v0.3.3 (2016-08-02)
Full Changelog

v0.3.2 (2016-08-02)
Full Changelog

v0.3.1 (2016-07-15)
Full Changelog

v0.3.0 (2016-06-23)
Full Changelog
**Implemented enhancements:**
• Add P(\text{uplift}>0) as a statistic #2

v0.2.5 (2016-05-30)
Full Changelog
**Implemented enhancements:**
• Implement \texttt{\_version\_} #14
**Closed issues:**
• upload full documentation! #1

v0.2.4 (2016-05-16)
Full Changelog
**Closed issues:**
• No module named experiment and test_data #13

v0.2.3 (2016-05-06)
Full Changelog
v0.2.2 (2016-05-06)

Full Changelog

v0.2.1 (2016-05-06)

Full Changelog

v0.2.0 (2016-05-06)

* This Change Log was automatically generated by `github_changelog_generator`
  
  <https://github.com/skywinder/Github-Changelog-Generator>`
  
* This Change Log was automatically generated by `github_changelog_generator`
  
  <https://github.com/skywinder/Github-Changelog-Generator>`
CHAPTER 8

Indices and tables

- genindex
- modindex
- search