# pysubgroup Documentation

***Release stable***

**mgbckr**

**Oct 27, 2025**

# CONTENTS

**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.

CHAPTER

# ONE

# 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.DFS().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 depth first 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
- SubgroupDicoveryResult: These are the main outcome of a subgroup disovery 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 am 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 Warning

- GP-growth is in an experimental stage.

### 1.1.6 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},
```

(continues on next page)

(continued from previous page)

```
  pages={658--662},
  year={2018}
}
```

### 1.1.7 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**



#### 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.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.GpGrowth().execute(task)
```

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='b_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 $A \cap B = \varnothing$ 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 funciton 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
    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 outpu 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
```

(continues on next page)

(continued from previous page)

```
    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 simple 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 $\alpha$ 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 contigency table which is then passed to the scipy chi2_contigency 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 othe 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 $\mu_{SG}$ and $\mu$ 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 will 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
        """
        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] 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.

### 1.3.1 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.

### 1.3.2 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 and [Conda-Forge], the following steps can be used to release a new version for `pysubgroup`:

#### PyPI

1. Merge `master` branch into `develop`.
2. Make sure all unit tests are successful on `develop`.
3. Create a pull request from the `develop` to the `master` branch.
4. Merge the pull request after all tests have passed.
5. Tag the current commit on the main branch (`master`) with a release tag, e.g., `git tag -a 0.7.7 -m "Release 0.7.7`.
6. Push the new tag to the upstream, e.g., `git push --tags`
7. GitHub Actions will now automatically push this version to
8. 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.
9. The

#### Conda Forge

TODO: Automate this with GitHub Actions?

Resources:

- Conda: Contributing packages

1. **Fork** and clone pysubgroup-feedstock
2. Create a branch with the current version number, e.g., `git branch 0.7.7; git checkout 0.7.7`.
3. Get the `sha256` hash from PyPI, i.e., click on `view hashes` of the `Source Distribution`.
4. Update the `recipe/meta.yaml`:

```
{% set version = "0.7.7" %}

# ...

source:
  sha256: <SHA256 from PyPI>
```

5. Commit and push branch
6. Create a pull request on main repository. For this, make sure to fill in all the fields and use closing tags for changes.
7. Merge the pull requests when all tests have successfully passed.

## 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.

      "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
```

(continues on next page)

(continued from previous page)

```
    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,
    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.
```

(continues on next page)

(continued from previous page)

```
   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.

   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,
```

(continues on next page)

(continued from previous page)

```
   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
   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");
```

(continues on next page)

(continued from previous page)

```
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 Main Contributors

- Florian Lemmerich (flemmerich) florian@lemmerich.net
- Martin Becker (mgbckr) mgbckr@informatik.uni-rostock.de
- Felix Stamm (Feelx234)

## 1.6 Changelog

### 1.6.1 [0.9.0] - 2024-xx-xx

- dropped support for Python 3.6 and 3.7
- Support boolean attribute values in `EqualitySelector.from_str()`

### 1.6.2 [0.7.7] - 2020-05-20

#### Fixed

- BeamSearch now correctly uses `beam_width` parameter

### 1.6.3 [0.7.6] - 2020-05-20

Some internal changes to the continuous integration pipeline on top of version 0.7.6.

### 1.6.4 [0.7.5] - 2020-05-20

Moved to pyscaffold, src/test structure and GitHub Actions.

### 1.6.5 [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 seperate 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 lign with the corresponding name shema for nominal.

#### Changed internally

- statistics are now also store along with score and description
- The function `ps.get_cover_array_and_size` was added, it allows for a consistent way to acces 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 lign 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.

### 1.6.6 [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

### 1.6.7 [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

### 1.6.8 [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, Disjuntion
    - __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

Implementation of the Apriori algorithm for subgroup discovery.

This class provides methods to perform level-wise search for subgroups using the Apriori algorithm.

**execute**(*task*)

Executes the Apriori algorithm on the given task.

**Parameters**
**task** – The subgroup discovery task to be executed.

**Returns**
A SubgroupDiscoveryResult containing the discovered subgroups.

**get_next_level**(*promising_candidates*)

Generates the next level of candidates based on the current promising candidates.

**Parameters**
**promising_candidates** – A list of promising candidate selectors.

**Returns**
A list of new candidate selectors for the next level.

**get_next_level_candidates**(*task*, *result*, *next_level_candidates*)

Evaluates candidates at the current level and filters promising ones for the next level.

**Parameters**

- **task** – The subgroup discovery task.
- **result** – The current list of discovered subgroups.
- **next_level_candidates** – List of subgroups to be evaluated at the current
- **level.**

**Returns**
A list of promising candidates (selectors) for the next level.

**get_next_level_candidates_vectorized**(*task*, *result*, *next_level_candidates*)

Vectorized evaluation of candidates at the current level to filter promising ones for the next level.

**Parameters**

- **task** – The subgroup discovery task.
- **result** – The current list of discovered subgroups.
- **next_level_candidates** – List of subgroups to be evaluated at the current
- **level.**

**Returns**

A list of promising candidates (selectors) for the next level.

**get_next_level_numba**(*promising_candidates*)

Generates the next level of candidates using Numba for acceleration.

**Parameters**

**promising_candidates** – A list of promising candidate selectors.

**Returns**

A list of new candidate selectors for the next level.

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

Bases: object

Implements the Beam Search algorithm for subgroup discovery.

**execute**(*task*)

Executes the Beam Search algorithm on the given task.

**Parameters**

**task** – The subgroup discovery task to be executed.

**Returns**

A SubgroupDiscoveryResult containing the discovered subgroups.

**class** pysubgroup.algorithms.**BestFirstSearch**

Bases: object

Implements the Best-First Search algorithm for subgroup discovery.

**execute**(*task*)

Executes the Best-First Search algorithm on the given task.

**Parameters**

**task** – The subgroup discovery task to be executed.

**Returns**

A SubgroupDiscoveryResult containing the discovered subgroups.

**class** pysubgroup.algorithms.**DFS**(*apply_representation=None*)

Bases: object

Depth-first search with look-ahead for a provided data structure.

**execute**(*task*)

Executes the DFS algorithm on the given task.

**Parameters**

**task** – The subgroup discovery task to be executed.

**Returns**
A SubgroupDiscoveryResult containing the discovered subgroups.

`search_internal`(*task*, *result*, *sg*)

Recursively searches for subgroups in a depth-first manner.

**Parameters**

- **task** – The subgroup discovery task.
- **result** – The current list of discovered subgroups.
- **sg** – The current subgroup being evaluated.

class pysubgroup.algorithms.**DFSNumeric**

Bases: object

Implements a specialized DFS algorithm for numeric quality functions.

`execute`(*task*)

Executes the DFSNumeric algorithm on the given task.

**Parameters**
**task** – The subgroup discovery task to be executed.

**Returns**
A SubgroupDiscoveryResult containing the discovered subgroups.

`search_internal`(*task*, *prefix*, *modification_set*, *result*, *bitset*)

Recursively searches in a dfs-manner for numeric quality functions.

**Parameters**

- **task** – The subgroup discovery task.
- **prefix** – The current list of selectors in the subgroup description.
- **modification_set** – The remaining selectors to consider.
- **result** – The current list of discovered subgroups.
- **bitset** – The current bitset representing the subgroup.

**Returns**
The updated list of discovered subgroups.

`tpl`

alias of `size_mean_parameters`

class pysubgroup.algorithms.**GeneralisingBFS**

Bases: object

Implements a Generalizing Best-First Search algorithm for subgroup discovery.

`execute`(*task*)

Executes the Generalizing Best-First Search algorithm on the given task.

**Parameters**
**task** – The subgroup discovery task to be executed.

**Returns**
A SubgroupDiscoveryResult containing the discovered subgroups.

class pysubgroup.algorithms.**SimpleDFS**

Bases: object

Implements a simple Depth-First Search algorithm for subgroup discovery. It is the most elementary (and thus probably slow) algorithm implementation.

**execute**(*task*, *use_optimistic_estimates=True*)

Executes the Simple DFS algorithm on the given task.

**Parameters**

- **task** – The subgroup discovery task to be executed.
- **use_optimistic_estimates** – Whether to use optimistic estimates for pruning.

**Returns**
A SubgroupDiscoveryResult containing the discovered subgroups.

**search_internal**(*task*, *prefix*, *modification_set*, *result*, *use_optimistic_estimates*)

Recursively searches for subgroups in a depth-first manner.

**Parameters**

- **task** – The subgroup discovery task.
- **prefix** – The current list of selectors in the subgroup description.
- **modification_set** – The remaining selectors to consider.
- **result** – The current list of discovered subgroups.
- **use_optimistic_estimates** – Whether to use optimistic estimates for pruning.

**Returns**
The updated list of discovered subgroups.

class pysubgroup.algorithms.**SimpleSearch**(*show_progress=True*)

Bases: object

Implements a simple exhaustive search algorithm for subgroup discovery.

**execute**(*task*)

Executes the Simple Search algorithm on the given task.

**Parameters**
**task** – The subgroup discovery task to be executed.

**Returns**
A SubgroupDiscoveryResult containing the discovered subgroups.

class pysubgroup.algorithms.**SubgroupDiscoveryTask**(*data*, *target*, *search_space*, *qf*, *result_set_size=10*, *depth=3*, *min_quality=-inf*, *constraints=None*)

Bases: object

Encapsulates all parameters required to perform standard subgroup discovery.

pysubgroup.algorithms.**constraints_satisfied**(*constraints*, *subgroup*, *statistics=None*, *data=None*)

Checks if all constraints are satisfied for a given subgroup.

**Parameters**

- **constraints** – A list of constraints to check.
- **subgroup** – The subgroup to be evaluated.

- **statistics** – Precomputed statistics for the subgroup (optional).
- **data** – The dataset to be analyzed (optional).

**Returns**
True if all constraints are satisfied, False otherwise.

### 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*

Binary target for classic subgroup discovery with boolean targets.

Stores the target attribute and value, and computes various statistics related to the target within a subgroup.

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

Calculate various statistics for the subgroup.

**Parameters**

- **subgroup** – The subgroup for which to calculate statistics.
- **data** (*pandas DataFrame*) – The dataset.
- **cached_statistics** (*dict, optional*) – Previously computed statistics.

**Returns**
A dictionary containing various statistical measures.

**Return type**
dict

**covers**(*instance*)

Determine whether the target selector covers the given instance.

**Parameters**
**instance** (*pandas DataFrame*) – The data instance to check.

**Returns**
Boolean array indicating coverage.

**Return type**
numpy.ndarray

**get_attributes**()

Get the attribute names used in the target.

**Returns**
A tuple containing the attribute name.

**Return type**
tuple

**get_base_statistics**(*subgroup*, *data*)

Compute basic statistics for the target within the subgroup and dataset.

**Parameters**

- **subgroup** – The subgroup for which to compute statistics.
- **data** (*pandas DataFrame*) – The dataset.

**Returns**

**Contains instances_dataset, positives_dataset,**
instances_subgroup, positives_subgroup.

**Return type**
tuple

```
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 tests for statistical independence of a subgroup against its complement.

Calculates the chi-squared statistic or p-value to measure the significance of the difference between the subgroup and the dataset.

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

Perform chi-squared test of statistical independence.

Tests whether a subgroup is statistically independent from its complement (see scipy.stats.chi2_contingency).

**Parameters**

- **instances_dataset** (*int*) – Total number of instances in the dataset.
- **positives_dataset** (*int*) – Total number of positive instances in the dataset.
- **instances_subgroup** (*int*) – Number of instances in the subgroup.
- **positives_subgroup** (*int*) – Number of positive instances in the subgroup.
- **min_instances** (*int, optional*) – Minimum required instances; return -inf if less.
- **bidirect** (*bool, optional*) – If True, both directions are considered interesting.
- **direction_positive** (*bool, optional*) – If bidirect is False, specifies the direction.
- **index** (*int, optional*) – Whether to return statistic (0) or p-value (1).

**Returns**
Chi-squared statistic or p-value, depending on the index parameter.

**Return type**
float

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

Perform chi-squared test for weighted data.

**Parameters**

- **subgroup** – The subgroup for which to calculate the statistic.
- **data** (*pandas DataFrame*) – The dataset.

- **weighting_attribute** (*str*) – The attribute used for weighting.
- **effective_sample_size** (*int, optional*) – Effective sample size.
- **min_instances** (*int, optional*) – Minimum required instances.

**Returns**
The p-value from the chi-squared test.

**Return type**
float

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

Evaluate the quality of the subgroup using the chi-squared test.

**Parameters**

- **subgroup** – The subgroup to evaluate.
- **target** (BinaryTarget) – The target definition.
- **data** (*pandas DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**
The chi-squared statistic or p-value.

**Return type**
float

**class** pysubgroup.binary_target.**GeneralizationAware_StandardQF**(*a*, *optimistic_estimate_strategy='default'*)

Bases: *GeneralizationAwareQF_stats*, *BoundedInterestingnessMeasure*

Generalization-Aware Standard Quality Function.

Extends the StandardQF to consider generalizations during subgroup discovery, providing methods for optimistic estimates and aggregate statistics.

**difference_based_agg_function**(*stats_subgroup*, *list_of_pairs*)

Aggregate statistics using the difference-based strategy.

**Parameters**

- **stats_subgroup** – Statistics of the current subgroup.
- **list_of_pairs** – List of (stats, agg_tuple) for all generalizations.

**Returns**
Aggregate statistics tuple.

**Return type**
namedtuple

**difference_based_optimistic_estimate**(*subgroup*, *target*, *data*, *statistics*)

Compute the optimistic estimate using the difference-based strategy.

**Parameters**

- **subgroup** – The subgroup for which to compute the estimate.
- **target** (BinaryTarget) – The target definition.
- **data** (*pandas DataFrame*) – The dataset.

- **statistics** (*any*) – Current statistics.

**Returns**
The optimistic estimate of the quality value.

**Return type**
float

**difference_based_read_p**(*agg_tuple*)

Read the p-value from the aggregate tuple using the difference-based strategy.

**Parameters**
**agg_tuple** – The aggregate statistics tuple.

**Returns**
The maximum percentage of positives.

**Return type**
float

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

Evaluate the quality of the subgroup considering generalizations.

**Parameters**

- **subgroup** – The subgroup to evaluate.
- **target** (BinaryTarget) – The target definition.
- **data** (*pandas DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**
The computed quality value.

**Return type**
float

**class ga_sQF_agg_tuple**(*max_p*, *min_delta_negatives*, *min_negatives*)

Bases: tuple

**max_p**
Alias for field number 0

**min_delta_negatives**
Alias for field number 1

**min_negatives**
Alias for field number 2

**max_based_aggregate_statistics**(*stats_subgroup*, *list_of_pairs*)

Aggregate statistics using the maximum-based strategy.

**Parameters**

- **stats_subgroup** – Statistics of the current subgroup.
- **list_of_pairs** – List of (stats, agg_tuple) for all generalizations.

**Returns**
The aggregated statistics.

**max_based_optimistic_estimate**(*subgroup*, *target*, *data*, *statistics=None*)

Compute the optimistic estimate using the maximum-based strategy.

**Parameters**

- **subgroup** – The subgroup for which to compute the estimate.
- **target** (BinaryTarget) – The target definition.
- **data** (*pandas DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**
The optimistic estimate of the quality value.

**Return type**
float

**max_based_read_p**(*agg_tuple*)

Read the p-value from the aggregate tuple using the maximum-based strategy.

**Parameters**
**agg_tuple** – The aggregate statistics tuple.

**Returns**
The ratio of positives in the aggregate statistics.

**Return type**
float

**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*

Quality function for binary targets based on positive instances.

**calculate_constant_statistics**(*data*, *target*)

Calculate statistics that remain constant for the dataset.

**Parameters**

- **data** (*pandas DataFrame*) – The dataset.
- **target** (BinaryTarget) – The target definition.

**Raises**
AssertionError – If the target is not an instance of BinaryTarget.

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

Calculate statistics specific to the subgroup.

**Parameters**

- **subgroup** – The subgroup for which to calculate statistics.
- **target** (BinaryTarget) – The target definition.
- **data** (*pandas DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**
Contains size_sg and positives_count for the subgroup.

**Return type**
namedtuple

**gp_get_null_vector**()

Get a null vector for initialization in GP-Growth algorithms.

**Returns**
Zero-initialized array of size 2.

**Return type**
numpy.ndarray

**gp_get_params**(*_cover_arr*, *v*)

Extract parameters from the statistics vector.

**Parameters**

- **_cover_arr** – Unused parameter.
- **v** (*numpy.ndarray*) – Statistics vector.

**Returns**
Contains size_sg and positives_count.

**Return type**
namedtuple

**gp_get_stats**(*row_index*)

Get statistics for a single row (used in GP-Growth algorithms).

**Parameters**
**row_index** (*int*) – The index of the row.

**Returns**
Array containing [1, positives[row_index]].

**Return type**
numpy.ndarray

**gp_merge**(*left*, *right*)

Merge two statistics vectors by summing them.

**Parameters**

- **left** (*numpy.ndarray*) – Left statistics vector.
- **right** (*numpy.ndarray*) – Right statistics vector.

**property gp_requires_cover_arr**

Indicate whether the GP-Growth algorithm requires a cover array.

**Returns**
False, since cover array is not required.

**Return type**
bool

**gp_size_sg**(*stats*)

Get the size of the subgroup from the statistics.

**Parameters**
**stats** (*numpy.ndarray*) – Statistics vector.

**Returns**
Size of the subgroup.

**Return type**
int

**gp_to_str**(*stats*)

Convert statistics to a string representation.

**Parameters**
**stats** (*numpy.ndarray*) – Statistics vector.

**Returns**
String representation of the statistics.

**Return type**
str

**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.

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

Evaluate the quality of the subgroup using the standard quality function.

**Parameters**

- **subgroup** – The subgroup to evaluate.
- **target** (BinaryTarget) – The target definition.
- **data** (*pandas DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**
The computed quality value.

**Return type**
float

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

Compute the optimistic estimate of the quality function.

**Parameters**

- **subgroup** – The subgroup for which to compute the optimistic estimate.
- **target** (*BinaryTarget*) – The target definition.
- **data** (*pandas DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**
The optimistic estimate of the quality value.

**Return type**
float

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

Compute the optimistic generalization of the quality function.

**Parameters**

- **subgroup** – The subgroup for which to compute the optimistic generalization.
- **target** (*BinaryTarget*) – The target definition.
- **data** (*pandas DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**
The optimistic generalization of the quality value.

**Return type**
float

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

Compute the standard quality function.

**Parameters**

- **a** (*float*) – Exponent to trade-off the relative size with difference in means.
- **instances_dataset** (*int*) – Total number of instances in the dataset.
- **positives_dataset** (*int*) – Total number of positive instances in the dataset.
- **instances_subgroup** (*int*) – Number of instances in the subgroup.
- **positives_subgroup** (*int*) – Number of positive instances in the subgroup.

**Returns**
The computed quality value.

**Return type**
float

**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.**ContainsValueConstraint**(*attribute_name*, *value*)

Bases: object

A constraint that ensures a subgroup contains in its cover at least one instance that has a specified value for a specified attribute.

**attribute_name**

The attribute that needs to contain the specified value in at least one instance.

**value**

The value that needs to be present in the specified attribute in at least one instance.

**property is_monotone**

Indicates whether the constraint is monotone.

**Returns**
True if the constraint is monotone, False otherwise.

**Return type**
bool

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

Checks if the subgroup satisfies the constraint.

**Parameters**

- **subgroup** – The subgroup to be evaluated.
- **statistics** – Precomputed statistics for the subgroup (optional).
- **data** – The dataset being analyzed (optional).

**Returns**

**True if the subgroup's cover contains at least one instance that has the specified value for the specified attribute (as defined during object construction),**
False otherwise.

**Return type**
bool

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

Bases: object

A constraint that ensures a subgroup has at least a minimum support.

**min_support**

The minimum number of instances that a subgroup must cover.

**Type**
int

**gp_is_satisfied**(*node*)

Checks if a node satisfies the constraint in the GP-Growth algorithm.

**Parameters**
**node** – The node to be evaluated.

**Returns**

**True if the node's size is at least the minimum support,**
False otherwise.

**Return type**
bool

`gp_prepare`(*qf*)

Prepares the constraint for the GP-Growth algorithm by accessing the size function.

**Parameters**
**qf** – The quality function used in the GP-Growth algorithm.

`property is_monotone`

Indicates whether the constraint is monotone.

**Returns**
True if the constraint is monotone, False otherwise.

**Return type**
bool

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

Checks if the subgroup satisfies the minimum support constraint.

**Parameters**

- **subgroup** – The subgroup to be evaluated.
- **statistics** – Precomputed statistics for the subgroup (optional).
- **data** – The dataset being analyzed (optional).

**Returns**

**True if the subgroup's size is at least the minimum support,**
False otherwise.

**Return type**
bool

`class` pysubgroup.constraints.`MinUniqueValuesConstraint`(*attribute_name*, *min_unique_values*)

Bases: `object`

A constraint that ensures a subgroup contains in its cover a minimum number of unique values for a specified attribute.

`attribute_name`

The attribute that needs to contain at least the specified number of values.

`min_unique_values`

The minimum number of unique values that must be present in the attribute in a subgroup cover.

`property is_monotone`

Indicates whether the constraint is monotone.

**Returns**
True if the constraint is monotone, False otherwise.

**Return type**
bool

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

Checks if the subgroup satisfies the constraint.

**Parameters**

- **subgroup** – The subgroup to be evaluated.

- **statistics** – Precomputed statistics for the subgroup (optional).
- **data** – The dataset being analyzed (optional).

**Returns**

**True if the subgroup's cover contains the minimum number of unique values for the specified attribute (as defined during object construction),**
False otherwise.

**Return type**
bool

### pysubgroup.datasets module

This module provides functions to load example datasets for testing and demonstration purposes. The datasets included are the German Credit Data and the Titanic dataset.

pysubgroup.datasets.**get_credit_data**()

Load the German Credit Data dataset.

The dataset is provided in ARFF format and includes various attributes related to creditworthiness.

**Returns**
A DataFrame containing the credit data.

**Return type**
pandas.DataFrame

pysubgroup.datasets.**get_titanic_data**()

Load the Titanic dataset.

The dataset includes information about the passengers on the Titanic, such as age, sex, class, and survival status.

**Returns**
A DataFrame containing the Titanic data.

**Return type**
pandas.DataFrame

### pysubgroup.fi_target module

Created on 29.09.2017

@author: lemmerfn

This module defines the FITarget and related quality functions for frequent itemset mining using the pysubgroup package.

**class** pysubgroup.fi_target.**AreaQF**

Bases: *SimpleCountQF*

Quality function that evaluates subgroups based on their area.

The area is computed as the size of the subgroup multiplied by the number of contained items

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

Evaluate the quality of the subgroup.

**Parameters**

- **subgroup** – The subgroup to evaluate.
- **target** – The target definition.

- **data** – The dataset.
- **statistics** (*any, optional*) – Previously computed statistics.

**Returns**
The area of the subgroup (size_sg * depth).

**Return type**
int

**class** pysubgroup.fi_target.**CountQF**

Bases: *SimpleCountQF*, *BoundedInterestingnessMeasure*

Quality function that evaluates subgroups based on their size.

Extends SimpleCountQF and BoundedInterestingnessMeasure.

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

Evaluate the quality of the subgroup.

**Parameters**

- **subgroup** – The subgroup to evaluate.
- **target** – The target definition.
- **data** – The dataset.
- **statistics** (*any, optional*) – Previously computed statistics.

**Returns**
The size of the subgroup.

**Return type**
int

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

Compute the optimistic estimate of the quality function.

**Parameters**

- **subgroup** – The subgroup for which to compute the optimistic estimate.
- **target** – The target definition.
- **data** – The dataset.
- **statistics** (*any, optional*) – Previously computed statistics.

**Returns**
The size of the subgroup.

**Return type**
int

**class** pysubgroup.fi_target.**FITarget**

Bases: *BaseTarget*

Target class for frequent itemset mining.

Represents the target for mining frequent itemsets, extending the BaseTarget class from pysubgroup.

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

Calculate statistics for the subgroup.

**Parameters**

- **subgroup_description** – The description of the subgroup.
- **data** – The dataset.
- **cached_statistics** (*dict, optional*) – Previously computed statistics.

**Returns**
A dictionary containing 'size_sg' and 'size_dataset'.

**Return type**
dict

`get_attributes()`
Return an empty list as attributes are not used in FITarget.

`get_base_statistics(subgroup, data)`
Compute the base statistics for the subgroup.

**Parameters**

- **subgroup** – The subgroup for which to compute statistics.
- **data** – The dataset.

**Returns**
The size of the subgroup.

**Return type**
int

`statistic_types = ('size_sg', 'size_dataset')`

`class pysubgroup.fi_target.SimpleCountQF`
Bases: *AbstractInterestingnessMeasure*

Quality function that counts the number of instances in a subgroup.

Provides basic counting functionality, useful for frequent itemset mining.

`calculate_constant_statistics(data, target)`
Calculate statistics that remain constant for the dataset.

**Parameters**

- **data** – The dataset.
- **target** – The target definition (unused in this implementation).

`calculate_statistics(subgroup_description, target, data, statistics=None)`
Calculate statistics specific to the subgroup.

**Parameters**

- **subgroup_description** – The description of the subgroup.
- **target** – The target definition (unused in this implementation).
- **data** – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**
Contains 'size_sg' for the subgroup.

**Return type**
namedtuple

**gp_get_null_vector**()

Get a null vector for initialization in GP-Growth algorithms.

**Returns**
A dictionary with 'size_sg' set to 0.

**Return type**
dict

**gp_get_params**(*_cover_arr*, *v*)

Extract parameters from the statistics dictionary.

**Parameters**

- **_cover_arr** – Unused parameter.
- **v** (*dict*) – Statistics dictionary.

**Returns**
Contains 'size_sg' from the statistics.

**Return type**
namedtuple

**gp_get_stats**(*_*)

Get statistics for a single instance (used in GP-Growth algorithms).

**Returns**
A dictionary with 'size_sg' set to 1.

**Return type**
dict

**gp_merge**(*left*, *right*)

Merge two statistics dictionaries by summing 'size_sg'.

**Parameters**

- **left** (*dict*) – Left statistics dictionary.
- **right** (*dict*) – Right statistics dictionary.

**gp_requires_cover_arr = False**

**gp_size_sg**(*stats*)

Get the size of the subgroup from the statistics.

**Parameters**
**stats** (*dict*) – Statistics dictionary.

**Returns**
Size of the subgroup.

**Return type**
int

**gp_to_str**(*stats*)

Convert statistics to a string representation.

**Parameters**
**stats** (*dict*) – Statistics dictionary.

**Returns**
String representation of 'size_sg'.

**Return type**
str

**tpl**

alias of CountQF_parameters

### pysubgroup.gp_growth module

class pysubgroup.gp_growth.**GpGrowth**(*mode='b_u'*)

Bases: object

Implementation of the GP-Growth algorithm.

GP-Growth is a generalization of FP-Growth and SD-Map capable of working with different Exceptional Model Mining targets on top of Frequent Itemset Mining and Subgroup Discovery.

This class provides methods to perform pattern mining using GP-Growth, supporting both bottom-up ('b_u') and top-down ('t_d') modes.

**GP_node**

Structure representing a node in the GP-tree.

**Type**
namedtuple

**minSupp**

Minimum support threshold (currently unused).

**Type**
int

**tqdm**

Function for progress bars (default is identity function).

**Type**
function

**depth**

Maximum depth of the search.

**Type**
int

**mode**

Mode of the algorithm ('b_u' for bottom-up, 't_d' for top-down).

**Type**
str

**constraints_monotone**

List of monotonic constraints.

**Type**
list

**results**

List to store the resulting subgroups.

**Type**
list

**task**

The subgroup discovery task to execute.

**Type**
*SubgroupDiscoveryTask*

**add_if_required**(*prefix*, *gp_stats*)

Adds a pattern to the result set if it meets the quality threshold.

**Parameters**

- **prefix** (*tuple*) – The current pattern (tuple of class indices).
- **gp_stats** – The aggregated statistics for the pattern.

**calculate_quality_function_for_patterns**(*task*, *results*, *arrs*)

Calculates the quality function for the given patterns.

**Parameters**

- **task** (SubgroupDiscoveryTask) – The task containing the quality function.
- **results** (*list*) – List of patterns with their aggregated parameters.
- **arrs** (*ndarray*) – The coverage arrays of the selectors.

**Returns**
A list of tuples containing quality, indices, and statistics.

**Return type**
list

**check_constraints**(*node*)

Checks if a node satisfies all monotonic constraints.

**Parameters**
**node** – The node to check.

**Returns**
True if the node satisfies all constraints, False otherwise.

**Return type**
bool

**check_tree_is_ordered**(*root*, *prefix=None*)

Verifies that the nodes of a tree are sorted in ascending order.

**Parameters**

- **root** (*GP_node*) – The root node of the tree.
- **prefix** (*list*) – The current path prefix.

**Returns**
A set of class labels in the tree.

**Return type**
set

**convert_results_to_subgroups**(*results*, *selectors_sorted*)

Converts patterns (indices) to actual subgroups.

**Parameters**

- **results** (*list*) – List of results containing qualities, indices, and statistics.

- **selectors_sorted** (*list*) – The list of sorted selectors.

**Returns**
A list of tuples containing quality, subgroup, and statistics.

**Return type**
list

**create_copy_of_path**(*nodes*, *new_nodes*, *stats*)

Creates a copy of a path in the tree, updating statistics.

**Parameters**

- **nodes** (*list*) – The list of nodes in the path.
- **new_nodes** (*dict*) – Dictionary to store new nodes.
- **stats** – The statistics to merge into the nodes.

**create_copy_of_tree_top_down**(*from_root*, *nodes=None*, *parent=None*, *is_valid_class=None*)

Creates a copy of the tree starting from a specific root in top-down mode.

**Parameters**

- **from_root** (*GP_node*) – The root node to copy from.
- **nodes** (*list*) – List to store the new nodes.
- **parent** (*GP_node*) – The parent of the new root node.
- **is_valid_class** (*dict*) – Dictionary indicating valid classes.

**Returns**
The new root node of the copied subtree.

**Return type**
GP_node

**create_initial_tree**(*arrs*)

Creates the initial FP-tree from the coverage arrays.

**Parameters**
**arrs** (*ndarray*) – A 2D NumPy array where each column corresponds to the coverage array of a selector.

**Returns**

**A tuple containing:**

- root (GP_node): The root node of the tree.
- nodes (list): A list of all nodes in the tree.

**Return type**
tuple

**create_new_tree_from_nodes**(*nodes*)

Creates a new tree from a list of nodes for recursive mining.

**Parameters**
**nodes** (*list*) – List of nodes to build the new tree from.

**Returns**
A dictionary mapping class labels to nodes in the new tree.

**Return type**
defaultdict

`execute`(*task*)

Executes the GP-Growth algorithm on the given task.

**Parameters**
**task** (SubgroupDiscoveryTask) – The subgroup discovery task to execute.

**Returns**
The result of the subgroup discovery.

**Return type**
*SubgroupDiscoveryResult*

`get_nodes_upwards`(*node*)

Retrieves all nodes from a given node up to the root.

**Parameters**
**node** (*GP_node*) – The starting node.

**Returns**
A list of nodes from the given node up to the root.

**Return type**
list

`get_stats_for_class`(*cls_nodes*)

Aggregates statistics for each class label.

**Parameters**
**cls_nodes** (*defaultdict*) – Dictionary mapping class labels to nodes.

**Returns**
A dictionary mapping class labels to aggregated statistics.

**Return type**
dict

`get_top_down_tree_for_class`(*cls_nodes*, *cls*, *is_valid_class*)

Creates a subtree for a specific class in top-down mode.

**Parameters**

- **cls_nodes** (*defaultdict*) – Dictionary mapping class labels to nodes.
- **cls** (*int*) – The class label to create the subtree for.
- **is_valid_class** (*dict*) – Dictionary indicating valid classes.

**Returns**

**A tuple containing:**

- base_root (GP_node): The root of the new subtree.
- nodes (list): A list of nodes in the new subtree.

**Return type**
tuple

`merge_trees_top_down`(*nodes*, *mutable_root*, *from_root*, *is_valid_class*)

Merges two trees in top-down mode.

**Parameters**

- **nodes** (*list*) – List of nodes in the mutable tree.
- **mutable_root** (*GP_node*) – The root of the mutable tree to merge into.
- **from_root** (*GP_node*) – The root of the tree to merge from.
- **is_valid_class** (*dict*) – Dictionary indicating valid classes.

**nodes_to_cls_nodes**(*nodes*)

Groups nodes by their class labels.

**Parameters**
**nodes** (*list*) – List of nodes to group.

**Returns**
A dictionary mapping class labels to lists of nodes.

**Return type**
defaultdict

**normal_insert**(*root*, *nodes*, *new_stats*, *classes*)

Inserts a transaction into the FP-tree.

**Parameters**

- **root** (*GP_node*) – The root node of the tree.
- **nodes** (*list*) – List of all nodes in the tree.
- **new_stats** – The statistics associated with the transaction.
- **classes** (*array-like*) – The class labels (selectors) present in the transaction.

**Returns**
The leaf node where the transaction ends.

**Return type**
GP_node

**prepare_selectors**(*search_space*, *data*)

Prepares the selectors by computing their coverage arrays and filtering based on constraints.

**Parameters**

- **search_space** (*list*) – The list of selectors to consider.
- **data** (*DataFrame*) – The dataset to be analyzed.

**Returns**

**A tuple containing:**

- selectors_sorted (list): The sorted list of selectors after filtering.
- **arrs (ndarray): A 2D NumPy array where each column corresponds to the** coverage array of a selector.

**Return type**
tuple

**recurse**(*cls_nodes*, *prefix*, *is_single_path=False*)

Recursively mines patterns in bottom-up mode.

**Parameters**

- **cls_nodes** (*defaultdict*) – Dictionary mapping class labels to nodes.

- **prefix** (*tuple*) – The current pattern prefix.
- **is_single_path** (*bool*) – Flag indicating if the current path is a single path.

**recurse_top_down**(*cls_nodes*, *root*, *depth_in=0*)

Recursively mines patterns in top-down mode.

**Parameters**

- **cls_nodes** (*defaultdict*) – Dictionary mapping class labels to nodes.
- **root** (*GP_node*) – The current root node.
- **depth_in** (*int*) – The current depth in the recursion.

**Returns**
A list of patterns with their aggregated statistics.

**Return type**
list

**remove_selectors_with_low_optimistic_estimate**(*s*, *search_space_size*)

Removes selectors from the list that have an optimistic estimate below the minimum required quality.

**Parameters**

- **s** (*list*) – List of selectors with their size and coverage arrays.
- **search_space_size** (*int*) – The size of the initial search space.

**setup**(*task*)

Prepares the algorithm by setting up the task, depth, constraints, and quality function.

**Parameters**
**task** (SubgroupDiscoveryTask) – The task to execute.

**setup_constraints**(*constraints*, *qf*)

Prepares constraints for use in the algorithm.

**Parameters**

- **constraints** (*list*) – List of constraints to apply.
- **qf** – The quality function used in the task.

**setup_from_quality_function**(*qf*)

Sets up function pointers from the quality function.

**Parameters**
**qf** – The quality function used in the task.

**to_file**(*task*, *path*)

Writes the tree to a file in a specific format.

**Parameters**

- **task** (SubgroupDiscoveryTask) – The task containing the quality function.
- **path** (*str or Path*) – The file path to write to.

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

Identity function used as a placeholder for tqdm when progress bars are not needed.

**Parameters**

- **x** – The input value to return.

- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

**Returns**
The input value x.

### 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*

A class that computes the generalization aware qf as follows: qf(sg) = qf(sg) - max_{generalizations} qf(sq)

**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*

An abstract base class that implements aggregation of stats of generalisations

**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_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_predictions_target module

Created on 16.08.2025

@author: Tom Siegl

class pysubgroup.model_predictions_target.**ARLQF**(*label_column: str*, *positive_label_value: any*, *subgroup_class_balance_weight: float = 0*, *subgroup_size_weight: float = 0*)

Bases: *BaseSoftClassifierPerformanceQF*

A quality function which scores binary soft classifier performance in a subgroup based on the difference of the classifier's average ranking loss (ARL) on the subgroup cover vs. the entire dataset. If the classifier performs worse on the subgroup (i.e. it has a greater ARL) compared to the entire dataset, then the quality is positive.

Weighting factors are provided to let the subgroup size and class balance influence the quality.

The overall quality is captured by the formula q = (ARL(subgroup) - ARL(dataset)) * |subgroup|^(size_weight) * class_balance(subgroup)^(class_balance_weight).

Implementation of phi^{rasl}_{alpha, beta} from the paper ["SubROC: AUC-Based Discovery of Exceptional Subgroup Performance for Binary Classifiers"](https://doi.org/10.48550/arXiv.2505.11283).

class pysubgroup.model_predictions_target.**BaseSoftClassifierPerformanceQF**(*performance_measure*, *performance_measure_type: Literal['score', 'loss']*, *performance_measure_bound=None*, *performance_measure_constraints: list[any] = []*, *subgroup_class_balance_weight: float = 0*, *subgroup_size_weight: float = 0*)

Bases: *BoundedInterestingnessMeasure*

**calculate_constant_statistics**(*data: DataFrame*, *target:* SoftClassifierTarget)

This function is called once for every search execution, it should do any preparation that is necessary prior to an execution.

**calculate_statistics**(*subgroup*, *target:* SoftClassifierTarget, *data: DataFrame*, *statistics={}*)

calculates necessary statistics this statistics object is passed on to the evaluate and optimistic_estimate functions

**evaluate**(*subgroup*, *target:* SoftClassifierTarget, *data: DataFrame*, *statistics=None*)

return the quality calculated from the statistics

**optimistic_estimate**(*subgroup*, *target:* SoftClassifierTarget, *data: DataFrame*, *statistics=None*)

returns optimistic estimate if one is available return it otherwise infinity

class pysubgroup.model_predictions_target.**PRAUCQF**(*label_column: str*, *positive_label_value: any*, *subgroup_class_balance_weight: float = 0*, *subgroup_size_weight: float = 0*)

Bases: *BaseSoftClassifierPerformanceQF*

A quality function which scores binary soft classifier performance in a subgroup based on the difference of the classifier's Area Under the Precision-Recall Curve (PR AUC) on the subgroup cover vs. the entire dataset. If the classifier performs worse on the subgroup (i.e. it has a lower PR AUC) compared to the entire dataset, then the quality is positive.

Weighting factors are provided to let the subgroup size and class balance influence the quality.

The overall quality is captured by the formula q = (PRAUC(subgroup) - PRAUC(dataset)) * |subgroup|^(size_weight) * class_balance(subgroup)^(class_balance_weight).

Implementation of phi^{rPRAUC}_{alpha, beta} from the paper ["SubROC: AUC-Based Discovery of Exceptional Subgroup Performance for Binary Classifiers"](https://doi.org/10.48550/arXiv.2505.11283).

class pysubgroup.model_predictions_target.**ROCAUCQF**(*label_column: str*, *subgroup_class_balance_weight: float = 0*, *subgroup_size_weight: float = 0*)

Bases: *BaseSoftClassifierPerformanceQF*

A quality function which scores binary soft classifier performance in a subgroup based on the difference of the classifier's Area Under the Receiver Operating Characteristic Curve (ROC AUC) on the subgroup cover vs. the entire dataset. If the classifier performs worse on the subgroup (i.e. it has a lower ROC AUC) compared to the entire dataset, then the quality is positive.

Weighting factors are provided to let the subgroup size and class balance influence the quality.

The overall quality is captured by the formula q = (ROCAUC(subgroup) - ROCAUC(dataset)) * |subgroup|^(size_weight) * class_balance(subgroup)^(class_balance_weight).

Implementation of phi^{rROCAUC}_{alpha, beta} from the paper ["SubROC: AUC-Based Discovery of Exceptional Subgroup Performance for Binary Classifiers"](https://doi.org/10.48550/arXiv.2505.11283).

class pysubgroup.model_predictions_target.**SoftClassifierTarget**(*label_column='label'*, *prediction_column='prediction'*)

Bases: object

Minimal target concept implementation to select label and prediction columns for binary soft classifier performance measures.

**calculate_statistics**(*subgroup*, *data: DataFrame*, *statistics={}*)

**get_target_columns**(*data: DataFrame*)

Select the label and prediction columns from object initialization.

**statistic_types = ()**

pysubgroup.model_predictions_target.**average_ranking_loss**(*y_true*, *y_pred*)

Implementation of the Average Ranking Loss (ARL) performance measure for binary soft classifiers based on the definitions in the paper ["Understanding Where Your Classifier Does (Not) Work – The SCaPE Model Class for EMM"](https://doi.org/10.1109/ICDM.2014.10).

**Parameters**

- **y_true** – Binary Labels, must be ordered to match y_pred.
- **y_pred** – Predicted Scores, must be in ascending order.

pysubgroup.model_predictions_target.**pr_auc_score**(*y_true*, *y_pred*)

Area Under the Precision-Recall Curve (PR AUC) performance measure for binary soft classifiers.

**Parameters**

- **y_true** – Binary Labels, must be ordered to match y_pred.
- **y_pred** – Predicted Scores.

### pysubgroup.model_target module

class pysubgroup.model_target.**EMM_Likelihood**(*model*)

Bases: *AbstractInterestingnessMeasure*

Exceptional Model Mining likelihood-based interestingness measure.

This class computes the difference in likelihoods between a subgroup model and the inverse (complement) model, providing a measure of how exceptional the subgroup is with respect to the entire dataset.

**calculate_constant_statistics**(*data*, *target*)

Calculate statistics that remain constant over all subgroups.

**Parameters**

- **data** – The dataset as a pandas DataFrame.
- **target** – The target variable (unused in this context).

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

Calculate statistics specific to a subgroup.

**Parameters**

- **subgroup** – The subgroup description.
- **target** – The target variable (unused in this context).
- **data** – The dataset as a pandas DataFrame.
- **statistics** – Previously calculated statistics (optional).

**Returns**
An EMM_Likelihood.tpl namedtuple containing model parameters, subgroup likelihood, inverse likelihood, and subgroup size.

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

Evaluate the interestingness of a subgroup.

**Parameters**

- **subgroup** – The subgroup description.
- **target** – The target variable (unused in this context).
- **data** – The dataset as a pandas DataFrame.
- **statistics** – Previously calculated statistics (optional).

**Returns**
The difference between subgroup likelihood and inverse likelihood.

**get_tuple**(*sg_size*, *params*, *cover_arr*)

Compute the likelihoods for the subgroup and its complement.

**Parameters**

- **sg_size** – Size of the subgroup.
- **params** – Model parameters obtained from fitting the subgroup.
- **cover_arr** – Boolean array indicating the instances in the subgroup.

**Returns**
An EMM_Likelihood.tpl namedtuple with the computed statistics.

**gp_get_params**(*cover_arr*, *v*)

Get parameters for GP-Growth algorithm.

**Parameters**

- **cover_arr** – Boolean array indicating the instances in the subgroup.
- **v** – Statistics vector from GP-Growth.

**Returns**
An EMM_Likelihood.tpl namedtuple with the computed statistics.

**property gp_requires_cover_arr**

Indicate whether the GP-Growth algorithm requires a cover array.

**Returns**
True, since the cover array is required.

**tpl**

alias of *EMM_Likelihood*

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

Bases: object

Polynomial Regression Model Class for Exceptional Model Mining.

Provides methods to fit a polynomial regression model to a subgroup and compute likelihoods for Exceptional Model Mining.

**calculate_constant_statistics**(*data*, *target*)

Calculate statistics that remain constant over all subgroups.

**Parameters**

- **data** – The dataset as a pandas DataFrame.
- **target** – The target variable (unused in this context).

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

Fit the polynomial regression model to the subgroup data.

**Parameters**

- **subgroup** – The subgroup description.
- **data** – The dataset as a pandas DataFrame (optional).

**Returns**
Contains regression coefficients and subgroup size.

**Return type**
*beta_tuple*

**gp_get_null_vector**()

Get a null vector for initialization in the GP-Growth algorithm.

**Returns**
Zero-initialized array of size 5.

**Return type**
numpy.ndarray

**gp_get_params**(*v*)

Extract model parameters from the statistics vector.

**Parameters**
**v** (*numpy.ndarray*) – Statistics vector.

**Returns**
Contains regression coefficients and subgroup size.

**Return type**
*beta_tuple*

**gp_get_stats**(*row_index*)

Get statistics for a single row (used in GP-Growth algorithm).

**Parameters**
**row_index** (*int*) – Index of the row in the dataset.

**Returns**
Statistics vector for the given row.

**Return type**
numpy.ndarray

`static gp_merge(u, v)`

Merge two statistics vectors for the GP-Growth algorithm.

**Parameters**

- **u** (*numpy.ndarray*) – Left statistics vector.
- **v** (*numpy.ndarray*) – Right statistics vector.

`property gp_requires_cover_arr`

Indicate whether the GP-Growth algorithm requires a cover array.

**Returns**
False, since the cover array is not required.

`gp_size_sg(stats)`

Get the size of the subgroup from the statistics.

**Parameters**
**stats** (*numpy.ndarray*) – Statistics vector.

**Returns**
Size of the subgroup.

**Return type**
float

`gp_to_str(stats)`

Convert statistics to a string representation.

**Parameters**
**stats** (*numpy.ndarray*) – Statistics vector.

**Returns**
String representation of the statistics.

**Return type**
str

`likelihood(stats, sg)`

Compute the likelihoods for the subgroup instances.

**Parameters**

- **stats** (beta_tuple) – Regression parameters and subgroup size.
- **sg** (*numpy.ndarray*) – Boolean array indicating subgroup instances.

**Returns**
Likelihood values for the subgroup instances.

**Return type**
numpy.ndarray

`loglikelihood(stats, sg)`

Compute the log-likelihoods for the subgroup instances.

**Parameters**

- **stats** (beta_tuple) – Regression parameters and subgroup size.
- **sg** (*numpy.ndarray*) – Boolean array indicating subgroup instances.

**Returns**
Log-likelihood values for the subgroup instances.

**Return type**
numpy.ndarray

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

This module defines the NumericTarget and associated quality functions for subgroup discovery when the target variable is numeric.

class pysubgroup.numeric_target.**GeneralizationAware_StandardQFNumeric**(*a*, *invert=False*, *estimator='default'*, *centroid='mean'*)

Bases: *GeneralizationAwareQF_stats*

Generalization-Aware Standard Quality Function for Numeric Targets.

Extends StandardQFNumeric to consider generalizations during subgroup discovery, providing methods for optimistic estimates and aggregate statistics.

**aggregate_statistics**(*stats_subgroup*, *list_of_pairs*)

Aggregate statistics from generalizations.

**Parameters**

- **stats_subgroup** – Statistics of the current subgroup.
- **list_of_pairs** – List of (stats, agg_stats) tuples from generalizations.

**Returns**
The aggregated statistics.

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

Evaluate the quality of the subgroup considering generalizations.

**Parameters**

- **subgroup** – The subgroup to evaluate.
- **target** (NumericTarget) – The target definition.
- **data** (*pandas.DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Previously computed statistics.

**Returns**
The computed quality value.

**Return type**
float

class pysubgroup.numeric_target.**NumericTarget**(*target_variable*)

Bases: object

Target class for numeric variables in subgroup discovery.

Represents a target where the variable of interest is numeric, and computes statistics such as mean, median, standard deviation within subgroups.

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

Calculate various statistics for the subgroup and dataset.

**Parameters**

- **subgroup** – The subgroup for which to calculate statistics.
- **data** (*pandas.DataFrame*) – The dataset.
- **cached_statistics** (*dict, optional*) – Previously computed statistics.

**Returns**
A dictionary containing statistical measures.

**Return type**
dict

**get_attributes**()

Get a list of attribute names used by the target.

**Returns**
A list containing the target variable name.

**Return type**
list

**get_base_statistics**(*subgroup*, *data*)

Compute basic statistics for the subgroup and dataset.

**Parameters**

- **subgroup** – The subgroup for which to compute statistics.
- **data** (*pandas.DataFrame*) – The dataset.

**Returns**
(instances_dataset, mean_dataset, instances_subgroup, mean_sg)

**Return type**
tuple

**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='default'*, *centroid='mean'*)

Bases: *BoundedInterestingnessMeasure*

Standard Quality Function for numeric targets.

This quality function computes interestingness of subgroups based on the difference between subgroup mean (or median) and dataset mean (or median), weighted by the size of the subgroup raised to the power of 'a'.

**a**

Exponent to trade off between subgroup size and difference in means.

**Type**
float

**invert**

Whether to invert the quality function (not used currently).

**Type**
bool

**estimator**

Strategy for optimistic estimation ('sum', 'max', 'order').

**Type**
str

**centroid**

Central tendency measure ('mean', 'median', 'sorted_median').

**Type**
str

**class Max_Estimator**(*qf*)

Bases: object

Estimator for optimistic estimate using maximum value strategy.

This estimator calculates the optimistic estimate based on the maximum value greater than the dataset centroid.

From Florian Lemmerich's Dissertation [section 4.2.2.1, Theorem 4 (page 82)]:

$$oe(sg) = n^a_{>\mu_0}(T^{\max}(sg) - \mu_0)$$

**calculate_constant_statistics**(*data*, *target*)

Calculate constant statistics needed for estimation.

**Parameters**

- **data** (*pandas.DataFrame*) – The dataset.
- **target** (NumericTarget) – The target definition.

**get_data**(*data*, *target*)

Prepare data for estimation (no changes for this estimator).

**Parameters**

- **data** (*pandas.DataFrame*) – The dataset.
- **target** (NumericTarget) – The target definition.

**Returns**
The unmodified dataset.

**Return type**
pandas.DataFrame

**get_estimate**(*subgroup*, *sg_size*, *sg_centroid*, *cover_arr*, *_*)

Compute the optimistic estimate for the subgroup.

**Parameters**

- **subgroup** – The subgroup description.
- **sg_size** (*int*) – Size of the subgroup.

- **sg_centroid** (*float*) – Mean or median of the subgroup.
- **cover_arr** (*numpy.ndarray*) – Boolean array indicating subgroup instances.
- **_** – Unused parameter.

**Returns**
The optimistic estimate.

**Return type**
float

**class MeanOrdering_Estimator**(*qf*)

Bases: object

Estimator for optimistic estimate using mean ordering strategy.

This estimator sorts the target values and computes the optimal subgroup by considering prefixes of the sorted list.

**calculate_constant_statistics**(*data*, *target*)

Set up the estimation function, possibly using Numba for speed.

**Parameters**

- **data** (*pandas.DataFrame*) – The dataset.
- **target** (NumericTarget) – The target definition.

**get_data**(*data*, *target*)

Prepare data by sorting according to the target variable.

**Parameters**

- **data** (*pandas.DataFrame*) – The dataset.
- **target** (NumericTarget) – The target definition.

**Returns**
The sorted dataset.

**Return type**
pandas.DataFrame

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

Compute the optimistic estimate for the subgroup.

**Parameters**

- **subgroup** – The subgroup description.
- **sg_size** (*int*) – Size of the subgroup.
- **sg_mean** (*float*) – Mean of the subgroup.
- **cover_arr** (*numpy.ndarray*) – Boolean array indicating subgroup instances.
- **target_values_sg** (*numpy.ndarray*) – Target values in the subgroup.

**Returns**
The optimistic estimate.

**Return type**
float

**get_estimate_numpy**(*values_sg*, *_*, *mean_dataset*)

Compute the optimistic estimate using NumPy.

**Parameters**

- **values_sg** (*numpy.ndarray*) – Sorted target values in the subgroup.
- **_** – Unused parameter.
- **mean_dataset** (*float*) – Mean of the dataset.

**Returns**
The optimistic estimate.

**Return type**
float

**class Summation_Estimator**(*qf*)

Bases: object

Estimator for optimistic estimate using summation strategy.

This estimator calculates the optimistic estimate as a hypothetical subgroup which contains only instances with value greater than the dataset mean and is of maximal size.

From Florian Lemmerich's Dissertation [section 4.2.2.1, Theorem 2 (page 81)]:

$$oe(sg) = \sum_{x \in sg, T(x) > 0} (T(sg) - \mu_0)$$

**calculate_constant_statistics**(*data*, *target*)

Calculate constant statistics needed for estimation.

**Parameters**

- **data** (*pandas.DataFrame*) – The dataset.
- **target** (NumericTarget) – The target definition.

**get_data**(*data*, *target*)

Prepare data for estimation (no changes for this estimator).

**Parameters**

- **data** (*pandas.DataFrame*) – The dataset.
- **target** (NumericTarget) – The target definition.

**Returns**
The unmodified dataset.

**Return type**
pandas.DataFrame

**get_estimate**(*subgroup*, *sg_size*, *sg_centroid*, *cover_arr*, *_*)

Compute the optimistic estimate for the subgroup.

**Parameters**

- **subgroup** – The subgroup description.
- **sg_size** (*int*) – Size of the subgroup.
- **sg_centroid** (*float*) – Mean or median of the subgroup.

- **cover_arr** (*numpy.ndarray*) – Boolean array indicating subgroup instances.
- **_** – Unused parameter.

**Returns**
The optimistic estimate.

**Return type**
float

**calculate_constant_statistics**(*data*, *target*)

Calculate statistics that remain constant for the dataset.

**Parameters**

- **data** (*pandas.DataFrame*) – The dataset.
- **target** (NumericTarget) – The target definition.

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

Calculate statistics specific to the subgroup.

**Parameters**

- **subgroup** – The subgroup for which to calculate statistics.
- **target** (NumericTarget) – The target definition.
- **data** (*pandas.DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**
Contains size_sg, mean or median, and estimate.

**Return type**
namedtuple

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

Evaluate the quality of the subgroup using the standard quality function.

**Parameters**

- **subgroup** – The subgroup to evaluate.
- **target** (NumericTarget) – The target definition.
- **data** (*pandas.DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Previously computed statistics.

**Returns**
The computed quality value.

**Return type**
float

**mean_tpl**

alias of StandardQFNumeric_parameters

**median_tpl**

alias of StandardQFNumeric_median_parameters

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

Compute the optimistic estimate of the quality function.

**Parameters**

- **subgroup** – The subgroup for which to compute the optimistic estimate.
- **target** (NumericTarget) – The target definition.
- **data** (*pandas.DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Previously computed statistics.

**Returns**

The optimistic estimate of the quality value.

**Return type**

float

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

Compute the standard quality function for numeric targets.

**Parameters**

- **a** (*float*) – Exponent for weighting the subgroup size.
- **_** – Unused parameter (size of dataset).
- **mean_dataset** (*float*) – Mean of the target variable in the dataset.
- **instances_subgroup** (*int*) – Number of instances in the subgroup.
- **mean_sg** (*float*) – Mean of the target variable in the subgroup.

**Returns**

The computed quality value.

**Return type**

float

**tpl**

alias of StandardQFNumeric_parameters

**class** pysubgroup.numeric_target.**StandardQFNumericMedian**

Bases: *BoundedInterestingnessMeasure*

Quality function for numeric targets using median (deprecated).

> **Note**
>
> This class is no longer supported. Use StandardQFNumeric with centroid='median' instead.

**tpl**

alias of StandardQFNumericMedian_parameters

**class** pysubgroup.numeric_target.**StandardQFNumericTscore**(*invert=False*)

Bases: *BoundedInterestingnessMeasure*

Quality function for numeric targets using T-score.

`calculate_constant_statistics`(*data*, *target*)

Calculate statistics that remain constant for the dataset.

**Parameters**

- **data** (*pandas.DataFrame*) – The dataset.
- **target** (NumericTarget) – The target definition.

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

Calculate statistics specific to the subgroup.

**Parameters**

- **subgroup** – The subgroup for which to calculate statistics.
- **target** (NumericTarget) – The target definition.
- **data** (*pandas.DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Unused in this implementation.

**Returns**

Contains size_sg, mean, std, and estimate.

**Return type**

namedtuple

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

Evaluate the quality of the subgroup using the T-score.

**Parameters**

- **subgroup** – The subgroup to evaluate.
- **target** (NumericTarget) – The target definition.
- **data** (*pandas.DataFrame*) – The dataset.
- **statistics** (*any, optional*) – Previously computed statistics.

**Returns**

The computed T-score.

**Return type**

float

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

Compute the optimistic estimate of the quality function.

**Parameters**

- **subgroup** – The subgroup for which to compute the optimistic estimate.
- **target** – The target definition.
- **data** – The dataset.
- **statistics** (*any, optional*) – Previously computed statistics.

**Returns**

The optimistic estimate of the quality value.

**Return type**

float

**static t_score**(*mean_dataset*, *instances_subgroup*, *mean_sg*, *std_sg*)

Compute the T-score for the subgroup.

**Parameters**

- **mean_dataset** (*float*) – Mean of the dataset.
- **instances_subgroup** (*int*) – Number of instances in the subgroup.
- **mean_sg** (*float*) – Mean of the subgroup.
- **std_sg** (*float*) – Standard deviation of the subgroup.

**Returns**
The computed T-score.

**Return type**
float

**tpl**

alias of StandardQFNumericTscore_parameters

pysubgroup.numeric_target.**calc_sorted_median**(*arr*)

Calculate the median of a sorted array.

**Parameters**
**arr** (*numpy.ndarray*) – A sorted array.

**Returns**
The median value.

**Return type**
float

pysubgroup.numeric_target.**read_mean**(*tpl*)

Extract the mean value from a namedtuple.

**Parameters**
**tpl** (*namedtuple*) – A namedtuple containing a 'mean' field.

**Returns**
The mean value.

**Return type**
float

pysubgroup.numeric_target.**read_median**(*tpl*)

Extract the median value from a namedtuple.

**Parameters**
**tpl** (*namedtuple*) – A namedtuple containing a 'median' field.

**Returns**
The median value.

**Return type**
float

### pysubgroup.permutation_test module

class pysubgroup.permutation_test.**NegativeClassCountRandomSelector**(*size*, *negative_class_count*, *np_rng*, *positive_class_indices*, *negative_class_indices*)

Bases: object

A selector that covers a random subset of the given indices, such that the number of covered instances as well as the number of negatives instances are always the same.

**covers**(*data_instance*)

**select**()

Randomize the cover of this selector.

**property selectors**

**set_descriptions**(*size*, *negative_class_count*, **args*, ***kwargs*)

pysubgroup.permutation_test.**permutation_test**(*qf: any*, *result: any*, *target: ~pysubgroup.model_predictions_target.SoftClassifierTarget*, *data: ~pandas.core.frame.DataFrame*, *num_random_samples: int*, *np_rng: ~numpy.random._generator.Generator = Generator(PCG64) at 0x791FC01ACAC0*, *max_random_sampling_retries: int = 10*, *alpha: float = 0.05*, *pos_label: any = 1*, *neg_label: any = 0*, *multitest_correction_method: str = 'fdr_by'*, *tqdm: any = None*)

Test the subgroups in the result for statistical significance by comparison to qualities of random samples from the data. Random samples are drawn such that the number of instances from each class in the sample is the same as in the tested subgroup.

Only for SoftClassifierTargets.

**Parameters**

- **qf** – Quality function to use as the test statistic.
- **result** – ps.SubgroupDiscoveryResult object holding the subgroups to test.
- **target** – Target concept to use in the quality function.
- **data** – Dataset to compute all qualities from. The qualities of the given subgroups are also recomputed on this data for the test.
- **num_random_samples** – How many random samples to draw. More samples allow to distinguish p-values more fine-grained.
- **np_rng** – Random generator object to use for drawing the samples. Use this to get reproducible results.
- **max_random_sampling_retries** – How often to repeat the drawing process for each sample to get a quality. Repetitions are used when the quality is undefined on a random sample.
- **pos_label** – Which value in the dataset to count as a positive class.
- **neg_label** – Which value in the dataset to count as a negative class.

- **multitest_correction_method** – Which method to correct the p-values against the multiple comparison problem with. Refer to statsmodels.stats.multitest.multipletests for all possible values.

**Return p_values_raw**
Uncorrected p-values for each subgroup

**Return reject**
Test result after multiple testing correction.

**Return p_values_corrected**
P-values after multiple testing correction.

**Return qualities**
Subgroup qualities on the testing data.

**Return samples**
The full random sample of qualities that was generated for each subgroup.

### pysubgroup.refinement_operator module

**class** pysubgroup.refinement_operator.**RefinementOperator**

Bases: object

Base class for refinement operators.

**class** pysubgroup.refinement_operator.**StaticGeneralizationOperator**(*selectors*)

Bases: object

Refinement operator for static generalization.

This operator generalizes subgroups by adding selectors from a predefined list, ensuring that each selector is used in a specific order.

**refinements**(*sG*)

Generate refinements of the given subgroup.

**Parameters**
**sG** – The subgroup to refine.

**Returns**
A generator of refined subgroups.

**class** pysubgroup.refinement_operator.**StaticSpecializationOperator**(*selectors*)

Bases: object

Refinement operator for static specialization.

This operator specializes subgroups by adding selectors in a predefined order, ensuring that each attribute is used only once in a subgroup description.

**refinements**(*subgroup*)

Generate refinements of the given subgroup.

**Parameters**
**subgroup** – The subgroup to refine.

**Returns**
A generator of refined subgroups.

### pysubgroup.representations module

class pysubgroup.representations.**BitSetRepresentation**(*df*, *selectors_to_patch*)

Bases: *RepresentationBase*

Representation class that uses bitsets for selectors and conjunctions.

**Conjunction**

alias of *BitSet_Conjunction*

**Disjunction**

alias of *BitSet_Disjunction*

**patch_classes**()

Patch class-level attributes before entering the context.

**patch_selector**(*sel*)

Patch a selector by computing its bitset representation.

**Parameters**
**sel** – Selector to patch.

class pysubgroup.representations.**BitSet_Conjunction**(**args*, ***kwargs*)

Bases: *Conjunction*

Conjunction subclass that uses bitsets for representation.

Provides efficient computation of the conjunction using numpy boolean arrays.

**append_and**(*to_append*)

Append a selector using logical AND and update the representation.

**Parameters**
**to_append** – Selector to append.

**compute_representation**()

Compute the bitset representation of the conjunction.

**Returns**
Numpy boolean array representing the instances covered by the conjunction.

**n_instances = 0**

**property size_sg**

Size of the subgroup represented by the conjunction.

class pysubgroup.representations.**BitSet_Disjunction**(**args*, ***kwargs*)

Bases: *Disjunction*

Disjunction subclass that uses bitsets for representation.

Provides efficient computation of the disjunction using numpy boolean arrays.

**append_or**(*to_append*)

Append a selector using logical OR and update the representation.

**Parameters**
**to_append** – Selector to append.

**compute_representation**()

Compute the bitset representation of the disjunction.

**Returns**
Numpy boolean array representing the instances covered by the disjunction.

**property size_sg**

Size of the subgroup represented by the disjunction.

**class** pysubgroup.representations.**NumpySetRepresentation**(*df*, *selectors_to_patch*)

Bases: *RepresentationBase*

Representation class that uses numpy arrays for selectors and conjunctions.

**Conjunction**

alias of *NumpySet_Conjunction*

**patch_classes**()

Patch class-level attributes before entering the context.

**patch_selector**(*sel*)

Patch a selector by computing its numpy array representation.

**Parameters**
**sel** – Selector to patch.

**class** pysubgroup.representations.**NumpySet_Conjunction**(**args*, ***kwargs*)

Bases: *Conjunction*

Conjunction subclass that uses numpy arrays for set representation.

**all_set = None**

**append_and**(*to_append*)

Append a selector using logical AND and update the representation.

**Parameters**
**to_append** – Selector to append.

**compute_representation**()

Compute the numpy array representation of the conjunction.

**Returns**
Numpy array of indices representing the instances covered by the conjunction.

**property size_sg**

Size of the subgroup represented by the conjunction.

**class** pysubgroup.representations.**RepresentationBase**(*new_conjunction*, *selectors_to_patch*)

Bases: object

Base class for different representation strategies.

Provides methods to patch selectors and manage class-level patches. Can be used as a context manager to ensure patches are applied and removed properly.

**patch_all_selectors**()

Patch all selectors in the selectors_to_patch list.

**patch_classes**()

Patch the required classes.

Can be overridden by subclasses to patch class-level attributes or methods.

**patch_selector**(*sel*)

Patch a single selector.

This method should be implemented by subclasses.

**undo_patch_classes**()

Undo patches applied to classes.

Can be overridden by subclasses to remove class-level patches.

**class** pysubgroup.representations.**SetRepresentation**(*df*, *selectors_to_patch*)

Bases: *RepresentationBase*

Representation class that uses sets for selectors and conjunctions.

**Conjunction**

alias of *Set_Conjunction*

**patch_classes**()

Patch class-level attributes before entering the context.

**patch_selector**(*sel*)

Patch a selector by computing its set representation.

**Parameters**
**sel** – Selector to patch.

**class** pysubgroup.representations.**Set_Conjunction**(**args*, ***kwargs*)

Bases: *Conjunction*

Conjunction subclass that uses sets for representation.

**all_set = {}**

**append_and**(*to_append*)

Append a selector using logical AND and update the representation.

**Parameters**
**to_append** – Selector to append.

**compute_representation**()

Compute the set representation of the conjunction.

**Returns**
Set of indices representing the instances covered by the conjunction.

**property size_sg**

Size of the subgroup represented by the conjunction.

### pysubgroup.subgroup_description module

Created on 28.04.2016

@author: lemmerfn

class pysubgroup.subgroup_description.**BooleanExpressionBase**

Bases: ABC

Base class for boolean expressions (conjunctions and disjunctions).

**abstractmethod append_and**(*to_append*)

Append a selector or expression using logical AND.

**abstractmethod append_or**(*to_append*)

Append a selector or expression using logical OR.

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

Bases: *BooleanExpressionBase*

Conjunction of selectors (logical AND).

**append_and**(*to_append*)

Append a selector or conjunction using logical AND.

**append_or**(*to_append*)

Append a selector or expression using logical OR (not supported).

**covers**(*instance*)

Determine which instances are covered by the conjunction.

**Parameters**
**instance** – pandas DataFrame containing the data.

**Returns**
A boolean array indicating which instances are covered.

**property depth**

Return the number of selectors in the conjunction.

**static from_str**(*s*)

Create a Conjunction from a string representation.

**Parameters**
**s** – String representation of the conjunction.

**Returns**
A Conjunction instance.

**pop_and**()

Remove and return the last selector added using AND.

**pop_or**()

Pop operation for OR is not supported in Conjunction.

**property selectors**

Return the selectors in the conjunction as a tuple.

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

Bases: *Disjunction*

Disjunctive Normal Form expression.

**append_and**(*to_append*)

Append a selector using logical AND to all conjunctions.

**append_or**(*to_append*)

Append a selector or conjunction using logical OR.

**pop_and**()

Remove and return the last selector added using AND from all conjunctions.

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

Bases: *BooleanExpressionBase*

Disjunction of selectors (logical OR).

**append_and**(*to_append*)

Append a selector or expression using logical AND (not supported).

**append_or**(*to_append*)

Append a selector or disjunction using logical OR.

**covers**(*instance*)

Determine which instances are covered by the disjunction.

**Parameters**
**instance** – pandas DataFrame containing the data.

**Returns**
A boolean array indicating which instances are covered.

**property selectors**

Return the selectors in the disjunction as a tuple.

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

Bases: *SelectorBase*

Selector that checks for equality with a specific value.

**property attribute_name**

Name of the attribute.

**property attribute_value**

Value of the attribute to compare for equality.

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

Compute the descriptions (hash, query, string) for the selector.

**covers**(*data*)

Determine which instances in data are covered by this selector.

**Parameters**
**data** – pandas DataFrame containing the data.

**Returns**
A boolean array indicating which instances are covered.

**static from_str**(*s*)

Create an EqualitySelector from a string representation.

**Parameters**
**s** – String representation of the selector.

**Returns**
An EqualitySelector instance.

**property selectors**

Return the selector itself as a tuple (for compatibility).

**set_descriptions**(*attribute_name*, *attribute_value*, *selector_name=None*)

Set the descriptions (query, string, hash) for the selector.

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

Bases: *SelectorBase*

Selector that checks if a value is within an interval.

**property attribute_name**

Name of the attribute.

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

Compute the descriptions (hash, query, string) for the interval selector.

**classmethod compute_string**(*attribute_name*, *lower_bound*, *upper_bound*, *rounding_digits*)

Compute the string representation of the interval selector.

**covers**(*data_instance*)

Determine which instances are covered by this interval selector.

**Parameters**
**data_instance** – pandas DataFrame containing the data.

**Returns**
A boolean array indicating which instances are within the interval.

**static from_str**(*s*)

Create an IntervalSelector from a string representation.

**Parameters**
**s** – String representation of the interval selector.

**Returns**
An IntervalSelector instance.

**property lower_bound**

Lower bound of the interval (inclusive).

**property selectors**

Return the selector itself as a tuple (for compatibility).

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

Set the descriptions (hash, query, string) for the interval selector.

**property upper_bound**

Upper bound of the interval (exclusive).

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

Bases: *SelectorBase*

Selector that negates another selector.

**property attribute_name**

Name of the attribute.

**covers**(*data_instance*)

Determine which instances are not covered by the underlying selector.

**Parameters**
**data_instance** – pandas DataFrame containing the data.

**Returns**
A boolean array indicating which instances are not covered.

**property selectors**

Return the selector itself as a tuple (for compatibility).

**set_descriptions**(*selector*)

Set the descriptions (query, hash) for the negated selector.

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

Bases: ABC

Base class for selectors, ensuring each selector instance is unique.

**abstractmethod set_descriptions**(**args*, ***kwargs*)

Set the descriptions for the selector.

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

Create equality selectors for nominal attributes.

**Parameters**

- **data** – pandas DataFrame containing the data.
- **ignore** – List of attribute names to ignore.

**Returns**
List of EqualitySelector instances.

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

Create equality selectors for a nominal attribute.

**Parameters**

- **data** – pandas DataFrame containing the data.
- **attribute_name** – Name of the attribute.
- **dtypes** – Data types of the data columns.

**Returns**
List of EqualitySelector instances for the attribute.

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

Create selectors for numeric attributes.

**Parameters**

- **data** – pandas DataFrame containing the data.
- **nbins** – Number of bins to use for discretization.
- **intervals_only** – If True, only create interval selectors.
- **weighting_attribute** – Optional attribute for weighting.

- **ignore** – List of attribute names to ignore.

**Returns**
List of numeric selectors.

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

Create selectors for a numeric attribute.

**Parameters**

- **data** – pandas DataFrame containing the data.
- **attr_name** – Name of the attribute.
- **nbins** – Number of bins to use for discretization.
- **intervals_only** – If True, only create interval selectors.
- **weighting_attribute** – Optional attribute for weighting.

**Returns**
List of numeric selectors for the attribute.

pysubgroup.subgroup_description.**create_selectors**(*data*, *nbins=5*, *intervals_only=True*, *ignore=None*)

Create a list of selectors for all attributes in the data.

**Parameters**

- **data** – pandas DataFrame containing the data.
- **nbins** – Number of bins to use for numeric attributes.
- **intervals_only** – If True, only create interval selectors for numeric attributes.
- **ignore** – List of attribute names to ignore.

**Returns**
List of selectors.

pysubgroup.subgroup_description.**get_cover_array_and_size**(*subgroup*, *data_len=None*, *data=None*)

Compute the cover array and its size for a given subgroup.

**Parameters**

- **subgroup** – The subgroup for which to compute the cover array and size.
- **data_len** – Optional length of the data.
- **data** – Optional data.

**Returns**
Tuple of (cover array, size).

pysubgroup.subgroup_description.**get_size**(*subgroup*, *data_len=None*, *data=None*)

Compute the size of the cover array for a given subgroup.

**Parameters**

- **subgroup** – The subgroup for which to compute the size.
- **data_len** – Optional length of the data.

- **data** – Optional data.

**Returns**
Size of the cover array.

pysubgroup.subgroup_description.**pandas_sparse_eq**(*col*, *value*)

Compare a pandas sparse column to a value.

**Parameters**

- **col** – pandas Series with SparseArray data.
- **value** – The value to compare with.

**Returns**
A pandas SparseArray of booleans indicating where col equals value.

pysubgroup.subgroup_description.**remove_target_attributes**(*selectors*, *target*)

Remove selectors that are based on target attributes.

**Parameters**

- **selectors** – List of selectors.
- **target** – The target object with get_attributes method.

**Returns**
List of selectors not based on target attributes.

### pysubgroup.utils module

Created on 02.05.2016

@author: lemmerfn

**class** pysubgroup.utils.**BaseTarget**

Bases: object

Base class for defining targets in subgroup discovery.

Provides a method to check if all required statistics are present.

**all_statistics_present**(*cached_statistics*)

Checks if all required statistics are present in the cached statistics.

**Parameters**
**cached_statistics** (*dict*) – The dictionary of cached statistics.

**Returns**
True if all required statistics are present, False otherwise.

**Return type**
bool

**class** pysubgroup.utils.**SubgroupDiscoveryResult**(*results*, *task*)

Bases: object

Represents the result of a subgroup discovery task.

Contains methods to convert results to different formats.

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

Converts the results to a pandas DataFrame.

**Parameters**

- **statistics_to_show** (*list, optional*) – The statistics to include in the DataFrame.
- **autoround** (*bool*) – If True, automatically rounds numerical columns.
- **include_target** (*bool*) – If True, includes the target in the DataFrame.

**Returns**
A pandas DataFrame representing the results.

**Return type**
DataFrame

**to_descriptions**(*include_stats=False*)

Converts the results to a list of subgroup descriptions.

**Parameters**
**include_stats** (*bool*) – If True, includes statistics in the output.

**Returns**
A list of subgroup descriptions.

**Return type**
list

**to_latex**(*statistics_to_show=None*, *escape_underscore=True*)

Converts the results to a LaTeX-formatted table.

**Parameters**

- **statistics_to_show** (*list, optional*) – The statistics to include in the LaTeX table.
- **escape_underscore** (*bool*) – If True, escapes underscores in strings.

**Returns**
A string containing the LaTeX-formatted table.

**Return type**
str

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

Converts the results to a table format.

**Parameters**

- **statistics_to_show** (*list, optional*) – The statistics to include in the table.
- **print_header** (*bool*) – If True, includes a header row.
- **include_target** (*bool*) – If True, includes the target in the table.

**Returns**
A list of rows representing the table.

**Return type**
list

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

Adds a subgroup to the result set if it meets the required quality and constraints.

> **Important**
>
> Only add/remove subgroups from *result* by using *heappop* and *heappush* to ensure order of subgroups by quality.

**Parameters**

- **result** (*list*) – The current list of subgroups (heap).
- **sg** – The subgroup to potentially add.
- **quality** (*float*) – The quality of the subgroup.
- **task** (SubgroupDiscoveryTask) – The task containing parameters and constraints.
- **check_for_duplicates** (*bool*) – If True, checks for duplicates before adding.
- **statistics** (*optional*) – Precomputed statistics for the subgroup.
- **explicit_result_set_size** (*int, optional*) – Overrides the task's result_set_size.

**Returns**
None

pysubgroup.utils.**conditional_invert**(*val*, *invert*)

Conditionally inverts a value based on a boolean flag.

**Parameters**

- **val** (*float*) – The value to potentially invert.
- **invert** (*bool*) – If True, the value is inverted.

**Returns**
The (possibly inverted) value.

**Return type**
float

pysubgroup.utils.**count_bits**(*bitset_as_int*)

Counts the number of set bits (1s) in a bitset represented as an integer.

**Parameters**
**bitset_as_int** (*int*) – The bitset represented as an integer.

**Returns**
The number of set bits.

**Return type**
int

pysubgroup.utils.**create_subgroup_with_representation**(*data*, *selectors*)

Create an object representing the conjunction of the given selectors, including a bitmask indicating which instances in the dataset are covered.

**Parameters**

- **data** – dataset to evaluate the cover on
- **selectors** – list of selectors to form the conjunction

pysubgroup.utils.**derive_effective_sample_size**(*weights*)

Calculates the effective sample size for weighted data.

**Parameters**
**weights** (*array-like*) – The weights assigned to the samples.

**Returns**
The effective sample size.

**Return type**
float

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

Discretizes a numerical attribute into bins with approximately equal frequency.

**Parameters**

- **data** (*DataFrame*) – The dataset containing the attribute to discretize.
- **attribute_name** (*str*) – The name of the attribute to discretize.
- **nbins** (*int*) – The number of bins to create.
- **weighting_attribute** (*str, optional*) – An optional attribute to weight the instances.

**Returns**
A list of cutpoints defining the bins.

**Return type**
list

pysubgroup.utils.**find_set_bits**(*bitset_as_int*)

Finds the indices of set bits in a bitset represented as an integer.

**Parameters**
**bitset_as_int** (*int*) – The bitset represented as an integer.

**Yields**
*int* – The index of each set bit.

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

Formats a float to a specified number of decimal places.

**Parameters**

- **x** (*float*) – The value to format.
- **digits** (*int*) – The number of decimal places.

**Returns**
The formatted string.

**Return type**
str

pysubgroup.utils.**intersect_of_ordered_list**(*list_1*, *list_2*)

Computes the intersection of two ordered lists.

**Parameters**

- **list_1** (*list*) – The first ordered list.
- **list_2** (*list*) – The second ordered list.

**Returns**
The intersection of the two lists.

**Return type**
list

pysubgroup.utils.**is_categorical_attribute**(*data*, *attribute_name*)

Determines if an attribute in the dataset is categorical.

**Parameters**

- **data** (*DataFrame*) – The dataset.
- **attribute_name** (*str*) – The name of the attribute.

**Returns**
True if the attribute is categorical, False otherwise.

**Return type**
bool

pysubgroup.utils.**is_numerical_attribute**(*data*, *attribute_name*)

Determines if an attribute in the dataset is numerical.

**Parameters**

- **data** (*DataFrame*) – The dataset.
- **attribute_name** (*str*) – The name of the attribute.

**Returns**
True if the attribute is numerical, False otherwise.

**Return type**
bool

pysubgroup.utils.**minimum_required_quality**(*result*, *task*)

Determines the minimum quality required for a subgroup to be considered for inclusion in the result set.

**Parameters**

- **result** (*list*) – The current list of subgroups (heap).
- **task** (SubgroupDiscoveryTask) – The task containing parameters like
- **min_quality.** (*result_set_size and*)

**Returns**
The minimum required quality for a subgroup to be added to the result set.

**Return type**
float

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

Calculates the Jaccard similarity between two subgroups based on their coverage.

**Parameters**

- **sg** – The first subgroup.
- **another_sg** – The second subgroup.
- **data** (*DataFrame*) – The dataset.

**Returns**
The Jaccard similarity between the two subgroups.

**Return type**
float

pysubgroup.utils.**perc_formatter**(*x*)

Formats a float as a percentage string with one decimal place.

**Parameters**
**x** (*float*) – The value to format.

**Returns**
The formatted percentage string.

**Return type**
str

pysubgroup.utils.**powerset**(*iterable*, *max_length=None*)

Generates the power set (all possible combinations) of an iterable up to a maximum length.

**Parameters**

- **iterable** (*iterable*) – The iterable to generate combinations from.
- **max_length** (*int, optional*) – The maximum length of combinations.

**Returns**
An iterator over the power set of the iterable.

**Return type**
iterator

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

Filters and sorts the result set of subgroups according to the task parameters.

**Parameters**

- **result** (*list*) – The list of subgroups (heap).
- **task** (SubgroupDiscoveryTask) – The task containing parameters like result_set_size and min_quality.

**Returns**
The filtered and sorted list of subgroups.

**Return type**
list

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

Removes selectors that are based on specified attributes.

**Parameters**

- **selector_list** (*list*) – The list of selectors to filter.
- **attribute_list** (*list*) – The list of attribute names to remove selectors for.

**Returns**
The filtered list of selectors.

**Return type**
list

pysubgroup.utils.**results_df_autoround**(*df*)

Automatically rounds numerical columns in a DataFrame for better readability.

**Parameters**
**df** (*DataFrame*) – The DataFrame containing the results.

**Returns**
The DataFrame with rounded numerical values.

**Return type**
DataFrame

pysubgroup.utils.**str_to_bool**(*s*)

Converts a string representation of a boolean value to a boolean type.

**Parameters**
**s** (*str*) – The string to convert (e.g., 'true', 'False', '1', '0').

**Returns**
The boolean value represented by the string.

**Return type**
bool

**Raises**
**ValueError** – If the string does not represent a valid boolean value.

pysubgroup.utils.**to_bits**(*list_of_ints*)

Converts a list of integers to a bitset represented as an integer.

**Parameters**
**list_of_ints** (*list*) – The list of integers to convert.

**Returns**
The bitset represented as an integer.

**Return type**
int

### pysubgroup.visualization module

pysubgroup.visualization.**plot_distribution_numeric**(*sg*, *target*, *data*, *bins*, *show_dataset=True*)

pysubgroup.visualization.**plot_npspace**(*result_df*, *data*, *annotate=True*, *fixed_limits=False*)

pysubgroup.visualization.**plot_qualities_on_sample_distribution**(*result*, *qualities*, *samples*, *alpha=0.05*, *side: Literal['left', 'right'] = 'right'*, *bins=25*)

Create plots of the empirical sample distribution as a histogram for each subgroup. Include indicators for the subgroup quality and the quality that the alpha threshold corresponds to.

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

pysubgroup.visualization.**plot_sgbars**(*result_df*, ** (Keyword-only parameters separator (PEP 3102))*, *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

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## p

# INDEX

## C

## D



## E

## F



## G

## N



## O



## P

## R



## S

## T



## U

## V



## W