ExpAn Documentation

Release 1.4.0

Zalando SE

Sep 20, 2019

Contents

Contents:

CHAPTER 1

ExpAn: Experiment Analysis

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 (**Exp**eriment **An**alysis) 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.

1.1 Documentation

The latest stable version is 1.4.0. Please check out our tutorial and documentation.

1.2 Installation

1.2.1 Stable release

To install ExpAn, run this command in your terminal:

```
$ pip install expan
```

1.2.2 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
```

1.3 License

CHAPTER 2

Tutorial

Here is a tutorial to use ExpAn. Let's get started!

2.1 Generate demo data

First, let's generate some random data for the tutorial.

```
from expan.core.util import generate_random_data
data, metadata = generate_random_data()
```

`data` is a pandas DataFrame. It must contain a column for entity identifier named **entity**, a column for variant, and one column per kpi/feature.

`metadata` is a python dict. It should contain the following keys:

- `experiment`: Name of the experiment, as known to stakeholders. It can be anything meaningful to you.
- `sources` (optional): Names of the data sources used in the preparation of this data.
- `experiment_id` (optional): This uniquely identifies the experiment. Could be a concatenation of the experiment name and the experiment start timestamp.
- `retrieval_time` (optional): Time that data was fetched from original sources.
- `primary_KPI` (optional): Primary evaluation criteria.

Currently, `metadata` is only used for including more information about the experiment, and is not taken into consideration for analysis.

2.2 Create an experiment

To use ExpAn for analysis, you first need to create an `Experiment` object.

```
from expan.core.experiment import Experiment
exp = Experiment(metadata=metadata)
```

This `Experiment` object has the following parameters:

- `metadata`: Specifies an experiment name as the mandatory and data source as the optional fields. Described above.

2.3 Create a statistical test

Now we need a `StatisticalTest` object to represent what statistical test to run. Each statistical test consist of a dataset, one kpi, treatment and control variant names, and the optional features. Dataset should contain necessary kpis, variants and features columns.

```
from expan.core.statistical_test import KPI, Variants, StatisticalTest

kpi = KPI('normal_same')
variants = Variants(variant_column_name='variant', control_name='B', treatment_name='A
↪')
test = StatisticalTest(data=data, kpi=kpi, features=[], variants=variants)
```

2.4 Let's start analyzing!

Running an analysis is very simple:

```
exp.analyze_statistical_test(test)
```

Currently `analyze_statistical_test` supports 4 test methods: `fixed_horizon` (default), `group_sequential`, `bayes_factor` and `bayes_precision`. All methods requires different additional parameters.

If you would like to change any of the default values, just pass them as parameters to delta. For example:

```
exp.analyze_statistical_test(test, test_method='fixed_horizon', assume_normal=True,␣
↪percentiles=[2.5, 97.5])
exp.analyze_statistical_test(test, test_method='group_sequential', estimated_sample_
↪size=1000)
exp.analyze_statistical_test(test, test_method='bayes_factor', distribution='normal')
```

Here is the list of additional parameters. You may also find the description in our API page.

fixed_horizon is the default method:

- `assume_normal=True`: Specifies whether normal distribution assumptions can be made. A t-test is performed under normal assumption. We use bootstrapping otherwise. Bootstrapping takes considerably longer time than assuming the normality before running experiment. If we do not have an explicit reason to use it, it is almost always better to leave it off.
- `alpha=0.05`: Type-I error rate.
- `min_observations=20`: Minimum number of observations needed.
- `nruns=10000`: Only used if assume normal is false.

- `relative=False`: If relative==True, then the values will be returned as distances below and above the mean, respectively, rather than the absolute values.

group_sequential is a frequentist approach for early stopping:

- `spending_function='obrien_fleming'`: Currently we support only Obrient-Fleming alpha spending function for the frequentist early stopping decision.
- `estimated_sample_size=None`: Sample size to be achieved towards the end of experiment. In other words, the actual size of data should be always smaller than estimated_sample_size.
- `alpha=0.05`: Type-I error rate.
- `cap=8`: Upper bound of the adapted z-score.

bayes_factor is a Bayesian approach for delta analysis and early stopping:

- `distribution='normal'`: The name of the KPI distribution model, which assumes a Stan model file with the same name exists. Currently we support *normal* and *poisson* models.
- `num_iters=25000`: Number of iterations of bayes sampling.
- `inference=sampling`: 'sampling' for MCMC sampling method or 'variational' for variational inference method to approximate the posterior distribution.

bayes_precision is another Bayesian approach similar as *bayes_factor*:

- `distribution='normal'`: The name of the KPI distribution model, which assumes a Stan model file with the same name exists. Currently we support *normal* and *poisson* models.
- `num_iters=25000`: Number of iterations of bayes sampling.
- `posterior_width=0.08`: The stopping criterion, threshold of the posterior width.
- `inference=sampling`: 'sampling' for MCMC sampling method or 'variational' for variational inference method to approximate the posterior distribution.

2.5 Interpreting result

The output of the `analyze_statistical_test` method is an instance of class `core.result.StatisticalTestResult`. Please refer to the API page for result structure as well as descriptions of all fields. An example of the result is shown below:


```
{
    "result": {
        "confidence_interval": [
        {
            "percentile": 2.5,
            "value": 0.1
        },
        {
            "percentile": 97.5,
            "value": 1.1
        }],
        "control_statistics": {
            "mean": 0.0,
            "sample_size": 1000,
            "variance": 1.0
        },
        "delta": 1.0,
```


(continues on next page)

(continued from previous page)

```
        "p": 0.04,
        "statistical_power": 0.8,
        "treatment_statistics": {
            "mean": 1.0,
            "sample_size": 1200,
            "variance": 1.0
        }
    },
    "test": {
        "features": [],
        "kpi": {
            "name": "revenue"
        },
        "variants": {
            "control_name": "control",
            "treatment_name": "treatment",
            "variant_column_name": "variant"
        }
    }
}
```

2.6 Subgroup analysis

Subgroup analysis in ExaAn will select subgroup (which is a segment of data) based on the input argument, and then perform a regular delta analysis per subgroup as described before. That is to say, we don't compare between subgroups, but compare treatment with control within each subgroup.

If you wish to perform the test on a specific subgroup, you can use the `FeatureFilter` object:

```
feature = FeatureFilter('feature', 'has')
test = StatisticalTest(data=data, kpi=kpi, features=[feature], variants=variants)
```

2.7 Statistical test suite

It is very common to run a suite of statistical tests. In this case, you need to create a `StatisticalTestSuite` object to represent the test suite. A `StatisticalTestSuite` object consists of a list of `StatisticalTest` and a correction method:

```
from expan.core.statistical_test import *

kpi = KPI('normal_same')
variants = Variants(variant_column_name='variant', control_name='B', treatment_name='A
↪')

feature_1 = FeatureFilter('feature', 'has')
feature_2 = FeatureFilter('feature', 'non')
feature_3 = FeatureFilter('feature', 'feature that only has one data point')

test_subgroup1 = StatisticalTest(data, kpi, [feature_1], variants)
test_subgroup2 = StatisticalTest(data, kpi, [feature_2], variants)
test_subgroup3 = StatisticalTest(data, kpi, [feature_3], variants)
```

(continues on next page)

(continued from previous page)

```
tests = [test_subgroup1, test_subgroup2, test_subgroup3]
test_suite = StatisticalTestSuite(tests=tests, correction_method=CorrectionMethod.BH)
```

And then you can use the `Experiment` instance to run the test suite. Method `analyze_statistical_test_suite` has the same arguments as `analyze_statistical_test`. For example:

```
exp.analyze_statistical_test_suite(test_suite)
exp.analyze_statistical_test_suite(test_suite, test_method='group_sequential',␣
↪estimated_sample_size=1000)
exp.analyze_statistical_test_suite(test_suite, test_method='bayes_factor',␣
↪distribution='normal')
```

2.8 Result of statistical test suite

The output of the `analyze_statistical_test_suite` method is an instance of class `core.result.MultipleTestSuiteResult`. Please refer to the API page for result structure as well as descriptions of all fields.

Following is an example of the analysis result of statistical test suite:


```
{
    "correction_method": "BH",
    "results": [
        {
            "test": {
                "features": [
                    {
                        "column_name": "device_type",
                        "column_value": "desktop"
                    }
                ],
                "kpi": {
                    "name": "revenue"
                },
                "variants": {
                    "control_name": "control",
                    "treatment_name": "treatment",
                    "variant_column_name": "variant"
                }
            },
            "result": {
                "corrected_test_statistics": {
                    "confidence_interval": [
                        {
                            "percentile": 1.0,
                            "value": -0.7
                        },
                        {
                            "percentile": 99.0,
                            "value": 0.7
                        }
                    ],
```


(continues on next page)

(continued from previous page)


```
                    "control_statistics": {
                        "mean": 0.0,
                        "sample_size": 1000,
                        "variance": 1.0
                    },
                    "delta": 1.0,
                    "p": 0.02,
                    "statistical_power": 0.8,
                    "treatment_statistics": {
                        "mean": 1.0,
                        "sample_size": 1200,
                        "variance": 1.0
                    }
                },
                "original_test_statistics": {
                    "confidence_interval": [
                        {
                            "percentile": 2.5,
                            "value": 0.1
                        },
                        {
                            "percentile": 97.5,
                            "value": 1.1
                        }
                    ],
                    "control_statistics": {
                        "mean": 0.0,
                        "sample_size": 1000,
                        "variance": 1.0
                    },
                    "delta": 1.0,
                    "p": 0.04,
                    "statistical_power": 0.8,
                    "treatment_statistics": {
                        "mean": 1.0,
                        "sample_size": 1200,
                        "variance": 1.0
                    }
                }
            }
        },
        {
            "test": {
                "features": [
                    {
                        "column_name": "device_type",
                        "column_value": "mobile"
                    }
                ],
                "kpi": {
                    "name": "revenue"
                },
                "variants": {
                    "control_name": "control",
                    "treatment_name": "treatment",
                    "variant_column_name": "variant"
                }
```


(continues on next page)

(continued from previous page)


```
            },
            "result": {
                "corrected_test_statistics": {
                    "confidence_interval": [
                        {
                            "percentile": 1.0,
                            "value": -0.7
                        },
                        {
                            "percentile": 99.0,
                            "value": 0.7
                        }
                    ],
                    "control_statistics": {
                        "mean": 0.0,
                        "sample_size": 1000,
                        "variance": 1.0
                    },
                    "delta": 1.0,
                    "p": 0.02,
                    "statistical_power": 0.8,
                    "stop": false,
                    "treatment_statistics": {
                        "mean": 1.0,
                        "sample_size": 1200,
                        "variance": 1.0
                    }
                },
                "original_test_statistics": {
                    "confidence_interval": [
                        {
                            "percentile": 2.5,
                            "value": 0.1
                        },
                        {
                            "percentile": 97.5,
                            "value": 1.1
                        }
                    ],
                    "control_statistics": {
                        "mean": 0.0,
                        "sample_size": 1000,
                        "variance": 1.0
                    },
                    "delta": 1.0,
                    "p": 0.04,
                    "statistical_power": 0.8,
                    "stop": true,
                    "treatment_statistics": {
                        "mean": 1.0,
                        "sample_size": 1200,
                        "variance": 1.0
                    }
                }
            }
        }
    ]
```


(continues on next page)

(continued from previous page)

```
}
```

That's it!

For API list and theoretical concepts, please read the next sections.

CHAPTER 3

API

3.1 Architecture

`core.experiment` is the most important module to use ExpAn. It provides interface for running different analysis.

`core.statistics` provides the underlying statistical functions. Functionalities in this module includes **bootstrap**, **delta**, **pooled standard deviation**, **power analysis**, etc.

`core.early_stopping` provides early stopping algorithms. It supports **group sequential**, **Bayes factor** and **Bayes precision**.

`core.correction` implements methods for multiple testing correction.

`core.statistical_test` holds structures of statistical tests. You will need the data structure in this module to run an experiment.

`core.results` holds structures of analysis result. This will be the running structure of an experiment.

`core.util` contains supplied common functions used by other modules such as **generate random data** and **drop nan values**, among many others.

`core.version` constructs versioning of the package.

`data.csv_fetcher` reads the raw data and constructs an experiment instance.

`core.binning` is now DEPRECATED. It implements categorical and numerical **binning algorithms**. It supports binning implementations which can be applied to unseen data as well.

3.2 API

Please visit the API list for detailed usage.

CHAPTER 4

Glossary

4.1 Assumptions used in analysis

1. Sample-size estimation
   - Treatment does not affect variance
   - Variance in treatment and control is identical
   - Mean of delta is normally distributed
2. Equal or unequal sample sizes, equal variance t-test
   - Mean of means is t-distributed (or normally distributed)
   - Variance of two distributions are same (so the variance of two groups of sample should be similar)
3. In general
   - Sample represents underlying population
   - Entities are independent

4.2 Derived KPIs, such as conversion rate

For each user, we have their number of orders and their number of sessions. We estimate the orders-per-session ("conversion rate") by computing the total number of orders across all users and divide that by the total number of sessions across all users. Equivalently, we can use the ratio of the means:

$$\overline{CR} = \text{estimated conversion rate} = \frac{\sum_{i=1}^{n} o_i}{\sum_{i=1}^{n} s_i} = \frac{\frac{1}{n}\sum_{i=1}^{n} o_i}{\frac{1}{n}\sum_{i=1}^{n} s_i} = \frac{\bar{o}}{\bar{s}}$$

As a side comment, you might be tempted to compute the ratio for each individual, $\frac{o_i}{s_i}$, and compute the mean of those ratios, $\overline{\left(\frac{o}{s}\right)_i}$. The problem with this is that it's an estimator with low accuracy; more formally, its variance is large.

Intuitively, we want to compute a mean by giving greater weight to ratios which have more sessions; this is how we derive the formula for $\overline{CR}$ above.

To calculate the variance of this estimate, and therefore apply a t-test, we need to compute the variance of this estimator. If we used the same data again, but randomly reassigned every user to a group (treatment or control), and recomputed $\overline{CR}$ many times, how would this estimate vary?

We model that the s_i are given (i.e. non-random), and the o_i are random variables whose distribution is a function of s_i.

For each user, the "error" (think linear regression) is:

$$e_i = o_i - s_i \cdot \overline{CR}$$

The subtracted portion $(-s_i \cdot \overline{CR})$ is essentially non-random for our purposes, allowing us to say - to a very good approximation - that $Var[o_i] = Var[e_i]$. Also, the **e** vector will have mean zero by construction.

Therefore, as input to the pooled variance calculation, we use this as the variance estimate:

$$\hat{Var}\left[\frac{o_i}{\bar{s}}\right] = \hat{Var}\left[\frac{e_i}{\bar{s}}\right] = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{e_i - \bar{e}}{\bar{s}}\right)^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{e_i}{\bar{s}}\right)^2$$

The variances are calculated as above for both the control and the treatment and fed into a pooled variance calculation as usual for a t-test.

See the test named `test_using_lots_of_AA_tests()` within `expan/tests/test_derived.py` for a demonstration of how this method gives a uniform p-value under the null; this confirms that the correct error rate is maintained.

Finally, this method doesn't suffer from the problem described in this blog post. In our notation, o_i is the sum of the orders for all session for user i. The method criticized in that blog post is to compute the variance estimate across every session, i.e. ignoring o_i and instead using the per-session orders individually. That is problematic because it ignores the fact that the sessions for a given user may be correlated with each other. Our approach is different and follows the linear regression procedure closely, and therefore is more robust to these issues.

4.3 Early stopping

Given samples x from treatment group, samples y from control group, we want to know whether there is a significant difference between the means $\delta = \mu(y)\mu(x)$. To save the cost of long-running experiments, we want to stop the test early if we are already certain that there is a statistically significant result.

You can find links to our detailed documentations for concept of early stopping and early stopping methods we investigated.

4.4 Subgroup analysis

Subgroup analysis in ExpAn will select subgroup (which is a segment of data) based on the input argument, and then perform a regular delta analysis per subgroup as described before.

That is to say, we don't compare between subgroups, but compare treatment with control within each subgroup.

To support automatic detection of those interesting subgroups, also known as Heterogeneous Treatment Effect, is under planning.

4.5 Multiple testing problem

Multiple testing problem occurs when one considers a set of statistical inferences simultaneously. Consider a set of 20 hypothesis that you wish to test at the significance level of 0.05. What is the probability of observing at least one significant result just due to chance?

$\Pr(\text{at least one significant result}) = 1 - \Pr(\text{no significant results}) = 1 - (1 - 0.05)^{20} \approx 0.64$

With 20 tests being considered, we have a 64% chance of observing at least one significant result, even if all of the tests are actually not significant. Methods for dealing with multiple testing frequently call for adjusting α in some way, so that the probability of observing at least one significant result due to chance remains below your desired significance level.

ExpAn allows you to control the correction method for your set of statistical tests (statistical test suite) yourself. There are three options for the correction method:

- **CorrectionMethod.BONFERRONI**- strict Bonferroni correction which controls the family-wise error rate.
- **CorrectionMethod.BH** - correction by Benjamini-Hochberg: less strict and more powerful correction method which decreases the false discovery rate.
- **CorrectionMethod.NONE** - no correction is used. Even this option is available in ExpAn we strongly recommend to do not neglect the importance of correction for multiple testing and always correct for multiple testing using Benjamini-Hochberg correction, as a default one (as currently set up in ExpAn).

Correction is performed per each statistical test suite, but you can use the correction methods separately by calling `benjamini_hochberg(false_discovery_rate, original_p_values)` or `bonferroni(false_positive_rate, original_p_values)` providing corresponding p-values for the correction.

Read more about each correction method:

- Benjamini-Hochberg or original paper "Hochberg, Y., and A. C. Tamhane. Multiple Comparison Procedures."
- Bonferroni

4.6 Chi-square test (Multinomial Goodness of Fit Test).

In ExpAn we have the possibility to conduct multinomial goodness of fit test (chi-square test). The test is applied when you have one categorical variable from a single population. It is used to determine whether sample data are consistent with a hypothesized distribution (allocation of traffic or split percentage).

This test, in our case, is used to check the variant split based on the claimed percentage. For example, we want 50% of the users to be exposed to control variant (for example, green checkout button) and 50% of the users to be exposed to treatment variant (for example, yellow checkout button). We conduct a random assignment of variants and would like to check whether the random assignment did the right job and we've got the correct split of the variants. We would also like to know whether the variant split consistent with the specified percentage after the outlier filtering as well.

The Ho is: the data are consistent with a specified distribution (or the variant split corresponds to the expected percentage) The Ha is: the data are not consistent with a specified distribution (or the variant split do not correspond to the expected percentage) Typically, the null hypothesis (Ho) specifies the proportion of observations at each level of the categorical variable. The alternative hypothesis (Ha) is that at least one of the specified proportions is not true.

Multinomial goodness of fit test is described with one intuitive formula:

$$\chi^2_{K-1} = \sum_{i=1}^{K} \frac{(O_i - E_i)^2}{E_i}$$

Here O denotes the observed number of users buckets in bucket i, and E denotes the expected number of users bucketed in each bucket. K - overall number of buckets. The statistics capture how much each bucket deviates from the expected value, and the summation captures the overall deviation. Source

We use 0.05 significance level as the default one. We compute p-value - the probability of observing a sample statistics as extreme as the test statistic - and compare it to the significance level. We reject the null hypothesis when the p-value is less than the significance level.

We can use this test to check the correct split for the subgroups as well.

- Multiple testing problem for chi-square testing

Since chi-square testing is also a hypothesis testing, you need to keep in mind that multiple chi-square testing brings the problem of increasing of false positives rate described in the previous section. Let say, you want to test the correctness of your variants split 5 times at different times with 0.05 alpha. For 5 tests your alpha is no longer 0.05, but $1 - (1 - 0.05)^5 \approx 0.23$. Correction for multiple chi-square testing is needed here. In this case, you can run several chi-square tests and collect p-values, orrect p-values with one of our correction methods (CorrectionMethod.BONFERRONI, CorrectionMethod.BH) to get new corrected alpha, and make a decision about correctness of the variants splits using that new alpha.

CHAPTER 5

Change Log

5.1 v1.4.0 (2019-07-05)

Full Changelog

Closed issues:

- Why features are iterated in a for loop? #243

Merged pull requests:

- Two-sided outlier filtering mode #249 (gbordyugov)
- Create .zappr.yaml #248 (perploug)
- Correct documentation #245 (shansfolder)

5.2 v1.3.9 (2018-09-10)

Full Changelog

5.3 v1.3.8 (2018-09-10)

Full Changelog

Merged pull requests:

- Changed chi-square test, removed frequencies computation #240 (daryadedik)

5.4 v1.3.7 (2018-09-04)

Full Changelog

Merged pull requests:

- Added observed and expected frequencies to chi-square statistics #239 (daryadedik)
- Stop warning about NaNs #238 (aaron-mcdaid-zalando)
- stop overriding the warning level #237 (aaron-mcdaid-zalando)

5.5 v1.3.6 (2018-08-06)

Full Changelog

Merged pull requests:

- If the number of *finite* samples is too small, then the data isn't valid for analysis #236 (aaron-mcdaid-zalando)

5.6 v1.3.5 (2018-08-02)

Full Changelog

Merged pull requests:

- Re-wrote chi-square, removed dropping buckets. #234 (daryadedik)
- check if there are no p values to correct #233 (gbordyugov)

5.7 v1.3.3 (2018-07-26)

Full Changelog

Merged pull requests:

- Chi-squared test for the variant split check #228 (daryadedik)

5.8 v1.3.2 (2018-07-23)

Full Changelog

Merged pull requests:

- Fix minor typos #231 (sdia)
- custom deepcopy() method for 'StatisticalTest', to save some memory #230 (aaron-mcdaid-zalando)
- Updated old multiple correction documentation #229 (daryadedik)
- contributing.rst: Improve the release procedure to ensure that the up… #226 (aaron-mcdaid-zalando)

5.9 v1.3.1 (2018-07-01)

Full Changelog

5.10 v1.2.6 (2018-07-01)

Full Changelog

Merged pull requests:

- Ensure that outlier detection works if there is NaN in the data #225 (aaron-mcdaid-zalando)
- More powerful derived kpis #222 (aaron-mcdaid-zalando)

5.11 v1.2.5 (2018-06-22)

Full Changelog

Merged pull requests:

- Counting bugfix and save memory #224 (aaron-mcdaid-zalando)
- Fix for the possibility that both variances are zero #221 (aaron-mcdaid-zalando)

5.12 v1.2.4 (2018-05-31)

Full Changelog

Merged pull requests:

- Remove null analysis results from the analysis results files #219 (daryadedik)

5.13 v1.2.3 (2018-05-30)

Full Changelog

Merged pull requests:

- Removed deep copy of the data in statistical test construction #218 (daryadedik)

5.14 v1.2.2 (2018-05-30)

Full Changelog

Merged pull requests:

- Fixing bugs and adding more logging #217 (daryadedik)

5.15 v1.2.1 (2018-05-29)

Full Changelog

Merged pull requests:

- Added merge_with class method for merging two multiple test suite results and tests #216 (daryadedik)
- List of filtered columns as filtered_columns metadata information #215 (daryadedik)

5.16 v1.2.0 (2018-05-25)

Full Changelog

Merged pull requests:

- Update outlier filter on derived kpis #214 (shansfolder)

5.17 v1.1.0 (2018-05-24)

Full Changelog

Merged pull requests:

- Experiment data restructure #213 (daryadedik)
- Original corrected results #212 (daryadedik)

5.18 v1.0.1 (2018-04-23)

Full Changelog

Merged pull requests:

- Fixed docstring #211 (daryadedik)
- raise ValueError on zero pooled std for power calculations #210 (gbordyugov)
- Changed structure for statistics without correction #209 (daryadedik)

5.19 v1.0.0 (2018-03-22)

Full Changelog

Merged pull requests:

- Finish Documentation #204 (shansfolder)
- Fix logging sga error logging #203 (igusher)
- Project Headache #194 (shansfolder)

5.20 v0.6.13 (2018-03-15)

Full Changelog

Implemented enhancements:

- Applying bins to data frames #165

Fixed bugs:

- Sample size with an unequal split ratio #187
- SGA Percentile Issue #178

Merged pull requests:

- Wrap sga in try catch #202 (igusher)
- Multiple correction method module #201 (shansfolder)
- Adapted util module and util unit tests #199 (daryadedik)
- Adapt early stopping #198 (daryadedik)
- Adapt statistics.py #197 (shansfolder)
- Adapt experiment module #196 (shansfolder)
- Make result classes JSON serializable #195 (shansfolder)
- Results data structure #193 (shansfolder)
- fixed small typos in percentiles and doc text #191 (daryadedik)
- fixing sample size estimation #188 (gbordyugov)

5.21 v0.6.12 (2018-01-24)

Full Changelog

Merged pull requests:

- Doc update #186 (shansfolder)
- AXO-103 include variance in delta / group-sequential reports #185 (gbordyugov)

5.22 v0.6.11 (2018-01-23)

Full Changelog

Merged pull requests:

- Axo-91 bug fix sga #184 (shansfolder)
- added code coverage badge and reformatted README.rst a bit #183 (mkolarek)

5.23 v0.6.10 (2018-01-12)

Full Changelog

5.24 v0.6.9 (2018-01-12)

Full Changelog

Merged pull requests:

- Update deployment flow #182 (shansfolder)

5.25 v0.6.8 (2018-01-12)

Full Changelog

5.26 v0.6.7 (2018-01-10)

Full Changelog

Closed issues:

- Group Sequential - Percentile Issue #176

Merged pull requests:

- Increase version to 0.6.7 #181 (shansfolder)
- fixed last command in "Deploying to PyPI" part of contributing.rst #180 (mkolarek)
- Extended multiple correction for group sequential, added doc for multiple correction. #179 (daryadedik)
- Fix information fraction calculation #177 (shansfolder)

5.27 v0.6.6 (2017-11-27)

Full Changelog

Closed issues:

- Infinitely large confidence intervals produced by group_sequential_delta() #172

Merged pull requests:

- Merging dev to master for new release #175 (mkolarek)
- AXO-35 implemented estimate_sample_size() for estimating sample size … #174 (mkolarek)
- Fix two-sided alpha value in power analysis #173 (shansfolder)
- Docs/update contrib doc #171 (mkolarek)
- Add some parameter checks #170 (shansfolder)
- Make applying bins to data frames more agreeable #169 (gbordyugov)
- OCTO-2181: Implement over time analysis. Time-based SGA #164 (daryadedik)

5.28 v0.6.5 (2017-10-24)

Full Changelog

Merged pull requests:

- updated version #168 (mkolarek)
- Bump version: 0.6.3 → 0.6.4 #167 (mkolarek)
- bump version to v0.6.3 #166 (mkolarek)

5.29 v0.6.3 (2017-10-24)

Full Changelog

Merged pull requests:

- OCTO-2214 Bugfix: Capping information fraction #163 (shansfolder)
- OCTO-2088: Implement multiple testing correction in ExpAn #161 (daryadedik)
- OCTO-1044 Improve readthedoc #160 (shansfolder)
- OCTO-1933 Subgroup analysis #159 (shansfolder)
- release 0.6.2 #156 (mkolarek)
- OCTO-1920, OCTO-1968, OCTO-1969 Refactor binning #155 (shansfolder)

5.30 v0.6.2 (2017-08-29)

Full Changelog

Fixed bugs:

- Result statistics in Baeysian methods #142

Closed issues:

- Default Parameters of Constructor of Experiment class #151
- Update to ExpAn-Intro.ipynb #141

Merged pull requests:

- make development requirements open ended #154 (mkolarek)
- Octo 1930 implement quantile filtering #153 (mkolarek)
- Not use empty list for method parameter #152 (shansfolder)
- OCTO-1971 Add variational inference for early stopping #150 (shansfolder)
- Updated intro documentation covering delta methods. #149 (daryadedik)
- Release v0.6.1 #148 (shansfolder)
- Merge pull request #137 from zalando/dev #147 (shansfolder)
- Add static html file from intro doc for v0.6.1 #146 (shansfolder)

5.31 v0.6.1 (2017-08-08)

Full Changelog

Implemented enhancements:

- Optimizing the control flow from `Experiment` to `Results` #82
- more meaningful dict keys for results #139 (gbordyugov)

Fixed bugs:

- reenable means and bounds functions on Results object #9

Closed issues:

- Results.to_json() implementation not flexible #65
- Results.to_json() doesn't support trend() results #64

Merged pull requests:

- Documentation updates for Expan 0.6.x. Covers OCTO-1961, OCTO-1970 #145 (daryadedik)
- Fix delta/alpha model para inconsistency #144 (shansfolder)
- Small improvement on default type of report_kpi_names #140 (shansfolder)
- slightly different json structure for results #138 (gbordyugov)
- merging dev to master #137 (gbordyugov)

5.32 v0.6.0 (2017-07-26)

Full Changelog

Closed issues:

- Improve binning performance #135
- Missing unit tests for to_json() on early stopping algos #128

Merged pull requests:

- Octo 1616 no experimentdata #134 (gbordyugov)
- Attempt to fix pickling bug #133 (shansfolder)
- Stan models compilation, exceptions catch, unit tests adaptation. #131 (daryadedik)
- Added try-finally block for the compulsory clean-up of .pkl compiled models #130 (daryadedik)
- OCTO-1837 fixed to_json() #129 (gbordyugov)

5.33 v0.5.3 (2017-06-26)

Full Changelog

Implemented enhancements:

- Weighted KPIs is only implemented in regular delta #114

Fixed bugs:

- Assumption of nan when computing weighted KPIs #119
- Weighted KPIs is only implemented in regular delta #114
- Percentiles value is lost during computing group_sequential_delta #108

Closed issues:

- Failing early stopping unit tests #85

Merged pull requests:

- Release new version 0.5.3 #127 (mkolarek)
- OCTO-1804: Optimize the loading of .stan model in expan. #126 (daryadedik)
- Test travis python version #125 (shansfolder)
- OCTO-1619 Cleanup ExpAn code #124 (shansfolder)
- OCTO-1748: Make number of iterations as a method argument in _bayes_sampling #123 (daryadedik)
- OCTO-1615 Use Python builtin logging instead of our own debugging.py #122 (shansfolder)
- OCTO-1711 Support weighted KPIs in early stopping #121 (shansfolder)
- Fixed a few bugs #120 (shansfolder)
- OCTO-1614 cleanup module structure #115 (shansfolder)
- OCTO-1677 : fix missing .stan files #113 (gbordyugov)
- Bump version 0.5.1 -> 0.5.2 #112 (mkolarek)

5.34 v0.5.2 (2017-05-11)

Full Changelog

Implemented enhancements:

- OCTO-1502: cleanup of call chains #110 (gbordyugov)

Merged pull requests:

- OCTO-1502 support **kwargs for four delta functions #111 (shansfolder)
- new version 0.5.1 #107 (mkolarek)

5.35 v0.5.1 (2017-04-20)

Full Changelog

Implemented enhancements:

- Derived KPIs are passed to Experiment.fixed_horizon_delta() but never used in there #96

Merged pull requests:

- updated CONTRIBUTING.rst with deployment flow #106 (mkolarek)
- OCTO-1501: bugfix in Results.to_json() #105 (gbordyugov)
- OCTO-1502 removed variant_subset parameter... #104 (gbordyugov)
- OCTO-1540 cleanup handling of derived kpis #102 (shansfolder)

- OCTO-1540: cleanup of derived kpi handling in Experiment.delta() and ... #97 (gbordyugov)
- Small refactoring #95 (shansfolder)
- Merge dev to master for v0.5.0 #94 (mkolarek)

5.36 v0.5.0 (2017-04-05)

Full Changelog

Implemented enhancements:

- Bad code duplication in experiment.py #81
- pip == 8.1.0 requirement #76

Fixed bugs:

- Experiment.sga() assumes features and KPIs are merged in self.metrics #87
- pctile can be undefined in `Results.to\_json\(\)` #78

Closed issues:

- Results.to_json() => TypeError: Object of type 'UserWarning' is not JSON serializable #77
- Rethink Results structure #66

Merged pull requests:

- new dataframe tree traverser in to_json() #92 (gbordyugov)
- updated requirements.txt to have 'greater than' dependencies instead ... #89 (mkolarek)
- pip version requirement #88 (gbordyugov)
- Test #86 (s4826)
- merging in categorical binning #84 (gbordyugov)
- Add documentation of the weighting logic #83 (jbao)
- Early stopping #80 (jbao)
- a couple of minor cleanups #79 (gbordyugov)
- Merge to_json() changes #75 (mkolarek)
- Feature/early stopping #73 (jbao)

5.37 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)

5.38 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)

5.39 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)

5.40 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)

5.41 v0.4.1 (2016-10-18)

Full Changelog

Merged pull requests:

- small doc cleanup #55 (jbao)

- Add comments to cli.py #54 (igusher)
- Feature/octo 545 add consolidate documentation #53 (mkolarek)
- added os.path.join instead of manual string concatenations with '/' #52 (mkolarek)
- Feature/octo 958 outlier filtering #50 (mkolarek)
- Sort KPIs in reverse order before matching them in the formula #49 (jbao)

5.42 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

Merged pull requests:

- merging dev to master #48 (jbao)
- OCTO-825 overall metric #47 (jbao)
- Bump version: 0.3.2 → 0.3.4 #46 (mkolarek)
- Bug/fix dependencies #45 (mkolarek)

5.43 v0.3.4 (2016-08-08)

Full Changelog

Closed issues:

- perform trend analysis cumulatively #31
- Python3 #21

Merged pull requests:

- Feature/2to3 #43 (mkolarek)

5.44 v0.3.3 (2016-08-02)

Full Changelog

Merged pull requests:

- Merge pull request #41 from zalando/master #42 (jbao)
- master to dev #41 (mkolarek)
- Bump version: 0.3.1 → 0.3.2 #40 (mkolarek)
- Revert "Merge pull request #35 from zalando/dev" #39 (mkolarek)
- Merge pull request #35 from zalando/dev #38 (mkolarek)

5.45 v0.3.2 (2016-08-02)

Full Changelog

Merged pull requests:

- Bugfix/trend analysis bin label #37 (jbao)
- Added cumulative trends analysis OCTO-814 #36 (domheger)
- Merging 0.3.1 to master #35 (domheger)

5.46 v0.3.1 (2016-07-15)

Full Changelog

Merged pull requests:

- Bugfix/prob uplift over 0 #34 (jbao)
- Master #30 (mkolarek)

5.47 v0.3.0 (2016-06-23)

Full Changelog

Implemented enhancements:

- Add P(uplift>0) as a statistic #2
- Added function to calculate P(uplift>0) #24 (jbao)

Merged pull requests:

- updated travis.yml #29 (mkolarek)
- Master #28 (mkolarek)
- Master #27 (mkolarek)
- only store the p-value in the chi-square test result object #22 (jbao)

5.48 v0.2.5 (2016-05-30)

Full Changelog

Implemented enhancements:

- Implement __version__ #14

Closed issues:

- upload full documentation! #1

Merged pull requests:

- implement expan.__version__ #19 (pangeran-bottor)
- Mainly documentation changes, as well as travis config updates #17 (robertmuil)

- Update README.rst #16 (pangeran-bottor)
- added cli module #11 (mkolarek)
- new travis config specifying that only master and dev should be built #4 (mkolarek)

5.49 v0.2.4 (2016-05-16)

Full Changelog

Closed issues:

- No module named experiment and test_data #13

Merged pull requests:

- new travis config specifying that only master and dev should be built #5 (mkolarek)

5.50 v0.2.3 (2016-05-06)

Full Changelog

5.51 v0.2.2 (2016-05-06)

Full Changelog

5.52 v0.2.1 (2016-05-06)

Full Changelog

5.53 v0.2.0 (2016-05-06)

Merged pull requests:

- Added detailed documentation with data formats #3 (robertmuil)

* *This Change Log was automatically generated by*github_changelog_generator

CHAPTER 6

Contributing

6.1 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

6.2 Testing

Easiest way to run tests is by running the command `tox` from the terminal. The default Python environments for testing are python 2.7 and python 3.6. You can also specify your own by running e.g. `tox -e py35`.

6.3 Branching

We currently use the gitflow workflow. Feature branches are created from and merged back to the `master` branch. Please always make a Pull Request when you contribute.

See also the much simpler github flow here

6.4 Release

To make a release and **deploy to PyPI**, please follow these steps (we highly suggest to leave the release to admins of ExpAn):

- assuming you have a master branch that is up to date, create a pull request from your feature branch 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
- create a new tag
- run documentation generation which includes creation of changelog
- push **tags** to **master** (a third travis job will be started, but this time it will also push to PyPI because tags were pushed)

The flow would then look like follows:

1. `bumpversion patch` or `bumpversion minor`
2. `git describe --tags`, and note this tag name. We will need to edit this tag later.
3. `make docs`, which will extend the changelog by reading information from github.com/zalando/expan .
4. `git add CHANGELOG.*`
5. `git commit --amend --no-edit`
6. `git show`. Carefully review this commit before proceeding. Ensure the changelog is updated with the expected text, in particular a fully up-to-date version number.
7. `git tag -d {TAGNAME}`, where `{TAGNAME}` is the tag name from step 2.
8. `git tag {TAGNAME}` to recreate the tag in the correct place.
9. `git push`
10. `git push --tags`

You can then check if the triggered Travis CI job is tagged (the name should be eg. 'v1.2.3' instead of 'master').

Note that this workflow has a flaw that changelog generator will not put the changes of the current release, because it reads the commit messages from git remote.

Solution: We need to run `make docs` on **master** once more *after the release* to update the documentation page.

A better solution could be to discard the automatic changelog generator and manually write the changelog before step 1, and then config `make docs` to use this changelog file.

We explain the individual steps below.

6.4.1 Sphinx documentation

`make docs` will create the html documentation if you have sphinx installed. You might need to install our theme explicitly by `pip install sphinx_rtd_theme`.

If you have encountered an error like this: `API rate limit exceeded for github_username`, you need to create a git token and set an environment variable for it. See instructions here.

6.4.2 Versioning

For the sake of reproducibility, always be sure to work with a release when doing the analysis. We use semantic versioning.

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:

```
'{major}.{minor}.{patch}'
```

6.4.3 Bumping Version

We 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.

6.4.4 Travis CI

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

A **build** with unit tests is triggered either

- a commit is pushed to **master**
- or a **pull request** to **master** is opened.

A release to PyPI will be triggered if a new tag is pushed to **master**.

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