
pysubgroup Documentation

Release 0.7.6

mgbckr

Aug 07, 2023

CONTENTS

1	Contents	3
1.1	pysubgroup	3
1.2	Components	6
1.3	Contributing	12
1.4	License	16
1.5	Contributors	20
1.6	Changelog	20
1.7	pysubgroup	24
2	Indices and tables	41
	Python Module Index	43
	Index	45

pysubgroup is a Python package that enables subgroup discovery in Python+pandas (scipy stack) data analysis environment. It provides for a lightweight, easy-to-use, extensible and freely available implementation of state-of-the-art algorithms, interestingness measures and presentation options.

Start reading here: [Overview](#)

Prototype phase

This library is still in a prototype phase. It has, however, been already successfully employed in active application projects.

CONTENTS

1.1 pysubgroup

pysubgroup is a Python package that enables subgroup discovery in Python+pandas (scipy stack) data analysis environment. It provides for a lightweight, easy-to-use, extensible and freely available implementation of state-of-the-art algorithms, interestingness measures and presentation options.

This library is still in a prototype phase. It has, however, been already successfully employed in active application projects.

1.1.1 Subgroup Discovery

Subgroup Discovery is a well established data mining technique that allows you to identify patterns in your data. More precisely, the goal of subgroup discovery is to identify descriptions of data subsets that show an interesting distribution with respect to a pre-specified target concept. For example, given a dataset of patients in a hospital, we could be interested in subgroups of patients, for which a certain treatment X was successful. One example result could then be stated as:

“While in general the operation is successful in only 60% of the cases”, for the subgroup of female patients under 50 that also have been treated with drug d, the success rate was 82%.”

Here, a variable *operation success* is the target concept, the identified subgroup has the interpretable description *female=True AND age<50 AND drug_D = True*. We call these single conditions (such as *female=True*) selection expressions or short *selectors*. The interesting behavior for this subgroup is that the distribution of the target concept differs significantly from the distribution in the overall general dataset. A discovered subgroup could also be seen as a rule:

```
female=True AND age<50 AND drug_D = True ==> Operation_outcome=SUCCESS
```

Computationally, subgroup discovery is challenging since a large number of such conjunctive subgroup descriptions have to be considered. Of course, finding computable criteria, which subgroups are likely interesting to a user is also an eternal struggle. Therefore, a lot of literature has been devoted to the topic of subgroup discovery (including some of my own work). Recent overviews on the topic are for example:

- Herrera, Franciso, et al. “An overview on subgroup discovery: foundations and applications.” Knowledge and information systems 29.3 (2011): 495-525.
- Atzmueller, Martin. “Subgroup discovery.” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 5.1 (2015): 35-49.
- And of course, my point of view on the topic is summarized in my dissertation:

Prerequisites and Installation

pysubgroup is built to fit in the standard Python data analysis environment from the *scipy-stack*. Thus, it can be used just having pandas (including its dependencies numpy, scipy, and matplotlib) installed. Visualizations are carried out with the matplotlib library.

pysubgroup consists of pure Python code. Thus, you can simply download the code from the repository and copy it in your `site-packages` directory. pysubgroup is also on PyPI and should be installable using: `pip install pysubgroup`

Note: Some users complained about the **pip installation not working**. If, after the installation, it still doesn't find the package, then do the following steps:

1. Find where the directory `site-packages` is.
2. Copy the folder `pysubgroup`, which contains the source code, into the `site-packages` directory. (WARNING: This is not the main repository folder. The `pysubgroup` folder is inside the main repository folder, at the same level as `doc`)
3. Now you can import the module with `import pysubgroup`.

1.1.2 How to use:

A simple use case (here using the well known *titanic* data) can be created in just a few lines of code:

```
import pysubgroup as ps

# Load the example dataset
from pysubgroup.datasets import get_titanic_data
data = get_titanic_data()

target = ps.BinaryTarget('Survived', True)
searchspace = ps.create_selectors(data, ignore=['Survived'])
task = ps.SubgroupDiscoveryTask (
    data,
    target,
    searchspace,
    result_set_size=5,
    depth=2,
    qf=ps.WRAccQF())
result = ps.BeamSearch().execute(task)
```

The first line imports *pysubgroup* package. The following lines load an example dataset (the popular *titanic* dataset).

Therafter, we define a target, i.e., the property we are mainly interested in (`'survived'`). Then, we define the searchspace as a list of basic selectors. Descriptions are built from this searchspace. We can create this list manually, or use an utility function. Next, we create a `SubgroupDiscoveryTask` object that encapsulates what we want to find in our search. In particular, that comprises the target, the search space, the depth of the search (maximum numbers of selectors combined in a subgroup description), and the interestingness measure for candidate scoring (here, the Weighted Relative Accuracy measure).

The last line executes the defined task by performing a search with an algorithm—in this case beam search. The result of this algorithm execution is stored in a `SubgroupDiscoveryResults` object.

To just print the result, we could for example do:


```
print(result.to_dataframe())
```

to get:

1.1.3 Key classes

Here is an outline on the most important classes:

- **Selector:** A Selector represents an atomic condition over the data, e.g., *age < 50*. There several subtypes of Selectors, i.e., *NominalSelector* (*color==BLUE*), *NumericSelector* (*age < 50*) and *NegatedSelector* (a wrapper such as *not selector1*)
- **SubgroupDiscoveryTask:** As mentioned before, encapsulates the specification of how an algorithm should search for interesting subgroups
- **SubgroupDiscoveryResult:** These are the main outcome of a subgroup discovery run. You can obtain a list of subgroups using the `to_subgroups()` or to a dataframe using `to_dataframe()`
- **Conjunction:** A conjunction is the most widely used *SubgroupDescription*, and indicates which data instances are covered by the subgroup. It can be seen as the left hand side of a rule.

1.1.4 License

We are happy about anyone using this software. Thus, this work is put under an Apache license. However, if this constitutes any hindrance to your application, please feel free to contact us, we are sure that we can work something out.

Copyright 2016-2019 Florian Lemmerich

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.1.5 Cite

If you are using pysubgroup for your research, please consider citing our demo paper:

Lemmerich, F., & Becker, M. (2018, September). *pysubgroup: Easy-to-use subgroup discovery in python*. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECMLPKDD)*. pp. 658-662.

bibtex:

```
@inproceedings{lemmerich2018pysubgroup,
  title={pysubgroup: Easy-to-use subgroup discovery in python},
  author={Lemmerich, Florian and Becker, Martin},
  booktitle={Joint European Conference on Machine Learning and Knowledge Discovery in
↪ Databases},
  pages={658--662},
  year={2018}
}
```

1.1.6 Note

This project has been set up using PyScaffold 4.5. For details and usage information on PyScaffold see <https://pyscaffold.org/>.

1.2 Components

1.2.1 GP-Growth

This tree based algorithm uses a condensed representation (a so called valuation basis) to find interesting subgroups. The main advantage of this approach is, that the (potentially large) database has to be scanned only twice and thereafter all the necessary information is represented as more compact pattern-tree. Gp-growth is a generalisation of the popular [fp-growth](#) algorithm. So refer to instructional material on fp-growth for more in depth knowledge on the workings of this tree based algorithm.

Contents

- *GP-Growth*
 - *Basic usage*
 - *Create a custom target*

Basic usage

The basic usage of the gp-growth algorithm is not very different from the usage of any other algorithm in this package.

```
import pysubgroup as ps

# Load the example dataset
from pysubgroup.datasets import get_titanic_data
data = get_titanic_data()

target = ps.NominalSelector ('Survived', True)
searchspace = ps.create_selectors(data, ignore=['Survived'])
task = ps.SubgroupDiscoveryTask (data, target, dearchspace, result_set_size=5, depth=2,
↪ qf=ps.WRAccQF())
GpGrowth.execute(task)
```

But beware that gp-growth is using an exhaustive search strategy! This can greatly increase the runtime for high search depth. You can specify the mode argument in the constructor of GpGrowth to run gp-growth either bottom up (mode='b_u') or top down (mode='t_u'). As gp growth is a generalisation of fp-growth you can also perform standard fp-growth using gp_growth by using the CountQF (*Frequent Itemset Targets*) quality function.

Create a custom target

If you consider to use the gp-growth algorithm for your custom target that is totally possible if you find a valuation basis. We will now first introduce the concept of a valuation basis and thereafter outline the gp-growth interface that you have to support to use your quality function with our gp-growth implementation.

Valuation Basis

Think of a valuation basis as a codensed representation of a subgroup that allows to quickly compute the same representation for a union of two disjoint subgroups.

We call the function which takes the valuation basis of two disjoint sets and computes the valuation basis for the unified set merge. The function that compute the necessary statistics from a valuation basis stats_from_basis.

Now we can formulate: Given two disjoint sets A and B with $A \cap B = \emptyset$ and their valuation bases $v(A)$ and $v(B)$ with their functions stats_from_basis and merge as defined above, we can compute the properties of $A \cup B$ instead of from the union of the instances from the merged valuation basis. This can be summarized through the following equation:

$$props_from_instances(A \cup B) = props_from_basis(merge(v(A), v(B)))$$

Required Methods

To make a target and quality function suitable for gp-growth you have to provide several methods (all methods start with gp_ to indicate that they are used in the gp-growth algorithm). In addition to the standard quality function methods (see *Custom Quality Function*) the following methods should be implemented to make a quality function usable with gp_growth.

```
class MyGpQualityFunction
    def gp_get_basis(self, row_index):
        """ returns the valuation basis of the element at this row_index """
        pass

    def gp_get_null_vector(self):
        """ returns the zero element of the valuation basis """
        pass

    @staticmethod
    def gp_merge(v_l, v_r):
        """ merges the v_r valuation basis into the v_l valuation basis inplace! """
        pass

    def gp_get_statistics(self, cover_arr, v):
        """ computes the statistics for this quality function from the valuation basis v """
        pass

    @property
```

(continues on next page)

(continued from previous page)

```
def gp_requires_cover_arr(self) -> bool:
    """ returns a boolean value that indicates whether a cover array is required when
    ↳ calling the gp_get_statistics function

    usually this value is False
    """
    pass
```

Saving a gp_tree

It is possible to save a gp tree to a txt file for e.g. debugging purpose. You therefor have to implementd the gp_to_str function which takes a valuation basis and returns a string representation. It is an intentional choide to not call the str function on the valuation basis directly.

```
def gp_to_str(self, basis) -> str:
    """ returns a string representation of the valuation basis """
    pass
```

1.2.2 Selectors

Selectors are objects that if applied to a dataset yield a set of instances. If an instance is retured from a selector we say that the selectors covers that instance. While the term selectors usually only refers to basic selectors, conjunctions and disjunctions as well as negated selectors are also in a general sense selectors. Broadly speaking anything that implements the code:covers function is a selector. We will first introduce the frequently used basic selectors and thereafter the more general selectors that are the conjunction and disjunction. We conclude the chapter by showing how to implement a selectors yourself.

Basic Selectors

The pysubgroup package provides two basic selectors: The EqualitySelector and the IntervalSelector. Lets start by exploring the EqualitySelector:

```
import pysubgroup as ps
import pandas as pd

# create dataset
first_names = ['Alex', 'Anna', 'Alex']
sur_names = ['Smith', 'Johnson', 'Williams']
ages = [40, 25, 32]
df = pd.DataFrame.from_dict({'First_name':first_names, 'Sur_name': sur_names, 'age':ages}
    ↳)

# create selector
alex_selector = ps.EqualitySelector('First_name', 'Alex')
age_selector = ps.EqualitySelector('age', 22)
# apply selectors to dataframe
print('instances with ', str(alex_selector), alex_selector.covers(df))
print('instances with', str(age_selector), age_selector.covers(df))
```

```
instances with First_name=='Alex' [ True False  True]
instances with age==22 [False False False]
```

The output indicates that the first and third instance in the dataset have a first name that is equal to 'Alex'. The second output shows that no instances in our dataset is of age 22. The EqualitySelector selector can be used on many different datatypes, but is most useful on binary, string and categorical data. In addition to the EqualitySelector the pysubgroup package also provides the IntervalSelector. The following codes selects all instances from the database, which are in the age range 18 (included) to 40 (excluded).

```
interval_selector = ps.IntervalSelector('age', 18, 40)
print(interval_selector.covers(df))
```

```
[False  True  True]
```

The output shows that the second and third instance in our dataset have an age within the interval [18, 40).

Selectors are the building block of all rules generated with the pysubgroup package. If you want to write your own custom selector that is not a problem see customselector for references.

Negations

The pysubgroup package also provides the NegatedSelector class that takes any selector (not just basic ones) and inverts it.

```
inverted_selector = ps.NegatedSelector(alex_selector)
print('instances with first name not equal to Alex', inverted_selector.covers(df))
```

```
instances with first name not equal to Alex [False  True False]
```

The output is: instances with first name not equal to Alex [False, True, False].

Conjunctions

Most of the rules that are generated with the pysubgroup package use conjunctions to form more complex queries. Continuing the running example from above we can find all persons whose name is Alex *and* which have an age in the interval [18, 40) like so:

```
conj = ps.Conjunction([interval_selector, alex_selector])
print('instances with', str(conj), conj.covers(df))
```

```
instances with First_name=='Alex' AND age: [18:40[ [False False  True]
```

The output shows that only the last instance is covered by our conjunction.

Disjunctions

The pysubgroup package also provides disjunctions with the `Disjunction` class. Continuing the running example we can find all persons whose name is Alex *or* which have an age in the interval $[18, 40)$ like so:

```
disj = ps.Disjunction([interval_selector, alex_selector])
print('instances with', str(disj), disj.covers(df))
```

```
instances with First_name=='Alex' OR age: [18:40[ [ True  True  True]
```

We can see that all instances are covered by our conjunction.

Implementing your own

As already mentioned in the introduction on selectors, anything that provides a cover function is a selector. In this case we will show how to implement a custom basic selector that checks whether a string contains a given substring:

```
class StrContainsSelector:
    def __init__(self, column, substr):
        self.column = column
        self.substr = substr

    def covers(self, df):
        return df[self.column].str.contains(self.substr).to_numpy()

contains_selector = StrContainsSelector('Sur_name', 'm')
print(contains_selector.covers(df))
```

```
[ True False  True]
```

The output shows that only the first and last instance contain an m in their name. In addition to the covers function it is certainly advised to also implement the `__str__` and `__repr__` functions. This selector can now be added to the searchspace for any algorithm execution.

1.2.3 Targets and Quality Functions

To define the goal of our subgroup discovery task, we use targets and quality functions. Targets are used to define which attributes play a significant role and can provide common statistics for a subgroup in question. Quality functions assign a score to each subgroup. These scores are used by all the algorithms to determine the most interesting subgroups.

Frequent Itemset Targets

The most simple target is the *FITarget* with its associated quality functions *CountQF* and *AreaQf*. The CountQF simply counts the number of instances covered by the subgroup in question. The AreaQF multiplies the depth or length of the subgroup description with the number of instances covered by that description.

Binary Targets

For Boolean or Binary Targets we provide the *ChiSquaredQF* as well as the *StandardQF* quality functions. The *StandardQF* quality function uses a parameter α to weight the relative size $\frac{N_{SG}}{N}$ of a subgroup and multiplies it with the differences in relations of positive instances p to the number of instances N

$$\left(\frac{N_{SG}}{N}\right)^{\alpha} \left(\frac{p_{SG}}{N_{SG}} - \frac{p}{N}\right)$$

The *StandardQF* also supports an optimistic estimate.

The *ChiSquaredQF* is calculated based on the following contingency table which is then passed to the scipy `chi2_contingency` function. The small n represents the number of negative instances and should not be confused with the capital N which represents the total number of instances.

p_{SG}	$p - p_{SG}$
n_{SG}	$n - n_{SG}$

Nominal Targets

Currently pysubgroup only supports nominal targets as binary targets. So you can look for deviations of one nominal value with respect to all other nominal values.

Numeric Targets

For numeric targets pysubgroup offers the *StandardQFNumeric* which is defined similar to the *StandardQF*

$$\left(\frac{N_{SG}}{N}\right)^{\alpha} (\mu_{SG} - \mu)$$

where μ_{SG} and μ are the mean value for the subgroup and entire dataset respectively. For the *StandardQFNumeric* we offer three optimistic estimates: Average, Summation and Ordering. These are in detail described in Florian Lemmerich's dissertation. You can choose between the different optimistic estimates by using the keyword argument `estimator` the different options are 'sum', 'average', and 'order'

Custom Quality Function

To create a custom quality function that works with all algorithms except `gp_growth`.

```
class MyQualityFunction:
    def calculate_constant_statistics(self, task):
        """ calculate_constant_statistics
            This function is called once for every execution,
            it should do any preparation that is necessary prior to an execution.
        """
        pass

    def calculate_statistics(self, subgroup, data=None):
        """ calculates necessary statistics
            this statistics object is passed on to the evaluate
            and optimistic_estimate functions
        """
```

(continues on next page)

(continued from previous page)

```
pass

def evaluate(self, subgroup, statistics_or_data=None):
    """ return the quality calculated from the statistics """
    pass

def optimistic_estimate(self, subgroup, statistics=None):
    """ returns optimistic estimate
        if one is available return it otherwise infinity"""
    pass
```

1.3 Contributing

TODO: UPDATE THIS

Welcome to pysubgroup contributor's guide.

This document focuses on getting any potential contributor familiarized with the development processes, but other kinds of contributions are also appreciated.

If you are new to using [git](#) or have never collaborated in a project previously, please have a look at [contribution-guide.org](#). Other resources are also listed in the excellent [guide created by FreeCodeCamp](#)¹.

Please notice, all users and contributors are expected to be **open, considerate, reasonable, and respectful**. When in doubt, [Python Software Foundation's Code of Conduct](#) is a good reference in terms of behavior guidelines.

1.3.1 Issue Reports

If you experience bugs or general issues with pysubgroup, please have a look on the [issue tracker](#). If you don't see anything useful there, please feel free to fire an issue report.

Tip: Please don't forget to include the closed issues in your search. Sometimes a solution was already reported, and the problem is considered **solved**.

New issue reports should include information about your programming environment (e.g., operating system, Python version) and steps to reproduce the problem. Please try also to simplify the reproduction steps to a very minimal example that still illustrates the problem you are facing. By removing other factors, you help us to identify the root cause of the issue.

¹ Even though, these resources focus on open source projects and communities, the general ideas behind collaborating with other developers to collectively create software are general and can be applied to all sorts of environments, including private companies and proprietary code bases.

1.3.2 Documentation Improvements

You can help improve pysubgroup docs by making them more readable and coherent, or by adding missing information and correcting mistakes.

pysubgroup documentation uses [Sphinx](#) as its main documentation compiler. This means that the docs are kept in the same repository as the project code, and that any documentation update is done in the same way as a code contribution. We are using [CommonMark](#) with [MyST](#) extensions as our markup language.

Tip: Please notice that the [GitHub web interface](#) provides a quick way of propose changes in pysubgroup's files. While this mechanism can be tricky for normal code contributions, it works perfectly fine for contributing to the docs, and can be quite handy.

If you are interested in trying this method out, please navigate to the docs folder in the source [repository](#), find which file you would like to propose changes and click in the little pencil icon at the top, to open [GitHub's code editor](#). Once you finish editing the file, please write a message in the form at the bottom of the page describing which changes have you made and what are the motivations behind them and submit your proposal.

When working on documentation changes in your local machine, you can compile them using [tox](#) :

```
tox -e docs
```

and use Python's built-in web server for a preview in your web browser (<http://localhost:8000>):

```
python3 -m http.server --directory 'docs/_build/html'
```

1.3.3 Code Contributions

Submit an issue

Before you work on any non-trivial code contribution it's best to first create a report in the [issue tracker](#) to start a discussion on the subject. This often provides additional considerations and avoids unnecessary work.

Create an environment

Before you start coding, we recommend creating an isolated [virtual environment](#) to avoid any problems with your installed Python packages. This can easily be done via either [virtualenv](#):

```
virtualenv <PATH TO VENV>
source <PATH TO VENV>/bin/activate
```

or [Miniconda](#):

```
conda create -n pysubgroup python=3 six virtualenv pytest pytest-cov
conda activate pysubgroup
```

Clone the repository

1. Create an user account on GitHub if you do not already have one.
2. Fork the project [repository](#): click on the *Fork* button near the top of the page. This creates a copy of the code under your account on GitHub.
3. Clone this copy to your local disk:

```
git clone git@github.com:YourLogin/pysubgroup.git
cd pysubgroup
```

4. You should run:

```
pip install -U pip setuptools -e .
```

to be able to import the package under development in the Python REPL.

5. Install [pre-commit](#):

```
pip install pre-commit
pre-commit install
```

pysubgroup comes with a lot of hooks configured to automatically help the developer to check the code being written.

Implement your changes

1. Create a branch to hold your changes:

```
git checkout -b my-feature
```

and start making changes. Never work on the main branch!

2. Start your work on this branch. Don't forget to add [docstrings](#) to new functions, modules and classes, especially if they are part of public APIs.
3. Add yourself to the list of contributors in `AUTHORS.rst`.
4. When you're done editing, do:

```
git add <MODIFIED FILES>
git commit
```

to record your changes in [git](#).

Please make sure to see the validation messages from [pre-commit](#) and fix any eventual issues. This should automatically use [flake8/black](#) to check/fix the code style in a way that is compatible with the project.

Important: Don't forget to add unit tests and documentation in case your contribution adds an additional feature and is not just a bugfix.

Moreover, writing a [descriptive commit message](#) is highly recommended. In case of doubt, you can check the commit history with:

```
git log --graph --decorate --pretty=oneline --abbrev-commit --all
```

to look for recurring communication patterns.

5. Please check that your changes don't break any unit tests with:

```
tox
```

(after having installed `tox` with `pip install tox` or `pipx`).

You can also use `tox` to run several other pre-configured tasks in the repository. Try `tox -av` to see a list of the available checks.

Submit your contribution

1. If everything works fine, push your local branch to the remote server with:

```
git push -u origin my-feature
```

2. Go to the web page of your fork and click "Create pull request" to send your changes for review.

Troubleshooting

The following tips can be used when facing problems to build or test the package:

1. Make sure to fetch all the tags from the upstream `repository`. The command `git describe --abbrev=0 --tags` should return the version you are expecting. If you are trying to run CI scripts in a fork repository, make sure to push all the tags. You can also try to remove all the egg files or the complete egg folder, i.e., `.eggs`, as well as the `*.egg-info` folders in the `src` folder or potentially in the root of your project.
2. Sometimes `tox` misses out when new dependencies are added, especially to `setup.cfg` and `docs/requirements.txt`. If you find any problems with missing dependencies when running a command with `tox`, try to recreate the `tox` environment using the `-r` flag. For example, instead of:

```
tox -e docs
```

Try running:

```
tox -r -e docs
```

3. Make sure to have a reliable `tox` installation that uses the correct Python version (e.g., 3.7+). When in doubt you can run:

```
tox --version
# OR
which tox
```

If you have trouble and are seeing weird errors upon running `tox`, you can also try to create a dedicated `virtual environment` with a `tox` binary freshly installed. For example:

```
virtualenv .venv
source .venv/bin/activate
.venv/bin/pip install tox
.venv/bin/tox -e all
```

4. `Pytest` can `drop you` in an interactive session in the case an error occurs. In order to do that you need to pass a `--pdb` option (for example by running `tox -- -k <NAME OF THE FALLING TEST> --pdb`). You can also setup breakpoints manually instead of using the `--pdb` option.

1.3.4 Maintainer tasks

Releases

If you are part of the group of maintainers and have correct user permissions on `PyPI`, the following steps can be used to release a new version for `pysubgroup`:

1. Make sure all unit tests are successful.
 2. Tag the current commit on the main branch with a release tag, e.g., `v1.2.3`.
 3. Push the new tag to the upstream `repository`, e.g., `git push upstream v1.2.3`
 4. Clean up the `dist` and `build` folders with `tox -e clean` (or `rm -rf dist build`) to avoid confusion with old builds and Sphinx docs.
 5. Run `tox -e build` and check that the files in `dist` have the correct version (no `.dirty` or `git` hash) according to the `git` tag. Also check the sizes of the distributions, if they are too big (e.g., `> 500KB`), unwanted clutter may have been accidentally included.
 6. Run `tox -e publish -- --repository pypi` and check that everything was uploaded to `PyPI` correctly.
-

1.4 License

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

(continues on next page)

(continued from previous page)

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made,

(continues on next page)

(continued from previous page)

use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

(continues on next page)

(continued from previous page)

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following

(continues on next page)

(continued from previous page)

boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright 2020 Florian Lemmerich

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

1.5 Contributors

- mgbckr mgbckr@informatik.uni-rostock.de

1.6 Changelog

1.6.1 [0.7.1] - 2020-05-20

Added

- you can now additionally provide **constraints** to SubgroupDiscovery
 - **MinSupportConstraint** added
- you can now run the slow tests py passing `--runslow` to pytest
- **Conjunction**, **Disjunction** and **Selectors** now all have the public property `.selectors` that provides all basic selectors involved

Removed

- support for weights has been removed, it will probably be added in the future as separate targets and Quality functions.

Changed

- `create_numeric_selector_for_attribute` has been renamed to `create_numeric_selectors_for_attribute` (inserting an s) This brings it in line with the corresponding name schema for nominal.

Changed internally

- statistics are now also stored along with score and description
- The function `ps.get_cover_array_and_size` was added, it allows for a consistent way to access a cover array (a.k.a. sth to be thrown into a dataframe or a numpy array)
- algorithm tests now also call the `to_subgroups` and `to_dataframe` methods to check they work with that algorithm
- the order of `calculate_statistics` and `get_base_statistics` are now in line with that of quality functions (first subgroup then data)
- the size of a subgroup specified in a statistics object is now called `size_sg` uniformly. This avoids confusion with the `size` attribute of numpy arrays etc.

1.6.2 [0.7.0] - 2020-04-24

This update prepares pysubgroup for a better future. To do so we had to break backwards compatibility. Many of the classes that you know and love have been renamed so as to make their purpose more clear.

Changed:

- `SubgroupDescription` is now called `Conjunction`
- `NominalTarget` is now called `BinaryTarget`
- algorithms now return a `SubgroupDiscoveryResult` object
- the structure of quality functions changed (see documentation for more info)

Added

- pysubgroup now has a bunch of tests
- some algorithms and quality functions support numba for just in time compilation
- `ModelTarget`
- `gp-growth`
- 3 types of Representations (bitset, set, numpy-set)
- Refinement operator
- `Disjunction`

- New algorithms

1.6.3 [0.6.2.1] - 2019-20-11

Added

- Apriori now has the option to disable numba using the `use_numba` flag
- SimpleSrach now has a progressbar (enabled via the `show_progress=True` flag)
- The number of quality function evaluations can now be tracked using the `CountCallsInterestingMeasure` as a wrapper
- StandardQfNumeric now offers three different options to calculate the optimistic estimate
 - ‘sum’ (default) sums the values larger then the dataset mean (cf. Lemmerich 2014 p. 81 top)
 - ‘average’ uses the maximum target values as estimate (cf. Lemmerich 2014 p. 82 center)
 - ‘order’ uses ordering based bounds (cf. Lemmerich 2014 p. 89 bottom)

Bugfix

- Apriori now calculates the constant statistics before using representation
- DFS now properly works with any quality function

Improvements

- Apriori now reuses the compiled numba function
- Nominal target now uses `subgroup.size` to access the size of a subgroup representation
- StaticSpecializationOperator now avoids checking refinements of the same attribute
- `test_algorithms_numeric` now checks more algorithms

1.6.4 [0.6.2] - 2019-31-10

Changed

- **SubgroupDescription** has been replaced with **Conjunction**
- Selector `.covers` function returns a numpy array instead of a pandas Series (speedup on dense data)
- Conjunction `.selectors` is renamed to `Conjunction._selectors`
- quality functions have a different interface
 - `calculate_constant_statistics(self, task)` caches necessary precomputation
 - `calculate_statistics(self, subgroup, data=None)` returns a namedtuple with necessary statistics
 - `evaluate(self, subgroup, statistics=None)` computes quality from provided statistics
 - `optimistic_estimate(self, subgroup, statistics=None)` computes optimistic estimate from provided statistics
-

Added

- Conjunction (replaces SubgroupDescription)
- Disjunction
- DNF (Disjunctive Normal Form)
- representations (given a dataset selectors are queried only once)
 - BitsetRepresentation
 - SetRepresentation
 - NumpySetRepresentation
- SimpleSearch algorithm
- DFS (Depth first search) using a representation for StandardQF
- tests
 - access to datasets for testing is provided through DataSets class
 - tests for selector classes (NominalSelector, NumericSelector)
 - * `__eq__`
 - * `__lt__`
 - * `__hash__` similarity
 - * uniqueness of selectors
 - * cover function for NominalSelector
 - tests for Conjunction, Disjunction
 - * `__eq__`
 - * `__lt__`
 - * `__hash__` similarity
 - * cover
 - tests for algorithms with nominal target concept on the creditg dataset (StandardQF(1) + NominalSearchSpace, StandardQF(1)+Nominal&Numeric Searchspace, StandardQF(0.5)+Nominal&Numeric Searchspace)
 - * Apriori
 - * SimpleDFS
 - * BeamSearch
 - * DFS_bitset
 - * DFS_set
 - * DFS_numpy_sets
 - * SimpleSearch
 - tests for algorithms with numeric target concept (StandardQFNumeric)
 - * Apriori
 - * SimpleDFS

- * DFSNumeric
- tests for algorithm with fi target (CountQF)
 - * Apriori
 - * DFS
- tests for algorithms to find the best Disjunctions
 - * Apriori
 - * Generalising BFS

Improvements

- Apriori algorithm now runs significantly faster due to precomputing and usage of list comprehension

1.7 pysubgroup

1.7.1 pysubgroup package

Submodules

pysubgroup.algorithms module

Created on 29.04.2016

@author: lemmerfn

```
class pysubgroup.algorithms.Apriori(representation_type=None, combination_name='Conjunction',  
                                     use_numba=True)
```

Bases: `object`

execute(task)

get_next_level(promising_candidates)

get_next_level_candidates(task, result, next_level_candidates)

get_next_level_candidates_vectorized(task, result, next_level_candidates)

get_next_level_numba(promising_candidates)

```
class pysubgroup.algorithms.BeamSearch(beam_width=20, beam_width_adaptive=False)
```

Bases: `object`

Implements the BeamSearch algorithm. Its a basic implementation

execute(task)

```
class pysubgroup.algorithms.BestFirstSearch
```

Bases: `object`

execute(task)

```

class pysubgroup.algorithms.DFS(apply_representation)
    Bases: object
    Implementation of a depth-first-search with look-ahead using a provided datastructure.
    execute(task)
    search_internal(task, result, sg)

class pysubgroup.algorithms.DFSNumeric
    Bases: object
    execute(task)
    search_internal(task, prefix, modification_set, result, bitset)
    tpl
        alias of size_mean_parameters

class pysubgroup.algorithms.GeneralisingBFS
    Bases: object
    execute(task)

class pysubgroup.algorithms.SimpleDFS
    Bases: object
    execute(task, use_optimistic_estimates=True)
    search_internal(task, prefix, modification_set, result, use_optimistic_estimates)

class pysubgroup.algorithms.SimpleSearch(show_progress=True)
    Bases: object
    execute(task)

class pysubgroup.algorithms.SubgroupDiscoveryTask(data, target, search_space, qf, result_set_size=10,
                                                    depth=3, min_quality=-inf, constraints=None)
    Bases: object
    Capsulates all parameters required to perform standard subgroup discovery
    pysubgroup.algorithms.constraints_satisfied(constraints, subgroup, statistics=None, data=None)

```

pysubgroup.binary_target module

Created on 29.09.2017

@author: lemmerfn

```

class pysubgroup.binary_target.BinaryTarget(target_attribute=None, target_value=None,
                                              target_selector=None)
    Bases: BaseTarget
    calculate_statistics(subgroup, data, cached_statistics=None)
    covers(instance)
    get_attributes()

```

```
get_base_statistics(subgroup, data)
```

```
statistic_types = ('size_sg', 'size_dataset', 'positives_sg', 'positives_dataset',
'size_complement', 'relative_size_sg', 'relative_size_complement', 'coverage_sg',
'coverage_complement', 'target_share_sg', 'target_share_complement',
'target_share_dataset', 'lift')
```

```
class pysubgroup.binary_target.ChiSquaredQF(direction='both', min_instances=5, stat='chi2')
```

Bases: [SimplePositivesQF](#)

ChiSquaredQF which test for statistical independence of a subgroup against it's complement

...

```
static chi_squared_qf(instances_dataset, positives_dataset, instances_subgroup, positives_subgroup,
min_instances=5, bidirect=True, direction_positive=True, index=0)
```

Performs chi2 test of statistical independence

Test whether a subgroup is statistically independent from it's complement (see `scipy.stats.chi2_contingency`).

Parameters

instances_dataset –

positives_dataset,

instances_subgroup, positives_subgroup : int

counts of subgroup and dataset

:param

[positives_dataset,]

instances_subgroup, positives_subgroup : int

counts of subgroup and dataset

Parameters

- **min_instances** (*int*, *optional*) – number of required instances, if less -inf is returned for that subgroup
- **bidirect** (*bool*, *optional*) – If true both directions are considered interesting else `direction_positive` decides which direction is interesting
- **direction_positive** (*bool*, *optional*) – Only used if `bidirect=False`; specifies whether you are interested in positive (True) or negative deviations
- **index** (*{0, 1}*, *optional*) – decides whether the test statistic (0) or the p-value (1) should be used

```
static chi_squared_qf_weighted(subgroup, data, weighting_attribute, effective_sample_size=0,
min_instances=5)
```

```
evaluate(subgroup, target, data, statistics=None)
```

```
class pysubgroup.binary_target.GeneralizationAware_StandardQF(a)
```

Bases: [GeneralizationAwareQF_stats](#)

```
evaluate(subgroup, target, data, statistics=None)
```

get_max(*args)

class pysubgroup.binary_target.LiftQF

Bases: *StandardQF*

Lift Quality Function

LiftQF is a StandardQF with $a=0$. Thus it treats the difference in ratios as the quality without caring about the relative size of a subgroup.

class pysubgroup.binary_target.SimpleBinomialQF

Bases: *StandardQF*

Simple Binomial Quality Function

SimpleBinomialQF is a StandardQF with $a=0.5$. It is an order equivalent approximation of the full binomial test if the subgroup size is much smaller than the size of the entire dataset.

class pysubgroup.binary_target.SimplePositivesQF

Bases: *AbstractInterestingnessMeasure*

calculate_constant_statistics(data, target)

calculate_statistics(subgroup, target, data, statistics=None)

gp_get_null_vector()

gp_get_params(cover_arr, v)

gp_get_stats(row_index)

gp_merge(left, right)

property gp_requires_cover_arr

gp_size_sg(stats)

gp_to_str(stats)

tpl

alias of PositivesQF_parameters

class pysubgroup.binary_target.StandardQF(a)

Bases: *SimplePositivesQF*, *BoundedInterestingnessMeasure*

StandardQF which weights the relative size against the difference in averages

The StandardQF is a general form of quality function which for different values of a is order equivalent to many popular quality measures.

a

used as an exponent to scale the relative size to the difference in averages

Type

float

evaluate(subgroup, target, data, statistics=None)

optimistic_estimate(subgroup, target, data, statistics=None)

optimistic_generalisation(subgroup, target, data, statistics=None)

```
static standard_qf(a, instances_dataset, positives_dataset, instances_subgroup, positives_subgroup)
```

```
class pysubgroup.binary_target.WRAccQF
```

Bases: [StandardQF](#)

Weighted Relative Accuracy Quality Function

WRAccQF is a StandardQF with $a=1$. It is order equivalent to the difference in the observed and expected number of positive instances.

pysubgroup.constraints module

```
class pysubgroup.constraints.MinSupportConstraint(min_support)
```

Bases: [object](#)

```
gp_is_satisfied(node)
```

```
gp_prepare(qf)
```

```
property is_monotone
```

```
is_satisfied(subgroup, statistics=None, data=None)
```

pysubgroup.datasets module

```
pysubgroup.datasets.get_credit_data()
```

```
pysubgroup.datasets.get_titanic_data()
```

pysubgroup.fi_target module

Created on 29.09.2017

@author: lemmerfn

```
class pysubgroup.fi_target.AreaQF
```

Bases: [SimpleCountQF](#)

```
evaluate(subgroup, target, data, statistics=None)
```

```
class pysubgroup.fi_target.CountQF
```

Bases: [SimpleCountQF](#), [BoundedInterestingnessMeasure](#)

```
evaluate(subgroup, target, data, statistics=None)
```

```
optimistic_estimate(subgroup, target, data, statistics=None)
```

```
class pysubgroup.fi_target.FITarget
```

Bases: [BaseTarget](#)

```
calculate_statistics(subgroup_description, data, cached_statistics=None)
```

```
get_attributes()
```

```
get_base_statistics(subgroup, data)
```



```

    statistic_types = ('size_sg', 'size_dataset')

class pysubgroup.fi_target.SimpleCountQF
    Bases: AbstractInterestingnessMeasure
    calculate_constant_statistics(data, target)
    calculate_statistics(subgroup_description, target, data, statistics=None)
    gp_get_null_vector()
    gp_get_params(_cover_arr, v)
    gp_get_stats(_)
    gp_merge(left, right)
    gp_requires_cover_arr = False
    gp_size_sg(stats)
    gp_to_str(stats)
    tpl
        alias of CountQF_parameters

```

pysubgroup.gp_growth module

```

class pysubgroup.gp_growth.GpGrowth(mode='b_u')
    Bases: object
    add_if_required(prefix, gp_stats)
    calculate_quality_function_for_patterns(task, results, arrs)
    check_constraints(node)
    check_tree_is_ordered(root, prefix=None)
        Verify that the nodes of a tree are sorted in ascending order
    convert_results_to_subgroups(results, selectors_sorted)
    create_copy_of_path(nodes, new_nodes, stats)
    create_copy_of_tree_top_down(from_root, nodes=None, parent=None, is_valid_class=None)
    create_initial_tree(arrs)
    create_new_tree_from_nodes(nodes)
    execute(task)
    get_nodes_upwards(node)
    get_stats_for_class(cls_nodes)
    get_top_down_tree_for_class(cls_nodes, cls, is_valid_class)
    merge_trees_top_down(nodes, mutable_root, from_root, is_valid_class)

```

```
nodes_to_cls_nodes(nodes)

normal_insert(root, nodes, new_stats, classes)

prepare_selectors(search_space, data)

recurse(cls_nodes, prefix, is_single_path=False)

recurse_top_down(cls_nodes, root, depth_in=0)

remove_selectors_with_low_optimistic_estimate(s, search_space_size)

setup(task)

setup_constraints(constraints, qf)

setup_from_quality_function(qf)

to_file(task, path)

pysubgroup.gp_growth.identity(x, *args, **kwargs)
```

pysubgroup.measures module

Created on 28.04.2016

@author: lemmerfn

```
class pysubgroup.measures.AbstractInterestingnessMeasure
    Bases: ABC
    ensure_statistics(subgroup, target, data, statistics=None)

class pysubgroup.measures.BoundedInterestingnessMeasure
    Bases: AbstractInterestingnessMeasure

class pysubgroup.measures.CombinedInterestingnessMeasure(measures, weights=None)
    Bases: BoundedInterestingnessMeasure
    calculate_constant_statistics(data, target)
    calculate_statistics(subgroup, target, data, cached_statistics=None)
    evaluate(subgroup, target, data, statistics=None)
    evaluate_from_statistics(instances_dataset, positives_dataset, instances_subgroup,
                             positives_subgroup)
    optimistic_estimate(subgroup, target, data, statistics=None)

class pysubgroup.measures.CountCallsInterestingMeasure(qf)
    Bases: BoundedInterestingnessMeasure
    calculate_statistics(sg, target, data, statistics=None)

class pysubgroup.measures.GeneralizationAwareQF(qf)
    Bases: AbstractInterestingnessMeasure
    calculate_constant_statistics(data, target)
```

```

calculate_statistics(subgroup, target, data, statistics=None)

evaluate(subgroup, target, data, statistics=None)

class ga_tuple(subgroup_quality, generalisation_quality)
    Bases: tuple
    generalisation_quality
        Alias for field number 1
    subgroup_quality
        Alias for field number 0

get_qual_and_previous_qual(subgroup, target, data)

class pysubgroup.measures.GeneralizationAwareQF_stats(qf)
    Bases: AbstractInterestingnessMeasure
    calculate_constant_statistics(data, target)

    calculate_statistics(subgroup, target, data, statistics=None)

    evaluate(subgroup, target, data, statistics=None)

    ga_tuple
        alias of ga_stats_tuple

    get_max(*args)

    get_stats_and_previous_stats(subgroup, target, data)

pysubgroup.measures.maximum_statistic_filter(result_set, statistic, maximum)

pysubgroup.measures.minimum_quality_filter(result_set, minimum)

pysubgroup.measures.minimum_statistic_filter(result_set, statistic, minimum, data)

pysubgroup.measures.overlap_filter(result_set, data, similarity_level=0.9)

pysubgroup.measures.overlaps_list(sg, list_of_sgs, data, similarity_level=0.9)

pysubgroup.measures.unique_attributes(result_set, data)

```

pysubgroup.model_target module

```

class pysubgroup.model_target.EMM_Likelihood(model)
    Bases: AbstractInterestingnessMeasure
    calculate_constant_statistics(data, target)

    calculate_statistics(subgroup, target, data, statistics=None)

    evaluate(subgroup, target, data, statistics=None)

    get_tuple(sg_size, params, cover_arr)

    gp_get_params(cover_arr, v)

```

property `gp_requires_cover_arr`

tpl

alias of `EMM_Likelihood`

class `pysubgroup.model_target.PolyRegression_ModelClass(x_name='x', y_name='y', degree=1)`

Bases: `object`

calculate_constant_statistics(*data*, *target*)

fit(*subgroup*, *data*=None)

gp_get_null_vector()

gp_get_params(*v*)

gp_get_stats(*row_index*)

static gp_merge(*u*, *v*)

property gp_requires_cover_arr

gp_size_sg(*stats*)

gp_to_str(*stats*)

likelihood(*stats*, *sg*)

loglikelihood(*stats*, *sg*)

class `pysubgroup.model_target.beta_tuple(beta, size_sg)`

Bases: `tuple`

beta

Alias for field number 0

size_sg

Alias for field number 1

pysubgroup.numeric_target module

Created on 29.09.2017

@author: lemmerfn

class `pysubgroup.numeric_target.NumericTarget(target_variable)`

Bases: `object`

calculate_statistics(*subgroup*, *data*, *cached_statistics*=None)

get_attributes()

get_base_statistics(*subgroup*, *data*)

statistic_types = ('size_sg', 'size_dataset', 'mean_sg', 'mean_dataset', 'std_sg', 'std_dataset', 'median_sg', 'median_dataset', 'max_sg', 'max_dataset', 'min_sg', 'min_dataset', 'mean_lift', 'median_lift')

```

class pysubgroup.numeric_target.StandardQFNumeric(a, invert=False, estimator='sum')
    Bases: BoundedInterestingnessMeasure
    class Average_Estimator(qf)
        Bases: object
        calculate_constant_statistics(data, target)

        get_data(data, target)

        get_estimate(subgroup, sg_size, sg_mean, cover_arr, _)

    class Ordering_Estimator(qf)
        Bases: object
        calculate_constant_statistics(data, target)

        get_data(data, target)

        get_estimate(subgroup, sg_size, sg_mean, cover_arr, target_values_sg)

        get_estimate_numpy(values_sg, _, mean_dataset)

    class Summation_Estimator(qf)
        Bases: object
        calculate_constant_statistics(data, target)

        get_data(data, target)

        get_estimate(subgroup, sg_size, sg_mean, cover_arr, _)

    calculate_constant_statistics(data, target)

    calculate_statistics(subgroup, target, data, statistics=None)

    evaluate(subgroup, target, data, statistics=None)

    optimistic_estimate(subgroup, target, data, statistics=None)

    static standard_qf_numeric(a, _, mean_dataset, instances_subgroup, mean_sg)

    tpl
        alias of StandardQFNumeric_parameters

class pysubgroup.numeric_target.StandardQFNumericMedian(a, invert=False, estimator='sum')
    Bases: BoundedInterestingnessMeasure
    class Average_Estimator(qf)
        Bases: object
        calculate_constant_statistics(data, target)

        get_data(data, target)

        get_estimate(subgroup, sg_size, sg_mean, cover_arr, _)

    class Ordering_Estimator(qf)
        Bases: object

```

```
    calculate_constant_statistics(data, target)

    get_data(data, target)

    get_estimate(subgroup, sg_size, sg_mean, cover_arr, target_values_sg)

    get_estimate_numpy(values_sg, _, mean_dataset)

class Summation_Estimator(qf)
    Bases: object
    calculate_constant_statistics(data, target)

    get_data(data, target)

    get_estimate(subgroup, sg_size, sg_median, cover_arr, _)

calculate_constant_statistics(data, target)

calculate_statistics(subgroup, target, data, statistics=None)

evaluate(subgroup, target, data, statistics=None)

optimistic_estimate(subgroup, target, data, statistics=None)

static standard_qf_numeric(a, _, median_dataset, instances_subgroup, median_sg)

tpl
    alias of StandardQFNumericMedian_parameters

class pysubgroup.numeric_target.StandardQFNumericTscore(a, invert=False, estimator='sum')
    Bases: BoundedInterestingnessMeasure

class Average_Estimator(qf)
    Bases: object
    calculate_constant_statistics(data, target)

    get_data(data, target)

    get_estimate(subgroup, sg_size, sg_mean, cover_arr, _)

class Ordering_Estimator(qf)
    Bases: object
    calculate_constant_statistics(data, target)

    get_data(data, target)

    get_estimate(subgroup, sg_size, sg_mean, cover_arr, target_values_sg)

    get_estimate_numpy(values_sg, _, mean_dataset)

class Summation_Estimator(qf)
    Bases: object
    calculate_constant_statistics(data, target)

    get_data(data, target)
```

```

    get_estimate(subgroup, sg_size, sg_mean, cover_arr, _)
calculate_constant_statistics(data, target)
calculate_statistics(subgroup, target, data, statistics=None)
evaluate(subgroup, target, data, statistics=None)
optimistic_estimate(subgroup, target, data, statistics=None)
static standard_qf_numeric(a, _, mean_dataset, instances_subgroup, mean_sg, std_sg)
tpl
    alias of StandardQFNumericTscore_parameters

```

pysubgroup.refinement_operator module

```

class pysubgroup.refinement_operator.RefinementOperator
    Bases: object
class pysubgroup.refinement_operator.StaticGeneralizationOperator(selectors)
    Bases: object
    refinements(sG)
class pysubgroup.refinement_operator.StaticSpecializationOperator(selectors)
    Bases: object
    refinements(subgroup)

```

pysubgroup.representations module

```

class pysubgroup.representations.BitSetRepresentation(df, selectors_to_patch)
    Bases: RepresentationBase
    Conjunction
        alias of BitSet\_Conjunction
    Disjunction
        alias of BitSet\_Disjunction
    patch_classes()
    patch_selector(sel)
class pysubgroup.representations.BitSet_Conjunction(*args, **kwargs)
    Bases: Conjunction
    append_and(to_append)
    compute_representation()
    n_instances = 0
    property size_sg

```

```
class pysubgroup.representations.BitSet_Disjunction(*args, **kwargs)
    Bases: Disjunction
    append_or(to_append)
    compute_representation()
    property size_sg

class pysubgroup.representations.NumpySetRepresentation(df, selectors_to_patch)
    Bases: RepresentationBase
    Conjunction
        alias of NumpySet_Conjunction
    patch_classes()
    patch_selector(sel)

class pysubgroup.representations.NumpySet_Conjunction(*args, **kwargs)
    Bases: Conjunction
    all_set = None
    append_and(to_append)
    compute_representation()
    property size_sg

class pysubgroup.representations.RepresentationBase(new_conjunction, selectors_to_patch)
    Bases: object
    patch_all_selectors()
    patch_classes()
    patch_selector(sel)
    undo_patch_classes()

class pysubgroup.representations.SetRepresentation(df, selectors_to_patch)
    Bases: RepresentationBase
    Conjunction
        alias of Set_Conjunction
    patch_classes()
    patch_selector(sel)

class pysubgroup.representations.Set_Conjunction(*args, **kwargs)
    Bases: Conjunction
    all_set = {}
    append_and(to_append)
    compute_representation()
    property size_sg
```


pysubgroup.subgroup_description module

Created on 28.04.2016

@author: lemmerfn

class pysubgroup.subgroup_description.**BooleanExpressionBase**

Bases: [ABC](#)

abstract **append_and**(*to_append*)

abstract **append_or**(*to_append*)

class pysubgroup.subgroup_description.**Conjunction**(*selectors*)

Bases: [BooleanExpressionBase](#)

append_and(*to_append*)

append_or(*to_append*)

covers(*instance*)

property **depth**

static **from_str**(*s*)

pop_and()

pop_or()

property **selectors**

class pysubgroup.subgroup_description.**DNF**(*selectors=None*)

Bases: [Disjunction](#)

append_and(*to_append*)

append_or(*to_append*)

pop_and()

class pysubgroup.subgroup_description.**Disjunction**(*selectors=None*)

Bases: [BooleanExpressionBase](#)

append_and(*to_append*)

append_or(*to_append*)

covers(*instance*)

property **selectors**

class pysubgroup.subgroup_description.**EqualitySelector**(**args*, ***kwargs*)

Bases: [SelectorBase](#)

property **attribute_name**

property **attribute_value**

classmethod **compute_descriptions**(*attribute_name*, *attribute_value*, *selector_name*)

```
covers(data)

static from_str(s)

property selectors

set_descriptions(attribute_name, attribute_value, selector_name=None)

class pysubgroup.subgroup_description.IntervalSelector(*args, **kwargs)
    Bases: SelectorBase
    property attribute_name

    classmethod compute_descriptions(attribute_name, lower_bound, upper_bound,
                                     selector_name=None)

    classmethod compute_string(attribute_name, lower_bound, upper_bound, rounding_digits)

    covers(data_instance)

    static from_str(s)

    property lower_bound

    property selectors

    set_descriptions(attribute_name, lower_bound, upper_bound, selector_name=None)

    property upper_bound

class pysubgroup.subgroup_description.NegatedSelector(*args, **kwargs)
    Bases: SelectorBase
    property attribute_name

    covers(data_instance)

    property selectors

    set_descriptions(selector)

class pysubgroup.subgroup_description.SelectorBase(*args, **kwargs)
    Bases: ABC
    abstract set_descriptions(*args, **kwargs)

pysubgroup.subgroup_description.create_nominal_selectors(data, ignore=None)

pysubgroup.subgroup_description.create_nominal_selectors_for_attribute(data, attribute_name,
                                                                       dtypes=None)

pysubgroup.subgroup_description.create_numeric_selectors(data, nbins=5, intervals_only=True,
                                                         weighting_attribute=None, ignore=None)

pysubgroup.subgroup_description.create_numeric_selectors_for_attribute(data, attr_name,
                                                                       nbins=5,
                                                                       intervals_only=True,
                                                                       weighting_attribute=None)
```

```

pysubgroup.subgroup_description.create_selectors(data, nbins=5, intervals_only=True, ignore=None)
pysubgroup.subgroup_description.get_cover_array_and_size(subgroup, data_len=None, data=None)
pysubgroup.subgroup_description.get_size(subgroup, data_len=None, data=None)
pysubgroup.subgroup_description.remove_target_attributes(selectors, target)

```

pysubgroup.utils module

Created on 02.05.2016

@author: lemmerfn

class pysubgroup.utils.BaseTarget

Bases: `object`

all_statistics_present(cached_statistics)

class pysubgroup.utils.SubgroupDiscoveryResult(results, task)

Bases: `object`

to_dataframe(statistics_to_show=None, autoround=False, include_target=False)

to_descriptions(include_stats=False)

to_latex(statistics_to_show=None)

to_table(statistics_to_show=None, print_header=True, include_target=False)

pysubgroup.utils.add_if_required(result, sg, quality, task, check_for_duplicates=False, statistics=None)

pysubgroup.utils.conditional_invert(val, invert)

pysubgroup.utils.count_bits(bitset_as_int)

pysubgroup.utils.derive_effective_sample_size(weights)

pysubgroup.utils.equal_frequency_discretization(data, attribute_name, nbins=5,
weighting_attribute=None)

pysubgroup.utils.find_set_bits(bitset_as_int)

pysubgroup.utils.float_formatter(x, digits=2)

pysubgroup.utils.intersect_of_ordered_list(list_1, list_2)

pysubgroup.utils.is_categorical_attribute(data, attribute_name)

pysubgroup.utils.is_numerical_attribute(data, attribute_name)

pysubgroup.utils.minimum_required_quality(result, task)

pysubgroup.utils.overlap(sg, another_sg, data)

pysubgroup.utils.perc_formatter(x)

pysubgroup.utils.powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)

`pysubgroup.utils.prepare_subgroup_discovery_result(result, task)`

`pysubgroup.utils.remove_selectors_with_attributes(selector_list, attribute_list)`

`pysubgroup.utils.results_df_around(df)`

`pysubgroup.utils.to_bits(list_of_ints)`

pysubgroup.visualization module

`pysubgroup.visualization.compare_distributions_numeric(sgs, data, bins)`

`pysubgroup.visualization.plot_distribution_numeric(sg, data, bins)`

`pysubgroup.visualization.plot_npSPACE(result_df, data, annotate=True, fixed_limits=False)`

`pysubgroup.visualization.plot_roc(result_df, data, qf=<pysubgroup.binary_target.StandardQF object>, levels=40, annotate=False)`

`pysubgroup.visualization.plot_sgbars(result_df, _, ylabel='target share', title='Discovered Subgroups', dynamic_widths=False, _suffix='')`

`pysubgroup.visualization.similarity_dendrogram(result, data)`

`pysubgroup.visualization.similarity_sgs(sgd_results, data, color=True)`

`pysubgroup.visualization.supportSetVisualization(result, in_order=True, drop_empty=True)`

Module contents

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

- [pysubgroup](#), 40
- [pysubgroup.algorithms](#), 24
- [pysubgroup.binary_target](#), 25
- [pysubgroup.constraints](#), 28
- [pysubgroup.datasets](#), 28
- [pysubgroup.fi_target](#), 28
- [pysubgroup.gp_growth](#), 29
- [pysubgroup.measures](#), 30
- [pysubgroup.model_target](#), 31
- [pysubgroup.numeric_target](#), 32
- [pysubgroup.refinement_operator](#), 35
- [pysubgroup.representations](#), 35
- [pysubgroup.subgroup_description](#), 37
- [pysubgroup.utils](#), 39
- [pysubgroup.visualization](#), 40

A

[a](#) ([pysubgroup.binary_target.StandardQF](#) attribute), 27
[AbstractInterestingnessMeasure](#) (class in [pysubgroup.measures](#)), 30
[add_if_required\(\)](#) (in module [pysubgroup.utils](#)), 39
[add_if_required\(\)](#) ([pysubgroup.gp_growth.GpGrowth](#) method), 29
[all_set](#) ([pysubgroup.representations.NumpySet_Conjunction](#) attribute), 36
[all_set](#) ([pysubgroup.representations.Set_Conjunction](#) attribute), 36
[all_statistics_present\(\)](#) ([pysubgroup.utils.BaseTarget](#) method), 39
[append_and\(\)](#) ([pysubgroup.representations.BitSet_Conjunction](#) method), 35
[append_and\(\)](#) ([pysubgroup.representations.NumpySet_Conjunction](#) method), 36
[append_and\(\)](#) ([pysubgroup.representations.Set_Conjunction](#) method), 36
[append_and\(\)](#) ([pysubgroup.subgroup_description.BooleanExpressionBase](#) method), 37
[append_and\(\)](#) ([pysubgroup.subgroup_description.Conjunction](#) method), 37
[append_and\(\)](#) ([pysubgroup.subgroup_description.Disjunction](#) method), 37
[append_and\(\)](#) ([pysubgroup.subgroup_description.DNF](#) method), 37
[append_or\(\)](#) ([pysubgroup.representations.BitSet_Disjunction](#) method), 36
[append_or\(\)](#) ([pysubgroup.subgroup_description.BooleanExpressionBase](#) method), 37
[append_or\(\)](#) ([pysubgroup.subgroup_description.Conjunction](#) method), 37
[append_or\(\)](#) ([pysubgroup.subgroup_description.Disjunction](#) method), 37
[append_or\(\)](#) ([pysubgroup.subgroup_description.DNF](#)

method), 37
[Apriori](#) (class in [pysubgroup.algorithms](#)), 24
[AreaQF](#) (class in [pysubgroup.fi_target](#)), 28
[attribute_name](#) ([pysubgroup.subgroup_description.EqualitySelector](#) property), 37
[attribute_name](#) ([pysubgroup.subgroup_description.IntervalSelector](#) property), 38
[attribute_name](#) ([pysubgroup.subgroup_description.NegatedSelector](#) property), 38
[attribute_value](#) ([pysubgroup.subgroup_description.EqualitySelector](#) property), 37

B

[BaseTarget](#) (class in [pysubgroup.utils](#)), 39
[BeamSearch](#) (class in [pysubgroup.algorithms](#)), 24
[BestFirstSearch](#) (class in [pysubgroup.algorithms](#)), 24
[beta](#) ([pysubgroup.model_target.beta_tuple](#) attribute), 32
[beta_tuple](#) (class in [pysubgroup.model_target](#)), 32
[BinaryTarget](#) (class in [pysubgroup.binary_target](#)), 25
[BitSet_Conjunction](#) (class in [pysubgroup.representations](#)), 35
[BitSet_Disjunction](#) (class in [pysubgroup.representations](#)), 35
[BitSetRepresentation](#) (class in [pysubgroup.representations](#)), 35
[BooleanExpressionBase](#) (class in [pysubgroup.subgroup_description](#)), 37
[BoundedInterestingnessMeasure](#) (class in [pysubgroup.measures](#)), 30

C

[calculate_constant_statistics\(\)](#) ([pysubgroup.binary_target.SimplePositivesQF](#) method), 27
[calculate_constant_statistics\(\)](#) ([pysubgroup.fi_target.SimpleCountQF](#) method), 29

calculate_constant_statistics()	(pysub-group.measures.CombinedInterestingnessMeasure method), 30	group.binary_target.BinaryTarget method), 25
calculate_constant_statistics()	(pysub-group.measures.GeneralizationAwareQF method), 30	calculate_statistics()
calculate_constant_statistics()	(pysub-group.measures.GeneralizationAwareQF_stats method), 31	group.binary_target.SimplePositivesQF method), 27
calculate_constant_statistics()	(pysub-group.model_target.EMM_Likelihood method), 31	calculate_statistics()
calculate_constant_statistics()	(pysub-group.model_target.PolyRegression_ModelClass method), 32	group.fi_target.FITarget method), 28
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumeric method), 33	calculate_statistics()
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumeric.Average method), 33	group.fi_target.SimpleCountQF method), 29
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumeric.Ordering method), 33	calculate_statistics()
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumeric.Summarization method), 33	group.measures.CombinedInterestingnessMeasure method), 30
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumericMedian method), 34	calculate_statistics()
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumericMedian.Average method), 33	group.measures.CountCallsInterestingnessMeasure method), 30
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumericMedian.Ordering method), 33	calculate_statistics()
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumericMedian.Summarization method), 33	group.measures.GeneralizationAwareQF method), 30
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumericScore method), 35	calculate_statistics()
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumericScore.Average method), 34	group.measures.GeneralizationAwareQF_stats method), 31
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumericScore.Ordering method), 34	calculate_statistics()
calculate_constant_statistics()	(pysub-group.numeric_target.StandardQFNumericScore.Summarization method), 34	group.model_target.EMM_Likelihood method), 31
calculate_quality_function_for_patterns()	(pysubgroup.gp_growth.GpGrowth method), 29	calculate_statistics()
calculate_statistics()	(pysub-	group.numeric_target.NumericTarget method), 32
		calculate_statistics()
		group.numeric_target.StandardQFNumeric method), 33
		calculate_statistics()
		group.numeric_target.StandardQFNumericMedian method), 34
		calculate_statistics()
		group.numeric_target.StandardQFNumericMedian.Average method), 33
		calculate_statistics()
		group.numeric_target.StandardQFNumericMedian.Ordering method), 33
		calculate_statistics()
		group.numeric_target.StandardQFNumericMedian.Summarization method), 33
		calculate_statistics()
		group.numeric_target.StandardQFNumericScore method), 35
		calculate_statistics()
		group.numeric_target.StandardQFNumericScore.Average method), 34
		calculate_statistics()
		group.numeric_target.StandardQFNumericScore.Ordering method), 34
		calculate_statistics()
		group.numeric_target.StandardQFNumericScore.Summarization method), 34
		compare_distributions_numeric()
		(in module py-subgroup.visualization), 40
		compute_descriptions()
		(pysub-
		group.subgroup_description.EqualitySelector

class method), 37
compute_descriptions() (pysubgroup.subgroup_description.IntervalSelector class method), 38
compute_representation() (pysubgroup.representations.BitSet_Conjunction method), 35
compute_representation() (pysubgroup.representations.BitSet_Disjunction method), 36
compute_representation() (pysubgroup.representations.NumpySet_Conjunction method), 36
compute_representation() (pysubgroup.representations.Set_Conjunction method), 36
compute_string() (pysubgroup.subgroup_description.IntervalSelector class method), 38
conditional_invert() (in module pysubgroup.utils), 39
Conjunction (class in pysubgroup.subgroup_description), 37
Conjunction(pysubgroup.representations.BitSetRepresentation attribute), 35
Conjunction(pysubgroup.representations.NumpySetRepresentation attribute), 36
Conjunction(pysubgroup.representations.SetRepresentation attribute), 36
constraints_satisfied() (in module pysubgroup.algorithms), 25
convert_results_to_subgroups() (pysubgroup.gp_growth.GpGrowth method), 29
count_bits() (in module pysubgroup.utils), 39
CountCallsInterestingMeasure (class in pysubgroup.measures), 30
CountQF (class in pysubgroup.fi_target), 28
covers() (pysubgroup.binary_target.BinaryTarget method), 25
covers() (pysubgroup.subgroup_description.Conjunction method), 37
covers() (pysubgroup.subgroup_description.Disjunction method), 37
covers() (pysubgroup.subgroup_description.EqualitySelector method), 37
covers() (pysubgroup.subgroup_description.IntervalSelector method), 38
covers() (pysubgroup.subgroup_description.NegatedSelector method), 38
create_copy_of_path() (pysubgroup.gp_growth.GpGrowth method), 29
create_copy_of_tree_top_down() (pysubgroup.gp_growth.GpGrowth method), 29
create_initial_tree() (pysubgroup.gp_growth.GpGrowth method), 29
create_new_tree_from_nodes() (pysubgroup.gp_growth.GpGrowth method), 29
create_nominal_selectors() (in module pysubgroup.subgroup_description), 38
create_nominal_selectors_for_attribute() (in module pysubgroup.subgroup_description), 38
create_numeric_selectors() (in module pysubgroup.subgroup_description), 38
create_numeric_selectors_for_attribute() (in module pysubgroup.subgroup_description), 38
create_selectors() (in module pysubgroup.subgroup_description), 38
D
depth (pysubgroup.subgroup_description.Conjunction property), 37
derive_effective_sample_size() (in module pysubgroup.utils), 39
DFS (class in pysubgroup.algorithms), 24
DFSNumeric (class in pysubgroup.algorithms), 25
Disjunction (class in pysubgroup.subgroup_description), 37
Disjunction(pysubgroup.representations.BitSetRepresentation attribute), 35
Disjunction(pysubgroup.representations.NumpySetRepresentation attribute), 36
Disjunction(pysubgroup.representations.SetRepresentation attribute), 36
E
EMM_Likelihood (class in pysubgroup.model_target), 31
ensure_statistics() (pysubgroup.measures.AbstractInterestingnessMeasure method), 30
equal_frequency_discretization() (in module pysubgroup.utils), 39
EqualitySelector (class in pysubgroup.subgroup_description), 37
evaluate() (pysubgroup.binary_target.ChiSquaredQF method), 26
evaluate() (pysubgroup.binary_target.GeneralizationAware_StandardQF method), 26
evaluate() (pysubgroup.binary_target.StandardQF method), 27
evaluate() (pysubgroup.fi_target.AreaQF method), 28
evaluate() (pysubgroup.fi_target.CountQF method), 28
evaluate() (pysubgroup.measures.CombinedInterestingnessMeasure method), 30
evaluate() (pysubgroup.measures.GeneralizationAwareQF method), 31
evaluate() (pysubgroup.measures.GeneralizationAwareQF_stats method), 31
evaluate() (pysubgroup.model_target.EMM_Likelihood method), 31
evaluate() (pysubgroup.numeric_target.StandardQFNumeric method), 33

[evaluate\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericMedian](#) method), 34
[evaluate\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericTscore](#) method), 28
[evaluate\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericTscore](#) method), 35
[evaluate_from_statistics\(\)](#) ([pysubgroup.measures.CombinedInterestingnessMeasure](#) method), 30
[execute\(\)](#) ([pysubgroup.algorithms.Apriori](#) method), 24
[execute\(\)](#) ([pysubgroup.algorithms.BeamSearch](#) method), 24
[execute\(\)](#) ([pysubgroup.algorithms.BestFirstSearch](#) method), 24
[execute\(\)](#) ([pysubgroup.algorithms.DFS](#) method), 25
[execute\(\)](#) ([pysubgroup.algorithms.DFSNumeric](#) method), 25
[execute\(\)](#) ([pysubgroup.algorithms.GeneralisingBFS](#) method), 25
[execute\(\)](#) ([pysubgroup.algorithms.SimpleDFS](#) method), 25
[execute\(\)](#) ([pysubgroup.algorithms.SimpleSearch](#) method), 25
[execute\(\)](#) ([pysubgroup.gp_growth.GpGrowth](#) method), 29

F

[find_set_bits\(\)](#) (in module [pysubgroup.utils](#)), 39
[fit\(\)](#) ([pysubgroup.model_target.PolyRegression_ModelClass](#) method), 32
[FITarget](#) (class in [pysubgroup.fi_target](#)), 28
[float_formatter\(\)](#) (in module [pysubgroup.utils](#)), 39
[from_str\(\)](#) ([pysubgroup.subgroup_description.Conjunction](#) static method), 37
[from_str\(\)](#) ([pysubgroup.subgroup_description.EqualitySelector](#) static method), 38
[from_str\(\)](#) ([pysubgroup.subgroup_description.IntervalSelector](#) static method), 38

G

[ga_tuple](#) ([pysubgroup.measures.GeneralizationAwareQF_stats](#) attribute), 31
[generalisation_quality](#) ([pysubgroup.measures.GeneralizationAwareQF.ga_tuple](#) attribute), 31
[GeneralisingBFS](#) (class in [pysubgroup.algorithms](#)), 25
[GeneralizationAware_StandardQF](#) (class in [pysubgroup.binary_target](#)), 26
[GeneralizationAwareQF](#) (class in [pysubgroup.measures](#)), 30
[GeneralizationAwareQF.ga_tuple](#) (class in [pysubgroup.measures](#)), 31
[GeneralizationAwareQF_stats](#) (class in [pysubgroup.measures](#)), 31
[get_attributes\(\)](#) ([pysubgroup.binary_target.BinaryTarget](#) method),
 [get_attributes\(\)](#) ([pysubgroup.fi_target.FITarget](#) method),
 [get_attributes\(\)](#) ([pysubgroup.numeric_target.NumericTarget](#) method), 32
[get_base_statistics\(\)](#) ([pysubgroup.binary_target.BinaryTarget](#) method), 25
[get_base_statistics\(\)](#) ([pysubgroup.fi_target.FITarget](#) method), 28
[get_base_statistics\(\)](#) ([pysubgroup.numeric_target.NumericTarget](#) method), 32
[get_cover_array_and_size\(\)](#) (in module [pysubgroup.subgroup_description](#)), 39
[get_credit_data\(\)](#) (in module [pysubgroup.datasets](#)), 28
[get_data\(\)](#) ([pysubgroup.numeric_target.StandardQFNumeric.Average_Estimator](#) method), 33
[get_data\(\)](#) ([pysubgroup.numeric_target.StandardQFNumeric.Ordering_Estimator](#) method), 33
[get_data\(\)](#) ([pysubgroup.numeric_target.StandardQFNumeric.Sumation_Estimator](#) method), 33
[get_data\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericMedian.Average_Estimator](#) method), 33
[get_data\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericMedian.Ordering_Estimator](#) method), 34
[get_data\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericMedian.Sumation_Estimator](#) method), 34
[get_data\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericTscore.Average_Estimator](#) method), 34
[get_data\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericTscore.Ordering_Estimator](#) method), 34
[get_data\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericTscore.Sumation_Estimator](#) method), 34
[get_estimate\(\)](#) ([pysubgroup.numeric_target.StandardQFNumeric.Average_Estimator](#) method), 33
[get_estimate\(\)](#) ([pysubgroup.numeric_target.StandardQFNumeric.Ordering_Estimator](#) method), 33
[get_estimate\(\)](#) ([pysubgroup.numeric_target.StandardQFNumeric.Sumation_Estimator](#) method), 33
[get_estimate\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericMedian.Average_Estimator](#) method), 33
[get_estimate\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericMedian.Ordering_Estimator](#) method), 34
[get_estimate\(\)](#) ([pysubgroup.numeric_target.StandardQFNumericMedian.Sumation_Estimator](#) method), 34

<code>get_estimate()</code>	(pysub- group.numeric_target.StandardQFNumericTscore.Average_Estimation method), 34	<code>gp_get_null_vector()</code>	(pysub- group.model_target.PolyRegression_ModelClass method), 32
<code>get_estimate()</code>	(pysub- group.numeric_target.StandardQFNumericTscore.Ordering_Estimation method), 34	<code>gp_get_params()</code>	(pysub- group.binary_target.SimplePositivesQF method), 27
<code>get_estimate()</code>	(pysub- group.numeric_target.StandardQFNumericTscore.Summation_Estimation method), 34	<code>gp_get_params()</code>	(pysub- group.fi_target.SimpleCountQF method), 29
<code>get_estimate_numpy()</code>	(pysub- group.numeric_target.StandardQFNumeric.Ordering_Estimation method), 33	<code>gp_get_params()</code>	(pysub- group.model_target.EMM_Likelihood method), 31
<code>get_estimate_numpy()</code>	(pysub- group.numeric_target.StandardQFNumericMedian.Ordering_Estimation method), 34	<code>gp_get_params()</code>	(pysub- group.model_target.PolyRegression_ModelClass method), 32
<code>get_estimate_numpy()</code>	(pysub- group.numeric_target.StandardQFNumericTscore.Ordering_Estimation method), 34	<code>gp_get_stats()</code>	(pysub- group.binary_target.SimplePositivesQF method), 27
<code>get_max()</code> (pysubgroup.binary_target.GeneralizationAwareQF method), 26	<code>gp_get_stats()</code>	<code>gp_get_stats()</code>	(pysubgroup.fi_target.SimpleCountQF method), 29
<code>get_max()</code> (pysubgroup.measures.GeneralizationAwareQF method), 31	<code>gp_get_stats()</code>	<code>gp_get_stats()</code>	(pysub- group.model_target.PolyRegression_ModelClass method), 32
<code>get_next_level()</code> (pysubgroup.algorithms.Apriori method), 24	<code>gp_is_satisfied()</code>	<code>gp_is_satisfied()</code>	(pysub- group.constraints.MinSupportConstraint method), 28
<code>get_next_level_candidates()</code> (pysub- group.algorithms.Apriori method), 24	<code>gp_merge()</code>	<code>gp_merge()</code>	(pysubgroup.binary_target.SimplePositivesQF method), 27
<code>get_next_level_candidates_vectorized()</code> (py- subgroup.algorithms.Apriori method), 24	<code>gp_merge()</code>	<code>gp_merge()</code>	(pysubgroup.fi_target.SimpleCountQF method), 29
<code>get_next_level_numba()</code> (pysub- group.algorithms.Apriori method), 24	<code>gp_merge()</code>	<code>gp_merge()</code>	(pysubgroup.model_target.PolyRegression_ModelClass static method), 32
<code>get_nodes_upwards()</code> (pysub- group.gp_growth.GpGrowth method), 29	<code>gp_prepare()</code>	<code>gp_prepare()</code>	(pysub- group.constraints.MinSupportConstraint method), 28
<code>get_qual_and_previous_qual()</code> (pysub- group.measures.GeneralizationAwareQF method), 31	<code>gp_requires_cover_arr</code>	<code>gp_requires_cover_arr</code>	(pysub- group.binary_target.SimplePositivesQF property), 27
<code>get_size()</code> (in module pysub- group.subgroup_description), 39	<code>gp_requires_cover_arr</code>	<code>gp_requires_cover_arr</code>	(pysub- group.fi_target.SimpleCountQF attribute), 29
<code>get_stats_and_previous_stats()</code> (pysub- group.measures.GeneralizationAwareQF_stats method), 31	<code>gp_requires_cover_arr</code>	<code>gp_requires_cover_arr</code>	(pysub- group.model_target.EMM_Likelihood prop- erty), 31
<code>get_stats_for_class()</code> (pysub- group.gp_growth.GpGrowth method), 29	<code>gp_requires_cover_arr</code>	<code>gp_requires_cover_arr</code>	(pysub- group.model_target.PolyRegression_ModelClass property), 32
<code>get_titanic_data()</code> (in module pysubgroup.datasets), 28	<code>gp_size_sg()</code>	<code>gp_size_sg()</code>	(pysub- group.binary_target.SimplePositivesQF method), 27
<code>get_top_down_tree_for_class()</code> (pysub- group.gp_growth.GpGrowth method), 29	<code>gp_size_sg()</code>	<code>gp_size_sg()</code>	(pysubgroup.fi_target.SimpleCountQF method), 29
<code>get_tuple()</code> (pysubgroup.model_target.EMM_Likelihood method), 31	<code>gp_size_sg()</code>	<code>gp_size_sg()</code>	(pysub- group.model_target.PolyRegression_ModelClass method), 29
<code>gp_get_null_vector()</code> (pysub- group.binary_target.SimplePositivesQF method), 27	<code>gp_size_sg()</code>	<code>gp_size_sg()</code>	(pysub- group.model_target.PolyRegression_ModelClass method), 29
<code>gp_get_null_vector()</code> (pysub- group.fi_target.SimpleCountQF method), 29	<code>gp_size_sg()</code>	<code>gp_size_sg()</code>	(pysub- group.model_target.PolyRegression_ModelClass method), 29

- method*), 32
- `gp_to_str()` (*pysubgroup.binary_target.SimplePositivesQF*
method), 27
- `gp_to_str()` (*pysubgroup.fi_target.SimpleCountQF*
method), 29
- `gp_to_str()` (*pysubgroup.model_target.PolyRegression_ModelClass*
method), 32
- `GpGrowth` (*class in pysubgroup.gp_growth*), 29
- I**
- `identity()` (*in module pysubgroup.gp_growth*), 30
- `intersect_of_ordered_list()` (*in module pysub-*
group.utils), 39
- `IntervalSelector` (*class in pysub-*
group.subgroup_description), 38
- `is_categorical_attribute()` (*in module pysub-*
group.utils), 39
- `is_monotone` (*pysubgroup.constraints.MinSupportConstraint*
property), 28
- `is_numerical_attribute()` (*in module pysub-*
group.utils), 39
- `is_satisfied()` (*pysub-*
group.constraints.MinSupportConstraint
method), 28
- L**
- `LiftQF` (*class in pysubgroup.binary_target*), 27
- `likelihood()` (*pysub-*
group.model_target.PolyRegression_ModelClass
method), 32
- `loglikelihood()` (*pysub-*
group.model_target.PolyRegression_ModelClass
method), 32
- `lower_bound` (*pysubgroup.subgroup_description.IntervalSelector*
property), 38
- M**
- `maximum_statistic_filter()` (*in module pysub-*
group.measures), 31
- `merge_trees_top_down()` (*pysub-*
group.gp_growth.GpGrowth method), 29
- `minimum_quality_filter()` (*in module pysub-*
group.measures), 31
- `minimum_required_quality()` (*in module pysub-*
group.utils), 39
- `minimum_statistic_filter()` (*in module pysub-*
group.measures), 31
- `MinSupportConstraint` (*class in pysub-*
group.constraints), 28
- `module`
 - `pysubgroup`, 40
 - `pysubgroup.algorithms`, 24
 - `pysubgroup.binary_target`, 25
 - `pysubgroup.constraints`, 28
 - `pysubgroup.datasets`, 28
 - `pysubgroup.fi_target`, 28
 - `pysubgroup.gp_growth`, 29
 - `pysubgroup.measures`, 30
 - `pysubgroup.model_target`, 31
 - `pysubgroup.numeric_target`, 32
 - `pysubgroup.refinement_operator`, 35
 - `pysubgroup.representations`, 35
 - `pysubgroup.subgroup_description`, 37
 - `pysubgroup.utils`, 39
 - `pysubgroup.visualization`, 40
- N**
- `n_instances` (*pysubgroup.representations.BitSet_Conjunction*
attribute), 35
- `NegatedSelector` (*class in pysub-*
group.subgroup_description), 38
- `nodes_to_cls_nodes()` (*pysub-*
group.gp_growth.GpGrowth method), 29
- `normal_insert()` (*pysubgroup.gp_growth.GpGrowth*
method), 30
- `NumericTarget` (*class in pysubgroup.numeric_target*),
32
- `NumpySet_Conjunction` (*class in pysub-*
group.representations), 36
- `NumpySetRepresentation` (*class in pysub-*
group.representations), 36
- O**
- `optimistic_estimate()` (*pysub-*
group.binary_target.StandardQF method),
27
- `optimistic_estimate()` (*pysub-*
group.fi_target.CountQF method), 28
- `optimistic_estimate()` (*pysub-*
group.measures.CombinedInterestingnessMeasure
method), 30
- `optimistic_estimate()` (*pysub-*
group.numeric_target.StandardQFNumeric
method), 33
- `optimistic_estimate()` (*pysub-*
group.numeric_target.StandardQFNumericMedian
method), 34
- `optimistic_estimate()` (*pysub-*
group.numeric_target.StandardQFNumericTscore
method), 35
- `optimistic_generalisation()` (*pysub-*
group.binary_target.StandardQF method),
27
- `overlap()` (*in module pysubgroup.utils*), 39
- `overlap_filter()` (*in module pysubgroup.measures*),
31
- `overlaps_list()` (*in module pysubgroup.measures*), 31

P

[patch_all_selectors\(\)](#) (pysubgroup.representations.RepresentationBase method), 36
[patch_classes\(\)](#) (pysubgroup.representations.BitSetRepresentation method), 35
[patch_classes\(\)](#) (pysubgroup.representations.NumpySetRepresentation method), 36
[patch_classes\(\)](#) (pysubgroup.representations.RepresentationBase method), 36
[patch_classes\(\)](#) (pysubgroup.representations.SetRepresentation method), 36
[patch_selector\(\)](#) (pysubgroup.representations.BitSetRepresentation method), 35
[patch_selector\(\)](#) (pysubgroup.representations.NumpySetRepresentation method), 36
[patch_selector\(\)](#) (pysubgroup.representations.RepresentationBase method), 36
[patch_selector\(\)](#) (pysubgroup.representations.SetRepresentation method), 36
[perc_formatter\(\)](#) (in module pysubgroup.utils), 39
[plot_distribution_numeric\(\)](#) (in module pysubgroup.visualization), 40
[plot_npSPACE\(\)](#) (in module pysubgroup.visualization), 40
[plot_roC\(\)](#) (in module pysubgroup.visualization), 40
[plot_sgbars\(\)](#) (in module pysubgroup.visualization), 40
[PolyRegression_ModelClass](#) (class in pysubgroup.model_target), 32
[pop_and\(\)](#) (pysubgroup.subgroup_description.Conjunction method), 37
[pop_and\(\)](#) (pysubgroup.subgroup_description.DNF method), 37
[pop_or\(\)](#) (pysubgroup.subgroup_description.Conjunction method), 37
[powerset\(\)](#) (in module pysubgroup.utils), 39
[prepare_selectors\(\)](#) (pysubgroup.gp_growth.GpGrowth method), 30
[prepare_subgroup_discovery_result\(\)](#) (in module pysubgroup.utils), 39
[pysubgroup](#)
[module](#), 40
[pysubgroup.algorithms](#)
[module](#), 24
[pysubgroup.binary_target](#)

[module](#), 25
[pysubgroup.constraints](#)
[module](#), 28
[pysubgroup.datasets](#)
[module](#), 28
[pysubgroup.fi_target](#)
[module](#), 28
[pysubgroup.gp_growth](#)
[module](#), 29
[pysubgroup.measures](#)
[module](#), 30
[pysubgroup.model_target](#)
[module](#), 31
[pysubgroup.numeric_target](#)
[module](#), 32
[pysubgroup.refinement_operator](#)
[module](#), 35
[pysubgroup.representations](#)
[module](#), 35
[pysubgroup.subgroup_description](#)
[module](#), 37
[pysubgroup.utils](#)
[module](#), 39
[pysubgroup.visualization](#)
[module](#), 40

R

[recurse\(\)](#) (pysubgroup.gp_growth.GpGrowth method), 30
[recurse_top_down\(\)](#) (pysubgroup.gp_growth.GpGrowth method), 30
[RefinementOperator](#) (class in pysubgroup.refinement_operator), 35
[refinements\(\)](#) (pysubgroup.refinement_operator.StaticGeneralizationOperator method), 35
[refinements\(\)](#) (pysubgroup.refinement_operator.StaticSpecializationOperator method), 35
[remove_selectors_with_attributes\(\)](#) (in module pysubgroup.utils), 40
[remove_selectors_with_low_optimistic_estimate\(\)](#) (pysubgroup.gp_growth.GpGrowth method), 30
[remove_target_attributes\(\)](#) (in module pysubgroup.subgroup_description), 39
[RepresentationBase](#) (class in pysubgroup.representations), 36
[results_df_autoround\(\)](#) (in module pysubgroup.utils), 40

S

[search_internal\(\)](#) (pysubgroup.algorithms.DFS method), 25

`search_internal()` (pysubgroup.algorithms.DFSNumeric method), 25
`search_internal()` (pysubgroup.algorithms.SimpleDFS method), 25
`SelectorBase` (class in pysubgroup.subgroup_description), 38
`selectors` (pysubgroup.subgroup_description.Conjunction property), 37
`selectors` (pysubgroup.subgroup_description.Disjunction property), 37
`selectors` (pysubgroup.subgroup_description.EqualitySelector property), 38
`selectors` (pysubgroup.subgroup_description.IntervalSelector property), 38
`selectors` (pysubgroup.subgroup_description.NegatedSelector property), 38
`Set_Conjunction` (class in pysubgroup.representations), 36
`set_descriptions()` (pysubgroup.subgroup_description.EqualitySelector method), 38
`set_descriptions()` (pysubgroup.subgroup_description.IntervalSelector method), 38
`set_descriptions()` (pysubgroup.subgroup_description.NegatedSelector method), 38
`set_descriptions()` (pysubgroup.subgroup_description.SelectorBase method), 38
`SetRepresentation` (class in pysubgroup.representations), 36
`setup()` (pysubgroup.gp_growth.GpGrowth method), 30
`setup_constraints()` (pysubgroup.gp_growth.GpGrowth method), 30
`setup_from_quality_function()` (pysubgroup.gp_growth.GpGrowth method), 30
`similarity_dendrogram()` (in module pysubgroup.visualization), 40
`similarity_sgs()` (in module pysubgroup.visualization), 40
`SimpleBinomialQF` (class in pysubgroup.binary_target), 27
`SimpleCountQF` (class in pysubgroup.fi_target), 29
`SimpleDFS` (class in pysubgroup.algorithms), 25
`SimplePositivesQF` (class in pysubgroup.binary_target), 27
`SimpleSearch` (class in pysubgroup.algorithms), 25
`size_sg` (pysubgroup.model_target.beta_tuple attribute), 32
`size_sg` (pysubgroup.representations.BitSet_Conjunction property), 35
`size_sg` (pysubgroup.representations.BitSet_Disjunction property), 36
`size_sg` (pysubgroup.representations.NumpySet_Conjunction property), 36
`size_sg` (pysubgroup.representations.Set_Conjunction property), 36
`standard_qf()` (pysubgroup.binary_target.StandardQF static method), 27
`standard_qf_numeric()` (pysubgroup.numeric_target.StandardQFNumeric static method), 33
`standard_qf_numeric()` (pysubgroup.numeric_target.StandardQFNumericMedian static method), 34
`standard_qf_numeric()` (pysubgroup.numeric_target.StandardQFNumericTscore static method), 35
`StandardQF` (class in pysubgroup.binary_target), 27
`StandardQFNumeric` (class in pysubgroup.numeric_target), 32
`StandardQFNumeric.Average_Estimator` (class in pysubgroup.numeric_target), 33
`StandardQFNumeric.Ordering_Estimator` (class in pysubgroup.numeric_target), 33
`StandardQFNumeric.Summation_Estimator` (class in pysubgroup.numeric_target), 33
`StandardQFNumericMedian` (class in pysubgroup.numeric_target), 33
`StandardQFNumericMedian.Average_Estimator` (class in pysubgroup.numeric_target), 33
`StandardQFNumericMedian.Ordering_Estimator` (class in pysubgroup.numeric_target), 33
`StandardQFNumericMedian.Summation_Estimator` (class in pysubgroup.numeric_target), 34
`StandardQFNumericTscore` (class in pysubgroup.numeric_target), 34
`StandardQFNumericTscore.Average_Estimator` (class in pysubgroup.numeric_target), 34
`StandardQFNumericTscore.Ordering_Estimator` (class in pysubgroup.numeric_target), 34
`StandardQFNumericTscore.Summation_Estimator` (class in pysubgroup.numeric_target), 34
`StaticGeneralizationOperator` (class in pysubgroup.refinement_operator), 35
`StaticSpecializationOperator` (class in pysubgroup.refinement_operator), 35
`statistic_types` (pysubgroup.binary_target.BinaryTarget attribute), 26
`statistic_types` (pysubgroup.fi_target.FITarget attribute), 28
`statistic_types` (pysubgroup.numeric_target.NumericTarget attribute), 32
`subgroup_quality` (pysub-

group.measures.GeneralizationAwareQF.ga_tuple
attribute), 31

SubgroupDiscoveryResult (class in *pysubgroup.utils*),
39

SubgroupDiscoveryTask (class in *pysub-*
group.algorithms), 25

supportSetVisualization() (in module *pysub-*
group.visualization), 40

T

to_bits() (in module *pysubgroup.utils*), 40

to_dataframe() (*pysub-*
group.utils.SubgroupDiscoveryResult method),
39

to_descriptions() (*pysub-*
group.utils.SubgroupDiscoveryResult method),
39

to_file() (*pysubgroup.gp_growth.GpGrowth* method),
30

to_latex() (*pysubgroup.utils.SubgroupDiscoveryResult*
method), 39

to_table() (*pysubgroup.utils.SubgroupDiscoveryResult*
method), 39

tpl (*pysubgroup.algorithms.DFSNumeric* attribute), 25

tpl (*pysubgroup.binary_target.SimplePositivesQF*
attribute), 27

tpl (*pysubgroup.fi_target.SimpleCountQF* attribute), 29

tpl (*pysubgroup.model_target.EMM_Likelihood* at-
tribute), 32

tpl (*pysubgroup.numeric_target.StandardQFNumeric* at-
tribute), 33

tpl (*pysubgroup.numeric_target.StandardQFNumericMedian*
attribute), 34

tpl (*pysubgroup.numeric_target.StandardQFNumericTscore*
attribute), 35

U

undo_patch_classes() (*pysub-*
group.representations.RepresentationBase
method), 36

unique_attributes() (in module *pysub-*
group.measures), 31

upper_bound (*pysubgroup.subgroup_description.IntervalSelector*
property), 38

W

WRaccQF (class in *pysubgroup.binary_target*), 28